Institut für Technik der Informationsverarbeitung (ITIV)

Handbuch zum Praktikum Informationstechnik 1

Stand : June 15, 2023

Institutsleitung:
Prof. Dr.-Ing Dr. h. c. J. Becker
Prof. Dr.-Ing. E. Sax
Prof. Dr. rer. nat. W. Stork

Contents

1 Vorwort

Herzlich Willkommen in Ihrem Team zur Entwicklung einer neuen Software zur Steuerung eines MagniSilver™. Dabei handelt es sich um eine fahrbarer Roboterplattform entwickelt von Ubiquity Robotics.
Ihre Aufgabe in den nächsten Wochen wird die Planung des Vorgehens, sowie die Umsetzung und das Testen von Software für den MagniSilver™ sein. Da es sich beim MagniSilver™ um ein neues Produkt handelt und Sie sich zunächst mit diesem vertraut machen müssen, stehen Ihnen erfahrene Entwickler (Tutoren) zur Seite.

Einige Softwarekomponenten, wie zum Beispiel der Regler des MagniSilver™, sind bereits vollständig implementiert. Andere Komponenten, wie beispielsweise die verwendeten Hardwaremodule (Motoren, Sensoren, ...) müssen durch Sie angesteuert werden.

Wir wünschen Ihnen bei dieser Aufgabe viel Spaß und freuen uns auf die nächsten Wochen!

2 Inhalt des Praktikums

2.1 Über dieses Handbuch

Dieses Dokument ist in vier große Bereiche unterteilt. Im ersten Teil erfahren Sie alles Nötige zum Aufbau und Ablauf des Praktikums. Anschließend, im Kapitel 3 lernen sie die Programme und Hardware kennen, die Sie zur erfolgreichen Bearbeitung des Praktikums benötigen. Mit diesem Wissen sind Sie dann bereit für die eigentliche Aufgabenstellung, die Sie im gleichnamigen Kapitel 4 finden. Zum anschließenden Testen und Debuggen ihres Programmcodes finden sie hilfreiche Tipps in Kapitel 5.

Dieses Handbuch ist lang und es ist gewachsen, um Ihnen einen besseren Überblick zu geben. Im Gegensatz zu den Datenblättern ist in diesem Dokument jedoch alles früher oder später für das Praktikum relevant. Nehmen Sie sich also die Zeit und arbeiten Sie es genau durch, damit Sie später wissen, wo Sie was finden. Wir versuchen Ihnen das Leben leichter zu machen, indem wir stets auf die entsprechenden Stellen in den Datenblättern verweisen. In den Datenblättern finden Sie viele Informationen, die Sie für dieses Praktikum nicht benötigen werden. Ziehen Sie diese daher nur bei konkreten Unklarheiten zu Rate! Tun Sie dies bitte jedoch bevor Sie in die Sprechstunde kommen, um den Andrang in den Sprechstunden in Grenzen zu halten.

2.2 Aufbau und Ablauf

Das Praktikum besteht aus folgenden Komponenten:

- Einführungsveranstaltungen
  Die Einführungsveranstaltungen dienen dazu, Ihnen den Einstieg in den Workshop zu erleichtern. Das Praktikum nutzt eine Reihe von Tools, mit denen Sie bisher noch nicht arbeiten mussten. Wie Sie diese richtig einrichten und damit arbeiten können, wird Ihnen in den Einführungsveranstaltungen vorgeführt. Daher gilt unsere dringende Empfehlung, die Einführungsveranstaltungen mit Ihrem Notebook zu besuchen. So können Sie der Einführung direkt folgen. Die Sprechstunden sind kein Ersatz!

- Selbstständige Bearbeitung aller Aufgaben
  Es gibt keine Pflichttermine, an denen Sie die Aufgaben bewältigen müssen. Je früher Sie aber anfangen, desto bessere können Sie bei Problemen Fragestunden wahrnehmen.

- Mündliche Prüfung nach der Abgabe
  Nach Ende des Bearbeitungszeitraums werden alle GitLab Projekte gesperrt. In der auf die Abgabe folgende Woche werden Sie als Gruppe zu den Praktikumsthemen befragt. Zusätzlich wird mit Ihnen ihr Quellcode durchgegangen und überprüft, ob dieser den Programmierrichtlinien entspricht. Beachten Sie, dass es nach Abgabe keine Korrekturmöglichkeiten gibt. Jedes Gruppenmitglied wird nur zu den eigens bearbeiteten Klassen befragt. (Mindestens eine Grundklasse + Mitarbeit an Gruppenaufgabe)

- Probefahrt
  Das Praktikum schließt mit der freiwilligen Probefahrt ab. Nachdem Sie wochenlang Ihr MagniSilver™ Programm entwickelt haben, dürften Sie die Ergebnisse auf dem Produkt MagniSilver™ ausgiebig testen.

2.3 Wichtige Informationen, Termine und Fristen

Notwendige Dokumente wie dieses Handbuch, sowie organisatorische und inhaltliche Informationen finden sie hier: https://git.scc.kit.edu/pit/infos.
Eine volle Übersicht der Inhalte sowie eine Erklärung was GitLab genau ist, finden Sie im Kapitel Vorbereitung.

Bevor Sie mit der Bearbeitung des Praktikums starten möchten wir Sie noch auf ein paar organisatorische Punkte zwecks Bewertung aufmerksam machen.

1. Jedes Gruppenmitglied bearbeitet genau eine der drei Grundklassen (PWM, UART, Distance) und einen Teil der Gruppenaufgabe (Steuerung, Batteriespannungsüberwachung). Die Bearbeitung von weniger (z.B. nur einer Grundklasse) ist nicht ausreichend um das Praktikum zu bestehen! Auch die alleinige Bearbeitung der Steuerung oder Batteriespannungsüberwachung zählt nicht als ausreichende Bearbeitung!

2. Das Debuggen des Codes ist ein Teil der Aufgabe. Das ledigliche ,,Runterschreiben" des Codes ohne Test zählt nicht als erfolgreiche Bearbeitung der Aufgabe! Sie haben genügend Zeit für die Bearbeitung der Aufgabe und falls Sie Hilfe benötigen können Sie gerne einen Tutor zu Rate ziehen.

3. Die Zusammenarbeit mit anderen Gruppen und gar das Kopieren des Quellcodes von anderen Gruppen (oder dem Internet) ist nicht gestattet und ist ein Täuschungsversuch! Dies führt zu sofortigem Nichtbestehen des Praktikums. Das Kopieren von Code ist ein Plagiat und kann nach KIT Richtlinien zur Exmatrikulation führen!

4. Dokumentation und Projektmanagement sind auch Teil dieser gemeinsamen Programmieraufgabe. Der Code ist also verpflichtend zu kommentieren und ein Projektablaufplan mit Aufgabenzuteilungen im Repository abzulegen.

3 Vorbereitung

3.1 GitLab

3.1.1 Was ist Git?

Vereinfacht gesagt ist Git die Cloud für Programmierer. Sobald mehrere Entwickler am gleichen Quellcode (der in einem sog. „Repository“ liegt) arbeiten, ist es nötig, die Beiträge („Commits“) der einzelnen Beteiligten zu verwalten und zusammenzuführen. Zusätzlich ermöglicht Git beliebig viele Entwicklungszweige, „Branches“ genannt. So können beispielsweise in einem Zweig experimentelle Funktionen getestet werden. Sobald sich diese Funktionen in einem stabilen Zustand befinden, können sie mit einem anderen Branch zusammengeführt werden („Merge“). Dabei werden die Unterschiede beider Quellcodes verglichen und die jeweils aktuellsten Teile übernommen. Sehr wichtig und praktisch ist auch, dass jeder Commit, also jede Änderung am Quellcode gespeichert wird. So bleibt immer und uneingeschränkt die Möglichkeit zu einer beliebigen vorherigen Version zurückzukehren.

Git bietet – auch dank der vielen Erweiterungen – noch zahlreiche weitere Funktionen. Für die tägliche Nutzung ist Git als Kommandozeilenprogramm aber eher umständlich, weshalb vielfältige auf Git aufbauende Plattformen frei verfügbar sind. In erster Hinsicht stellen diese Plattformen einen Server zur Verfügung mit dem Entwickler Quellcode synchronisieren können. Zudem wird eine Benutzeroberfläche zur Verfügung gestellt. So ziemlich alle Plattformen bieten ein „Issue“-System (ein Issue wird auch „Ticket“ genannt). Dabei handelt es sich um eine Art Forum für das Repository mit dem auf Bugs und mögliche Verbesserungen hingewiesen und darüber diskutiert werden kann. Das Repository mit den Quelldateien, das Issue-System und eventuelle weitere Funktionen bilden zusammen ein Projekt.

Den Kern, nämlich Git selbst, rühren all diese Plattformen nicht an. Das führt dazu, dass i.d.R. jeder Client (also das Programm auf Ihrem Rechner, mit dem Sie den Code hoch- und runterladen) mit jeder Plattform kompatibel ist. Das am KIT verwendete GitLab, welches wir im Praktikum verwenden, richtet sich an die Softwareentwicklung innerhalb von Firmen. Es unterstützt gängige Benutzerkontenverwaltungen, weshalb Sie sich mit Ihrem KIT Account anmelden können. Im Rahmen dieses Praktikums brauchen Sie nur einen Bruchteil der verfügbaren Funktionen. Sie müssen auch nur das Mindeste einrichten; soweit wie möglich wurde alles für Sie vorkonfiguriert.

3.1.2 Wie nutzt das Praktikum GitLab?

Das gesamte Praktikum befindet sich auf GitLab in der Gruppe Informationstechnik 1 Praktikum, in der Sie drei Inhalte sehen: die öffentlichen Repositories Dokumente und Infos und Playground, sowie die Gruppe Gruppen, in der Sie das Projekt Ihrer Gruppe (mit Ihrem Quellcode) finden. In dieser Gruppe liegen – ausschließlich für die Tutoren sichtbar – die Projekte aller Studierendengruppen.

3.1.3 Wie kann ich Git benutzen?

Um GitLab zu nutzen bedarf es der Einrichtung Ihres GitLab-Kontos, sowie eines Programmes, mit dem Sie das Repository Ihrer Gruppe mit Ihrem Rechner synchronisieren können. Ihr GitLab-Konto wird bei der ersten Anmeldung automatisch erstellt. Besuchen Sie dazu https://git.scc.kit.edu und melden Sie sich mit Ihrem SCC-Konto an.

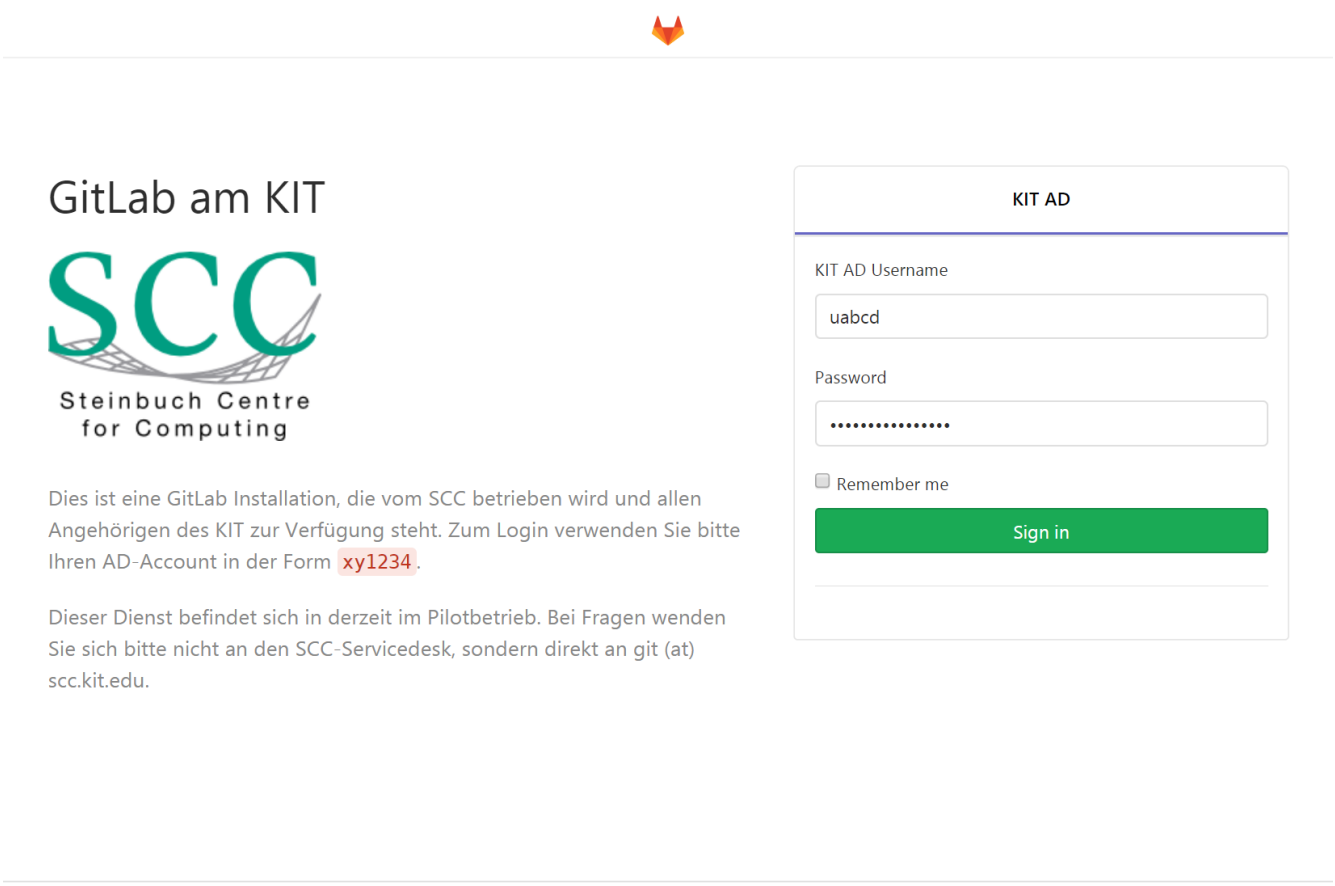

Figure 1: GitLab Anmeldeseite

Es wird automatisch ein GitLab-Konto für Sie angelegt und Sie werden auf die Startseite weitergeleitet.

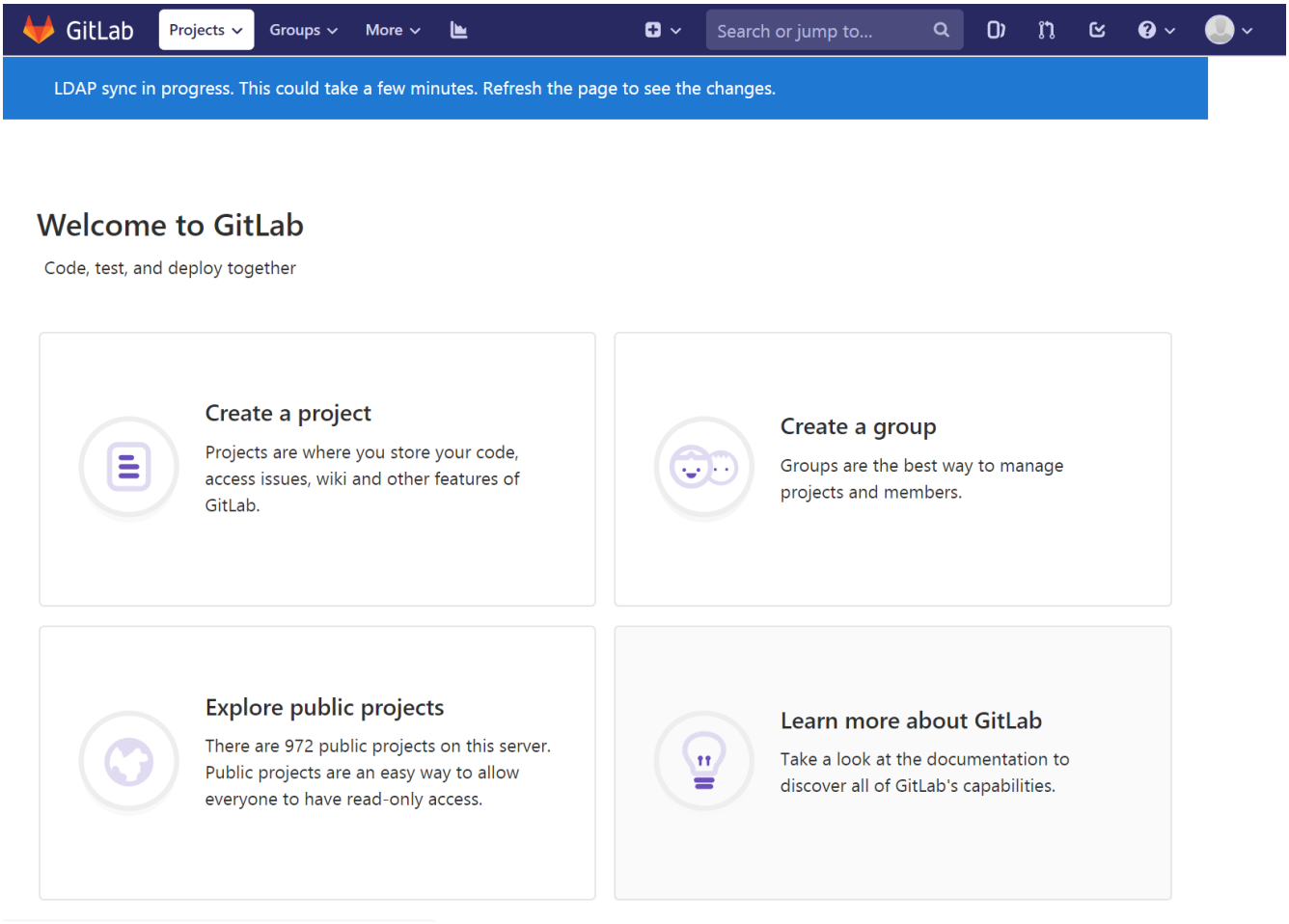

Figure 2: GitLab nach Anmeldung

Sollten Sie eine Warnung bezüglich SSH erhalten, können Sie sie ignorieren, da Sie diese Funktion nicht benötigen.

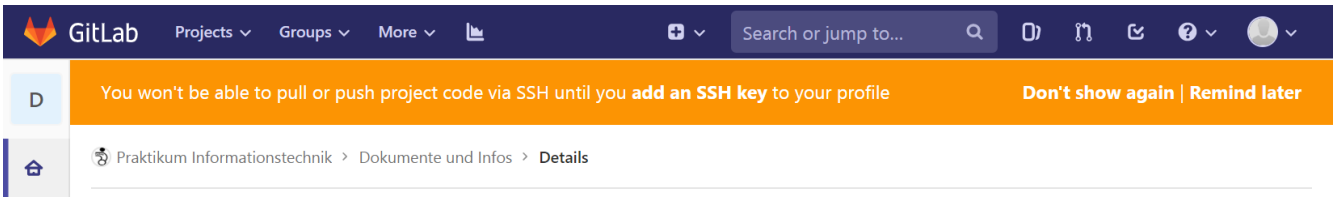

Figure 3: SSH Key Warnung GitLab

Über das Auswahlmenü oben rechts kommen Sie in die Einstellungen.

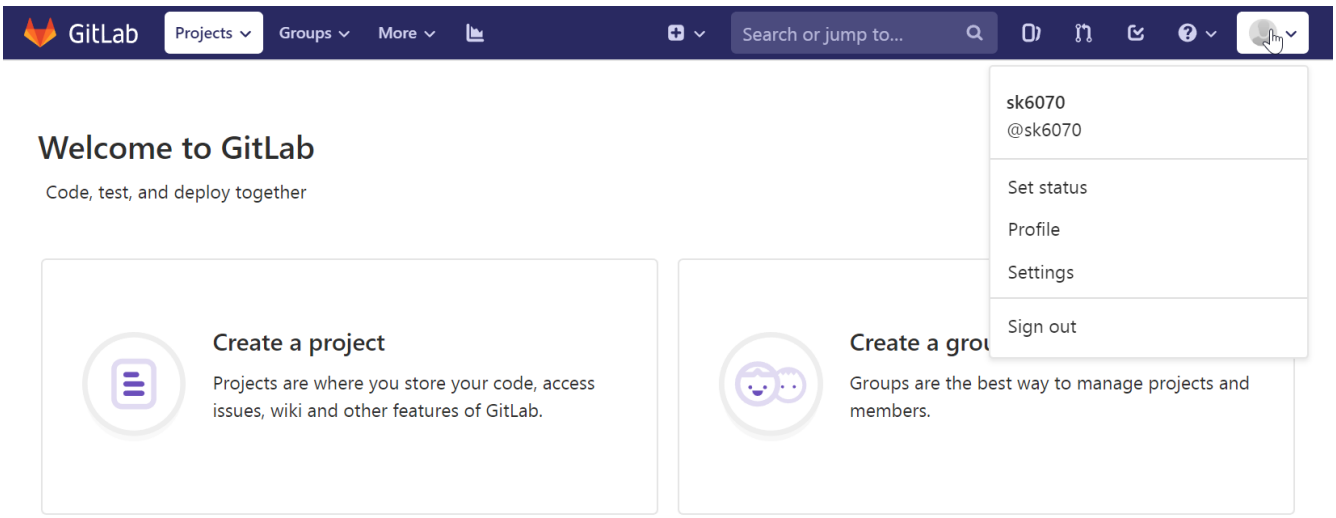

Figure 4: Zugang zu den Einstellungen in GitLab

Dort können Sie u.a. Ihren Nutzernamen, oder Ihr Profilbild ändern. In der Leiste rechts finden Sie zahlreiche weitere Einstellungsseiten. Unter ,,Preferences“ können Sie z.B. das Aussehen und die Sprache von GitLab ändern.

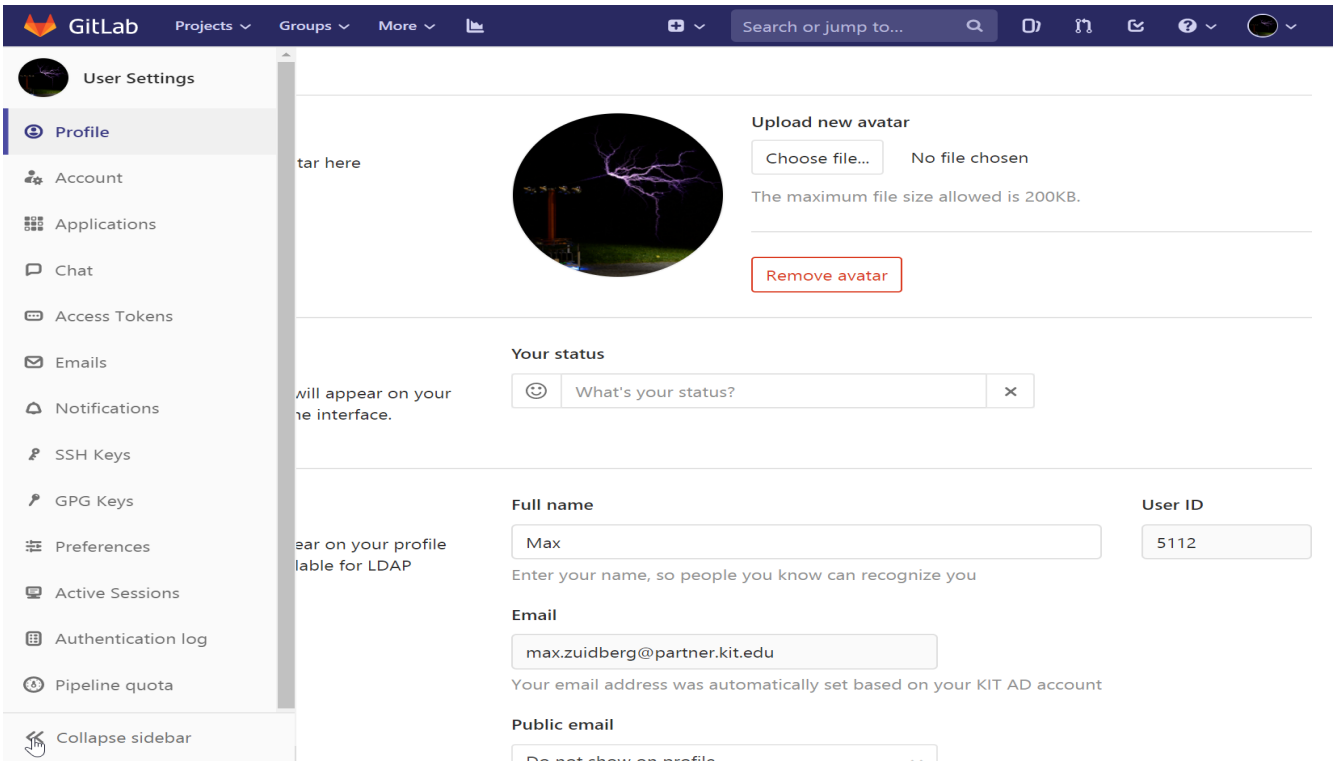

Figure 5: GitLab Einstellungen

Gehen Sie zu https://git.scc.kit.edu/pit und aktivieren Sie sämtliche Benachrichtigungen für die Gruppe („Watch“).

