Karlsruher Institut für Technologie
Institut für Technik der Informationsverarbeitung

Handbuch zum Informationstechnik I Praktikum

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

Inhaltsverzeichnis

1 Vorwort

Herzlich Willkommen in Ihrem Team zur Entwicklung einer neuen Software zur Steuerung eines MiniSeg™. Wie man vielleicht am Namen des Produktes erkennen kann, handelt es sich beim MiniSeg™ um einen vereinfachten Aufbau der bekannten Segway™ Transportmittel.

Ihre Aufgabe in den nächsten Wochen wird die Planung des Vorgehens, sowie die Umsetzung und das Testen von Software für das MiniSeg™ sein. Da es sich beim MiniSeg™ 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 MiniSeg™, sind bereits vollständig implementiert. Die verwendeten Hardwaremodule (Motoren, Potentiometer, ...) müssen durch Sie angesteuert werden.

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

2 Inhalte 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 Vorbereitung, 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 im gleichnamigen Kapitel folgt. Zum anschließenden Testen und Debuggen Ihres Programmcodes finden Sie hilfreiche Tipps im darauffolgenden Kapitel. Abschließend folgt mit Unterlagen die Sammlung aller Ablaufdiagramme, Klassenbeschreibungen, Schaltpläne, etc., die Sie unterwegs immer wieder benötigen werden. Am Ende des Handbuchs finden Sie auch einen Abschnitt, welcher auf weitere Dokumente verweist.

Dieses Handbuch ist lang; 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 schwierigen Einstieg in den Workshop zu erleichtern. Das Praktikum nutzt eine Reihe von sehr praktischen Tools, mit denen Sie bisher noch nicht arbeiten mussten. Wie Sie diese richtig einrichten und effektiv damit arbeiten können, wird Ihnen in den Einführungsveranstaltungen vorgeführt. Daher gilt unsere dringende Empfehlung, die Einführungsveranstaltungen mit Ihrem Notebook zu besuchen. Die Sprechstunden sind kein Ersatz und keine Alternative!
- Selbständige Bearbeitung aller Aufgaben
  Es gibt keine Pflichttermine, an denen Sie die Aufgaben bewältigen müssen. Sorgen Sie für eine faire Aufteilung innerhalb Ihrer Gruppe und beachten Sie die Fristen.
- Sprechstunden
  Wenn Sie nicht weiterkommen, und die Antwort auch nach gründlichem Lesen dieses Handbuchs, der entsprechenden Stellen in den „TivaC Mikrocontroller“ und dem „TivaWare™ Treiberbibliothek“ Datenblättern sowie der Issues auf GitLab nicht finden können, kommen Sie gerne in die Sprechstunde. Sobald Sie eine Teilaufgabe fertig haben, sollten Sie ebenfalls vorbeikommen, damit Sie Probleme oder Fehler *vor* der mündlichen Prüfung beseitigen können. Um den Andrang zu Beginn der Sprechstunden in Grenzen zu halten und Ihnen lange Wartezeiten zu ersparen werden die wöchentlichen Termine unterteilt. Es gibt kurze Fragenslots (15 Minuten) und lange Fragenslots (30 Minuten). Falls Sie zur Sprechstunde kommen möchten, müssen Sie sich vorher zu Ihrem Wunschslot anmelden, die Anmeldungslinks zum WIWI Portal finden Sie auf GitLab und in den Folien der ersten Einführungsveranstaltung. Wir möchten jeder Gruppe die Chance geben, Ihre Fragen zu stellen. Aus diesem Grund werden wir uns (im Notfall!) nicht davor drücken, einer Gruppe weiterzuhelfen, die sich nicht anmelden konnte. Seien Sie jedoch fair gegenüber Ihren Kommilitonen und den Tutoren und erscheinen Sie nur zu dem Terminslot, für den Sie sich auch angemeldet haben! Die Teilnehmerzahl der Slots ist bedingt durch die Anzahl der Tutoren begrenzt.

- Mündliche Prüfung nach der Abgabe
  Nach Ende des Bearbeitungszeitraumes werden alle GitLab Projekte gesperrt. In der folgenden Abgabewoche werden Sie als Gruppe mündlich von zwei Tutoren zu den Praktikumsthemen befragt. Zusätzlich gehen sie mit Ihnen Ihren Quellcode durch und überprüfen, ob dieser den Programmierrichtlinien entspricht. Beachten Sie, dass es keine Korrekturmöglichkeit gibt.
- Probefahrt
  Das Praktikum schließt mit einer freiwilligen Probefahrt ab. Nachdem Sie wochenlang Ihr MiniSeg™ entwickelt haben, können Sie die Ergebnisse auf einem großen TivSeg™ (Nachbau des Segway™ vom ITIV) ausgiebig erproben.

2.3 Aktuelle Infos, 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.

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 Studentengruppen.

Dokumente und Infos stellt Ihnen nicht nur alle Materialien zur Verfügung, die Sie für dieses Praktikum benötigen, sondern bietet durch das zugehörige Issue-System auch die Möglichkeit Feedback zu geben, Fragen zu stellen, oder auf Fehler aufmerksam zu machen. Wichtige Ankündigungen erhalten Sie ebenfalls dort.

Playground dient dazu, Merges (mehr dazu unter Arbeiten mit GitHub Desktop) kennenzulernen und wird Ihnen in der ersten Einführungsveranstaltung gezeigt.

3.1.3 Wie kann ich GitLab nutzen?

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.

3.1.3.1 Kontoerstellung

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.