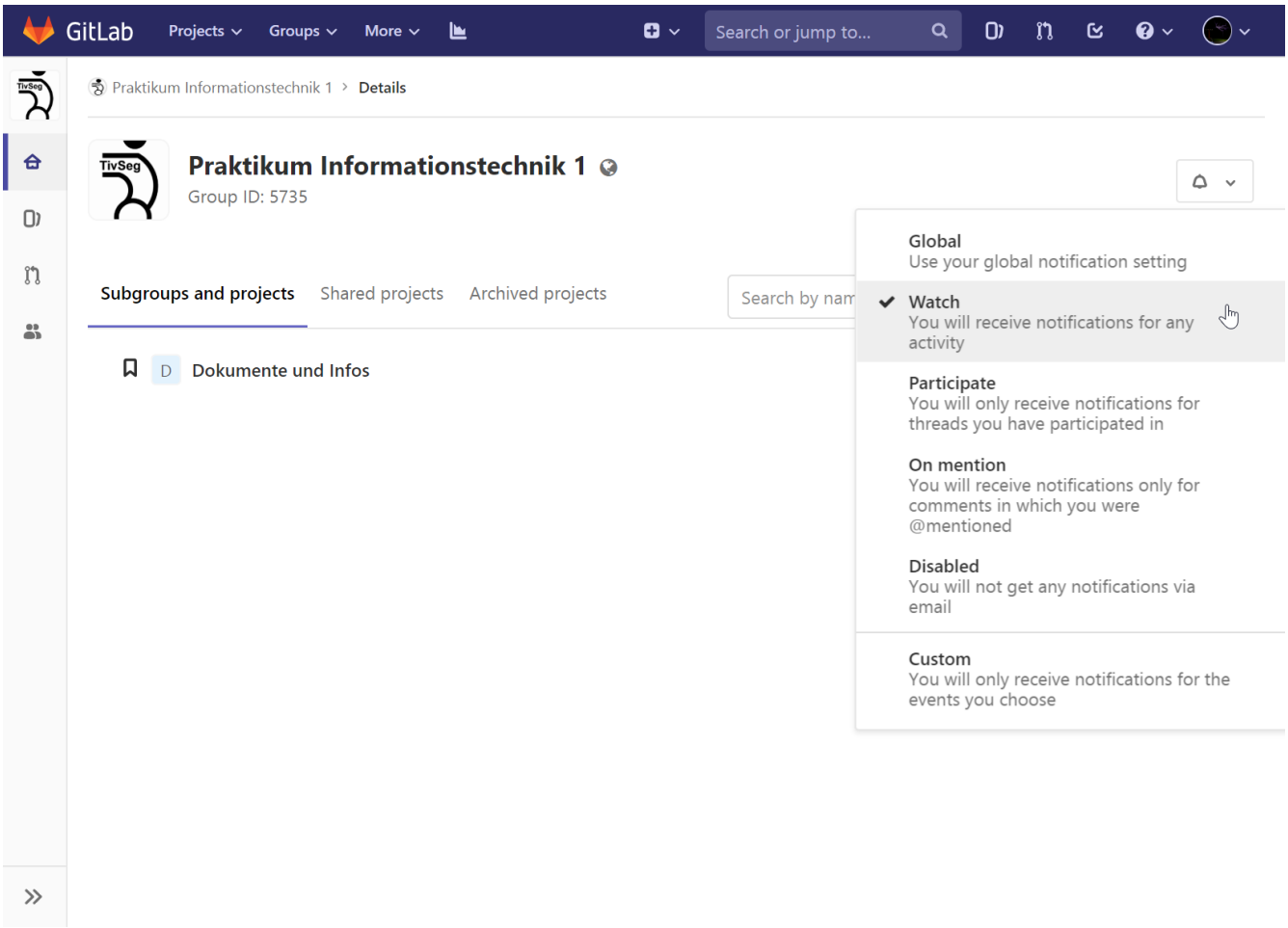

Figure 6: GitLab Benachrichtigungen aktivieren

Nach der Gruppeneinteilung finden Sie in der Praktikumsgruppe (siehe oben) ein fertig eingerichtetes Git-Projekt für Ihre Gruppe. Für die Synchronisierung dieses Repository benötigen Sie den unter „Clone“ angezeigten HTTPS-Link.

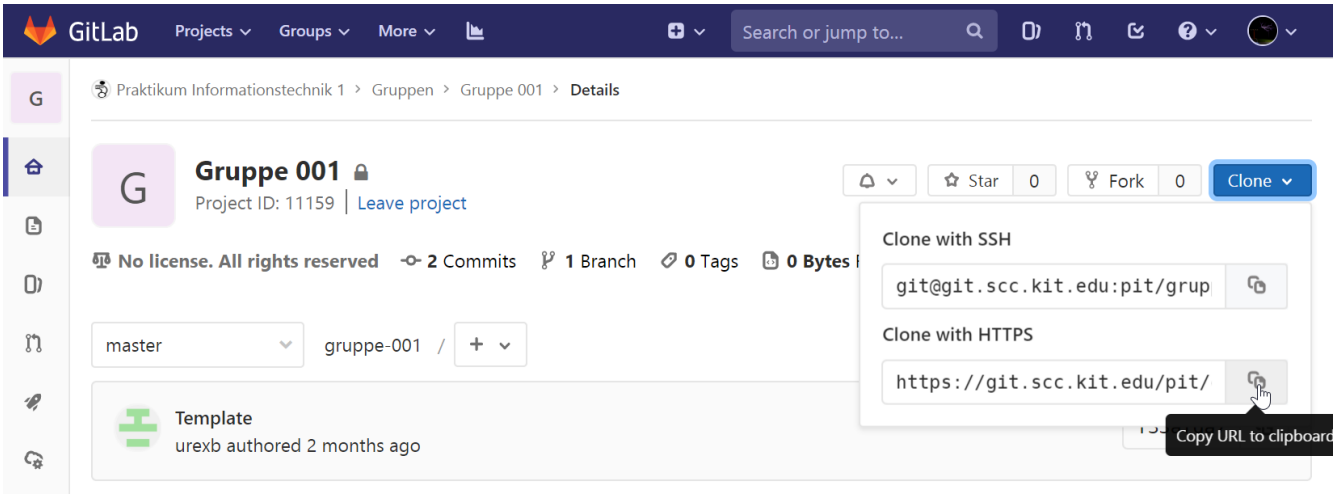

Figure 7: HTTPS Link für Clone Vorgang in GitLab

3.1.4 Github Desktop

Von den zahlreichen verfügbaren Clients sticht ein Gratis-Programm durch seine Einfachheit hervor: GitHub Desktop. Wie der Name bereits verrät, ist dieser Client eigentlich zum Arbeiten mit der Plattform GitHub gedacht, kann aber auch mit beliebigen anderen Plattformen arbeiten.

Sie können das Programm für macOS und Windows herunterladen: `https://desktop.github.com/`. Vorschläge für Linux Benutzer gibt es im entsprechenden GitLab Issue.

Bei der Installation werden Sie nach einem GitHub-Konto gefragt. Sofern Sie keins haben, oder keines angeben wollen, überspringen Sie diesen Schritt.

Ohne Konto werden Sie gebeten anzugeben, unter welchem Namen und welcher E-Mail-Adresse Ihre Commits eingereicht werden sollen. Sowohl Name als auch E-Mail-Adresse sollten identisch mit den Angaben auf GitLab sein, müssen es aber nicht.

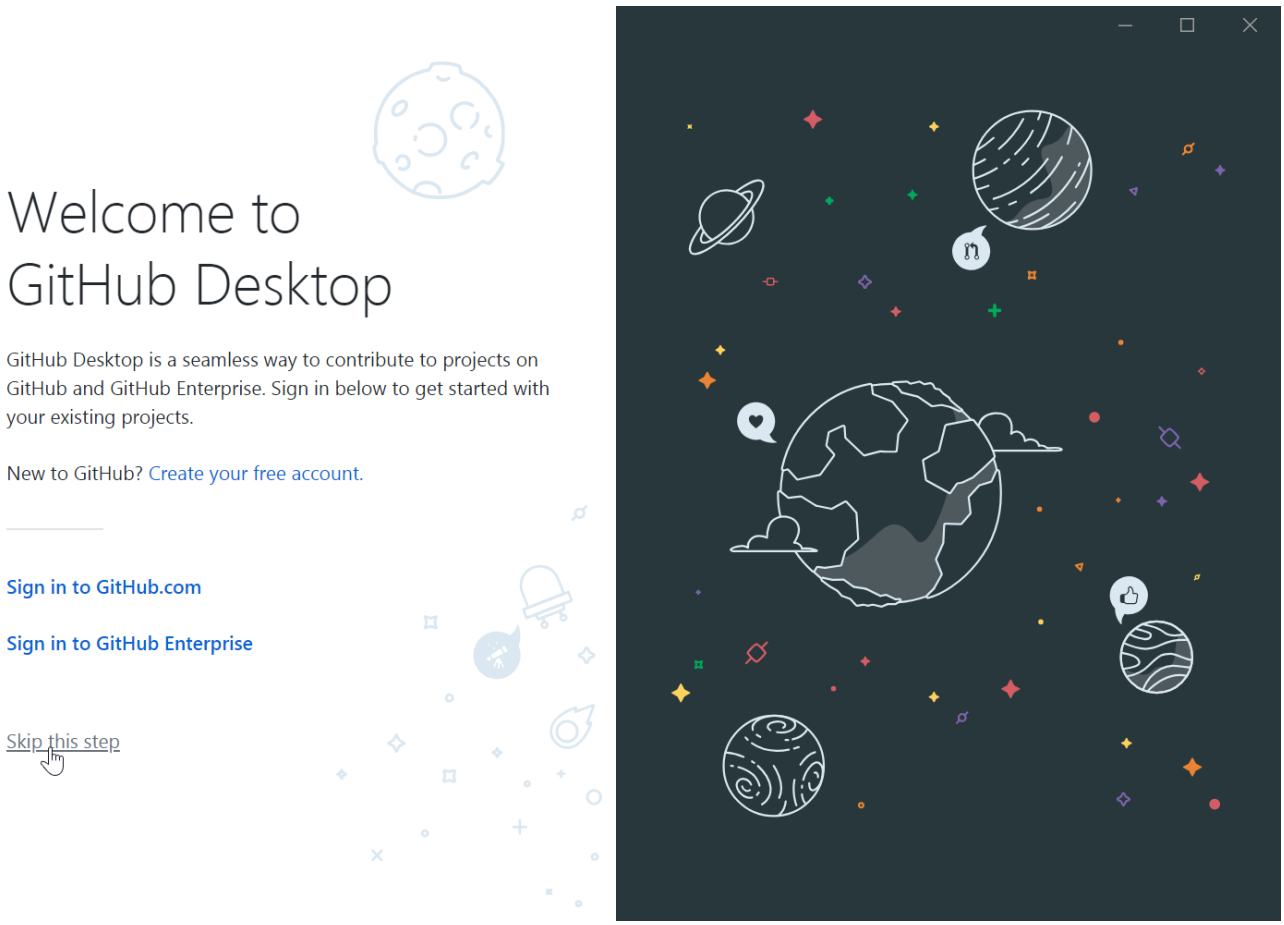

Figure 8: GitHub Desktop Konfiguration

Abschließend können Sie noch entscheiden, ob Sie Nutzungsstatistiken an GitHub übermitteln wollen oder nicht.
Nach der Installation gelangen Sie auf die Startseite von GitHub Desktop. Dort wählen Sie ,,Clone a repository from the Internet. . . “.

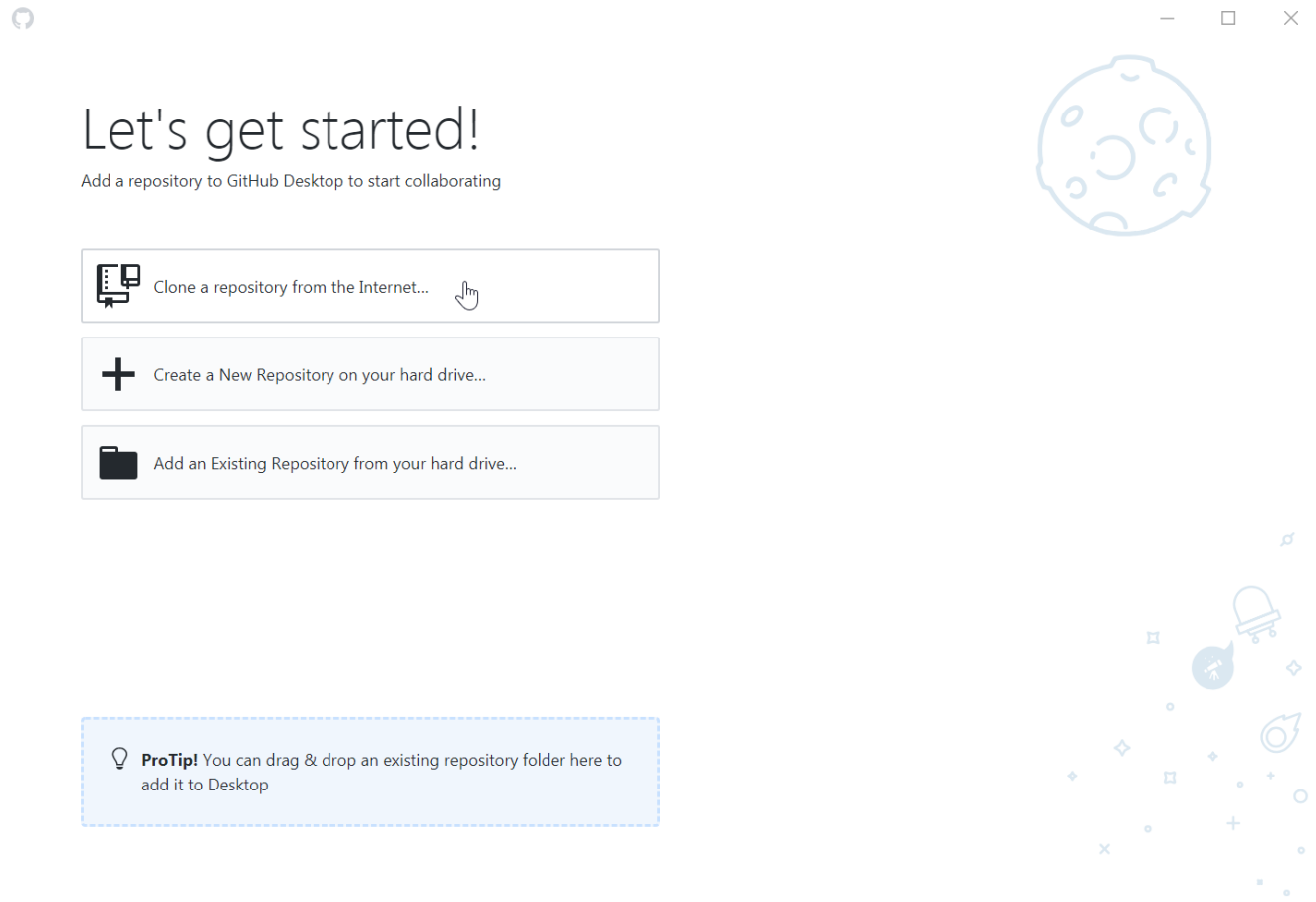

Figure 9: Klonen eines Repository in GitHub Desktop

In dem aufkommenden Fenster wählen Sie den Tab ,,URL“ und geben den zuvor bei Ihrem GitLab-Projekt kopierten Link ein. Sie können ebenfalls festlegen, wo das Repository gespeichert werden soll.

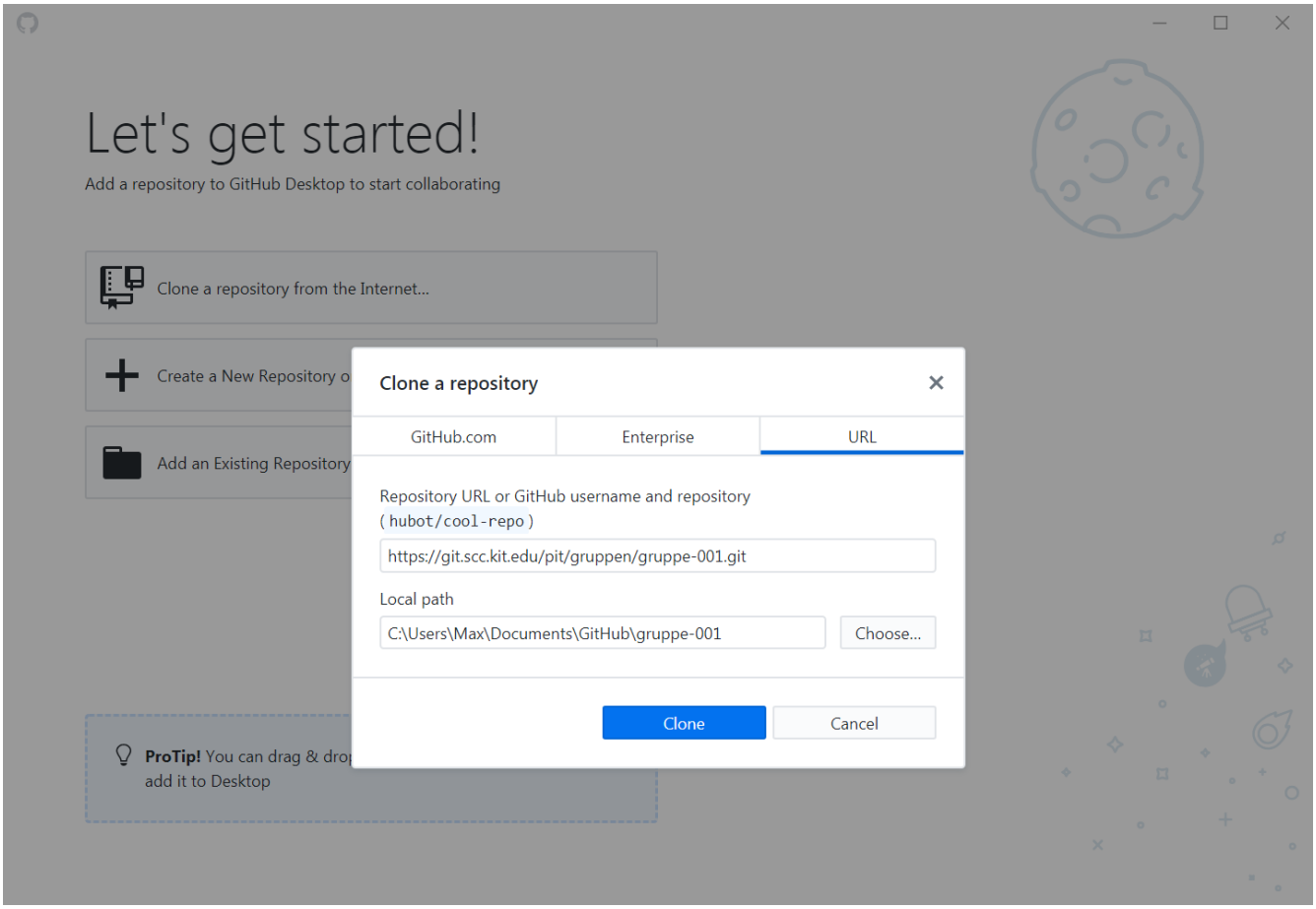

Figure 10: URL für Clone Prozess in GitHub Desktop eingeben

Kurz darauf wird GitHub Desktop nach Login Daten fragen. Geben Sie Ihre GitLab Anmeldedaten, also Ihr SCC-Konto ein.

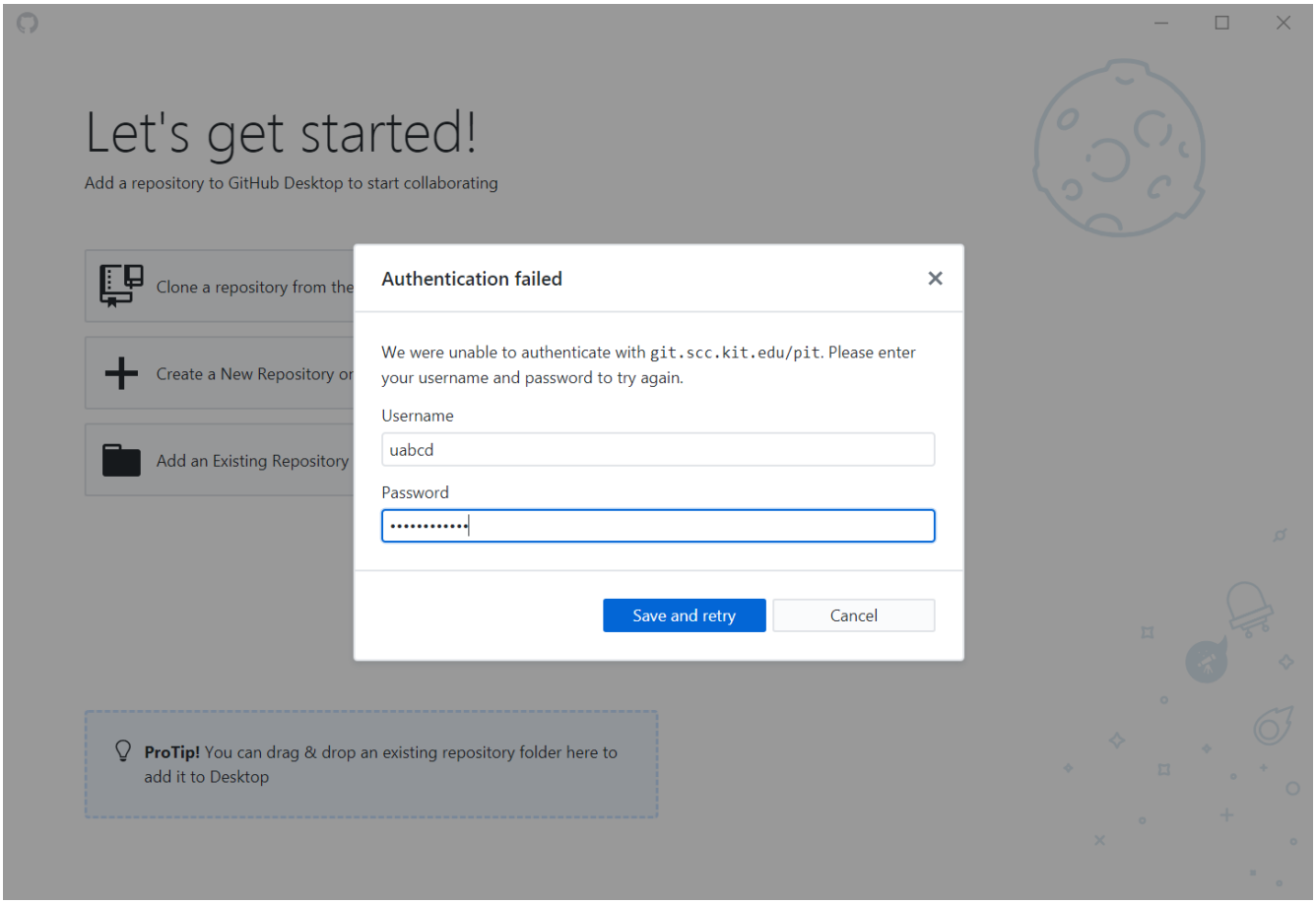

Figure 11: Eingabe GitLab Anmeldedaten in GitHub Desktop

Wenn Sie es bevorzugen, Ihr Passwort nicht anzugeben, können Sie in den GitLab Einstellungen auch einen sogenannten Token erstellen, und diesen anstelle des Passwortes angeben. Den Token können Sie unter ,,Settings → Access Token“ erstellen. Geben Sie ihm einen Namen, ggf. ein Ablaufdatum und setzen Sie ein Häkchen bei ,,api“. Anschließend können Sie Ihn zum Anmelden verwenden und jederzeit widerrufen.

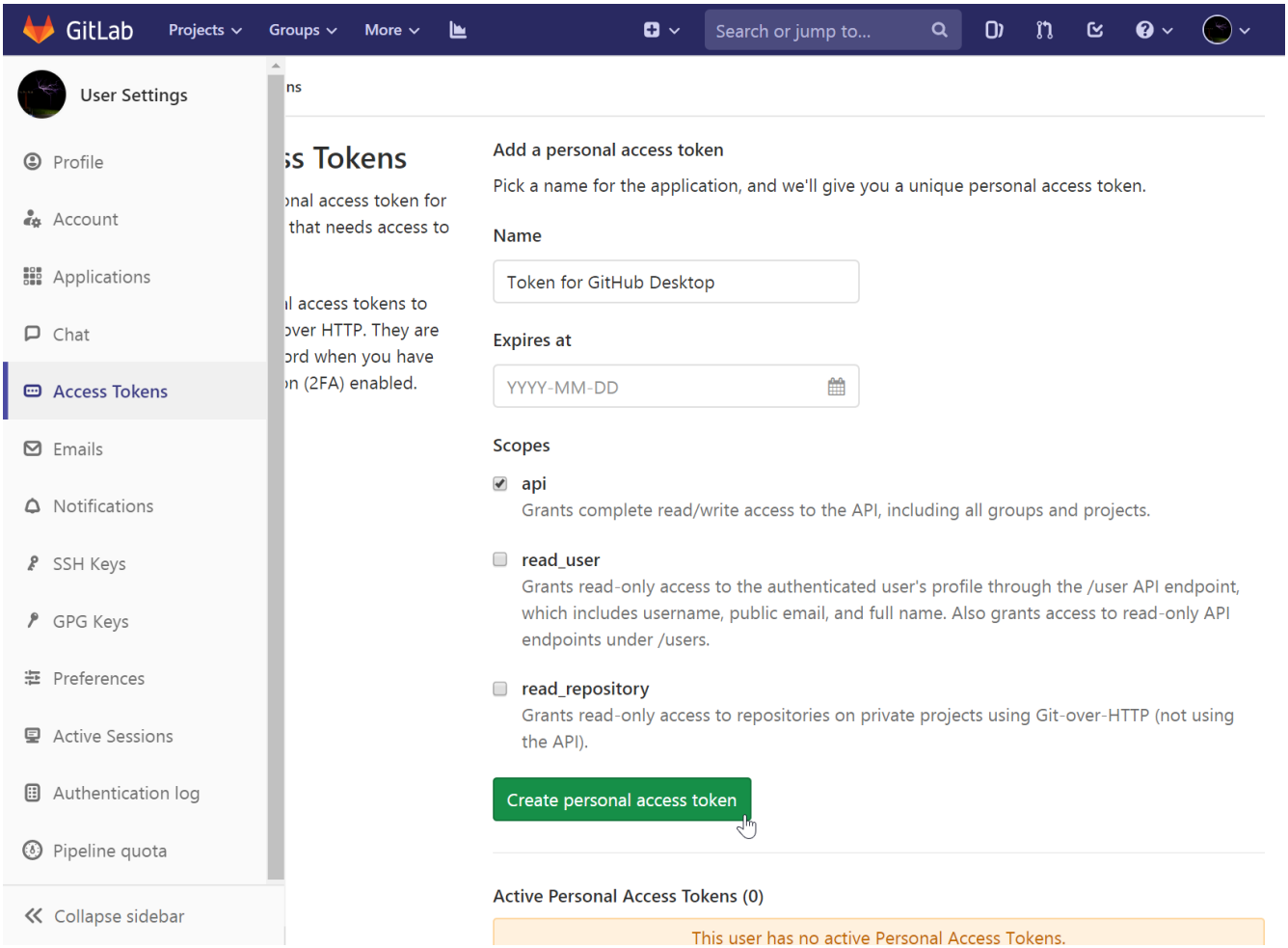

Figure 12: (Optional) Erstellen eines Access Tokens in GitLab

Nach Bestätigung Ihrer Login Daten synchronisiert GitHub Desktop das Repository und Sie gelangen zur Übersichtsseite.

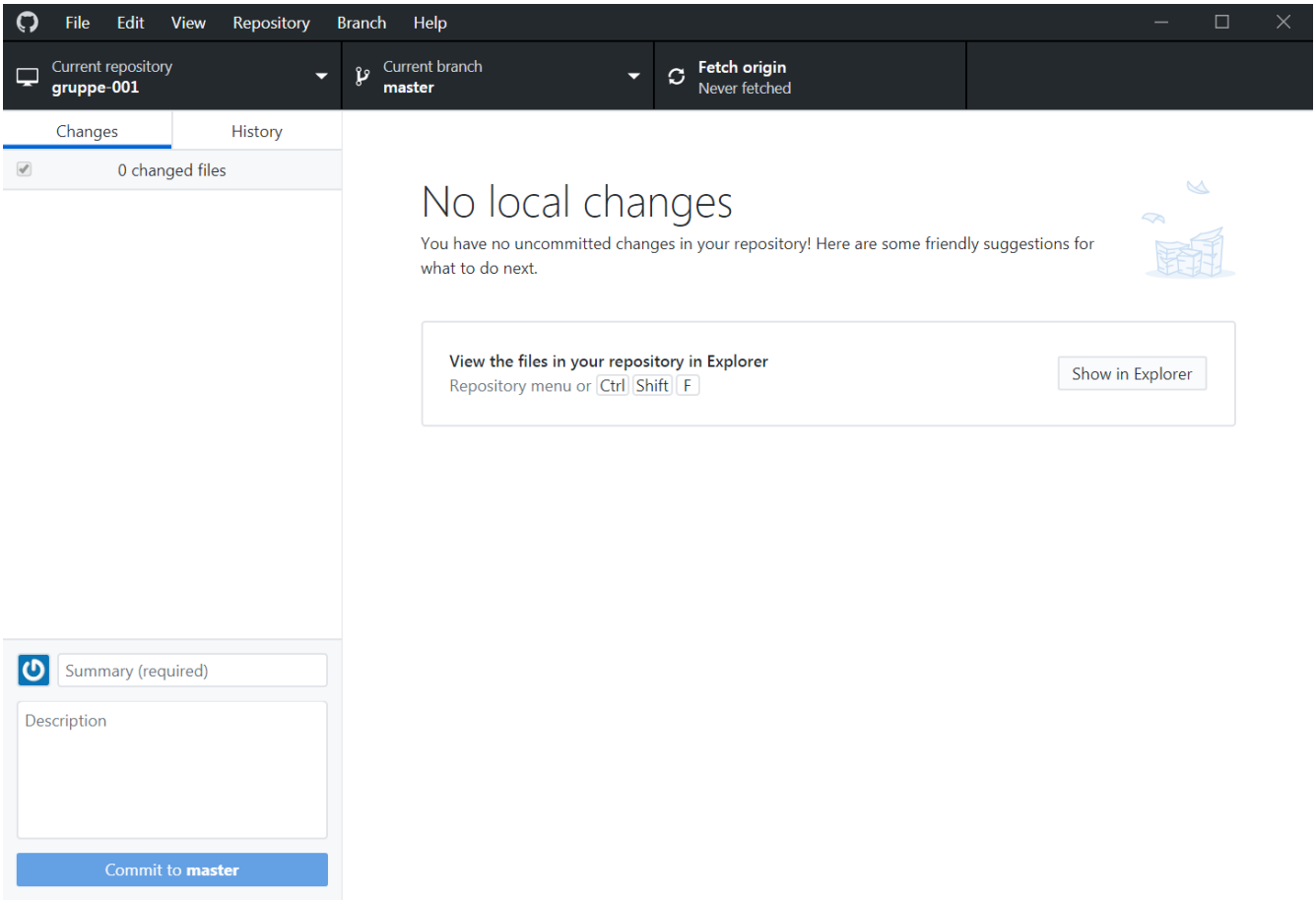

Figure 13: Übersichtsseite GitHub Desktop

Dieser Vorgang lässt sich für beliebig viele weitere Repositories wiederholen. Beginnen Sie mit „File → Clone repository. . . “.

Der grundsätzliche Ablauf beim Arbeiten mit GitHub Desktop ist folgender:

- Sicherstellen, dass der richtige Branch ausgewählt ist („Current branch“ oben in der Mitte). „master“ ist der Haupt-Branch Ihres Projektes.
- Sicherstellen, dass die lokale Version aktuell ist (oben rechts „Fetch origin“)
- Ggf. Änderungen herunterladen („Pull origin“)
- Am Code arbeiten
- Einen Commit mit den Änderungen erstellen
- Die Änderungen hochladen („Push origin“)

Um Konflikte durch unterschiedliche Versionen der Dateien zu vermeiden, achten Sie darauf, diesen Ablauf stets einzuhalten. Sorgen Sie zudem dafür, dass nie mehrere Gruppenmitglieder zeitgleich an den gleichen Dateien im gleichen Branch arbeiten. Legen Sie sich daher ggf. mehrere Branches

an, die Sie im Anschluss mergen.

Verwenden Sie stets GitHub Desktop, um den Quellcode und das Dokumente und Infos Repository aus GitLab auf Ihren Rechner zu laden. Nutzen Sie dafür nicht die Download-Funktion Ihres Browsers. Somit stellen Sie sicher, dass alle Dateien stets im richtigen Verzeichnis gespeichert und aktuell sind.
Wenn Sie am Code gearbeitet haben, werden Ihnen die Änderungen in GitHub Desktop angezeigt. Wollen Sie diese nun hochladen, geben Sie unten links eine Kurzbeschreibung dessen ein, was Sie geändert haben, sowie eine optionale ausführlichere Beschreibung. Schließlich können Sie den Commit erstellen („Commit to . . . “).

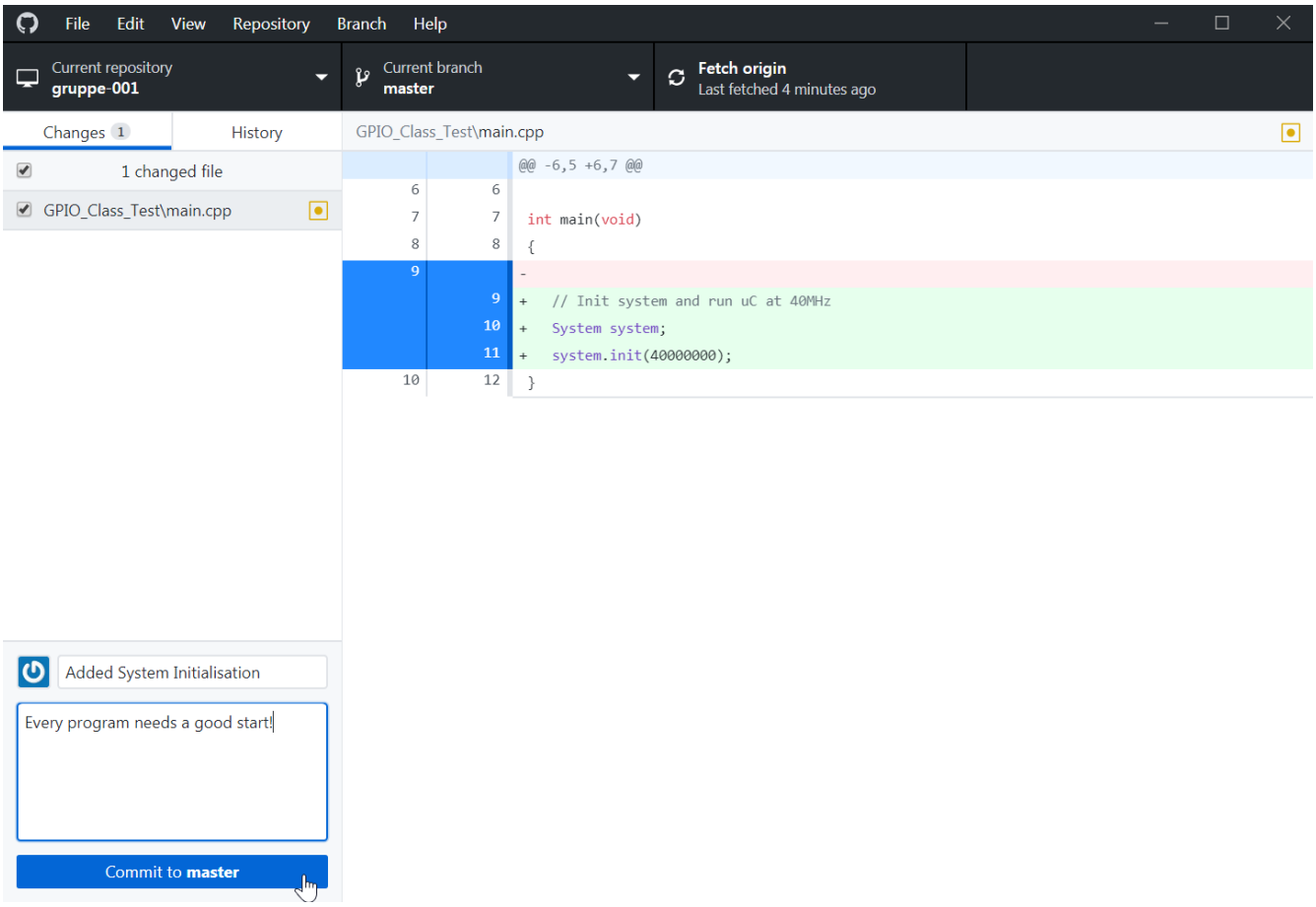

Figure 14: Erstellen eines Commits in GitHub Desktop

Solange der Commit nicht hochgeladen ist, können Sie ihn noch widerrufen („Undo“ unten links), danach jedoch nicht mehr. Laden Sie ihn mit „Push origin“ hoch.

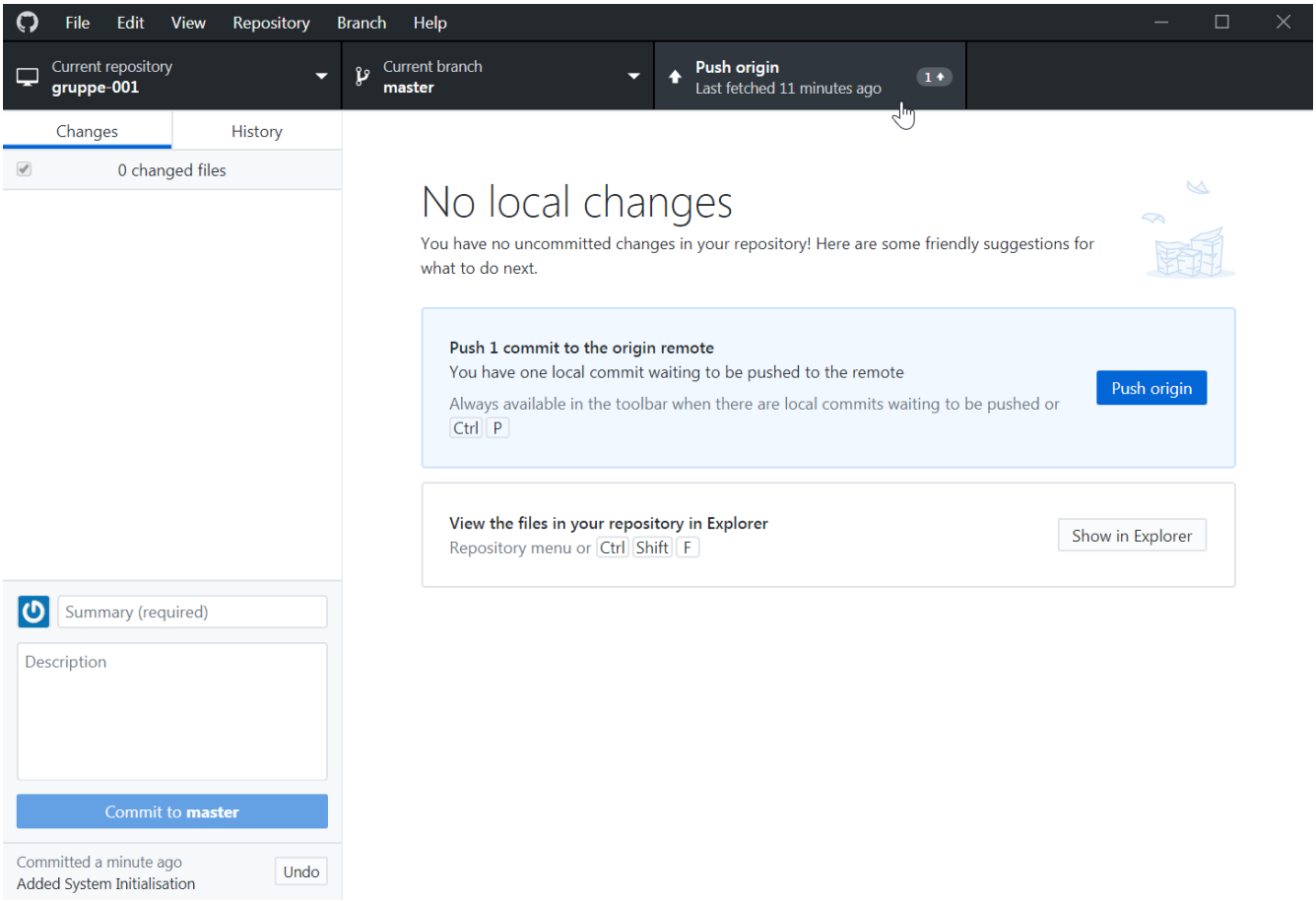

Figure 15: Pushen des Commits in GitHub Desktop

3.1.5 Weshalb benutzt das Praktikum Git?

Es gibt verschiedene Situationen in denen ein gut gepflegtes Git Repository praktisch ist.

- Es geht nichts verloren
  Falls Sie in einer Sackgasse gelandet sind, oder Ihr Code nicht mehr funktioniert und Sie den Fehler nicht finden können, können einfach zu einem vorherigen Commit zurück, in dem noch alles in Ordnung war. Besser noch als zurückgehen zu müssen, ist es allerdings einen Branch nur mit fertigem Code zu haben (üblicherweise der ,,master“ Branch) und für jeden Entwicklungsschritt einen eigenen Entwicklungszweig (z.B. ,,feature“ Branch) zu erstellen. In diesem können neue Funktionen und Ideen entwickelt und ausprobiert werden, ohne den stabilen Code im ,,master“ Branch zu beeinträchtigen.

- Übersichtliches und konfliktfreies Zusammenarbeiten möglich
  Jedes Gruppenmitglied erstellt sich einen eigenen Branch für den Abschnitt, den es entwickeln möchte. Hat es seinen Teil erfolgreich beendet, wird dieser Branch mit dem ,,master“ Branch zusammengeführt (,,Merge“). Hat ein anderes Gruppenmitglied eine andere Stelle in der gleichen Datei (in seinem Branch) bearbeitet, so ist das meist beim Mergen kein Problem, da Git automatisch erkennt, was wohin kommt und so das ,,zusammenflicken“ von Codezeilen entfällt. Je öfter man aber in bereits existierendem Code herumgebastelt

hat, desto höher ist natürlich auch die Gefahr, dass ein anderes Gruppenmitglied die gleiche Stelle bearbeitet hat und Git nicht mehr automatisch entscheiden kann, welche Version jetzt verwendet werden soll. Daher sollten die in einem Branch bearbeiteten Features, bzw. Codeabschnitte möglichst genau eingegrenzt sein und zeitig gemerged werden (natürlich erst wenn es funktioniert).

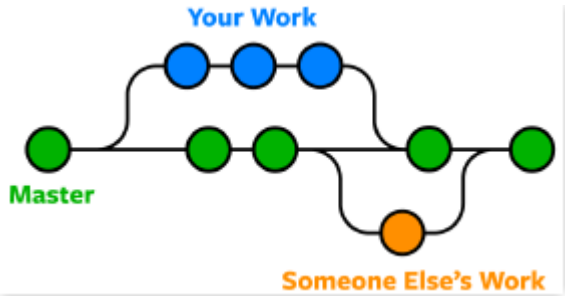

Figure 16: Prinzip der Branches in Git

3.1.6 Inhalte des Gruppen-Repository

Innerhalb Ihres Gruppen-Repository finden Sie folgende Ordner und Dateien vor:

- .gitignore
  Nicht jede Datei, die beim Programmieren entsteht, gehört in ein Repository. So ist es beispielsweise weder nötig noch erwünscht, dass temporäre Dateien, die beim Kompilieren anfallen, zusammen mit dem Quellcode auf GitLab landen. Derartige Ausschlüsse werden durch die .gitignore Datei festgelegt. Sie ist bereits vollständig eingerichtet, und muss nicht angerührt werden!

- Common_Classes
  Dieser Ordner enthält alle Klassen, die Sie in diesem Praktikum benutzen. Da Sie von mehreren unterschiedlichen Programmen, bzw. Projekten genutzt werden, liegen sie gesammelt an einem gemeinsamen Ort.

- libs
  Um Sie bei der Entwicklung zu unterstützten, liegen in diesem Ordner kompilierte Teile der Musterlösung. Den Ordner müssen Sie nicht verändern. Wie Sie die Musterlösung verwenden können, ist erklärt in Kapitel Einbinden der Klassenbibliotheken mit Musterlösung.

- *_Class_Test
  Die Testprogramme für die einzelnen Aufgabenteile (siehe Aufgabenstellung) liegen jeweils in einem eigenen Ordner (da es unabhängige CCS-Projekte sind). Sowohl der Verweis auf den Ordner Common_Classes, als auch alle anderen Einstellungen (wie z.B. der verwendete Mikrocontroller) sind bereits enthalten.

3.2 Code Composer Studio™ (CCS)

3.2.1 Was ist CCS?

CCS ist eine Entwicklungsumgebung von Texas Instruments, die auf die Mikrocontroller und eingebetteten Prozessoren des Herstellers angepasst ist. Sie ermöglicht das Schreiben, Kompilieren,

Flashen und Debuggen von Code. Das Programm steht für Windows, Linux und macOS zur Verfügung.

In der zweiten Einführungsveranstaltung zum Praktikum wird mit ihnen CCS gemeinsam eingerichtet. Im Handbuch finden Sie daher keine vollständige Anleitung zum Programm, sondern nur Angaben zu den wichtigsten praktikumsspezifischen Funktionen und deren Einrichtung. Falls Ihnen die Informationen im folgenden Kapitel nicht ausreichen und Ihnen der Foliensatz der Einführungsveranstaltung nicht weiterhilft, sollten Sie sich das zweite Kapitel des ,,TivaC LaunchPad Workshop“ (siehe Weitere Dokumente, Datenblätter, C++ Unterlagen) durchlesen.

CCS bietet eine Reihe nützlicher Shortcuts, die Ihnen viele Mausklicks ersparen. Eine Auflistung finden Sie im Dokument ,,CCS Shortcuts“ (siehe Weitere Dokumente, Datenblätter, C++ Unterlagen).

3.2.2 Einbinden von TivaWare™

Die Einrichtung von CCS für das Praktikum erfordert die korrekte Einbindung der Treiberbibliothek TivaWare™. Letztere liegt im Dokumente und Infos Repository und wird beim Synchronisieren mit GitHub Desktop automatisch auf Ihrem Rechner abgespeichert. TivaWare™beinhaltet alle notwendigen Treiber zur Ansteuerung der Peripherie des LaunchPads. Mithilfe der bereitgestellten Bibliothek wird die Entwicklung von Applikationen wesentlich beschleunigt. Im Folgenden wird auf das Einbinden von TivaWare™in CCS eingegangen. Öffnen Sie hierfür ihr CCS Workspace und navigieren Sie unter ,,Window“ zu ,,Preferences“ (Abbildung 17). Unter ,,Code Composer Studio → Build → Variables“ können Sie TivaWare™als Build Variable auf Workspace Ebene hinzufügen (Add. . .) (Abbildung 18).

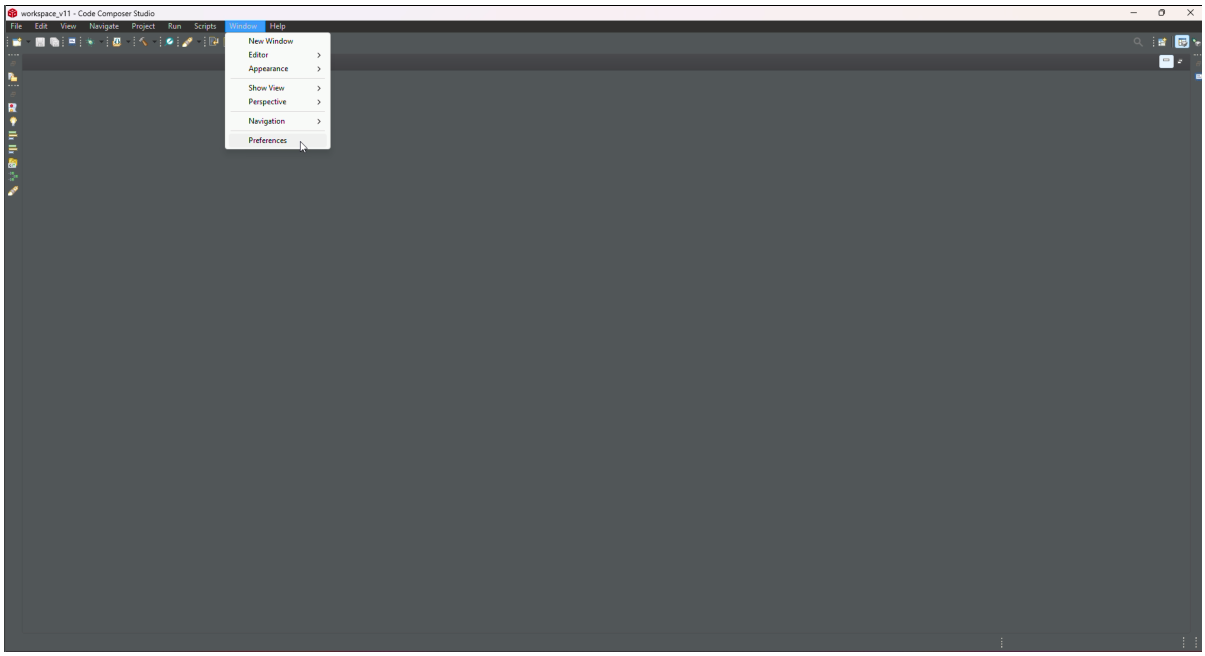

Figure 17: Öffnen der Code Composer Studio Preferences

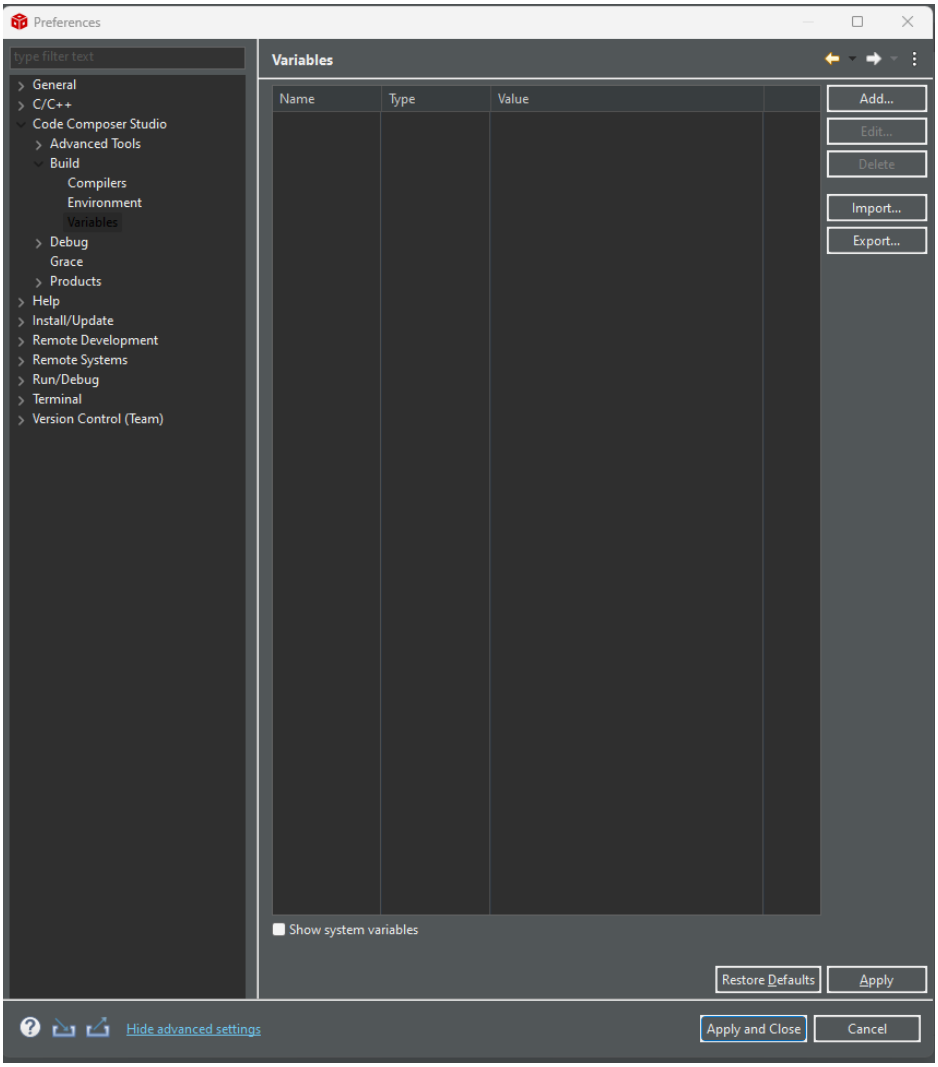

Figure 18: Hinzufügen einer Build Variablen

Als Variablenname müssen Sie ,,TIVAWARE_INSTALL“ eingeben. Unter ,,Type“ muss ,,Directory“ ausgewählt werden, damit Sie unter ,,Value“ zu Ihrem vorhin festgelegten TivaWare™ Ordner navigieren können.