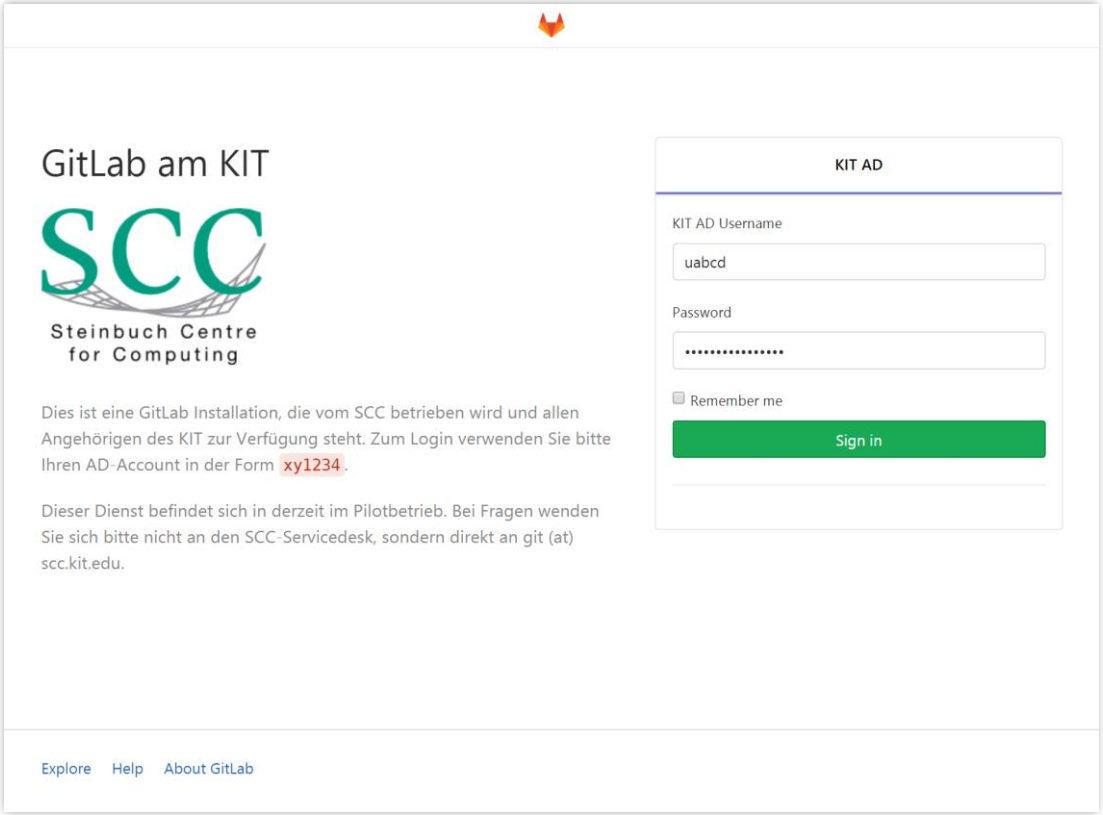

Abbildung 1: GitLab Anmeldeseite

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

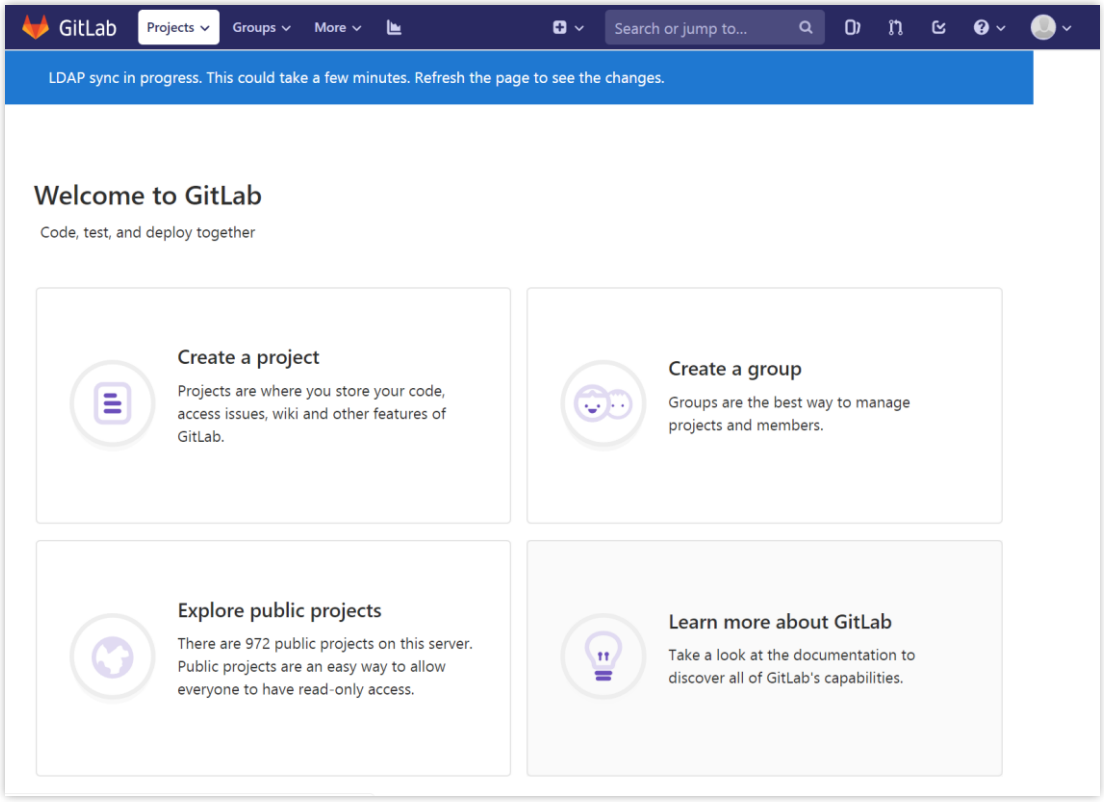

Abbildung 2: GitLab nach Anmeldung

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

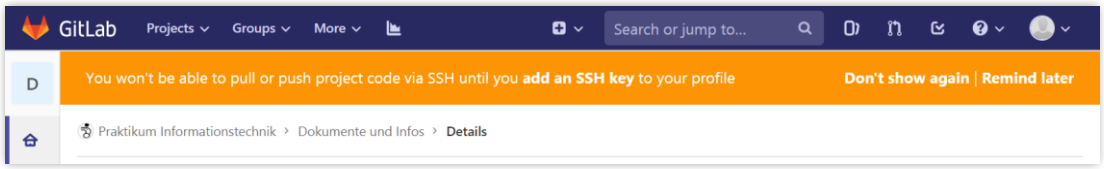

Abbildung 3: SSH Key Warnung GitLab

3.1.3.2 Einstellungen

Über das Auswahlmenü oben rechts kommen Sie in die Einstellungen.

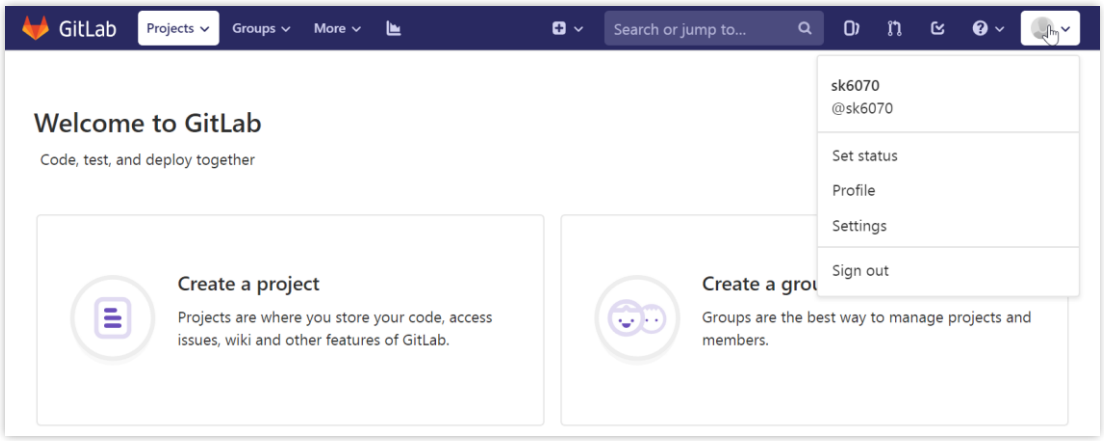

Abbildung 4: Zugang zu den Einstellungen in GitLab

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.