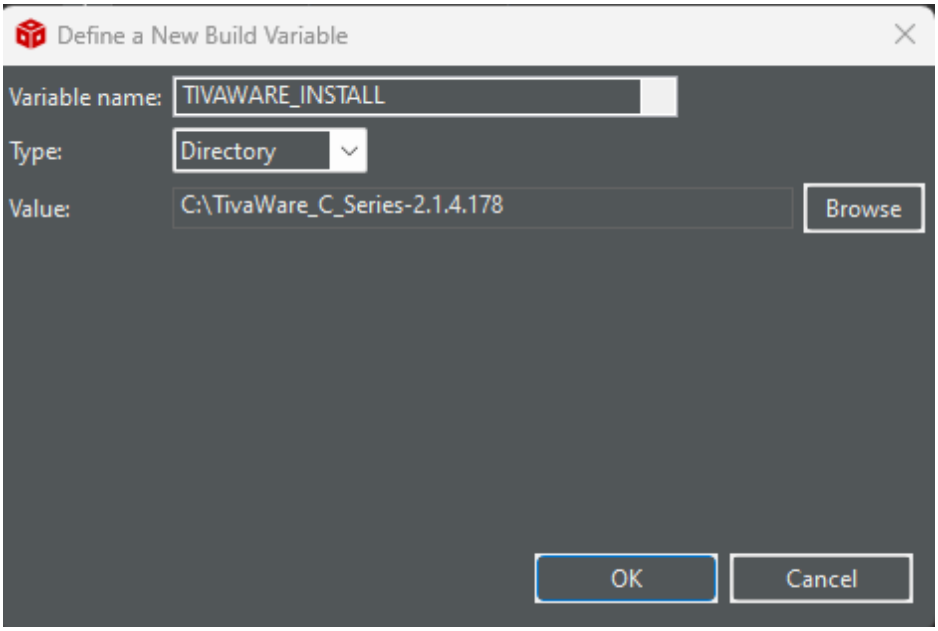

Figure 19: Definieren der TivaWare™ Build Variablen

3.2.3 Importieren eines CCS Projekts

Ihr GitLab Repository enthält bereits fertig eingerichtete CCS Projekte. Nach dem Herunterladen mit GitHub Desktop gilt es nun diese in Code Composer Studio zu importieren, um damit arbeiten zu können. Öffnen Sie hierfür CCS und klicken Sie auf ,,File -¿ Open Projects from File System. . . “.

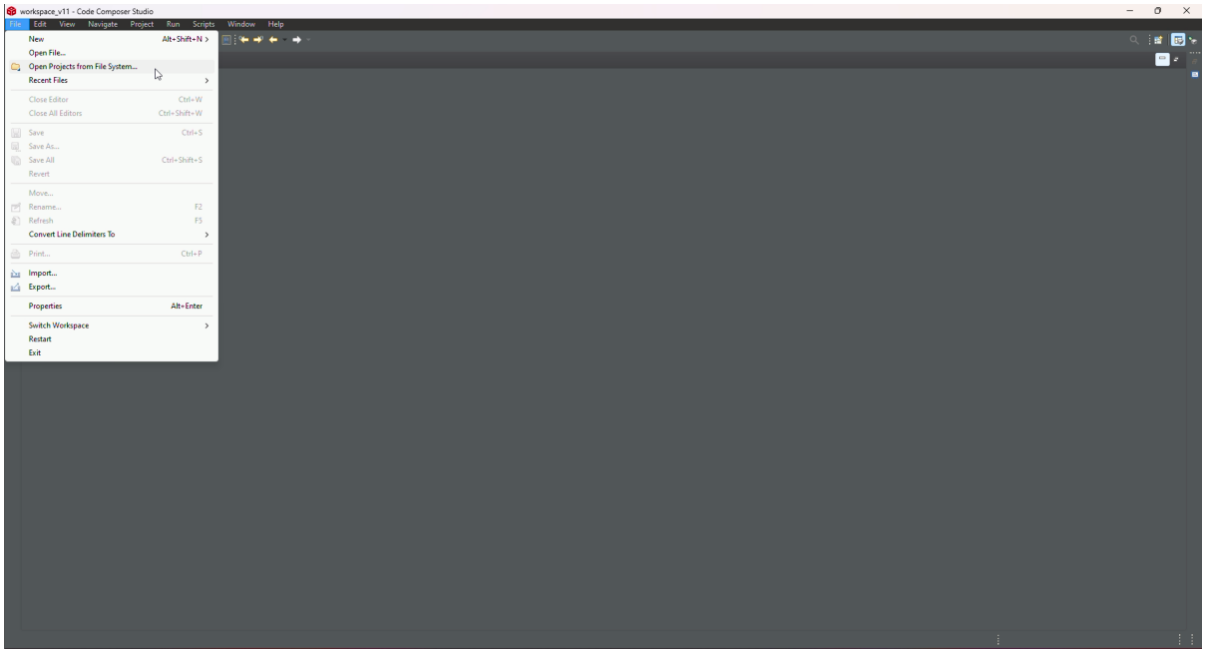

Figure 20: Importvorgang in CCS

Im neuen Fenster navigieren Sie mit ,,Directory...” in den Ordner, in dem sich das mit GitHub Desktop heruntergeladene Gruppen Repository befindet. Es werden anschließend alle gefundenen Ordner und Projekte importiert. Wählen Sie den Gruppenordner (erster Eintrag) ab und

bestätigen Sie den Import der anderen Projekte mit ,,Finish". Es befinden sich nun alle Projekte im Project Explorer (linke Seite in CCS).

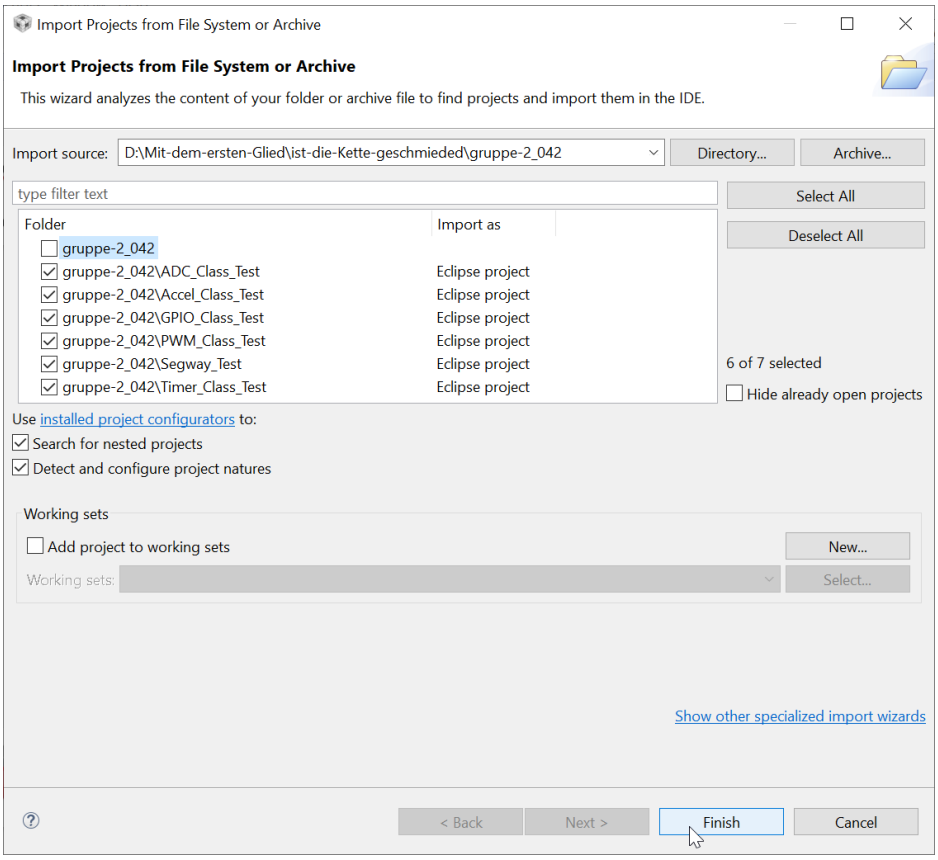

Figure 21: Auswahl der CCS Projekte

3.2.4 Debugging

Bei Fehlfunktionen des Quellcodes können Sie mittels der Debugging Schnittstelle des Tiva Launch-Pads nach Fehlern suchen. Dies ist in den meisten Fällen wesentlich effektiver als das ,,händische" durchsuchen im Codeeditor. Vorraussetzung ist, dass Ihr Code trotzdem fehlerfrei kompiliert, daher müssen Sie ggf. zunächst noch entspreechende Compilerfehler (,,Rechtschreibfehler") beheben, bevor Sie mit dem Debugger nach zusätzlichen Fehlern (,,Logikfehler") suchen können. Um die Debugging Schnittstelle zu nutzen muss das LaunchPad mit der ,,Debug" USB Buchse mit Ihrem Rechner verbunden sein.

Indem man den grünen Käfer drückt, wird der Quellcode kompiliert und im Debugging Modus auf das Board geladen. Das Programm pausiert zunächst an seinem Eintrittspunkt, der ersten Zeile der *main()*. Falss noch nicht geschehen, kann man mittels Doppelklick in den linken Rand der gewünschten Codezeile (oder *STRG + SHIFT + B*) einen ,,Breakpoint" setzen. Das Programm läuft nun (durch Drücken von ,,Resume") bis es diese Zeile erreicht. Des weiteren gibt es:

- Step Into:
  Hier wird die in der aktuellen Codezeile befindliche Methode aufgerufen und der Debugger ,,betritt" die Methode.

- Step Over:
  Hier wird der Code Zeile für Zeile ausgeführt, jedoch werden aufgerufene Methoden zunächst in einem Schritt ausgeführt, ehe das Programm wieder pausiert.

- Step Return:
  Dies ist das Gegenteil von Step Into. Es wird der gesamte Code in der aktuellen Methode ausgeführt. Das Programm stoppt bei der Rückkehr zum übergeordneten Code. Dies ist praktisch, wenn Sie mit Step Into in eine große Methode reingesprungen sind, aber nur einen kleinen Teil davon wikrlich mit Step Over sehen wollen. Den Rest dieser Methode können Sie danach mmit Step Return automatisch ausführen lassen. Step Into und Step Return hintereinander auszuführen ist identisch mit Step Over.

Der sichere Umgang mit dem Debugger erfordert Einarbeitung. Bevor Sie Ihren eigenen Code debuggen, sollten Sie den Debugger mit der fertigen GPIO-Klasse und dessen Testprogramm ausprobieren. Starten Sie hierzu das Testprogramm im Debugging Modus und beobachten Sie, wie Sie mit den Bedienelementen den Code zuelenweise ausführen können. Sie können auch gezielt Fehler in das Programm einbauen, um herauszufinden wie der Debugger darauf reagiert - hier sind Ihnen keine Grenzen gesetzt.

3.3 Arduino IDE

Die Arduino IDE ist eine einfache Programmierumgebung für macOS, Windows oder Linux. Üblicherweise dient sie zur Erstellung von Programmen für den Mikroprozessor eines Arduinos. Das Informationstechnik I Praktikum nutzt nur den Serial Plotter der IDE.

3.3.1 Installation

Die Installation benötigt nur wenige Schritte: Laden Sie den Installer herunter (siehe Arduino IDE) und führen Sie die Installation mit den Standard-Einstellungen aus. Danach können Sie das Programm starten. Die mindestens benötigte Verion is 1.8.10.

3.4 Der serielle Plotter der Arduino IDE

Der serielle Plotter stellt ein Medium zur Verfügung, durch das Sie in Echtzeit ein Diagramm der Daten sehen können, die am seriellen Anschluss des Mikrocontrollers ausgegeben werden (in diesem Fall der USB-Anschluss).

Um ihn nutzen zu können, müssen Sie erst festlegen, wie genau die Arduino IDE mit dem LaunchPad kommunizieren soll. Bevor Sie Ihr LaunchPad anschließen, sollten Sie einen Blick auf die Liste unter ,,Werkzeuge -¿ Port" werfen (siehe Abbildung). Oft stehen dort bereits Kommunikationskanäle (,,COMx"). Diese gehören dann z.B. zum Bluetooth-Modul. Wenn Sie nun Ihr LaunchPad anschließen, müsste ein weiterer COM-Port auftauchen, der für das LaunchPad steht. Wählen Sie diesen aus. Wenn kein neuer Port auftaucht, stecken Sie das LaunchPad wieder ab, starten Sie die Arduino IDE neu und versuchen Sie es erneut. Manchmal hilft es auch einen anderen USB Port zu probieren. Wenn selbst nach einem Neustart keine Verbindung möglich ist, sollten Sie zu uns in die Sprechstunde kommen.

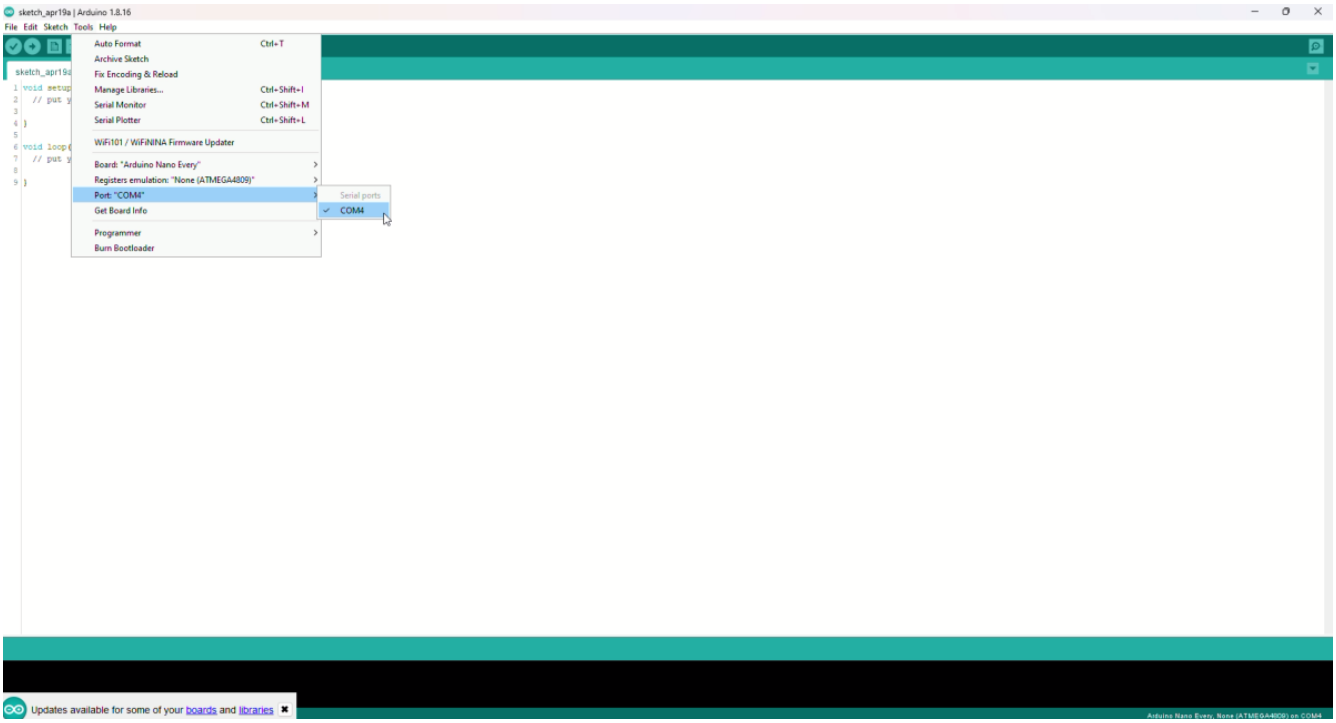

Figure 22: Auswählen des richtigen Kommunikationskanals (COMx)

Haben Sie den richtigen COM-Port ausgewählt, können Sie den seriellen Plotter starten. Er ist ebenfalls unter ,,Werkzeuge" zu finden (siehe Abbildung). Es öffnet sich ein Fenster, in dem sofort die Daten gezeichnet werden sollten. Ist dies nicht der Fall, überprüfen Sie, ob unten links die Baudrate auf 115200 Baud eingestellt ist und starten Sie den Plotter ggf nochmals neu. Falls dies auch nicht hilft, stellen Sie sicher, dass das Programm auf dem TivaBoard läuft und sich nicht in einem Breakpoint/Error befindet. Sollte es nun immer noch nicht funktionieren, gehen Sie wie oben beschrieben vor, um eine funktionierende Verbindung aufzubauen.

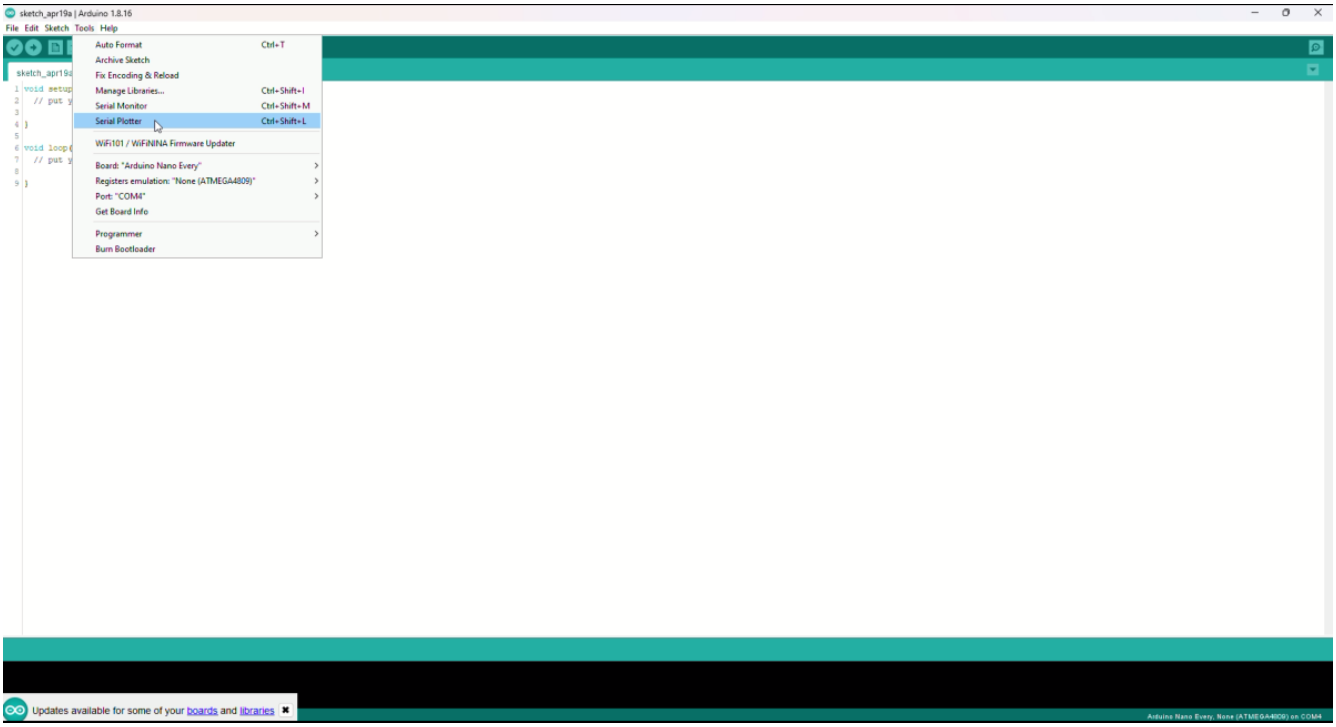

Figure 23: Starten des Serial Plotters

Details dazu, wie Sie Werte aus Ihrem Quellcode übermitteln können, finden Sie in der Dokumentation der System Klasse (Methoden System::setDebugVal und System::sendDebugVals).

3.5 Programmierrichtlinien

3.5.1 Motivation und Anwendung

„Der Zweck eines definierten Programmierstils ist die Erleichterung der Arbeit aller an einem Programmierprojekt beteiligten Teammitglieder. Das bezieht sich insbesondere auf die Lesbarkeit, Verständlichkeit und Wartbarkeit von Programm-Quelltext bzw. der Eliminierung vermeidbarer Fehlerquellen in Programmen.

Im Sinne der Verständlichkeit und Wartbarkeit kann eine Richtlinie die Verwendung von programmsprachlich erlaubten (aber "unsauberen") Programmkonstrukten einschränken oder ganz verbieten. Die Einhaltung von vorgängig definierten Nomenklaturen für Variablen, Prozeduren und Klassennamen kann Lesbarkeit und Wartbarkeit eines Programmcodes wesentlich verbessern.

Für die Wartbarkeit von Programmcode ist die Einhaltung eines definierten Programmierstils noch wichtiger als während der Entwicklung. Als Richtwert gilt, dass 80 % der Lebenszeit eines Softwareprotdukts auf die Wartung entfallen. Nur selten wird ein Produkt vom ursprünglichen Entwickler gewartet. Umso wichtiger ist es, dass bereits vom ersten Augenblick an ein guter Programmierstil verwendet wird." (Quelle)

Die nachfolgenden Richtlinien gliedern sich in zwei Kategorien: Pflichtrichtlinien und Empfehlungen. Erstere müssen eingehalten werden, letztere sind vor allem als Hilfe für diejenigen mit weniger

Programmiererfahrung gedacht und nicht verpflichtend. Welche Kategorie die Richtlinie angehört, ist durch ein P bzw. ein E gekennzeichnet.

Die von uns zur Verfügung gestellten Klassen basieren ebenfalls auf diesen Richtlinien. Sie können sich im Zweifelsfall daran orientieren. Ein hilfreiches Werkzeug ist die Autoformatierung von CCS (Markieren des Codes und anschließend die Tastenkombination $STRG + I$ eingeben). Sie stimmt mit den Regeln überein, kann aber nicht alle hier aufgeführten Punkte umsetzen.

3.5.2 Programmierrichtlinien

(P) Es ist absolut erforderlich, dass Sie sich innerhalb Ihrer Gruppe auf genau einen einzigen Programmierstil einigen und diesen einhalten.

(P) Wählen Sie aussagekräftige Namen. Die einzige erlaubte und sinnvolle Ausnahme sind Zählvariablen.
Schlecht:

```
uint32_t a = 42
for(uint8_t zaehlVariable = 0; zaehlVariable < 3; zaehlVariable++)
```

Gut:

```
uint32_t antwortAufAlles = 42
for (uint8_t i = 0; i < 3; i++)
```

(P) Verwenden Sie eine einzige Sprache im Quellcode. Alle Klassen-, Methoden-, und Variablennamen sind in der gleichen Sprache zu verfassen.

(P) Konstanten werden komplett in Großbuchstaben geschrieben. Besteht der Bezeichner aus mehreren Wörtern, werden diese mit einem Unterstrich voneinander getrennt.
Beispiel:

```
#define ANZAHL_STUDENTEN 256
const uint32_t WICHTIGES_DATUM = 11102161;
```

(P) Variablen- und Methodennamen beginnen immer mit einem Kleinbuchstaben. Wenn ein Name aus mehreren Wörtern besteht, beginnt jedes Folgeword mit einem Großbuchstaben.
Beispiel:

```
uint32_t segwaySpeed = 98;
```

(P) Klassennamen beginnen immer mit einem Großbuchstaben. Wenn ein Name aus mehreren Wörtern besteht, beginnt jedes Folgewort mit einem Großbuchstaben.
Beispiel:

```
class WatchdogTimer
```

(P) Ganzzahlen aller Art sind mit den in <stdint.h>definierten Variablentypen anzugeben. Anders als bei short, int und long ist die Größe eindeutig und auf den ersten Blick erkennbar.
Schlecht:

```
unsigned long long int sandkoernerAmStrand = 314159265358979384;
```

Gut:

```
uint64_t sandkoernerAmStrand = 314159265358979384;
```

(P) Gleitkommazahlen müssen immer vom Typ **float** (32 Bit) sein, da nur dieser von der Gleitkommaeinheit (FPU) des Mikrocontrollers verarbeitet werden kann. Normale Gleitkomma-Konstanten werden immer als **double** (64 Bit) interpretiert und müssen deswegen explizit als **float** markiert werden.
Schlecht:

```
float winkelInGrad = winkelInRad / 3.14159 * 180;
```

Gut:

```
float winkelInGrad = winkelInRad / 3.14159f * 180.0f;
```

(E) Weisen Sie Variablen bereits bei der Erstellung einen Wert zu. Dies gilt insbesondere für Instanzvariablen. So können Sie u.a. schwer auffindbare Fehler wie z.B. einen zu großen Array-Index vermeiden.

(P) Nur eine Anweisung pro Zeile.
Schlecht:

```
uint32_t a = b + 1; machWas(a);
```

Gut:

```
uint32_t a = b + 1
machWas(a);
```

(P) Ein Leerzeichen steht vor und nach Operatoren, nach Kontrollanweisungen und nach Kommas. Ausnahmen sind nur der ++ und der – Operator. Zwischen Methodenname und Klammern, sowie direkt nach einer geöffneten Klammer (bzw. direkt vor einer gescchlossenen Klammer) steht kein Leerzeichen.

(P) Geschweifte Klammern stehen in einer eigenen Zeile. Ausgenommen sind Definitionen von Arrays.

(P) Switches werden wie folgt formatiert. Es handelt sich dabei um die Standardeinstellung von

CCS.

```
switch (frucht)
{
case BANANE:
    farbe = GELB;
    hatSchale = true;
    break;
case ERDBEERE:
    farbe = ROT;
    hatSchale = false;
    break;
case PFLAUME:
    farbe = VIOLETT;
    hatSchale = true;
    break;
}
```

(E) Begrenzen Sie die Breite Ihrer Zeilen. Die Autoformatierung von CCS limitiert die Breite auf 80 Zeichen und auch nicht immer an sinnvollen Stellen innerhalb einer Zeile. Sie können Werte größer als 80 wählen, müssen sich aber auf einen Wert einigen.

(E) Kommentare stehen in der Zeile über der Anweisung. Methodenkommentare können sowohl vor der Methode, als auch zu Beginn der Methode stehen.

(E) Kommentare, die länger als zwei Zeilen sind, werden mit /* ... */ angeschrieben und nicht mit // ...

(E) Fügen Sie Leerzeilen ein, um Ihren Code in Blöcke zu unterteilen. SIe können auch vor einem Kommentar eine Leerzeile einfügen, um die Zugehörigkeit zur nächsten Zeile hervorzuheben.

(E) Vermeiden Sie die Verwendung bedingter Ausdrücke. Sie sind zwar kompakter, können jedoch schnell unübersichtlich werden.
Statt:

```
reading = potentiometer.read() > 0 ? reading / maxVal : -reading / minVal;
```

Lieber:

```
if (potentiometer.read() > 0)
{
    reading /= maxVal;
}
else
{
    reading /= -minVal;
}
```

(1)

(P) Zu jeder Methode gehört ein Kommentar, der die Funktion der Methode und, falls vorhanden, die Bedeutung der Übergabeparameter sowie des Rückgabewertes erklärt.

(P) Der gesamte Quellcode muss so kommentiert sein, dass keine Nachfragen zum Verständnis der Funktionalität notwendig sind. Sie können vorraussetzen, dass der Leser mit der Syntax von C++ vertraut ist, nicht jedoch, dass er weiß, was genau Sie sich überlegt haben. Die Kommentare enthalten deswegen weniger Beschreibung davon, was in der Zeile steht, sondern warum.
Am Beispiel der GPIO::init statt:

```
// Set current and pin type
current = GPIO_STRENGTH_2MA;
pinType = GPIO_PIN_TYPE_STD;
```

lieber:

```
// The default current and pin type (also see gpio.c line 1552-1614)
current = GPIO_STRENGTH_2MA;
pinType = GPIO_PIN_TYPE_STD;
```

(P) Achten Sie darauf, Ihre Kommentare aktuell zu halten! Insbesondere bei der Fehlersuche ist es üblich, Zeilen auszukommentieren oder umzuändern. Zum Debuggen gehört allerdings auch, dass Sie im Anschluss aufräumen.

(E) Wenn Sie Informationen aus externen Quellen wie z.B. Datenblättern verwenden, schreiben Sie Ihre Quelle sowie die Seitenzahl in den Kommentar. Keiner kennt diese Dokumente auswendig und auch Sie werden sich am Ende des Workshops nicht mehr an jede Information erinnern, die Sie zu Beginn gelesen haben.

(P) Das Verwenden der zur Verfügung gestellten Klassen ist Pflicht! Schreiben Sie keine eigene Lösung für die gleiche Funktion. Nutzen Sie auch nicht eine Methode der TivaWare™API, wenn bereits eine Klassenmethode dafür verfügbar ist.
Schlecht:

```
SysCtlDelay(2);
```

Gut:

```
system.delayCycles(5);
```

Ⓟ Die Signatur (Rückgabewert, Parametertyp und Name) der von Ihnen zu programmierenden Klassenmethoden sowie die von uns bereits gegebenen Methoden dürfen nicht verändert werden, da sie nicht nur in Ihrem Aufgabenbereich verwendet werden. Ihnen steht es jedoch frei, Methoden hinzuzufügen oder zu überladen.

Ⓟ Instanzvariablen sind nie **public**. Schreiben Sie bei Bedarf entsprechende get- bzw. set-Methoden.

Ⓟ Sowohl Konstruktoren als auch Destruktoren müssen leer bleiben. Dies hängt damit zusammen, wie die Klassen innerhalb anderer Klassen verwendet werden.

Ⓟ Verwenden Sie die Konstanten der TivaWare™API. Definieren Sie keine eigenen Konstanten für die gleiche Aufgabe.
Schlecht:

```
// Adresse aus API kopiert.
#define GUTER_PORT 0x40004000

// Setze je nach Port andere Werte.
if (port == GUTER_PORT)
{
    GPIOPinWrite(GUTER_PORT, 0x02, 0xff);
}
else if (port == 0x40025000)
{
    GPIOPinWrite(0x40025000, 0x04, 0xff);
}
```

Gut:

```
// Setze je nach Port andere Werte.
if (port == GPIO_PORTA_BASE)
{
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_1, 0xff);
}
else if (port == GPIO_PORTA_BASE)
{
    GPIOPinWrite(GPIO_PORTA_BASE, GPIO_PIN_3, 0xff);
}
```

Ⓟ Die Zusammenarbeit mit anderen Gruppen oder gar der Austausch von Quellcode ist nicht gestattet. Es muss in der Prüfung nachvollziehbar sein, dass Sie die präsentierte Lösungen selbstständig erarbeitet haben. Sollten wir einen Austausch von Code zwischen zwei Personen feststellen, führt dies zum Nichtbestehen für **BEIDE** Personen.

Ⓟ Erstellen Sie keine Variablen mit **new**. Diese landen nicht auf dem Stack, sondern auf dem Heap, der jedoch nicht verfügbar ist.

(P) Beispielwerte sind verbindlich. Wenn für eine Variable oder Übergabeparameter ein Beispielwert von uns gegeben ist, so muss die Methode mit diesem Wert aufgerufen werden können, bzw. die Variable die Information in der aufgezeichneten Form speichern.
Beispiel: In der GPIO-Klasse ist für die Variable port der Beispielwert GPIO_PORTF_BASE gegeben. Sie darf den Port daher nicht mit 1, 2, 3,... oder 'a', 'b', 'c',... speichern, sondern nur als GPIO_PORTA_BASE, GPIO_PORTB_BASE, GPIO_PORTC_BASE,...

(E) Wenn optionale Übergabeparameter gefordert sind, so können Sie dieses Problem immer mit default-Argumenten lösen.

4 Aufgabenstellung

Die Aufgabenstellung beinhaltet Bonusaufgaben, die **freiwillig** sind. Diese sind mit einem (B) gekennzeichnet.

4.1 PWM Klasse

4.1.1 Was ist PWM?

Ein Mikrocontroller hat nur digitale Ausgänge und kann deshalb nicht direkt einen analogen, also kontinuierlichen, Verlauf ausgeben. Sie können z.B. eine LED nur ein, oder ausschalten, aber nicht mit halber Helligkeit leuchten lassen. Dies reicht für viele Anwendungen nicht aus. Beispielsweise wird ein Modellauto nicht immer nur Vollgas gefahren oder gestoppt.

Um das Problem zu umgehen, wird ein analoges Signal ,,simuliert“, indem der Mikrocontroller einen Pin sehr schnell an- und ausschaltet. Dieses Verfahren wird Pulsweitenmodulation, kurz PWM, genannt.

Das Schalten geschieht mit einer festen Frequenz aber variablen Tastgrad. Dieser gibt an wie das Verhältnis zwischen HIGH und LOW ist. Also den Anteil den der Pin an- oder ausgeschaltet ist. Im Falle des MagniSilver™ bestimmt dieser Wert wie viel Leistung an die Motoren weitergegeben wird. Die Erzeugung dieser Rechtecksignale übernehmen im Tiva Mikrocontroller zwei PWM-Module mit je vier PWM Generatoren. Jeder Generator kann zwei unabhängige PWM Signale mit der gleichen Frequenz ausgeben.

Im einfachsten Fall besteht ein PWM Generator aus einem 16-Bit Zähler (Timer), zwei PWM-Komparatoren und einem Signalgenerator. Totzeit, Fehlerbehandlung, Interrupts und Anderes benötigen wir in unserer Anwendung nicht. Der Zähler zählt von seinem Load-Wert bis auf 0 herunter und fängt dann wieder bei seinem Load-Wert an. Der Load-Wert gibt also die Frequenz an. Der Compare-Wert entscheidet, bis zu welchem Zähler-Wert die Ausgänge auf HIGH bleiben und bestimmt so zusammen mit dem Load-Wert den Tastgrad. Die Frequenz mit der gezählt wird, ist die CPU-Taktfrequenz geteilt durch den ,,PWM Unit Clock Divider“, welcher in der System-Klasse festgelegt wird.

Folglich bestimmt der Load-Wert die Frequenz des PWM-Signals, während der Compare-Wert den Tastgrad (bei einer LED: Helligkeit) bestimmt. Üblicherweise wird die Frequenz einmalig festgelegt, und anschließend nur der Tastgrad verändert.

Wenn Sie das Blockdiagramm des PWM-Moduls betrachten, werden Sie feststellen, dass die Signale durch einen ,,Output Control Logic“-Block gehen. Dieser Block entscheidet, ob, und wie die Signale tatsächlich den Mikrocontroller verlassen. So können einzelne Signale invertiert oder abgeschaltet werden, Details finden sich im ,,TivaC Mikrocontroller Datenblatt“ auf den Seiten 1231, 1232 und 1239.

4.1.2 Anforderungen an die Klasse

Mithilfe der PWM-Klasse können Motoren über eine H-Brücke (auch Vollbrücke genannt) in Geschwindigkeit und Richtung gesteuert werden. Die zwei PWM-Module des Mikrocontrollers können jeweils vier Motoren ansteuern. Insgesamt ermöglicht die PWM-Klasse also das gleichzeitige Steuern von bis zu acht unabhängigen Motoren.

4.1.3 PWM::init

- Rückgabewert/-typ: **void**
- Übergabeparameter:

  System *sys: Zeiger auf die laufende Instanz der Systemklasse

  GPIO *dirLeft: Zeiger auf den Richtungspin des linken Motors.

  GPIO *dirRight: Zeiger auf den Richtungspin des rechten Motors.

  uint32_t portBase: Adresse des verwendeten GPIO Ports, z.B. GPIO_PORTF_BASE (NICHT die Konstante des PWM Moduls)

  uint32_t pin1, pin2: Adresse der verwendeten Pins, z.B. GPIO_PIN_6, GPIO_PIN_7. Die Pins müssen zum gleichen PWM Generator gehören. In welcher Reihenfolge sie angegeben werden ist allerdings egal.

  bool invert: Optionaler Übergabeparameter, der angibt, ob das PWM Signal invertiert werden soll(An = Aus, Aus=An und NICHT positiver/negativer Tastgrad).
  Der Standardwert ist **false**.

  uint32_t freq: Optionaler Übergabeparameter, der die Frequenz des PWM Generators bestimmt. Der Standardwert ist 5kHz.

- Funktion/Umsetzung:
  Analog zur GPIO::init-Methode führt diese Methode die Initialisierung des PWM-Moduls und des PWM-Objektes durch. Jedes PWM-Objekt beansprucht dabei genau einen PWM-Generator. Zur Initialisierung gehört u.a. die Bestimmung der PWM-Taktfrequenz, das Setzen der Ausgangsfrequenz, das Konfigurieren der Ausgangslogik, sowie die Aktivierung des Generators.

  Den korrekten PWM-Generator sowie alle weiteren zur Konfiguration benötigten Konstanten können Sie anhand der übergebenen Werte für **portBase**, **pin1** und **pin2** bestimmen.

  ⓘ Anstatt **portBase**, **pin1** und **pin2** einzeln abzugleichen, können alle drei Variablen auf einmal verglichen werden, indem man sie bitweise verodert. Soll z.B. geprüft werden, ob es sich um die Pins **PA0** und **PA1** handelt, kann das wie folgt geprüft werden:

  ```
  if((portBase|pin1|pin2) == (GPIO_PORTA_BASE|GPIO_PIN_0|GPIO_PIN_1))
  ```

  (Siehe auch "inc/hw_memmap.h" Zeile 53 folgend und "driverlib/gpio.h" Zeile 60 folgend). Aufgrund der Vielzahl an zulässigen Pin-Kombinationen sollten Sie jedoch keine if-else-Verzweigung benutzen. Auch eine switch-case-Anweisung führt in den meisten

Fällen zu sehr langen Lösungen. Versuchen Sie stattdessen, mit einem Array zu arbeiten, den Sie nach der passenden Pin-Kombination durchsuchen. Einzelne Generatoren können mehrere Pinkombinationen ansteuern. Wählen Sie eine Zuordnung, bei dem jeder Generator eine einzigartige Pinkombination erhält.

Alle PWM Module werden durch den PWM Unit Clock Divisor mit einem Bruchteil der System Taktfrequenz versorgt. Sowohl den Teiler als auch die System Taktfrequenz können Sie von der System-Klasse erhalten.

Nach Durchlaufen der PWM::init-Methode muss der PWM-Generator zwar aktiviert sein, damit die anderen Klassenmethoden funktionieren, an den Ausgängen darf aber kein Signal anliegen, bis die Methode setDuty erstmals mit einem Tastgrad verschieden von Null aufgerufen wurde.

Denken Sie daran, bei Bedarf die FPU zu aktivieren (siehe System-Klasse)

4.1.4 PWM::setFreq

- Rückgabewert/-typ: **void**
- Übergabeparameter: **uint32_t** freq: Frequenz in Hz
- Funktion/Umsetzung:
  Konfiguriert den PWM-Generator so, dass seine Frequenz dem übergebenen Wert entspricht.

  (B) Die System Klasse initialisiert den PWM Unit Clock Divisor bereits auf einen Wert, der für alle Aufgaben ausreichend ist. Um die PWM-Klasse noch universeller zu gestalten, soll diese Methode selbst den optimalen Teiler wählen und mithilfe der entsprechenden System Methode setzen. Ziel ist es, einen möglichst großen Frequenzbereich mit möglichst großer Auflösung abzudecken. Da diese Funktion nur unter bestimmten Voraussetzungen verwendet werden kann, müssen Sie einen zusätzlichen optionalen Übergabeparameter **adjustPWMClockDiv** hinzufügen, der den default Wert **false** hat. Nur wenn dieser explizit auf **true** gesetzt wird, darf die Methode den Teiler ändern.

4.1.5 PWM::setDuty

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  float dutyLeft, dutyRight: Tastgrad als Gleitkommazahl zwischen -1.0f und 1.0f für beide Motoren. Das Vorzeichen bestimmt die Drehrichtung.
- Funktion/Umsetzung:
  Diese Methode setzt die gewünschte Pulsweite und die Richtungspins der Motoren. Je nach Richtung, also je nachdem ob der übergebene Tastgrad positiv oder negativ ist, muss der jeweilige Richtungspin (siehe Config.h) gesetzt werden. Als Konvention gilt, dass ein Tastgrad größer als Null ,,vorwärts" bedeutet und in dem Fall der Richtungspin auf HIGH liegt. Weil die PWM-Generatoren bei einem Compare-Wert von Null ein PWM Signal mit 100% Tastgrad ausgeben (statt 0%), müssen in diesem Fall beide Ausgänge deaktiviert werden.

4.1.6 Testprogramm

- Mindestens zu prüfende Methoden

  1. PWM::init
  2. PWM::setFreq
  3. PWM::setDuty

- Programmvorschlag
  Das PWM-Testprogramm ermöglichst es Ihnen, zwei Motoren und deren Ansteuerung mit dem Treiber SN754410 zu testen. Der Tastgrad soll dabei schrittweise um 1% erhöht werden bis dieser 100% erreicht und danach wieder verringert werden bis dieser -100% erreicht. Wenn Sie dabei den vorgeschlagenen Hardware Aufbau benutzen, können Sie die Richtung anhand von LEDs erkennen. Denken sie allerdings daran, die Richtungspins vorher als GPIO Objekte zu initialisieren und an die PWM Klasse zu übergeben.

  Betreiben Sie die Motoren einmal mit einer Frequenz von 2500Hz und ein anderes Mal mit der Standardfrequenz der PWM Klasse.

  Denken Sie daran, dass sie ein zusätzliches GPIO-Objekt für den Enable-Pin des Treibers benötigen. Ansonsten werden sich die Motoren nicht drehen. Falls sich trotz Enable nichts tut, können Sie auch die on-board LEDs als PWM-Ausgang verwenden (PF2, PF3). So können Sie schnell bestimmen, ob der Fehler in Ihrem Code oder Ihrem Aufbau liegt.

4.2 UART Klasse

4.2.1 Was ist UART?

Da es sich um ein Ferngesteuertes Fahrzeug handelt, gibt es auf der Fernbedienung und dem Fahrzeug jeweils ein TivaBoard. Damit das TivaBoard auf der Fernbedienung mit dem Tivaboard auf dem MagniSilver kommunizieren kann, muss eine Kommunikationsschnittstelle initalisiert werden. Hierfür soll das meistgenutzte Hardware Kommunikationsprotokoll UART (universal asynchronous receiver-transmitter) verwendet werden. Die Kommunikation basiert dabei auf zwei getrennten Kommunikationsleitungen (einer Empfängerleitung r_x und einer Sendeleitung t_X). Da es sich um eine asynchrone Kommunikationsschnittstelle handelt, müssen Sender und Empfänger synchronisiert werden. Das wird durch ein bis zwei Startbit(s) realisiert (siehe Figure 24). Die tatsächliche Nachricht kann dabei 8-9 bit beanspruchen. Um eventuelle Bitfehler bei der Übertragung festzustellen, kann auch noch ein Paritätsbit ergänzt werden. Optional können noch bis zu zwei Stopbits hinzugefügt werden, um deutlich zu machen, dass das Datenbyte hier endet. Um die jeweiligen Bits korrekt erkennen zu können, müssen Sender und Empfänger auf der gleichen Frequenz laufen, da sonst keine Kommunikation möglich ist. Die Frequenz wird als sogenannte Baudrate bezeichnet, die Anzahl der Symbole pro Sekunde. Die Konfiguration des UART Moduls (Anzahl der Start/Paritäts/Stopbits,...) ist Ihnen selbst überlassen.

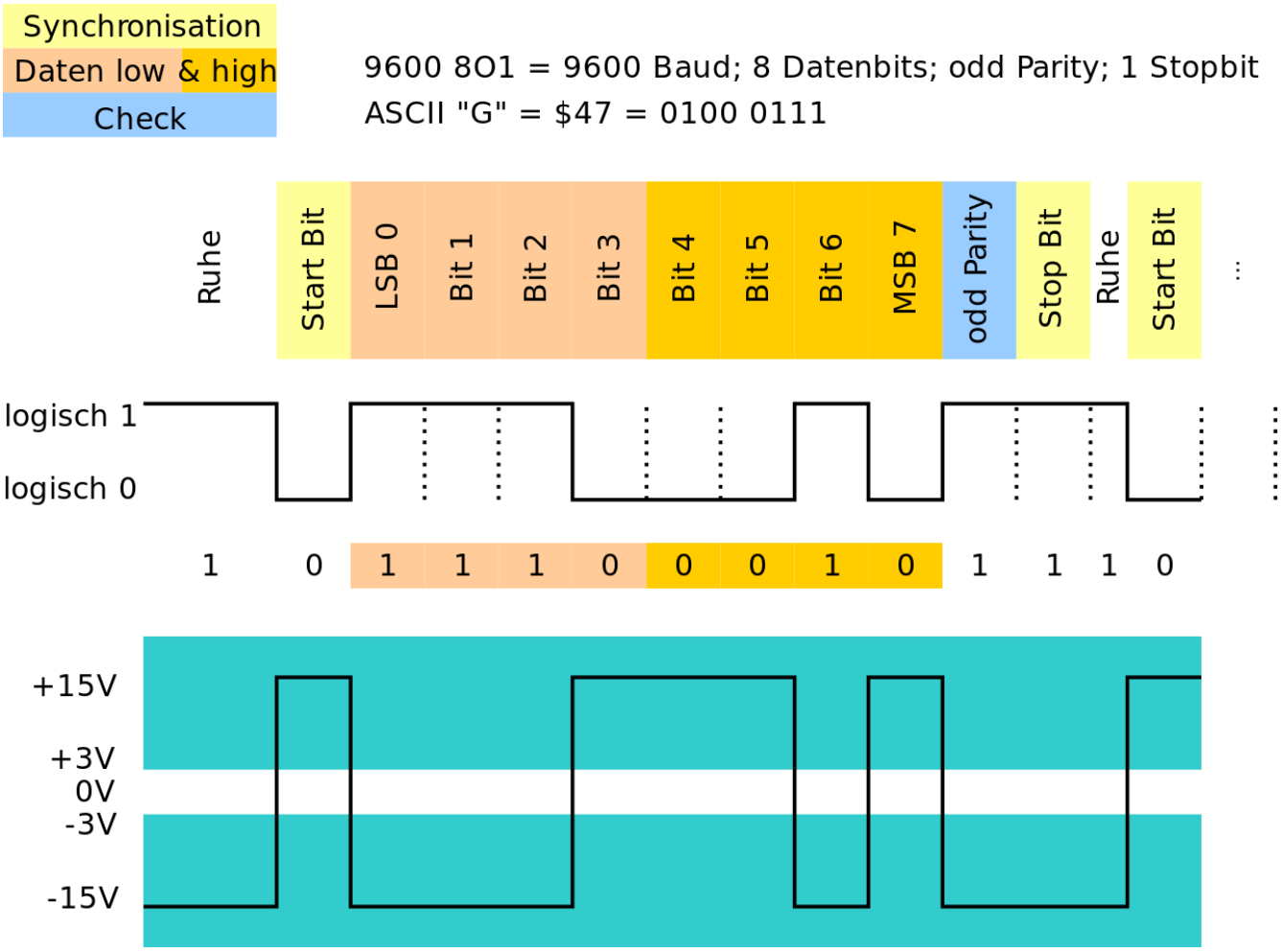

Figure 24: Schematische Darstellung eines UART Bytes [1]

Da die Kommunikation zwischen Magni und Fernbedienung mehrere Datenbytes beansprucht wird ein Synchronisationswort (siehe Config.h) an den Beginn der Nachricht gehängt, um den Beginn der Nachricht zu signalisieren. Dies soll sicherstellen, dass der erste empfangene Wert auch der erste Wert der Nachricht ist. Zur Kommunikation zwischen Magni und Fernbedienung wird ein ESP8266 verwendet. Es kommt vor, dass dieser die zu sendende Nachricht nicht direkt hintereinander sendet, was dazu führt, dass nur eine halbe Nachricht ankommt oder dass 1.5 Nachrichten ankommen. Es sollen nur vollständig empfangene Nachrichten verwendet werden. Wenn nicht genau eine vollständige Nachricht empfangen wird, soll die Nachricht gelöscht werden.

4.2.2 UART::init

- Rückgabewert/-typ: **void**
- Übergabeparameter:

  System *sys: Zeiger auf die laufende System Instanz

  uint32_t portBase: Adresse des GPIO Ports der für die Kommunikation genutzt wird, z.B. GPIO_PORTA_BASE

uint32_t tx: Adresse des Pins der zum Senden von Nachrichten verwendet wird, z.B. GPIO_PIN_0

uint32_t rx: Adresse des Pins, der zum Empfangen von Nachrichten verwendet wird, z.B. GPIO_PIN_1

- Funktion/Umsetzung:
  Analog zur GPIO::init Methode führt diese Methode die Initalisierung des UART Moduls durch. Jedes UART-Objekt beansprucht dabei genau ein UART Modul. Das TivaBoard besitzt 8 UART Module. Zur Initalisierung gehören u.a. die Freischaltung der GPIO und UART Ports, die Konfigurierung der GPIO Pins, die Konfigurierung des UART Protokolls, sowie das Freischalten des FIFO.

  Das korrekte UART Modul sowie alle weiteren zur Konfiguration benötigten Konstanten können sie anhand der übergebenen Werte für **portBase**, **tx** und **rx** bestimmen.

  ⓘ Anstatt **portBase**, **tx** und **rx** einzeln abzugleichen, können alle drei Variablen auf einmal verglichen werden, indem man sie bitweise verodert. Soll z.B. geprüft werden, ob es sich um die Pins **PB0** und **PB1** handelt, kann das wie folgt geprüft werden:

  ```
  if((portBase|tx|rx) == (GPIO_PORTB_BASE|GPIO_PIN_0|GPIO_PIN_1))
  ```

  (Siehe auch "inc/hw_memmap.h" Zeile 53 folgend und "driverlib/gpio.h" Zeile 60 folgend). Aufgrund der Vielzahl an zulässigen Pinkombinationen sollten sie jedoch keine if-else-Verzweigung benutzen. Auch eine switch-case-Anweisung führt in den meisten Fällen zu sehr langen Lösungen. Versuchen sie stattdessen, mit einem Array zu arbeiten, den Sie nach der passenden Pin-Kombination durchsuchen.

  ⓘ Die Werte für UARTx_BASE sind nicht zufällig. Tatsächlich gilt
  UART(x+1)_BASE = UARTx_BASE + 0x1000.
  Überlegen Sie wie sie dies nutzen können.

  Nach Durchlaufen der UART::init-Methode soll das UART Modul funktionsfähig sein und durch die beiden Methoden UART::send und UART::receive vollständig gesteuert werden können.