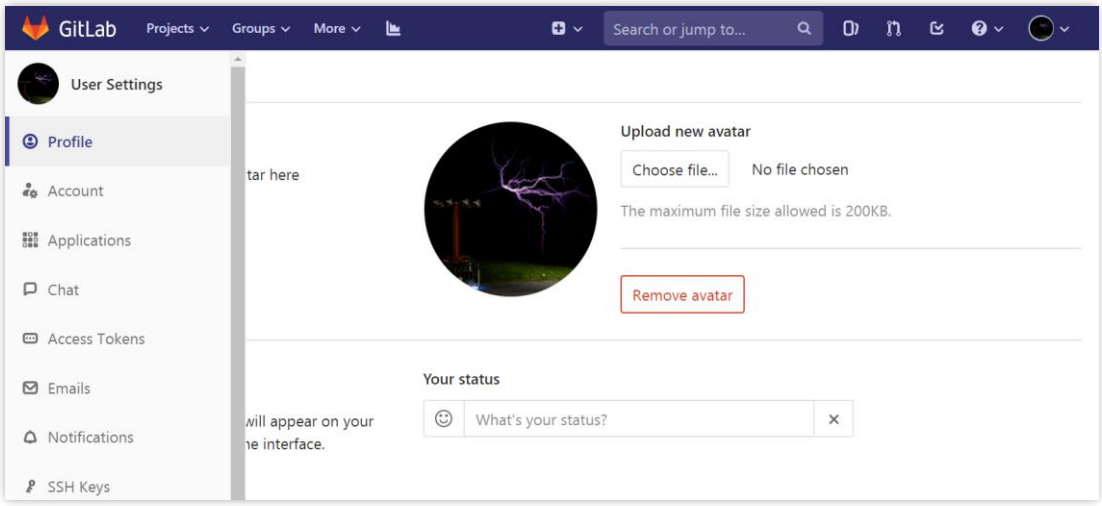

Abbildung 5: GitLab Einstellungen

3.1.3.3 Benachrichtigungen aktivieren

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

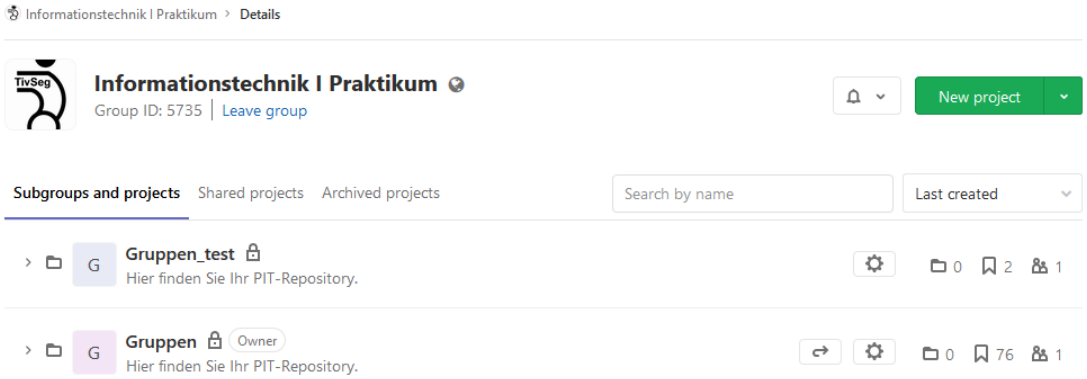

Abbildung 6: GitLab Benachrichtigungen aktivieren

3.1.3.4 Ihr Projekt

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.

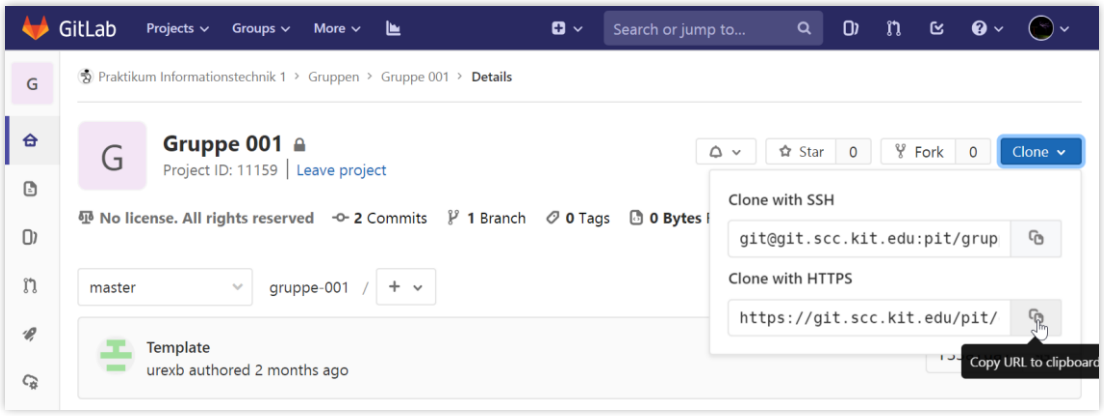

Abbildung 7: HTTPS Link für Clone Vorgang in GitLab

3.1.3.5 Installation von 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.

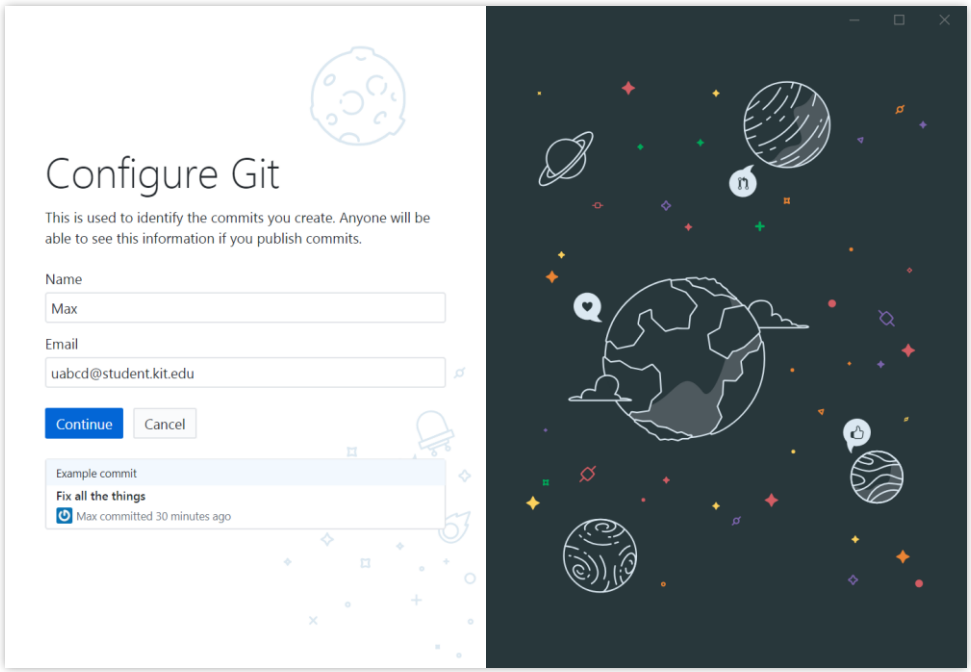

Abbildung 8: GitHub Desktop Konfiguration

Abschließend können Sie noch entscheiden, ob Sie Nutzungsstatistiken an GitHub übermitteln wollen oder nicht.

3.1.3.6 Einrichten von GitHub Desktop

Nach der Installation gelangen Sie auf die Startseite von GitHub Desktop. Dort wählen Sie „Clone a repository from the Internet…“.

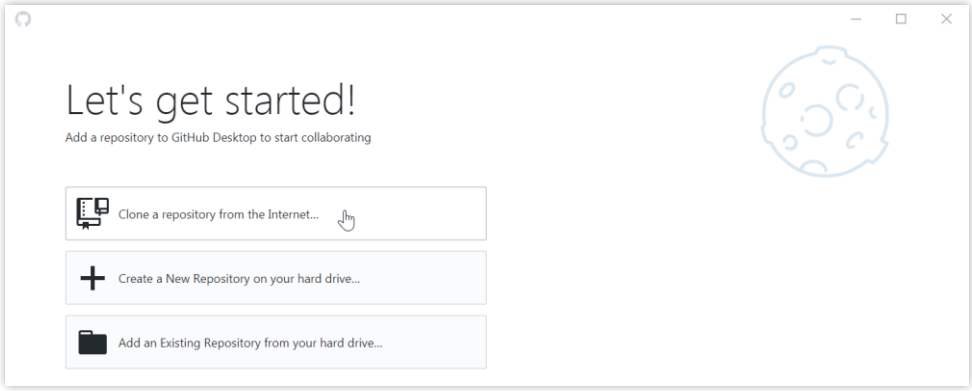

Abbildung 9: Clone eines Repository in GitHub Desktop

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.