4.2.3 UART::send

- Rückgabewert/-typ: **void**
- Übergabeparameter:

  const char *transmit: Zeiger auf das Char-Array, dass gesendet werden soll.

  uint32_t length: Länge der zu sendenden Nachricht

- Funktion/Umsetzung:
  Sendet die Nachrichten die durch das ,,transmit" Array übergeben wurden.

4.2.4 UART::receive

- Rückgabewert/-typ: **bool** successful
- Übergabeparameter:

  const char *receive: Zeiger auf das Array, in der die Empfangsnachricht gespeichert werden soll.

  uint32_t length: Die erwartete Länge der empfangenenen Nachricht.

- Funktion/Umsetzung:
  Liest das FIFO Empfangsregister aus, dekodiert die empfangene Nachricht und speichert die Daten im Array dessen Adresse übergeben wurde.

  Die Methode gibt **true** zurück, wenn die Nachricht korrekt empfangen wurde, andernfalls gibt sie **false** zurück.

  Bitte beachten Sie, dass die Nachricht, wie in Kapitel 4.2.1 beschrieben wird, ein Synchronisationsbyte besitzt, dass nicht in das Datenarray geschrieben werden soll. Ebenso sollen alle Nachrichten, die nicht vollständig angekommen sind, gelöscht werden.

4.2.5 Testprogramm

- Mindestens zu prüfende Methoden

  1. UART::init
  2. UART::send
  3. UART::receive

- Programmvorschlag:
  Das UART Testprogramm ermöglicht es Ihnen, die Funktionalität Ihrer UART Klasse mit nur einem TivaBoard zu testen. Initialiseren sie zwei Objekte der UART Klasse. Schicken Sie eine beliebige Nachricht (beginnend mit dem Synchonisationswort) von einem Objekt zum anderen. Wenn dieses die Nachricht korrekt empfängt, lassen sie eine LED leuchten.

4.3 Distance Klasse

4.3.1 Einleitung

Der MagniSilver besitzt 5 Ultraschallsensoren, die für die Distanzmessung verantwortlich sind. Der verwendete Sensor ist ein HC-SR04. Das Datenblatt des Sensors finden Sie im Dokumente und Infos Repository. Der Sensor erwartet einen min. $10\mu s$ Puls auf dem ,,trigger" Pin, um die Messung zu starten. Der Sensor sendet dabei acht 40kHz Signale aus und wartet dann, ob er ein reflektiertes Signal empfängt. Sobald die Messung gestartet wird, geht der ECHO Pin auf High. Wenn ein Signal empfangen wird, dann wird der ECHO Pin wieder auf LOW gezogen. Die Breite des Pulses des ECHO Pins korreliert direkt mit dem Abstand des Objektes. Die Umrechnung ist

wie folgt:

$$\text{Distanz}\,[cm] = \text{Pulslänge}\,[s] \cdot \text{Schallgeschwindigkeit}\,[m/s] \cdot 100/2 \tag{2}$$

Die Sensor kann eine Distanz von 2cm - 4m auflösen. Die Aufgabe der Distance Klasse ist es diesen Sensor anzusteuern und auszulesen. Dafür sollen die 32/64-Bit Wide Timer verwendet werden. Hint: Sie wollen die Breite des Pulses bestimmen, beachten Sie aber auch was passiert, wenn kein Objekt detektiert wird.

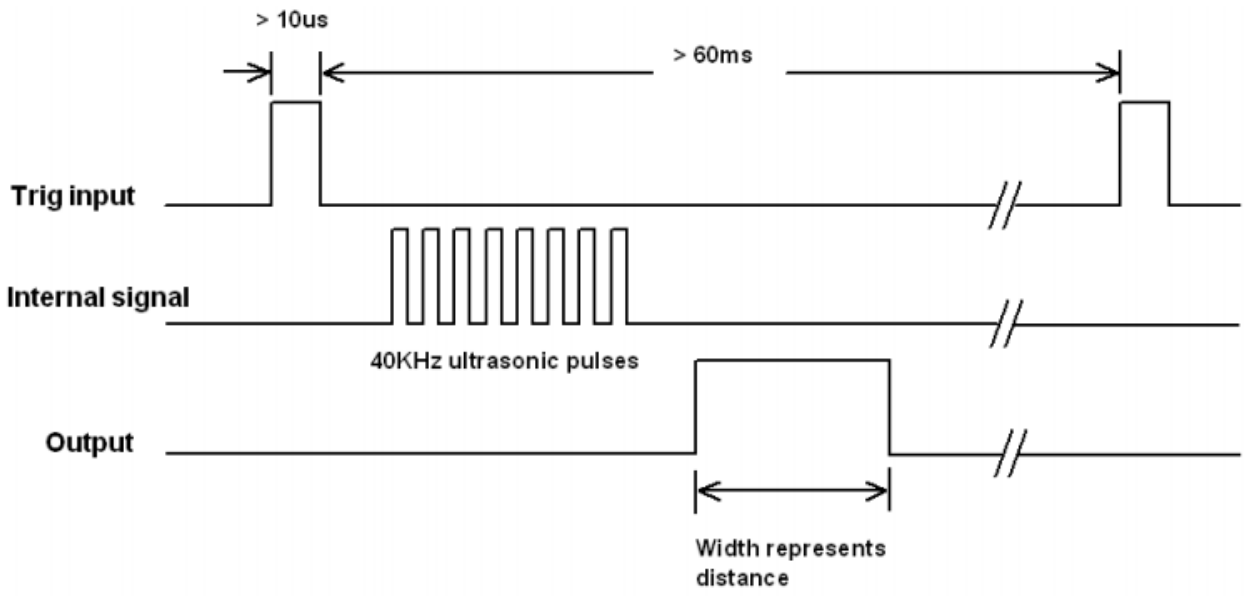

Figure 25: Timing des Sensors [2]

4.3.2 Distance::init

- Rückgabewert/-typ: **void**
- Übergabeparameter:

  System *sys: Zeiger auf die laufende Systeminstanz

  uint32_t echoPort: Adresse des GPIO Ports der für die Messung des Signals verantwortlich ist, z.B. GPIO_PORTA_BASE

  uint32_t echoPin: Adresse des GPIO Pins der für die Messung des Signals verantwortlich ist, z.B. GPIO_PIN_0

  uint32_t triggerPort: Adresse des GPIO Ports der für die Auslösung des Sensors verantwortlich ist, z.B. GPIO_PORTA_BASE

  uint32_t triggerPin: Adresse des GPIO Pins der für die Auslösung des Sensors verantwortlich ist, z.B. GPIO_PIN_0

  void (*ISR)**(void)**: Zeiger auf die Interrupt Service Routine die ausgelöst werden soll, wenn ein Interrupt ausgelöst wird.

- Funktion/Umsetzung:
  Analog zur GPIO::init führt diese Methode die Initialisierung des Distance Objektes durch. Jedes Objekt darf dabei genau einen Wide Timer beanspruchen. Zur Initalisierung gehören die Freischaltung der GPIO Module, das Konfigurieren der Pins als Timer Pins sowie die Freischaltung und Konfigurierung der 32/64 Bit Wide Timer.

4.3.3 Distance::trigger

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Startet die Messung des Ultraschallsensors.

4.3.4 Distance::edgeDetect

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Diese Methode wird aufgerufen wenn eine Flanke durch den Timer erkannt wird und berechnet beim auftreten der zweiten Flanke die gemessene Distanz.

4.3.5 Distance::getDistance

- Rückgabewert/-typ: **float** distance
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Gibt die letzte gemessene Distanz zurück.

4.3.6 Testprogramm

- Mindestens zu prüfende Methoden
  1. Distance::init
  2. Distance::trigger
  3. Distance::edgeDetect
  4. Distance::getDistance
- Programmvorschlag:
  Das Distance Testprogramm ermöglichst es Ihnen, die Funktionalität ihrer Distance Klasse mit dem Tivaboard zu testen. Die Aufgabe ist es einen Ultraschallsensor zu initialisieren und zu steuern. Wenn die Distanz zu einem Objekt kleiner ist als 30cm dann soll eine LED anfangen zu leuchten. Ist die Distanz größer als 30cm dann soll die LED wieder ausgeschaltet werden.

4.4 Steering Klasse

Die Lenkung besteht aus einem 2D-Joystick auf der Fernsteuerung. Der Joystick besteht aus zwei linearen Potentiometern, je einer für jede Achse (x,y). Eine Neigung des Joysticks resultiert in einer Änderung der Spannung. Die Spannungsänderung kann durch einen ADC Pin ausgelesen werden. Jede Achse kann unabhängig voneinander abgelesen werden. Zum Testen der Steering Klasse, benötigen Sie die Remote_Class_Test,main

4.4.1 Steering::init

- Rückgabewert/-typ: **void**
- Übergabeparameter:

  System *sys: Zeiger auf die laufende System Instanz

  uint32_t baseX: Adresse des ADC Moduls das den Wert für die x-Achse ausliest, z.B. ADC0_BASE

  uint32_t baseY: Adresse des ADC Moduls das den Wert für die y-Achse ausliest, z.B. ADC0_BASE

  uint32_t sseqX: Adresse des ADC Sample Sequencers, der den Wert für die x-Achse ausliest (0, 1, 2, 3)

  uint32_t sseqY: Adresse des ADC Sample Sequencers, der den Wert für die y-Achse ausliest (0, 1, 2, 3)

  uint32_t analogInX: Analoger Eingang, der den Wert für die x-Achse ausliest, z.B. ADC_CTL_CH0

  uint32_t analogInY: Analoger Eingang, der den Wert für die y-Achse ausliest, z.B. ADC_CTL_CH0

- Funktion/Umsetzung:
  Initialisiert die ADC Pins, die zum Auslesen der x,y Achsen verwendet werden.

4.4.2 Steering:calcValue

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Liest die Spannung an beiden ADC Pins aus und berechnet daraufhin PWM Werte für die Motoren.

4.4.3 Steering:getLeftSpeed

- Rückgabewert/-typ: **float** leftSpeed
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Gibt den berechneten PWM Wert (-1.0f...1.0f) für den linken Motor zurück

4.4.4 Steering:getRightSpeed

- Rückgabewert/-typ: **float** rightSpeed
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Gibt den berechneten PWM Wert(-1.0f...1.0f) für den rechten Motor zurück

4.4.5 Testprogramm

Die Steering Klasse kann mithilfe des Remote Testprogramms getestet werden. Dabei können die jeweiligen Werte mithilfe der Methode „setDebuggingValues" gesetzt werden und über die Arduino IDE ausgelesen werden.

4.5 Batteriespannungsüberwachung

4.5.1 Motivation und Anforderungen

Die Spannung eines Akkus darf nicht unter einen bestimmten Wert sinken, da er ansonsten Schaden nimmt oder gar unbrauchbar wird. Für das Akkupack des MagniSilver liegt diese Grenze bei 21V. Obwohl mit fortschreitender Entladung auch die Leistungsfähigkeit nachlässt, ist es für eine außenstehende Person nicht möglich zu erkennen, ab wann der Akku Schaden nimmt. Es bedarf daher einer genauen Spannungsüberwachung. Der entsprechende ADC-Pin ist auf dem MagniSilver an die Akkuspannung angeschlossen. Alles was für sie relevant ist, dass sie zu jedem Zeitpunkt 3/28 der aktuellen Batteriespannung messen. Da Ihr Modellaufbau keine Akkus besitzt, können Sie die Spannung mittels eines Potentiometers beliebig einstellen und so ihre Batteriespannungsüberwachung testen.

Der MagniSilver steht im ständigen Austausch mit der Fernbedienung und kann somit auch dem Benutzer eine zu niedrige Batteriespannung anzeigen. Dafür existiert im Kommunikationsprotokoll die sogenannten ErrorCodes. Integrieren Sie eine Batteriespannungsüberwachung in die Magni::background-Methode die einen BatteryLow Error Code an die Fernbedienung schickt, sobald die Batteriespannung länger als 5 Sekunden durchgehend unter dem Grenzwert liegt.

Nutzen Sie bei der Umsetzung soweit wie möglich die bereits in der Config.h hinterlegten Konstanten (siehe Arbeiten mit der Config.h).

4.6 Dokumentation

Zu jedem Code-Projekt gehört Dokumentation. Für das erfolgreiche Bestehen wird von ihnen daher ein kommentierter Code, der über aussagekräftige Commits gepushed wurde, erwartet. Zusätzlich müssen Sie verpflichtend ein Gantt-Chart mit einer Planung über den Bearbeitungszeitraum bis Ende der dritten Bearbeitungswoche hochladen. Darin sollen die Arbeitspakete klar erkenntlich voneinander getrennt sein und die Zuständigkeit der einzelnen Teammitglieder geklärt sein. Anhand der Zuteilung werden die Fragen in der Codebesprechung gestellt. Geben Sie außerdem den kritischen Pfad an.

5 Test und Debugging

5.1 Arbeiten mit dem CCS Debugger

CCS sollte Ihnen mittlerweile mehr als nur vertraut sein, auch den Debugger und dessen Funktionen haben Sie bereits kennengelernt. Im Folgenden geben wir Ihnen Tipps, wie Sie dieses äußerst mächtige Werkzeug sinnvoll einsetzen können.

Falls Ihr Code noch nicht ordnungsgemäß funktioniert, dann ist es zunächst hilfreich das Programm im Debugging-Modus auszuführen und anschließend zu pausieren. Nun gibt es einige Szenarien, die auftreten können.

- Der Code pausiert in der FaultISR:
  Dies geschieht, wenn das Programm etwas tun möchte, was nicht möglich oder nicht erlaubt ist, beispielsweise einen GPIO Pin anzusteuern, wenn das Modul noch gar nicht freigeschaltet ist.
  Hier bietet es sich an, Ihren Code in der main mittels ,,Step Over" zu untersuchen. Haben Sie nun herausgefunden, aus welcher Zeile das Programm in die FaultISR springt, können Sie dort einen Breakpoint setzen und das Programm neu starten. Nun können Sie den Code ausführen lassen und er wird in der Zeile pausiert, in der Sie den Breakpoint gesetzt haben. Jetzt bietet sich ,,Step Into" an, um die aufgerufene Methode Zeile für Zeile zu untersuchen. Dieses ,,Step Over" → Breakpoint → ,,Step Into" Prozedere wiederholen Sie so oft, bis Sie in einer Zeile angelangt sind, und er eine TivaWare™API Methode aufgerufen wird. Da Sie davon aufgehen können, dass letztere korrekt funktionieren, bedeutet das, dass entweder die Übergabeparameter Ihrer Methode fehlerhaft sind, oder nötige Hardware-Konfigurationsschritte vorher nicht unternommen worden sind.

- Es wird eine leere Seite angezeigt:
  Sollten Sie das Projekt zum ersten Mal debuggen, ist dies normal. Das Programm hat innerhalb der TivaWare ™Bibliothek pausiert, der Debugger weiß aber noch nicht, wo der zugehörige Quellcode liegt. Sie müssen mittels des ,,Browse" Buttons zu Ihrem TivaWare™ Ordner navigieren und innerhalb dessen den Ordner ,,driverlib" auswählen. Nun sollte Ihnen der Quellcode der TivaWare™Bibliothek richtig angezeigt werden.
  Es wird Ihnen allerdings auch danach noch vorkommen, dass Sie diese weiße Seite sehen. Das liegt daran, wie Delays auf dem Mikrocontroller funktionieren. Wenn Sei innerhalb eines Delays pausieren (was je nach Dauer sehr wahrscheinlich ist), kann Ihnen kein Quellcode angezeigt werden. Sie können mithilfe der Step-Funktionen und Breakpoints (siehe oben) den Fall umgehen.

- Falls der Code nicht in der FaultISR pausiert:
  In diesem Fall bleibt Ihnen meist nicht anderes übrig, als die Funktionsweise des Codes Schritt für Schritt mittels ,,Step Into" und ,,Step Over" zu überprüfen. Hierbei kann auch die Beobachtung von Variablen im ,,Variables" und ,,Expressions" Fenster bei der Fehlersuche helfen. Im ,,Variables" Fenster werden automatisch alle Variablen angezeigt, die in der beobachteten Methode verwendet werden (also z.B. keine privaten Variablen anderer Objekte). Im ,,Expressions" Fenster können Sie sämtliche Variablen, die im Programm vorkommen - auch Objekte sowie deren Zeiger - dauerhaft zum Beobachten hinzufügen (,,Add new

Expressions"). Für ganz harte Fälle ließen sich noch die einzelnen Register im entsprechenden Fenster auslesen, dies ist jedoch normalerweise nicht nötig und erfordert einiges Wissen über die genaue Funktion der einzelnen Register. Bevor Sie sich an die Register wagen, empfehlen wir, dass Sie sich die Informationen in diesem Handbuch und die Issues auf GitLab nochmal ansehen. Eventuell finden Sie hier die nötigen Informationen, um Ihr Problem zu identifizieren. Falls dies nicht zum Ziel führt und Ihre Teamkameraden Ihnen ebenfalls nicht helfen können, sollten Sie die Sprechstunde besuchen, wo Ihnen nach Möglichkeit geholfen wird.

5.2 Verwenden der System::error-Methode

Der CCS Debugger ist ein äußerst hilfreiches und mächtiges Tool, jedoch nicht immer einfach in der Handhabung. Vor allem bei Overflows und der vorhin beschriebenene FaultISR ann das Debuggen sehr umständlich sein. Um diese Problem teilweise zu umgeben, wurde in der System-Klasse die System::error- Methode implementiert. Das Aufrufen dieser Funktion stoppt alle Interrupts und setzt den Code in eine Endlosschleife. Beim Aufruf können optional ein Fehlercode und bis zu drei Variablen übergeben werden. Hierzu gibt es die Datei *Errorcodes.h*, die eine Aufzählung an Fehlercodes enthält. Ebenfalls enthalten ist die Liste der Variablen, die jeweils zusammen mit dem Fehlercode zum Debuggen übergeben werden. Die bereits enthaltenen Fehlercodes dürfen Sie verwenden, aus Kompatibilitätsgründen jedoch nicht ändern oder entfernen. Es steht Ihnen frei, weitere Fehlercodes hinzuzufügen.

Um mit der System::error-Methode zu arbeiten, müssen Sie diese an den kritischen Stellen in Ihren Klassen einbauen. Ziel ist es, alle problematischen Fälle (Overflow, FaultISR,...) zu vermeiden und stattdessen in dieser Methode zu landen. Der Fehlercode und die Variablen zeigen Ihnen an, von welcher Stelle aus Sie dort gelandet sind. Mit dem Debugger können Sie dies Schritt für Schritt nachverfolgen. In der fertigen GPIO-Klasse ist die System::error bereits implementiert, nutzen sie diese, um sich damit näher vertraut zu machen.

5.3 Einbinden der Klassenbibliotheken mit Musterlösung

5.3.1 Zweck

Der Arbeitsaufwand des Praktikums wurde derart von uns berechnet, dass eine Dreiergruppe gemeinsam am Projekt arbeitet. Besteht Ihre Gruppe nur aus zwei aktiven Mitgliedern oder arbeiten Sie ganz alleine, steht außer Frage, dass Sie nicht das gesamte Praktikum bearbeiten müssen. Um trotzdem den Gesamtaufbau zu testen, sind von uns für jede Klasse Bibliotheken mit der Musterlösung bereitsgestellt. Diese Bibliotheken dürfen Sie für die nicht von Ihnen bearbeiteten Teile der Aufgabenstellung verwenden. Richten Sie sich im Zweifelsfall in den Sprechstunden an die Verantwortlichen des Praktikums.

Auch wenn Sie Teil einer Dreiergruppe sind und alle Mitglieder gewissenhaft mitarbeiten, können sie von den Klassenbibliotheken Gebrauch machen. Nachdem Sie eine Klasse und das zugehörige Testprogramm geschrieben haben und es nicht läuft, können Sie die Bibliothek zur Fehlersuche nutzen: Funktioniert Ihr Testprogramm mit der Bibliothek, so liegt der Fehler definitiv in der Klasse (und nicht im Testprogramm). Bei der Abgabe und der mündlichen Prüfung dürfen die Bibliotheken natürlich **nicht** mehr eingebunden sein!

5.3.2 Funktionsweise

In der Vorlage sind die Bibliotheken für die zu schreibenden Klassen enthalten. Sie sind bereits in Ihren Programmvorlagen integriert und müssen bei Bedarf nur noch eingebunden werden. Hierzu gibt es in den .cpp-Dateien der entsprechenden Klassen am Anfang ein # **define**, das standardmäßig auskommentiert ist. Durch Entfernen der beiden Schrägstriche wird die Codezeile berücksichtigt und die Bibliothek eingebunden. In diesem Fall wird die Musterlösung verwendet und Ihr Code innerhalb der Klasse nicht mehr vom Compiler beachtet. Möchten Sie im Anschluss Ihren Code wieder nutzen, so kommentieren Sie das # **define** wieder aus.

5.4 Testen mit dem Testaufbau

Nachdem Sie Ihren Testaufbau wie im Kapitel Testaufbau beschrieben aufgebaut haben, können Sie Ihren Code auf das LaunchPad flashen.
Auf das TivaBoard, welches mit dem Joystick verbunden ist, muss das Remote Test Programm geflasht werden. Auf das andere TivaBoard wird das Magni Test Programm geflasht. Nun können Sie mithilfe des Joysticks die Motoren steuern. Halten Sie nun ein Objekt vor den Ultraschallsensor kommen die Motoren zum Stillstand.

Zuletzt können Sie noch Ihre Akkuspannungsüberwachung testen. Stellen Sie dafür das Potentiometer in eine Extremstellung („Akku vollständig entladen"). Nach fünf Sekunden müssen die Motoren zum Stillstand kommen. Erst nach Zurücksetzen des Potentiometers und einem Reset des LaunchPad ist der Testaufbau wieder betriebsbereit.

5.5 Häufige Fehler

5.5.1 C++ Fehler

- Arrays sind in C++ nullindiziert, das heißt, dass das erste Element den Index Null hat und das letzte Element den Index „Länge minus eins".
- Variablen innerhalb eines Gültigkeitsbereichs (Scope) blenden Variablen mit gleichem Namen aus dem äußeren Scope aus (siehe Variable Shadowing). Um auf Klassenvariablen zuzugreifen, wenn es eine gleichnamige Variable in dem aktuellen Scope gibt, kann der Pointer auf die aktuelle Instanz der Klasse (this-¿) verwendet werden.
- Da der Mikrocontroller nur 32 Bit Gleitkommazahlen effizient verarbeiten kann, wurde der Compiler so konfiguriert, dass nur diese akzeptiert werden. Der Standardtyp für Gleitkommazahlen in C++ ist jedoch **double** (64 Bit). Um entsprechende Fehlermeldungen zu vermeiden, müssen Sie unbedingt den Datentyp **float** benutzen bzw. ein f hinter die Gleitkommazahl hinzufügen (bspw. 3.0f).
- Teilt man einen Integer (Ganzzahl) durch einen anderen Integer, so wird eine Ganzzahldivision durchgeführt (bspw. $7/2 = 3$, $5/3 = 1$, $1/4 = 0$). Dies kann vermieden werden, indem Sie explizit angeben, dass eine der Variablen als float, also als eine Gleitkommazahl behandelt werden soll (Stichwort Type casting).

  Da jede Konvertierung zusätzlichen Rechenbedarf bedeutet und Gleitkommarechnungen auf Mikrocontrollern oft aufwändiger zu verarbeiten sind, ist es ratsam, sie wo nur möglich zu

umgehen. Die Probleme der Ganzzahldivision lassen sich oft durch ein geschicktes Umstellen der Rechnung erreichen.

Beispiel: Gewünscht ist die Funktion, die einen Preisnachlass berechnet. Übergeben werden der ursprüngliche Preis und der gewährte Rabatt in Prozent.

Schlecht:

```
uint32_t besterPreis(uint32_t alterPreis, uint32_t prozent)
{
    uint32_t nachlass = prozent / 100 * alterPreis;
    return alterPreis - nachlass;
}
```

Wir erinnern uns, in C++ werden Operationen gleicher Priorität von links nach rechts ausgewertet. Das bedeutet, dass zuerst prozent / 100 geteilt wird und das Ergebnis dann erst mit alterPreis multipliziert wird. Da prozent aber nie größer als 100 ist, kommt bei der Ganzzahldivision fast immer Null raus (und nur für prozent = 100 eine Eins). Dieser Code kann somit keinne Nachlass von z.b. 10%, 20%, ... berechnen.

Gut:

```
uint32_t besterPreis(uint32_t alterPreis, uint32_t prozent)
{
    uint32_t nachlass = (prozent * alterPreis) / 100;
    return alterPreis - nachlass;
}
```

Dadurch, dass wir nun zuerst eine Multiplikation durchführen, wird die Zahl deutlich größer als 100 und der Genauigkeitsverlust bei der Division wird minimiert. Diese Rechnung liefert genau das gleiche Resultat wie die Berechnung mit Gleitkommazahlen (und anschließender Rundung), ist aber einfacher und schneller.

Die Klammern sind streng genommen nicht nötig und sollen nur verdeutlichen, dass die Multiplikation mit Absicht zuerst ausgeführt wird.

- Ein uintxx_t ist eine Ganzzahl ohne Vorzeichen, also positiv. Darüber hinaus hat sie einen festen Wertebereich, den sie abdeckt. So kann ein uint8_t nur Zahlen von 0 bis 255 beschreiben. Wenn Sie einen von beiden Grenzen überschreiten landen Sie am anderen Ende der Skala. Im genannten Beispiel wäre daher 0-1 = 255 und 250 + 10 = 4.
- Ein intxx_t ist ebenfalls eine Ganzzahl, allerdings mit Vorzeichen. Es ist ein ,,verschobener" uintxx_t, d.h. die Hälfte des Wertebereichs steht für negative Zahlen, die andere für positive. Die Größe des Intervalls bleibt die Gleiche. So deckt ein int8_t den Bereich von -128 bis 127 ab, also 256 Werte, genau wie ein unint_t. Ebenfalls gleich ist das Verhalten an den Enden der Skala: 120 - 10 = 126 und 100 + 100 = -56.