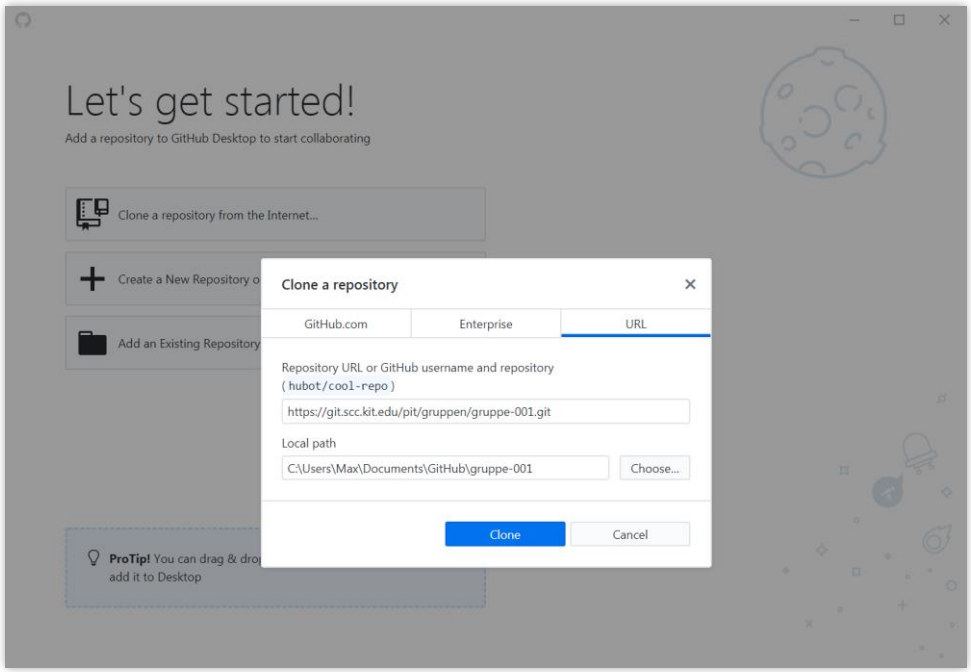

Abbildung 10: URL für Clone Prozess in GitHub Desktop eingeben

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

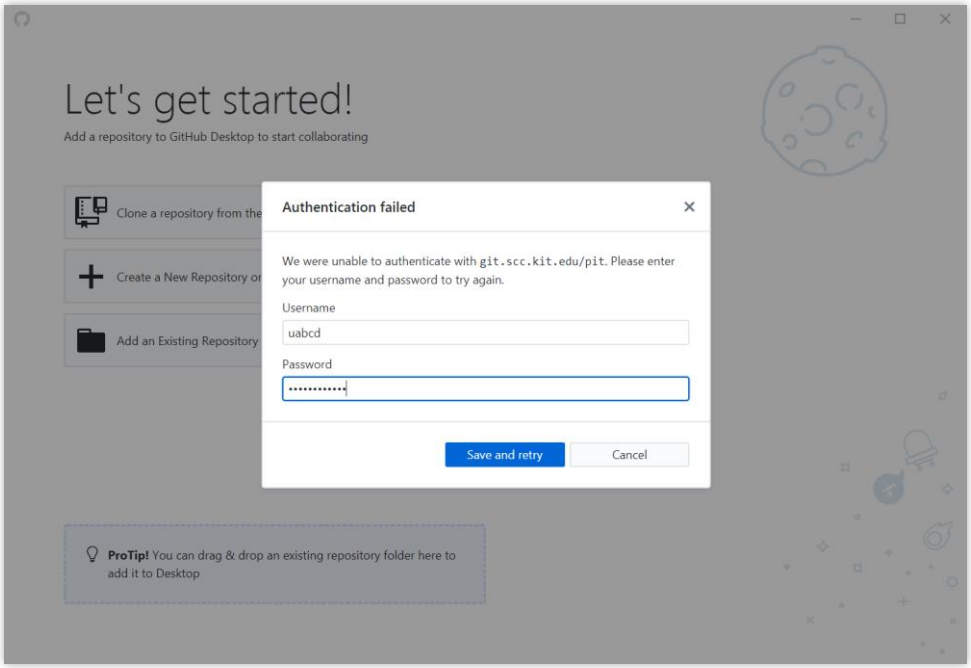

Abbildung 11: Eingabe GitLab Anmeldedaten in GitHub Desktop

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.

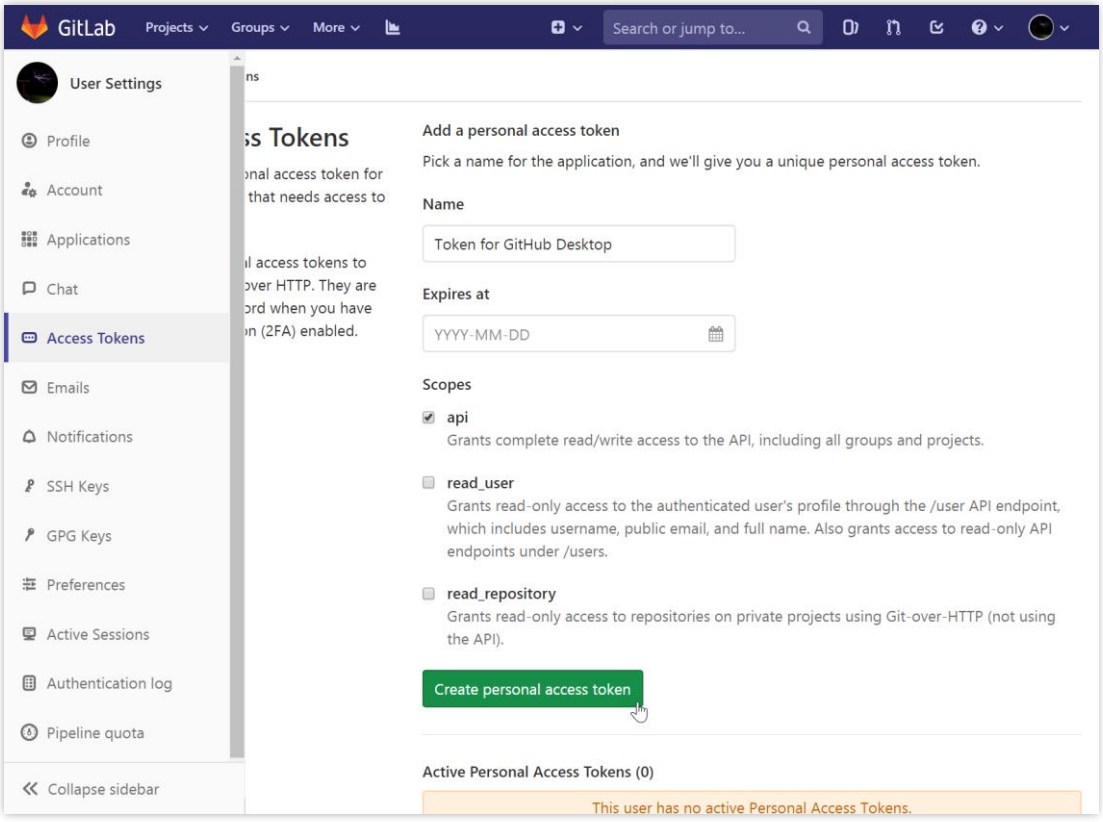

Abbildung 12: (Optional) Erstellen eines Access Tokens in GitLab

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

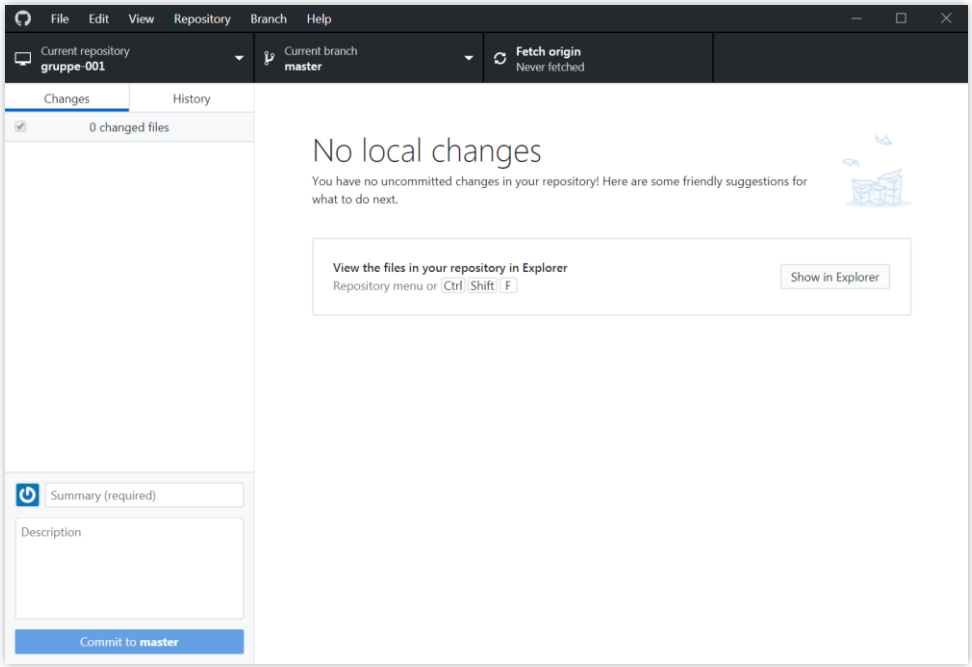

Abbildung 13: Übersichtsseite GitHub Desktop

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

3.1.3.7 Arbeiten mit GitHub Desktop

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 …“).

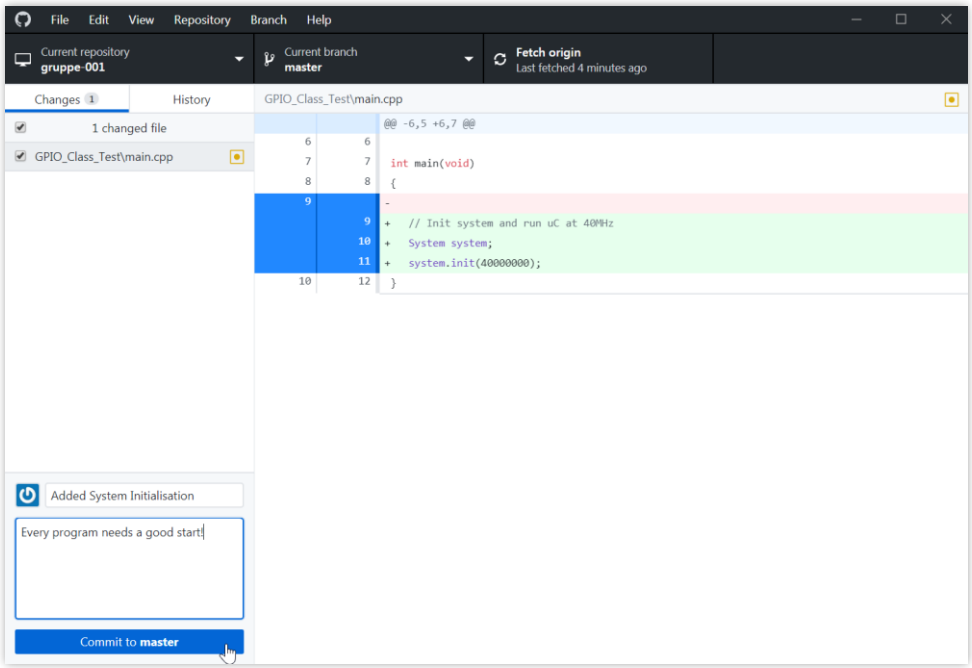

Abbildung 14: Erstellen eines Commits in GitHub Desktop

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.

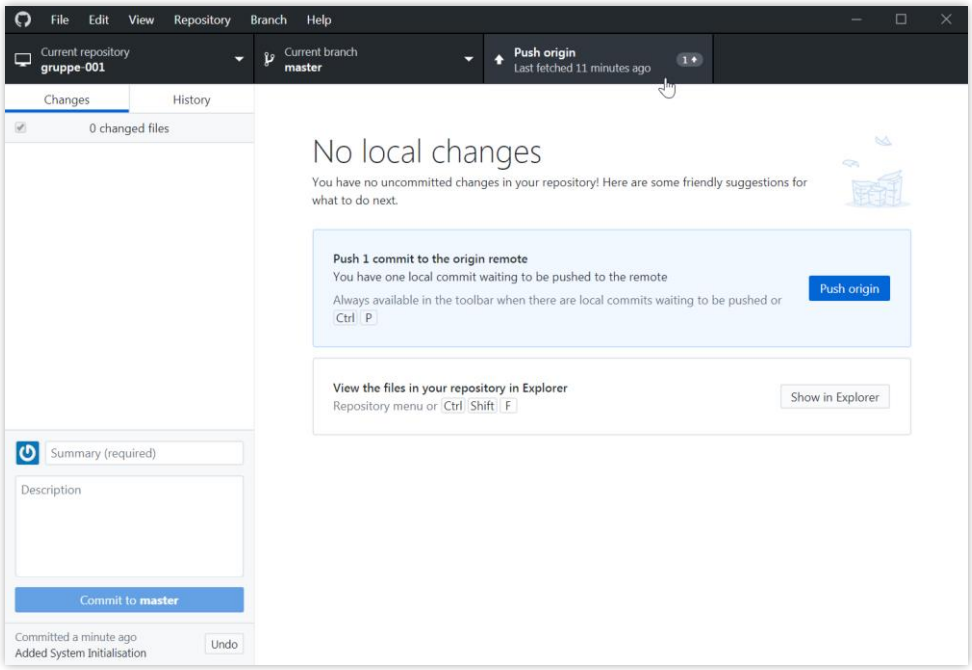

Abbildung 15: Pushen des Commits in GitHub Desktop

3.1.4 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).

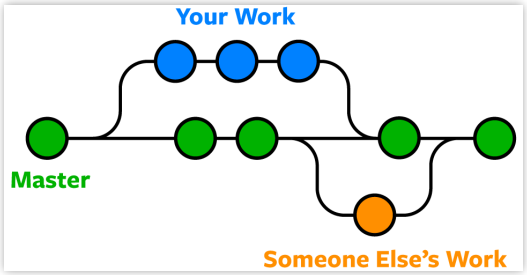

Abbildung 16: Prinzip der Branches in Git

3.1.5 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 werden Sie CCS gemeinsam einrichten und auch kennenlernen. 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).

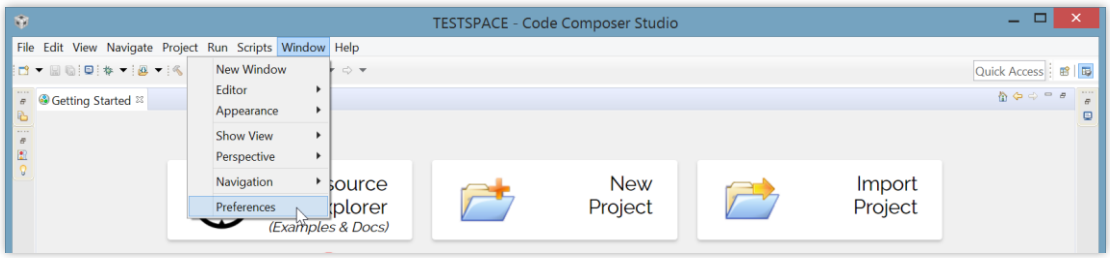

Abbildung 17: Öffnen der Code Composer Studio „Preferences“

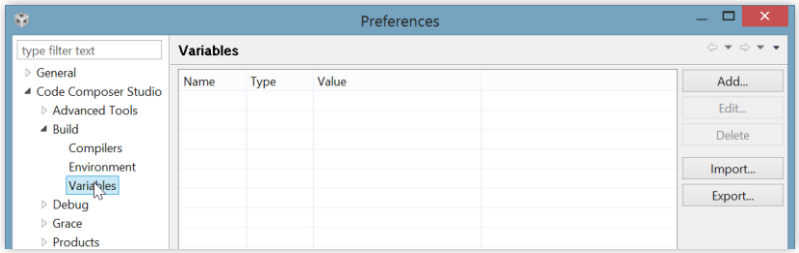

Abbildung 18: Hinzufügen einer Build Variable

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.