- Mehrfachabgleiche funktionieren in C++ nicht. (1 < variable < 15) ist äquivalent zu ((1 ¡ variable) < 15). Da (1 < variable) **true** oder **false** (also 1 oder 0) und somit immer kleiner als 15 ist, kommt bei der Mehrfachabfrage immer **true** raus. Es müssen daher zwei Einzelabfragen kombiniert werden: ((1 < variable) && (variable < 15)). Nun wird richtigerweise geprüft, ob variable sowohl größer als 1 als auch kleiner als 15 ist. Genauso ist zu beachten, dass für den ist-gleich Operator == anstatt des Zuweisungsoperators = verwendet wird. (variable = xxx) weist variable nicht nur den Wert xxx zu, sondern liefert diesen ebenfalls zurück. Dagegen vergleicht == die zwei Werte ohne sie zu ändern und gibt zurück, ob beide gleich sind oder nicht.

- Wenn Sie eine Variable vor ihrer Initialisierung verwenden, so hat sie einen zufälligen Wert. Das Programm wird somit immer kompiliert, allerdings kann es zu schwer auffindbaren Fehlern kommen, wenn Sie vergessen diesen Zufallswert mit sinnvollen Daten zu überschreiben. Die Empfehlung lautet daher, allen Variablen direkt bei der Deklaration einen bestimmten Wert zuzuweisen.

5.5.2 TI Besonderheit

- Falls Sie beim Testen in der FaultISR landen: Haben Sie dran gedacht, dass der Port freigeschaltet werden muss und dann noch gewartet werden muss, bis dies in der Hardware tatsächlich geschehen ist? Auch in anderen Bereichen ist es unter Umständen nötig Wartezeiten vorzusehen (siehe auch Arbeiten mit dem CCS Debugger).

- Achten Sie auf folgende Besonderheit von PWMGenPeriodSet(): Sie übergeben dieser Methode den Perioden-Wert für den Zähler des PWM Generators. Normalerweise müssten Sie für den Load-Wert (also der Wert, der ins Register geschrieben wird) noch 1 abziehen, doch in diesem Fall wird das von der Funktion selbst erledigt. Im PWM Register steht also immer ein Wert, der um 1 kleiner als der Übergabewert ist.
  Dies ist wichtig für den Compare Wert („Pulse-Width"). Da dieser stets kleiner als der Wert im Load Register sein muss, müssen Sie bei dessen Berechnung mit (periode - 1) rechnen.

5.5.3 GitLab

- Sie haben ein Wiki in Ihrem Repository, welches Sie gerne benutzen dürfen, wenn Ihnen danach ist. Allerdings können Sie aus Gründen der Rechteverwaltung keine Seiten löschen. Sie können einen Tutor mittels Issue darum bitten, es wird allerdings nicht seine höchste Priorität sein, das zu erledigen.

- Bitte posten Sie ihre Fragen nur im Fragen Repository. Wenn Sie in Dokumente und Infos posten, werden alle Studenten eine Mail dafür bekommen. So gehen wichtige Informationen und Ankündigungen unter. Wir werden keine Fragen im Dokumente und Infos Projekt beantworten.

- Falls Sie eine Frage auf Gitlab stellen, so laden Sie bitte Ihren aktuellen Quellcode in Ihr Repository hoch (nicht in das Issue kopieren!!) und verweisen Sie in Ihrem Issue auf die Stelle im Code an der Sie Fragen haben. Geben Sie unbedingt den Branch an, falls Sie nicht im master-Branch arbeiten!

Sie können, wenn Sie auf Gitlab ihren Quellcode öffnen, per Rechtsklick auf die Zeilennummern einen Link zu genau dieser Stelle kopieren. Alternativ Dokumentname und Zeilennummer/-bereich im Issue mit angeben.

6 Unterlagen

6.1 Programmstruktur

Im Rahmen des IT1 Praktikums werden Sie nicht den gesamten Code schreiben müssen, um den MagniSilver zum Fahren zu bringen. Die Grundarchitektur ist Ihnen bereits gegeben. Die Abstraktionsebenen sind in der folgenden Abbildung dargestellt. Nachfolgend finden Sie auch eine ausführliche Dokumentation der einzelnen gegebenen Klassen.

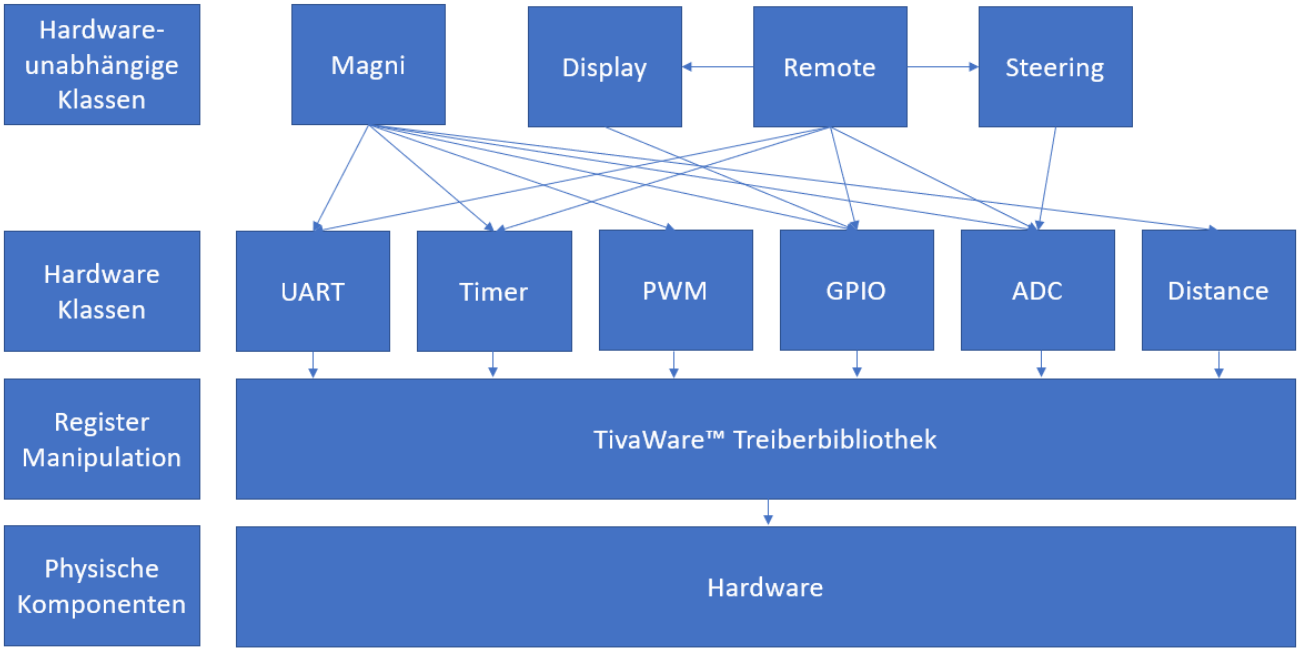

Figure 26: Programmstruktur

Zu beachten ist hierbei, das zwar in der Magni- und der Remote Klasse die ,,Magie" geschieht, jedoch die in der main() initialisierten Timer dafür sorgen, dass das Programm läuft!

6.2 Magni-Programm

6.2.1 Grundsätzliche Funktion

Solange Sie den Joystick nicht bewegen und sich keine Objekte vor den Ultraschallsensoren befindet, bewegen sich keine Motoren. Hat die Fernbedienung eine Verbindung zum MagniSilver und die Batteriespannung ist hoch genug, kann der MagniSilver mithilfe des Joysticks bewegt werden.

6.2.2 Programmablauf

6.2.2.1 Hauptprogramm

Die main() Funktion innerhalb des Magni_Class_Test/main.cpp initialisiert das System und den MagniSilver. Anschließend führt es die Hintergrundaufgaben in der Dauerschleife aus. Die Interrupt Service Routinen der beiden Timer innerhalb der main.cpp werden in Abbildung dargestellt.

Die Interrupt Service Routinen SensorxISR und die ISRLeft/ISRRight dienen der Auslesung der Sensoren, da eine Interrupt Service Routine nicht als Klassenmethode implementiert werden kann.

6.2.2.2 Magni::update

Diese Methode steuert das Verhalten des MagniSilver. Sie liest die Sensordaten ein, liest die empfangene Nachricht aus und aktualisiert die Motorsteuerung. Da diese Methode als periodischer Interrupt durchgeführt wird, enhält Sie nur Aufgaben, die zeitkritisch sind. Eine Interrupt Service Routine (ISR) enthält nur so viel Code wie nötig.

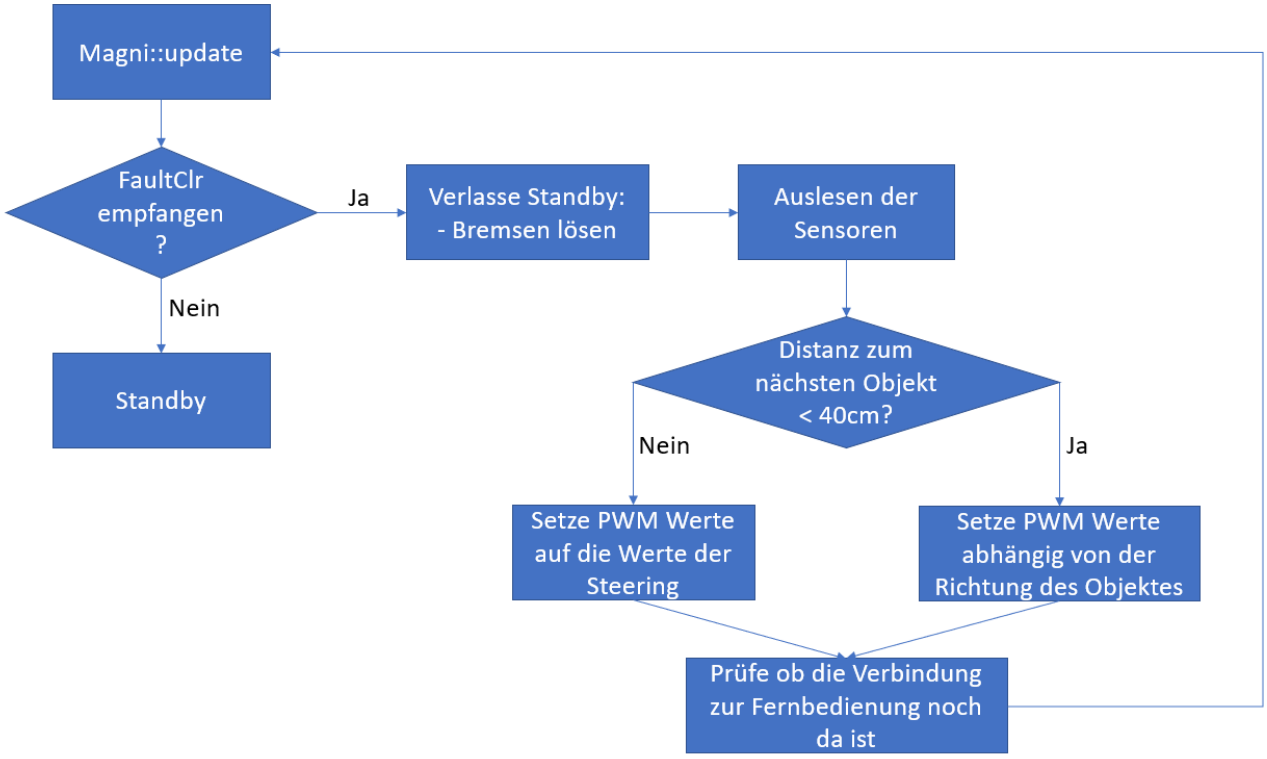

Figure 27: Programmstruktur der Magni::update Methode

6.2.2.3 Magni::background

Neben der Steuerung des MagniSilver müssen auch noch weitere Aufgaben im Hintergrund durchgeführt werden. Diese beinhalten das Senden der Daten zwischen MagniSilver und Fernbedienung sowie die Batteriespannungsüberwachung.

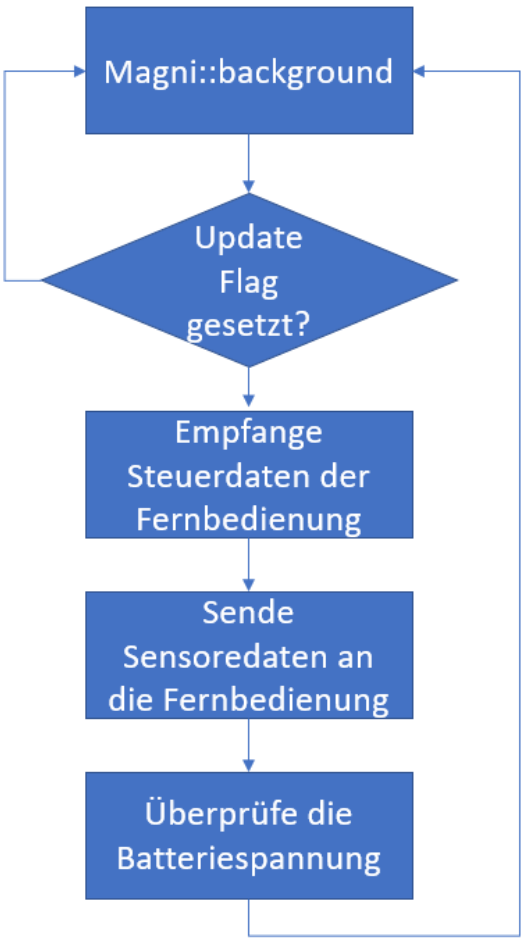

Figure 28: Programmstruktur der Magni::background Methode

6.2.2.4 Remote::update

Diese Methode steuert das Verhalten der Fernbedienung. Sie liest den Joystick aus, wandelt diese Werte in PWM Werte um, sendet und empfängt Nachrichten zwischen MagniSilver und Fernbedienung und steuert das Steuerungsmenü.

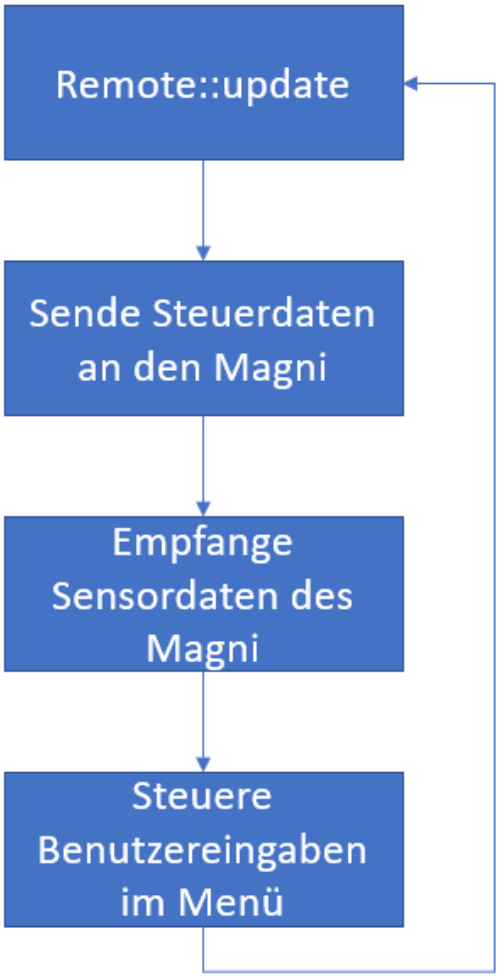

Figure 29: Programmstruktur der Remote::update Methode

6.2.2.5 Remote::updateDisplay

Diese Methode aktualisiert die Werte auf dem Display.

6.2.3 Konfiguration

Die gesamten Einstellungen, wie z.B. das Pinout oder die PWM-Frequenz, liegen in der Datei Config.h. Falls Sie Ihren Testaufbau wie im Schaltplan beschrieben aufgebaut haben, dann können Sie diese Datei auch in Ihre Testprogramme einbinden. Der Testaufbau und das tatsächliche MagniSilver nutzen bekanntlich den gleichen Programmcode und einen fast identischen Aufbau. Es gibt allerdings einige Unterschiede im Programmcode die berücksichtigt werden müssen.
Um diese Unterschiede zu berücksichtigen, müssen sie in der Config.h angeben, für welchen Aufbau sie das Programm momentan nutzen möchten. Welche Werte genutzt werden, können Sie über die zwei **#define** zu Beginn der Datei festlegen. Standardmäßig ist die Config.h auf den Testaufbau ausgelegt.

Wenn Sie beim Arbeiten am Segway Programm Konstanten und Einstellungen benötigen, prüfen Sie zunächst, ob sie nicht schon in der Config.h vorhanden sind. Falls nein, fügen Sie sie dort hinzu und nicht als Variable oder Konstante innerhalb der Klasse.

Es gibt eine Konfiguration für den MagniSilver und eine Konfiguration für die Fernbedienung. Beide befinden sich in der Config.h und werden durch ein **#define** gewechselt. Die genaue Konfiguration ist bereits in den Einstellungen der Testprogramme hinterlegt und muss nicht geändert werden. Um herauszufinden, welche Konfiguration benutzt wird, schauen Sie welchen Teil des Testaufbaus sie benutzen. Testen Sie z.B die Steuerung so wird die Remote Konfiguration benutzt wohingegen der PWM Test mit der Magni Konfiguration durchgeführt wird.

6.3 Gegebene Klassen

6.3.1 Einleitung

Nachfolgend finden Sie die vollständige Dokumentation der zur Verfügung gestellten Klassen. Trotz ausführlicher Beschreibung der Methoden sollten sie dich den kommentierten Quellcode durchlesen, um ein besseres Verständnis für die Funktion der Methoden zu erhalten. Insbesondere die GPIO-Klasse, welche Sie auch in der zweiten Einführungsveranstaltung bearbeiten werden, sollten Sie verstehen, ehe Sie mit der Aufgabenstellung beginnen. Aus diesem Grund ist sie hier als Aufgabenstellung beschrieben, so dass Sie selbst programmieren und mit der Musterlösung abgleichen können. Dies bereitet Sie optimal auf die restlichen Aufgaben vor.

6.3.2 GPIO Klasse

Die GPIO-Klasse dient dazu, die Pins des Tiva™ Boards als General Purpose Input/Output zu verwenden, also digitale Ein- und Ausgänge. So können z.B. Taster ausgelesen oder LEDs angesteuert werden. Sie ist damit die wichtigste Grundlage der Interaktion zwischen dem Mikrocontroller und der Umgebung.

6.3.2.1 GPIO::init

- Rückgabewert/-typ: **void**
- Übergabeparameter

  System *sys: Zeiger auf die laufenden Systeminstanz

  uint32_t portBase: Basisadresse des GPIO Ports, z.B. GPIO_PORTA_BASE

  uint32_t pin: Pin innerhalb des Ports, z.B. GPIO_PIN_0

  uint32_t dir: Legt fest, ob es sich um einen Eingang oder einen Ausgang handelt. Kann GPIO_DIR_MODE_IN oder GPIO_DIR_MODE_OUT sein.

  bool pullup: Optionale boolsche Variable, die angibt, ob der Pull-up-Widerstand verwendet werden soll, oder nicht. Der Standardwert ist false.

- Funktion/Umsetzung:
  Die GPIO::init-Methode führt alle nötigen Schritte aus, damit der Pin verwendet werden kann. Dazu gehört: alle Pin-Eigenschaften als private Variablen im Objekt speichern, den Port freischalten, den Pin als Ein- bzw. Ausgang konfigurieren und ggf. den Pull-up-Widerstand aktivieren.
  Die zu speichernden Eigenschaften sind:
  - uint32_t portBase: Die Basisadresse des Ports (z.B. GPIO_PORTF_BASE)
  - uint32_t pin: Der Pin innerhalb des Ports (z.B. GPIO_PIN_3)
  - uint32_t dir: Angabe, ob es sich um einen Ein- oder Ausgang handelt (z.B. GPIO_DIR_MODE_IN)
  - uint32_t current: Der aktuelle Maximalstrom in Form der API-Konstanten (z.B. GPIO_STRENGTH
  - uint32_t pinType: Der Pin-Typ und somit ob Pull-up oder Pull-down Widerstände aktiviert sind (z.B. GPIO_PIN_TYPE_STD)

  Es handelt sich dabei um private Variablen. Beachten Sie auch, dass manche API Funktionen Standardeinstellungen vornehmen. Aktualisieren Sie ggf. die zugehörigen Objektvariablen. Um zu bestimmen, welcher Registerzugriff für die Freischaltung dieses Ports nötig ist, verwenden Sie eine Switch Case Anweisung, die als Argument den Wert von portBase nutzt.

6.3.2.2 GPIO::read

- Rückgabewert/-typ: **bool**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Liefert **true** zurück, falls der Pin HIGH ist und **false** falls er LOW ist.

6.3.2.4 GPIO::write

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  bool state: Der zu setzende Wert.
- Funktion/Umsetzung
  Setzt den Pin auf den übergebenen Wert.

6.3.2.5 GPIO::getCurrent

- Rückgabewert/-typ:
  uint32_t Maximalstrom des Pins in mA
- Übergabeparameter: **void**

- Funktion/Umsetzung:
  Liefert den aktuellen Maximalstrom des Pins in mA zurück. Neben der Möglichkeit wie in der init-Methode eine Switch Case Anweisung zu nutzen, kann auch eine if-else Verzweigung verwendet werden. Bestimmen Sie mit letzterer den Wert der privaten Variable current und geben Sie den entsprechenden Strom in mA zurück.
  Überlegen Sie sich, was diese Funktion zurückliefern muss, wenn GPIO::setCurrent noch nicht aufgerufen wurde.

6.3.2.6 GPIO::setCurrent

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t current: Gewünschter Maximalstrom des Pins in mA
- Funktion/Umsetzung:
  Ändert den Maximalstrom des Pins auf den gegebenen Wert und aktualisiert die zugehörige private Variable current. Hierzu müssen Sie aus dem Übergabeparameter die entsprechende Konstante der TivaWare™ API bestimmen.
  Die möglichen Werte für current sind: 2,4,6,8,10,12. Teilen Sie current durch zwei erhalten Sie die Werte 1,2,3,4,5,6 also fortlaufende Zahlen. Dies bedeutet, dass Sie keine if-else Verzweigung oder Switch Case Anweisung brauchen , sondern mit diesen fortlaufenden Zahlen direkt den gewünschten Wert aus einem Array auslesen können.
  Legen Sie einen privaten, konstanten Array an, der die zugehörigen Konstanten (z.B. GPIO_STRENGTH_4MA) enthält. Diesen können Sie nun einfach mit current / 2 - 1 adressieren.

6.3.2.7 GPIO::setPullup

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  bool pullup: Angabe, ob der Pull-up Widerstand aktiviert oder deaktiviert werden soll.
- Funktion/Umsetzung:
  Aktiviert oder deaktiviert den Pull-up Widerstand und aktualisiert die zugehörige private Variable pinType

6.3.2.8 GPIO::setPulldown

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  bool pulldown: Angabe, ob der Pull-down Widerstand aktiviert oder deaktiviert werden soll.
- Funktion/Umsetzung:
  Aktiviert oder deaktiviert den Pull-down Widerstand und aktualisiert die zugehörige private Variable pinType

6.3.2.9 Testprogramm

- Geprüfte Methoden
  - GPIO::init
  - GPIO::read
  - GPIO::write
  - GPIO::setPullup
- Funktionsbeschreibung:
  Das Testprogramm schaltet die onboard-LED auf dem LaunchPad um, wenn der Taster SW_1 gedrückt wird. Die LED wechselt also bei einem Tasterdruck von aus auf an bzw. wenn sie eingeschaltet war, von an auf aus.
  Um Kontaktprellen zu vermeiden können Sie auf folgenden Pseudo-Code zurückgreifen. Nutzen Sie ihn bei Bedarf auch in den anderen Testprogrammen.

```
if (Taster gedrückt)
{
    warte(50ms);
    while (Taster gedrückt);

    // Eigentlicher Code

    warte(50ms);
}
```

6.3.3 Timer Klasse

6.3.3.1 Timer::init

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  System *sys: Zeiger auf die laufende Instanz der System-Klasse
  uint32_t base: Basis-Adresse des Timer-Moduls, (z.B. TIMER0_BASE)
  void (*ISR) (**void**): Zeiger auf die globale Funktion, die vom Timer aufgerufen werden soll.
  uint32_t freq: Optionale Angabe der gewünschten Timer-Frequenz in Hz. Der Standardwert ist 0Hz.
- Funktion/Umsetzung:
  Analog zur GPIO::init-Methode führt diese Methode die Initialisierung des Timer-Moduls und des Timer-Objektes durch.

6.3.3.2 Timer::start

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**

- Funktion/Umsetzung:
  Aktiviert den Timer.

6.3.3.3 Timer::stop

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Deaktiviert den Timer. Nach Aufruf dieser Methode soll jedoch keine ISR mehr abgearbeitet werden. Beim Stoppen des Timers mit dieser Methode werden die Frequenz und die Periodendauer nicht verändert, so dass der Timer mit Timer::start wieder weiterlaufen kann.

6.3.3.4 Timer::getFreq

- Rückgabewert/-typ:
  uint32_t: Aktuelle Timer-Frequenz in Hz.
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Liefert die aktuell eingestellte Frequenz des Timers in Hz zurück. Diese Methode soll stets einen korrekten Wert zurückliefern, unabhängig davon ob Timer::setFreq oder Timer::setPeriodUS oder keine von beiden Methoden vorher aufgerufen wurden.

6.3.3.5 Timer::getPeriodUS

- Rückgabewert/-typ:
  uint32_t: Aktuelle Timer-Periode in μs.
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Liefert die aktuell eingestellte Periode des Timers in Mikrosekunden zurück. Diese Methode soll stets einen korrekten Wert zurückliefern, unabhängig davon ob Timer::setFreq oder Timer::setPeriodUS oder keine von beiden Methoden vorher aufgerufen wurden.