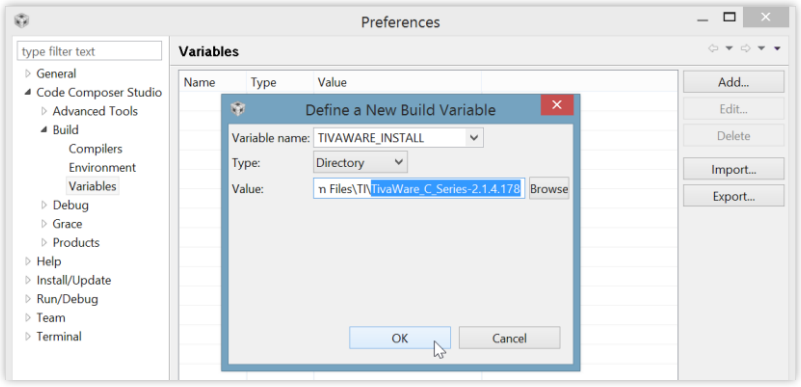

Abbildung 19: Definieren der TivaWare™ Build Variable

3.2.3 Importieren eines CCS Projektes

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...“.

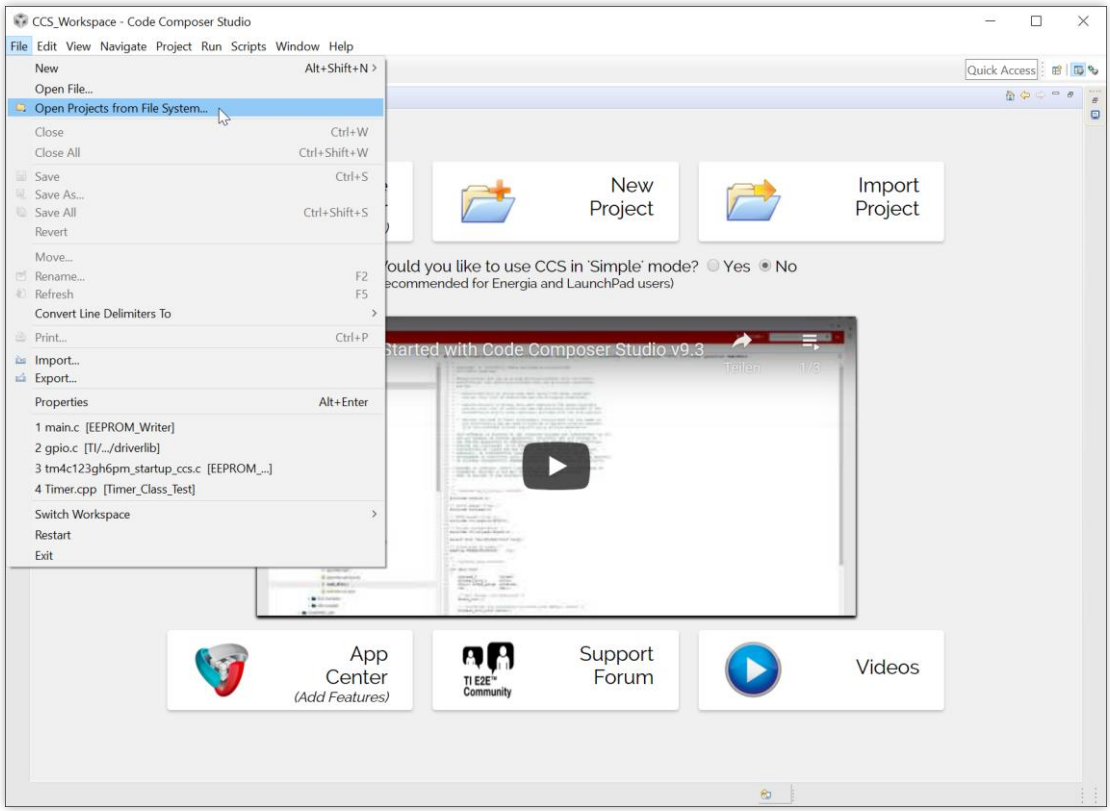

Abbildung 20: Importvorgang in CCS

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 gefundene Ordner und Projekte aufgelistet. Wählen sie den Gruppenordner ab (erster Eintrag), und bestätigen Sie dem Import der anderen Projekte mit „Finish“ (Abbildung 21). Es befinden sich nun alle Projekte im Project Explorer (linke Seite in CCS).

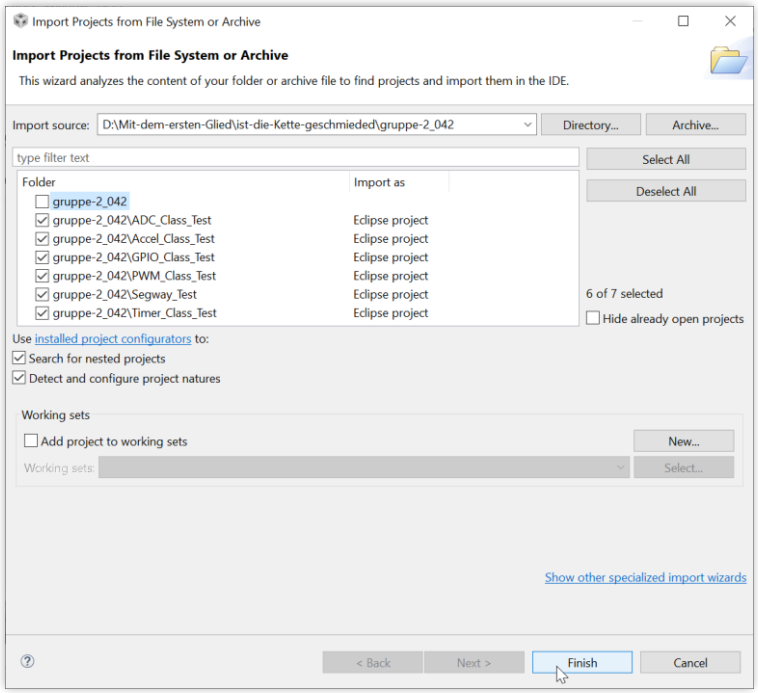

Abbildung 21: Auswahl der CCS Projekte

3.2.4 Debugging

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

3.2.4.1 Debugging in CCS

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()`. Falls 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 wirklich mit Step Over sehen wollen. Den Rest dieser Methode können Sie danach mit Step Return automatisch ausführen lassen. Step Into und Step Return hintereinander ausführen 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 das Testprogramm im Debugging-Modus und beobachten Sie, wie Sie mit den Bedienelementen den Code zeilenweise ausführen können. Sie können auch gezielt Fehler in das Programm einbauen und herausfinden, 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 der Arduino IDE

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 Version ist 1.8.10.

3.3.2 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 (Abbildung 22). Oft stehen dort bereits einige 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.

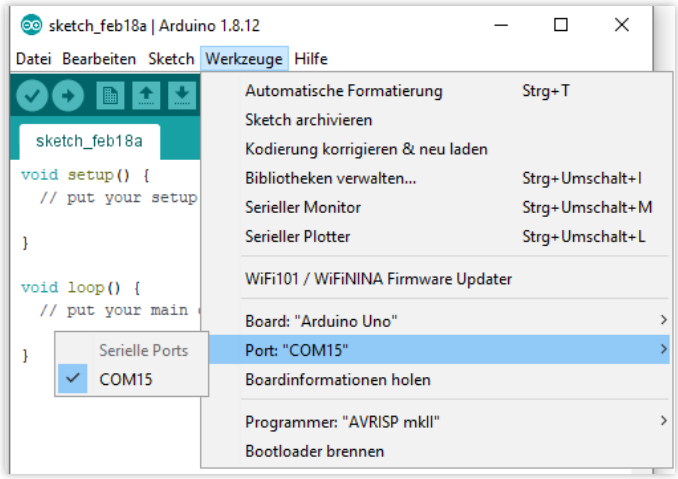

Abbildung 22: Auswahl des Ports in der Arduino IDE (Serieller Plotter)

Haben Sie den richtigen COM-Port ausgewählt, können Sie den Seriellen Plotter starten. Er ist ebenfalls unter „Werkzeuge“ zu finden (Abbildung 23). 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. Ansonsten wie oben beschrieben vorgehen um eine funktionierende Verbindung aufzubauen.