6.3.3.6 Timer::setFreq

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t freq: Gewünschte Timer-Frequenz in Hz.
- Funktion/Umsetzung:
  Lässt den Timer Interrupts mit der gegebenen Frequenz auslösen, Ein Wert von 0 Hz stoppt den Timer und setzt die Frequenz sowie die Periodendauer auf 0. Bevor der Timer wieder mit Timer::start aktiviert werden kann, muss eine neue Frequenz oder Periodendauer eingestellt werden.

6.3.3.7 Timer::setPeriodUS

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t period: Gewünschte Timer-Periode in μs.
- Funktion/Umsetzung:
  Lässt den Timer Interrupts mit der gewünschten Periode auslösen. Ein Wert on 0 μs stoppt den Timer und setzt die Frequenz sowie die Periodendauer auf 0. Bevor der Timer wieder mit Timer::start aktiviert werden kann, muss eine neue Frequenz oder Periodendauer eingestellt werden.

6.3.3.8 Timer::clearInterruptFlag

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Löscht das ,,Timeout Interrupt Flag" dieses Timers.

6.3.4 ADC Klasse

6.3.4.1 ADC::init

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  System *sys: Zeiger auf die laufende Instanz der System-Klasse
  uint32_t base: Basis-Adresse des ADC-Moduls (z.B. ADC0_BASE)
  uint32_t sampleSeq: Zu verwendender Sample Sequencer innerhalb des ADC Moduls (0,1,2 oder 3)
  uint32_t analogInput: Analoger Eingang, der gelesen werden soll (z.B. ADC_CTL_CH0)
- Funktion/Umsetzung:
  Analog zur GPIO::init-Methode führt diese Methode die Initialisierung des ADC-Moduls und des ADC-Objektes durch.

6.3.4.2 ADC::setHWAveraging

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t averaging: Anzahl an Messwerten, die zu einem Wert gemittelt werden. Muss eine Zweierpotenz bis einschließlich 64 sein oder 0 falls die Mittelung deaktiviert werden soll.
- Funktion/Umsetzung:
  Aktiviert oder deaktiviert die Hardware-Mittelung (auch Oversampling genannt).

6.4.4.3 ADC::read

- Rückgabewert/-typ: **void**
  uint32_t: Wert von 0 bis 4095 entsprechend einer Spannung von 0V bis 3,3V

- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Liefert für den zugehörigen Pin einen Wert zwischen 0 und 4095 zurück, der einer Spannung von 0V bis 3,3V entspricht. Die Methode startet die AD-Konvertierung und wartet das Ergebnis ab.

6.4.4.4 ADC::readVolt

- Rückgabewert/-typ:
  float: Spannung in V
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Liefert für den zugehörigen Pin die anliegende Spannung in Volt zurück.

6.3.5 System Klasse

Die System-Klasse ist die zentrale Klasse und stellt von allen Klassen gemeinsam genutzte bzw. erforderliche Funktionen zur Verfügung. Dazu gehört u.a. das Setzen der CPU-Taktfrequenz, Delay-Methoden und eine Debugging Schnittstelle.
In jedem Programm läuft nur eine einzige Instanz der System Klasse. Jede Klasse, die auf die System-Klasse zugreifen soll, erhält als erstes Argument einen Zeiger auf die laufende Instanz der System-Klasse. Dieses System-Objekt muss vor allen anderen initialisiert werden.

6.3.5.1 System::init

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t clk: Gewünschte CPU Taktfrequenz in Hz. Kann 40 MHz, 50 MHz oder 80 MHz sein.
- Funktion/Umsetzung:
  Initialisiert den Mikrocontroller indem die Taktfrequenz der CPU auf die gewünsche Frequenz konfiguriert wird. Zusätzlich werden der PWM Unit Clock Divisor gesetzt, alle gesperrten GPIO-Pins entsperrt, die Debugging Schnittstelle konfiguriert und die Interrups freigeschaltet.

6.3.5.2 System::error

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  ErrorCodes errorCode: Optionaler Fehlerparameter (siehe ErrorCodes.h) angibt.
  void *faultOrigin: Bis zu 3 optionale Zeiger auf die Fehlerquelle(n).
- Funktion/Umsetzung:
  Diese Methode kann genutzt werden, um Fehler aller Art innerhalb der Klassen abzufangen. Dadurch kann zum einen unvorhersehbares Verhalten vermieden werden, wie es z.B. bei

Overflows vorkommen kann und zum anderen die FaultISR der TivaWare™ API umgangen werden, die deutlich schwerer zu debuggen ist.
Die Methode deaktiviert alle Module des Mikrocontrollers, um sicher zu stellen, dass der MagniSilver™ abschaltet. Landen Sie nämlich in der FaultISR, werden Sie merken, dass die Motoren einfach weiterdrehen.
Sie können weitere Funktionen hinzufügen (z.B. Blinken der on-board LED). Beachten Sie nur, dass Sie wegen eines Fehlers hier gelandet sind, und eventuell nicht merh alles zu 100% funktioniert. Halten Sie daher den Code innerhalb der Methode so knapp wie möglich.

Siehe auch Verwenden der System::error-Methode

6.3.5.3 System::getClockFreq

- Rückgabewert/-typ:
  uint32_t: Aktuelle CPU Taktfrequenz in Hz.
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Liefert die aktuelle CPU Taktfrequenz in Hz zurück.

6.3.5.4 System::enableFPU

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t: Wert für den PWM Unit Clock Divisors. Eine Zweierpotenz von 1 bis 64.
- Funktion/Umsetzung:
  Der PWM Unit Clock Divisor ermöglicht das Einstellen einer PWM-Taktfrequenz unterhalb der CPU Taktfrequenz (siehe ,,TivaC Mikrocontroller Datenblatt" Seite 222 und Seite 1234). Der entsprechende Teiler kann mit dieser Funktion gesetzt werden.

6.3.5.5 System::setPWMClockDiv

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t: Wert für den PWM Unit Clock Divisior. Eine Zweierpotenz von 1 bis 64
- Funktion/Umsetzung:
  Der PWM Unit Clock Divisor ermöglicht das Einstellen einer PWM-Taktfrequenz unterhalb der CPU Taktfrequenz (siehe ,,TivaC Mikrocontroller Datenblatt" Seite 222 und Seite 1234). Der entsprechende Teiler kann mit dieser Funktion gesetzt werden.

6.3.5.6 System::getPWMClockDiv

- Rückgabewert/-typ:
  uint32_t: Wert des PWM Unit Clock Divisors. Eine Zweierpotenz von 1 bis 64.
- Übergabeparameter: **void**

- Funktion/Umsetzung:
  Der PWM Unit Clock Divisor ermöglicht das Einstellen einer PWM-Taktfrequenz unterhalb der CPU Taktfrequenz (siehe ,,TivaC Mikrocontroller Datenblatt“ Seite 222 und Seite 1234). Der entsprechende Teiler kann mit dieser Funktion ausgelesen werden.

6.3.5.7 System::delayCycles

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t cycles: Gewünschte Verzögerung in Taktzyklen
- Funktion/Umsetzung:
  Pausiert die CPU für mindestens die übergebene Anzahl an CPU-Taktzyklen. Da sie auf der TivaWare™ Funktion SysCtlDelay basiert, sind nur Mehrfache von 3 Taktzyklen möglich. Andere Werte werden aufgerundet.

6.3.5.8 System::delayUS

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint32_t us: Gewünschte Verzögerung in μs
- Funktion/Umsetzung:
  Pausiert die CPU für mindestens die übergebene Anzahl an Mikrosekunden. Da sie auf der TivaWare™ Funktion SysCtlDelay basiert, sind nur Mehrfache von 3 Taktzyklen möglich. Andere Werte werden aufgerundet.

6.3.5.9 System::setDebugging

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  bool debug: Gibt an, ob Debugging aktiviert oder deaktiviert werden soll.
- Funktion/Umsetzung: Mit dieser Funktion kann das Senden von Debugging Daten aktiviert oder unterbunden werden. Beim Start ist Debugging aktiviert.

6.3.5.10 System::setDebugVal

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  const char* name: Name der Variablen, der in der Legende des Arduino Plotters angezeigt werden soll. Darf kein Leerzeichen, Komma, Tab ('\t') oder Zeilenumbruch ('\n') enthalten.

  int32_t value: Wert als Integer, der im Arduino Plotter angezeigt werden soll. Bei Fließkommavariablen empfiehlt es sich unter Umständen sie mit einem Faktor 100 zu multiplizieren, um die gewünschte Auflösung zu erreichen.

- Funktion/Umsetzung:
  Ermöglicht das Speichern von Werten zum Übermitteln mit System::sendDebugVals. Beachten Sie bitte, dass momentan nur acht Werte parallel übermittelt werden können, da der Arduino Plotter nur acht verschiedene Farben zur Verfügung stellt. Um mehr Variablen zu übermitteln, erhöhen sie einfach die Variable System::maxDebugVals. Eine detailliertere Erklärung inkl. eines Verwendungsbeispiels sind im Quellcode der Methode gegeben.

6.3.5.11 System::sendDebugVals

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Übermittelt bis zu acht gespeicherte Werte mithilfe der seriellen Schnittstelle über USB an einen Computer. Die Werte können mit System::setDebugVals gesetzt werden. Die Daten sind Tab-separiert und können z.B. mit dem Serial Plotter der quelloffenen Arduino IDE dargestellt werden. Diese Methode sollte am besten periodisch aufgerufen werden, z.B. durch einen 10Hz Timer Interrupt. In dem gegebenen Quellcode des Magni-Programms ist dies bereits der Fall.

6.3.6 Display Klasse

Die Display Klasse steuert ein 4x20 Segment Textdisplay. **6.3.6.1 Display::init**

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  System *sys: Zeiger auf die laufende Instanz der System-Klasse
- Funktion/Umsetzung:
  Analog zur GPIO::init führt diese Methode die Initialisierung des Display-Objektes durch.

6.3.6.2 Display::sendByte

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint8_t data: Datenbyte, dass an das Display gesendet werden soll.
  bool: Zeigt an, ob es sich bei dem Datenbyte um Daten zum Anzeigen handelt (**true**) oder ob es sich um eine Konfiguration (**false**) handelt.
- Funktion/Umsetzung:
  Sendet Daten/Konfiguration an das Display.

6.3.6.3 Display::enable

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Lässt das Display die anliegenden Daten einlesen.

6.3.6.4 Display::clearDisplay

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Löscht alle angezeigten Daten des Displays.

6.3.6.5 Display::resetCursor

- Rückgabewert/-typ: **void**
- Übergabeparameter: **void**
- Funktion/Umsetzung:
  Setzt den Cursor auf die Ursprungsposition zurück.

6.3.6.6 Display::cursorPosition

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint8_t column: Gibt die Spalte an (Wert zwischen 0 und 19) in die sich der Cursor bewegen soll.
  uint8_t row: Gibt die Reihe an (Wert zwischen 0 und 3) in die sich der Cursor bewegen soll.
- Funktion/Umsetzung:
  Setzt den Cursor auf die übergebene Position.

6.3.6.7 Display::printText

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  char text: Das anzuzeigende Symbol
- Funktion/Umsetzung:
  Sendet das anzuzeigende Symbol an das Display.

6.3.6.8 Display::printText

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  const char *text: Zeiger auf das anzuzeigende Char Array
- Funktion/Umsetzung:
  Überladung der Funktion 6.3.6.7, damit ein kompletter String an Symbolen an das Display gesendet werden kann.

6.3.6.9 Display::printNumber

- Rückgabewert/-typ: **void**

- Übergabeparameter:
  uint16_t number: Die anzuzeigende Ganzzahl
- Funktion/Umsetzung:
  Sendet die übergebene Zahl an das Display.

6.3.6.10 Display::printFloat

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  float number: Die anzuzeigende Fließkommazahl
- Funktion/Umsetzung:
  Sendet die übergebene Zahl an das Display.

6.3.6.11 Display::clearSegment

- Rückgabewert/-typ: **void**
- Übergabeparameter:
  uint8_t length: Angabe wie viele Segmente inkl. des aktuellen Segments gelöscht werden sollen.
- Funktion/Umsetzung:
  Löscht die angegebene Anzahl an Segmenten von der aktuellen Position aus.

6.4 Testaufbau

Bitte beachten Sie: Der hier gezeigte Testaufbau, dient dem Test des Gesamtprogramms, also der Funktionsfähigkeit in Magni_Class_Test und Remote_Class_Test im Zusammenspiel. Um die Low-Level Klassen, und die beiden genannten Klassen im Einzelnen, zu Testen, müssen Sie sich eigene Testaufbauten überlegen. Denken Sie dabei möglichst einfach! Schließen Sie beispielsweise nach Möglichkeit direkt am Board an!
Für den Aufbau eines eigenen MiniMagni™ (in ihrer Gruppe) haben Sie die Bauteile bereits erhalten. Beachten Sie bitte, dass Sie pro Person nur einen Motor und ein Tiva C Series TM4C123G LaunchPad ausgeteilt bekommen. Sollten ihnen Bauteile fehlen finden Sie weitere Informationen auf den Folien der Einführungsveranstaltungen. Tabelle 6.4 listet die wesentlichen Bauteile auf. Zusätzlich zu diesen Bauteilen benötigen Sie zudem passende Kabel.

Anzahl	Bauteil
2x	Tiva C Series TM4C123G LaunchPad
2x	Gleichstrommotor (3V, max. 1,8W)
1x	Vierfach H-Brücke SN754410
1x	Ultraschallsensor HC-SR04
1x	Joystick
1x	Steckbrett (Breadboard)
1x	Potentiometer (100 kΩ)
2x	Elektrolytkondensator (100 μF)
2x	Keramikkondensator (100 nF)
2x	Widerstand (10 kΩ)
2x	Widerstand (330 Ω)
2x	LED Rot

Table 1: Bauteile für den Aufbau des MiniMagni™

Einen möglichen Aufbau des MiniMagni™ finden Sie in Abbildung 30.
Wie Sie darauf erkennen können, wurde an dem Launchpad ein anderer Joystick, als der ausgeteilte Joystick angeschlossen. In Zukunft werden wir versuchen hier eine neue Abbildung einzufügen. Für den Moment ist im folgenden erklärt, wie der neue Joystick angeschlossen werden kann:
Abbildung 31 zeigt eine alternative Möglichkeit den Joystick zu verbinden. Dafür schließen Sie sechs Verbindungskabel direkt an die Potentiometer des Joysticks an. Diese Verbindungskabel müssen Sie über das Breadboard mit dem Launchpad verbinden. Dabei müssen Sie das blaue und pinke Kabel (die beiden Mittleren) mit Ground. Dafür stecken Sie diese auf die dritte Reihe von rechts auf das Breadboard in Abbildung 30) verbinden. Als nächstes werden beiden äußeren Kabel (lila und weiß) mit der Spannungsquelle verbunden. Dafür stecken Sie diese in die unterste Reihe auf dem Breadboard. Als letztes werden die Potentiometer mit ADC-Eingängen auf dem Launchpad verbunden. Ziehen Sie dafür ein weiteres Kabel von PE3 auf dem Launchpad zur zweiten Reihe von Rechts auf dem Breadboard. Stecken Sie außerdem das oragene Kabel in dieselbe Reihe. Verbinden Sie als nächstes PE0 auf dem Launchpad mit der Reihe ganz rechts auf dem Breadboard. Stecken Sie das letzte Kabel (oliv) in diese Reihe. Ihr Joystick ist nun Verbunden und nutzbar.

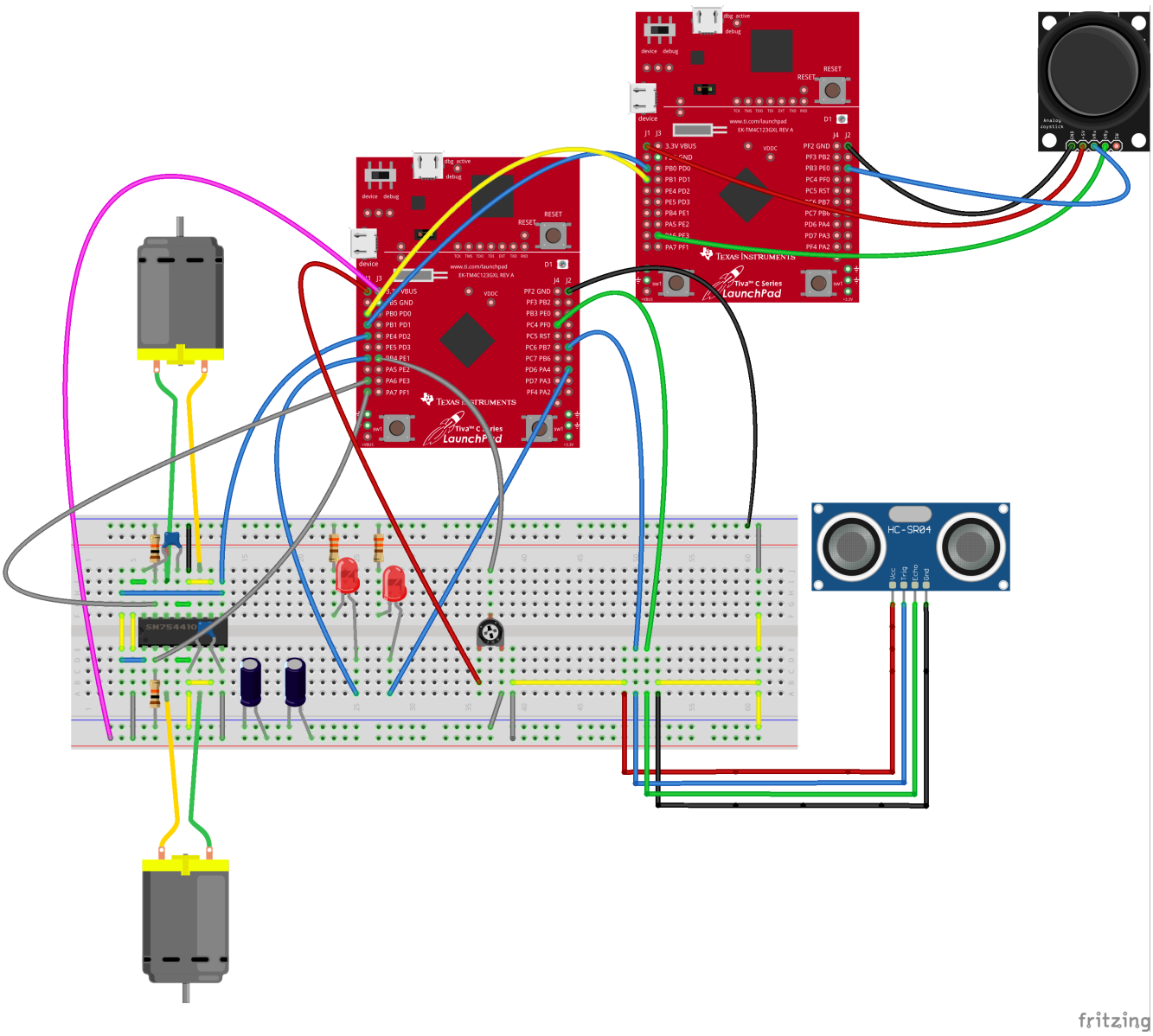

Figure 30: Schematischer Testaufbau auf dem Steckbrett

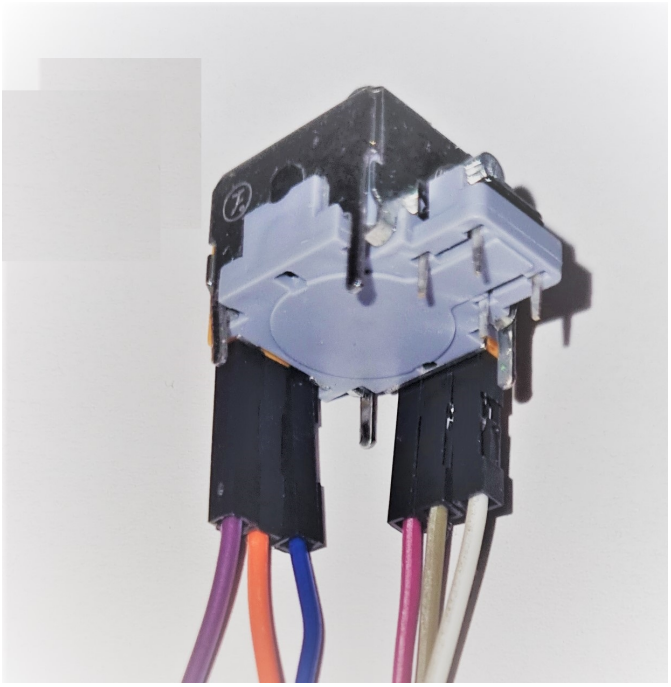

Figure 31: Anschlussbeispiel Joystick

6.5 Weitere Dokumente, Datenblätter, C++ Unterlagen

6.5.1 Unterlagen im Dokumente und Infos Repository

Alle aktuellen Unterlagen finden Sie im Dokumente und Infos Repository, das Sie mit GitHub Desktop geklont haben sollten. Zur Bearbeitung unumgänglich sind das Datenblatt des Mikrocontrollers ,,TivaC Mikrocontroller Datenblatt" und die "TivaWare™ Treiberbibliothek", da diese wichtige Informationen zum Aufbau des Mikrocontrollers und der API enthalten. Diese umfassen zwar deutlich mehr Themen als für das IT1 Praktikum nötig, bieten aber alle benötigten Informationen. Sie werden später auch anhand von Datenblättern arbeiten müssen, also versuchen Sie sich nicht von den mehreren tausend Seiten erschlagen zu lassen und suchen Sie systematisch nach den für Sie relevanten Kapiteln. Wir haben versucht das Essentielle in den Grundlagenklassen auszuarbeiten, damit Sie nicht in das sprichwörtliche kalte Wasser geschmissen werden. Um den Einstieg in die Datenblätter zu erleichtern; finden Sie nachfolgend eine kurze Beschreibung der wichtigsten Dokumente:

- Das Dokument ,,TivaWare™ Treiberbibliothek" enthält die vollständige Dokumentation der TivaWare API. Anders gesagt: Hier werden Sie zu jeder Funktion, die die API zur Verfügung stellt, eine Beschreibung finden. Zudem finden Sie hier kurze Programmierbeispiele zu den jeweiligen Themen. Da Sie den Mikrocontroller mithilfe der Treiberbibliothek programmieren, ist dieses Datenblatt für die Bearbeitung des Workshops essenziell. Falls dieses Dokument eine Frage nicht genau genug beantwortet oder Sie generell mehr über den Aufbau des Mikrocontrollers wissen wollen bzw. müssen, lohnt sich ein Blick in das ,,TivaC Mikrocontroller Datenblatt".

- Das ,,TivaC Mikrocontroller Datenblatt" bietet detaillierte Informationen über den Aufbau des Mirkocontrollers und seine Funktionen. Theoretisch findet man hier alles was man wissen muss, um mit dem Tiva LaunchPad zu arbeiten. Jedoch sind alle Informationen sehr hardwarenah und komplex. Da Sie das Tiva LaunchPad mithilfe der Treiberbibliothek programmieren, ist es ratsam erst in der Treiberbibliothek nach Problemlösungen zu suchen.

- Der ,,TivaC LaunchPad Workshop" ist optional. Wie der Name schon sagt, ist das kein Datenblatt, sondern ein Workshop. Die Informationen hier sind viel einfacher und oberflächlicher gehalten als in den beiden oben beschriebenen Datenblättern. Außerdem werden die Themen zu zahlreichen Beispielen erläutert. Dadurch eignet sich dieses Dokument gut, um einen guten Überblick über das Tiva LaunchPad und dessen Programmierung zu erhalten. Wichtig: Es liefert keine ausreichenden Informationen, um das Praktikum vollständig bearbeiten zu können, sondern dient lediglich als Einstiegshilfe.

- ,,DC Motoren per PWM steuern Heise Make" ist ein Auszug aus dem Make Magazin, der mit den Grundlagen von Gleichstrommotoren beginnt und bei der integrierten H-Brücke, wie wir sie verwenden, aufhört. Falls Ihnen H-Brücken nicht vertraut sind, sollten Sie sich insbesondere die Seiten 7 folgend anschauen. Hier wird das Prinzip der H-Brücke genau erklärt.

6.5.2 Weitere hilfreiche Quellen

6.5.2.1 Tiva Mikrocontroller

Die Website Luis Electronic Projects bietet einige gute Tutorials zum Tiva Mikrocontroller und Tivaware: https://sites.google.com/site/luiselectronicprojects/tutorials/tiva-tutorials

6.5.2.2 C++

Falls Sie noch keine oder wenig Programmiererfahrung (in C++) haben, so finden Sie nachfolgend die wichtigsten Konzepte, mit denen Sie im Praktikum umgehen können müssen. Sie werden Ihnen auch in der IT1 Übung beigebracht.

- Pointer
- Mehrdimensionale Arrays
- Klassen
- Präprozessordirektiven (Header Guards, **#include**, **#define**)
- enum
- if/else, switch
- typecast

Falls Sie neben der Übung noch weitere Beispiele zum Erlernen von C++ benötigen, finden Sie diese auf der englischsprachigen Website http://www.learncpp.com inkl. ausführlichen Erklärungen. Die folgenden Kapitel sind für das Praktikum relevant:

- 1: C++ Basics
- 2: C++ Basics: Functions and Files
- 3: Debugging C++ Programs
- 4: Fundamental Data Types
- 5: Operators
- 7: Control Flow and Error Handling
- 9: Compound Types: References and Pointers
- 11: Arrays, Strings and Dynamic Allocation
- 13: Basic Object-oriented Programming

Es steht Ihnen natürlich frei, andere Unterlagen (Bücher, Internetseiten) zum Lernen der Programmiersprache zu verwenden.

References

[1] "Universal Asynchronous Receiver Transmitter," zuletzt geöffnet 27.02.2023. [Online]. Available: https://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter

[2] G. Anjitha, "Obstacle avoidance robot using fpga," 05 2020.