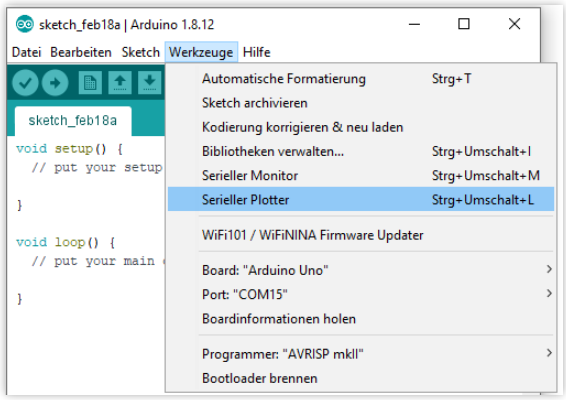

Abbildung 23: Auswählen des seriellen Plotters in der Arduino IDE

Details dazu, wie Sie Werte aus Ihrem Quellcode übermitteln können, finden Sie in der Dokumentation des Segway die in der System-Klasse beschriebenen Methoden `System::setDebugVal` und `System::sendDebugVals`.

3.4 Programmierrichtlinien

3.4.1 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 Wartung von Programmcode ist die Einhaltung eines definierten Programmierstils noch wichtiger als während der Entwicklung. Als Richtwert gilt, dass 80% der Lebenszeit eines Softwareprodukts 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. Welcher Kategorie die Richtlinie angehört, ist durch ein Ⓟ, bzw. ein Ⓔ 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.

3.4.2 Einheitlichkeit innerhalb der Gruppe

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

3.4.3 Bezeichner

Ⓟ 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++)
```

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

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

```
#define ANZAHL_STUDENTEN 265
const uint32_t WICHTIGES_DATUM = 11102161;
```

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

```
uint32_t segwaySpeed = 98
```

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

```
class WatchdogTimer
```

3.4.4 Variablen

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

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

Gut:

```
uint64_t sandkoernerAmStrand = 314159265358979384;
```

Ⓟ 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;
```

Ⓔ 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.

3.4.5 Formatierung

Ⓟ Nur eine Anweisung pro Zeile.
Schlecht:

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

Gut:

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

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

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

Ⓟ Switches werden wie folgend 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;
}
```

Ⓟ Begrenzen Sie die Breite Ihrer Zeilen. Die Autoformatierung von CCS limitiert die Breite auf 80 Zeichen, und auch nicht immer an sinnvollen Stellen innerhalb der Zeile. Sie können Werte größer als 80 wählen, müssen sich aber auf einen Wert einigen. Vereinzelte (!) Ausnahmen, z.B. für Funktionsköpfe sind erlaubt. Bei Kommentaren muss die gewählte Grenze immer eingehalten werden.

Ⓟ Kommentare stehen nie am Zeilenende, sondern immer in der Zeile darüber. Methodenkommentare können sowohl vor der Methode, als auch zu Beginn der Methode stehen.

Ⓔ Kommentare, die länger als zwei Zeilen sind, werden mit `/* … */` angeschrieben, nicht mit `// …` .

Ⓔ 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.

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

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

Gut:

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

3.4.6 Kommentieren

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

Ⓟ Der gesamte Quellcode muss so kommentiert sein, dass keine Nachfragen zum Verständnis der Funktionalität nötig sind. Sie können voraussetzen, 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*.
Schlecht (aus der `GPIO::init`):

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

Gut (aus der `GPIO::init`):

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

Ⓟ 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.

Ⓔ 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.

3.4.7 Klassen

Ⓟ 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 (Rückgabewerttyp, 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 Konstruktor als auch Destruktor müssen leer bleiben. Dies hängt damit zusammen, wie die Klassen innerhalb anderer Klassen verwendet werden.

Ⓔ Benutzen Sie immer CCS zum Erstellen neuer Klassen. Im Project Explorer (links) einen Rechtsklick auf das aktuelle CCS-Projekt, dann „New → Class“. Entfernen Sie den Haken bei „Namespace“ und geben Sie der Klasse den gewünschten Namen.

3.4.8 Konstanten

Ⓟ 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_PORTF_BASE)
{
    GPIOPinWrite(GPIO_PORTF_BASE, GPIO_PIN_3, 0xff);
}
```

3.4.9 Allgemeines

Ⓟ Die Zusammenarbeit mit anderen Gruppen oder gar der Austausch von Quellcode ist nicht gestattet. Es muss in der Prüfung nachvollziehbar sein, dass Sie die präsentierten Lösungen selbstständig erarbeitet haben.

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

Ⓟ 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`, …

Ⓔ Wenn optionale Übergabeparameter gefordert sind, so können Sie dies immer mit default-Argumenten lösen.

4 Aufgabenstellung

4.1 Hinweise zur Aufgabenstellung

4.1.1 Allgemeine Informationen

Diese Aufgabenstellung wurde mit der Zielsetzung geschrieben, eine vollständige und präzise Beschreibung der Anforderungen zu ergeben. Es ist zwingend erforderlich, dass Sie jeden Satz aufmerksam lesen.

Nachfolgende Aufgaben setzen voraus, dass Sie insbesondere das Kapitel Vorbereitung durchgearbeitet haben. Sie sollten ebenfalls einen Überblick über das Kapitel Unterlagen haben. Die Aufgabenstellung hat nicht die Funktion, Ihnen die benötigten Grundlagen zu erklären. Teil des Praktikums ist es, dass Sie sich das benötigte Handwerk selbstständig erarbeiten – wie es in der Praxis oft erforderlich ist.

4.1.2 Hinweise

Um Ihnen Programmieraufwand zu ersparen, stehen an einigen Stellen der Aufgabenstellung Hinweise, die Sie auf eine gute Lösungsmöglichkeit aufmerksam machen sollen. Es ist allerdings nicht schlimm, wenn Sie die Hinweise nicht umsetzen; viele Wege führen nach Rom.

Hinweise sind mit einem ⓘ markiert.

4.1.3 Bonusaufgaben

Für diejenigen, die etwas interessantere Programmieraufgaben suchen, gibt es an einigen Stellen Bonus-Aufgaben. Ihr Schwierigkeitsgrad liegt deutlich über dem der normalen Praktikumsaufgaben und soll zum Nachdenken und Tüfteln anregen. Die Aufgaben sind vollkommen optional. Sie werden nicht für eine erfolgreiche Probefahrt benötigt und haben keinerlei Einfluss auf die Bewertung in der mündlichen Prüfung.

Bonusaufgaben sind mit einem Ⓑ markiert.

4.2 Grundlegende Klassen

4.2.1 Einleitung

Nachfolgende 3 Klassen bilden zusammen mit den gegebenen Klassen das Grundgerüst für alle Programme in diesem Workshop – insbesondere für das Segway-Programm. Sie dienen dazu, mit den wichtigsten Hardwaremodulen des Tiva™ Boards objektorientiert arbeiten zu können.

Zu jeder Klasse gibt es ein vorgeschlagenes Testprogramm, mit dem Sie die einzelnen Methoden Ihrer Klasse überprüfen können. Es steht Ihnen frei, von diesem Vorschlag abzuweichen, und eigene Testprogramme zu schreiben. Dies jedoch unter der Bedingung, dass Ihr Programm mindestens die jeweils aufgeführten Methoden durchläuft und somit deren Funktionsfähigkeit demonstriert.

Da sowohl Konstruktor als auch Destruktor aller Klassen leer bleiben müssen, werden sie hier nirgends explizit aufgeführt.

Alle Klassen benötigen einen Zeiger auf die laufende Instanz der System-Klasse. Dies ist zugleich die einzige Klasse, die Sie innerhalb dieser vier Klassen verwenden dürfen.

Da die GPIO-Klasse sich gut als Einstieg in die Programmierumgebung des IT1 Praktikums eignet und einige wichtige Grundlagen und Vorgehensweisen lehrt, werden Ihnen Teile davon in der zweiten Einführungsveranstaltung erklärt. Obwohl Sie in Ihrer Vorlage bereits die Musterlösung der Klasse haben, raten wir Ihnen dringend, die Einführungsveranstaltungen zu besuchen und sich selbst mit dem Quellcode der Klasse auseinander zu setzen.

Stellen Sie sicher, dass Sie die GPIO-Klasse verstehen, ehe Sie mit den anderen Klassen beginnen!

4.2.2 Timer-Klasse

4.2.2.1 Was ist ein Timer?

Der Timer wird benutzt, um Anweisungen nach einer definierten Zeit auszuführen. So können Sie z.B. Messungen in regelmäßigen Abständen durchführen, oder eine LED blinken lassen. Der große Vorteil von Timern, bzw. Interrupts ist, dass sie den Prozessor nur während der eigentlichen Aufgabe beanspruchen, und dieser zwischen den Interrupts für andere Zwecke verwendet werden kann.

Im Grunde genommen ist ein Timer nichts anderes als ein Zähler, der beim so genannten „Load"-Wert startet und bei jedem Taktzyklus dekrementiert (herunterzählt). Der Timer kann nur in einem fixen Wertebereich arbeiten, da die Register des Timers nur eine bestimmte Größe haben. Ein 16 Bit Timer kann daher nur zwischen 0 und 2^{16}-1 (65535) zählen.

Auf den LaunchPads gibt es 6 Timer-Module mit je zwei 16 Bit Timern, und 6 Wide-Timer-Module mit je zwei 32 Bit Timern. Diese zwei Timer in jedem Modul, genannt Timer A und B, können einzeln verwendet werden, oder aber zusammengelegt werden und als ein einziger Timer genutzt werden. Damit werden die Register von Timer A um die von Timer B erweitert werden, wodurch der vergrößerte Timer A nun z.B. statt bis 2^{16}-1 (65535) bis 2^{32}-1 (4294967295) zählen kann. Ein Timer kann entweder periodisch zählen („periodic"), d.h. er startet von selbst immer wieder beim Load-Wert, oder auch nur ein einziges Mal („single-shot"). Mit einem Timer können Änderungen an digitalen Ausgängen (ähnlich der PWM-Funktion, siehe Was ist PWM?), ADC Lesevorgänge oder Interrupts ausgelöst werden. Hat der Timer bis auf 0 heruntergezählt, setzt das Timer Modul einen Flag um dem Prozessor zu signalisieren, dass die Zeit abgelaufen ist. Dieser Flag heißt „Timeout Interrupt Flag". Wenn der zugehörige Interrupt aktiviert ist, wird das laufende Programm unterbrochen und die festgelegte („registrierte") „Interrupt Service Routine" (ISR) wird ausgeführt. Der Code in dieser Routine sollte so kurz wie möglich gehalten werden um sicherzugehen, dass er vollständig ausgeführt wird bevor das Interrupt das nächste Mal ausgelöst wird und ebenfalls noch Rechenzeit für das Hauptprogramm übrig ist. Die ISR kann keine Objektmethode sein und muss, wie bereits erwähnt, registriert werden, damit der Prozessor weiß, welche Methode er ausführen muss. Es ist für die korrekte Funktion sehr wichtig, dass zu Beginn der ISR der „Timeout Interrupt Flag" gelöscht wird, da sonst die ISR in einer Dauerschleife ausgeführt wird.

Weitere Informationen finden sich in der „TivaWare™ Treiberbibliothek" Seite 531 und im „TivaC Mikrocontroller Datenblatt" Seite 704.

4.2.2.2 Anforderungen an die Klasse

Mit dieser Klasse können bis zu 6 Timerobjekte mit 32 Bit Auflösung erstellt werden. Jedes Objekt ruft periodisch eine gegebene Interrupt Service Routine auf. Arbeiten Sie mit den 16/32 Bit Timer Modulen. Nachfolgend finden Sie, welche Methoden die Klasse zur Verfügung stellen muss.

4.2.2.3 `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 oHz.

- Funktion/Umsetzung
  Analog zur `GPIO::init`-Methode führt diese Methode die Initialisierung des Timer-Moduls und des Timer-Objektes durch.

  ① Die Konstanten `TIMERx_BASE` sind keine Zufallswerte. Tatsächlich gilt: `TIMER(x+1)_BASE = TIMERx_BASE + 0x1000` (Siehe `"inc/hw_memmap.h"` Zeile 81 folgend). Überlegen Sie sich, wie Sie dies nutzen können.

4.2.2.4 `Timer::start`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `void`
- Funktion/Umsetzung
  Aktiviert den Timer.

4.2.2.5 `Timer::stop`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `void`
- Funktion/Umsetzung
  Deaktiviert den Timer. Beachten Sie, dass eventuell noch ein Interrupt Flag gesetzt ist. 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.

4.2.2.6 `Timer::getFreq`

- Rückgabewert/-typ
  `uint32_t`: Aktuelle Timer-Frequenz in Hz.
- Übergabeparameter
  `void`
- Funktion/Umsetzung
  Liefert die aktuell eingestellte Frequenz des Timers in Hertz zurück. Beachten Sie, dass diese Funktion stets einen korrekten Wert zurückliefern muss, unabhängig davon ob `Timer::setFreq` oder `Timer::setPeriodUS` oder keine von beiden Methoden vorher aufgerufen wurden.

4.2.2.7 `Timer::getPeriodUS`

- Rückgabewert/-typ
  `uint32_t`: Aktuelle Timer-Periode in µs.
- Übergabeparameter
  `void`
- Funktion/Umsetzun
  Liefert die aktuell eingestellte Periode des Timers in Mikrosekunden zurück. Beachten Sie, dass diese Funktion stets einen korrekten Wert zurückliefern muss, unabhängig davon ob `Timer::setFreq` oder `Timer::setPeriodUS` oder keine von beiden Methoden vorher aufgerufen wurden.

4.2.2.8 `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.

4.2.2.9 `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 von 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.

4.2.2.10 `Timer::clearInterruptFlag`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `void`
- Funktion/Umsetzung
  Löscht das „Timeout Interrupt Flag“ dieses Timers.

4.2.2.11 Testprogramm

- Mindestens zu prüfende Methoden
  - `Timer::init`
  - `Timer::start`
  - `Timer::setFreq`
  - `Timer::setPeriodUS`
  - `Timer::clearInterruptFlag`
- Programmvorschlag
  Das Testprogramm erlaubt es, die blaue LED mit unterschiedlichen Frequenzen blinken zu lassen. Durch Drücken von SW1 kann der steuernde Timer gestartet oder gestoppt werden. Durch Drücken von SW2 kann zwischen zwei Blinkfrequenzen gewechselt werden: entweder die LED blinkt 1 mal alle 3 Sekunden, oder sie blinkt 2 mal pro Sekunde.

  Ⓑ Der folgende Programmvorschlag ist etwas anspruchsvoller und kann als zusätzliche Übung betrachtet werden. Das Testprogramm lässt die blaue LED schrittweise schneller blinken und anschließend die rote LED schrittweise langsamer blinken. Dies wiederholt sich dauerhaft. Die Blinkfrequenz steigt dabei von 1 Hz auf 5 Hz und fällt anschließend von 5 Hz auf 1 Hz. Der Wechsel von einer Frequenz auf die nächste erfolgt alle 2 Sekunden. Die LEDs sind dabei genau so lange ein- wie ausgeschaltet (50% Tastgrad). Weiterhin darf die main-Funktion neben der Initialisierung nur eine leere while-Schleife enthalten. Delays sind selbstverständlich auch verboten (Siehe auch Mitte des 4. Absatzes im Kapitel Was ist ein Timer?).

4.2.3 ADC-Klasse

4.2.3.1 Was ist ein ADC?

Ein ADC ist ein „Analog-to-Digital-Converter“ oder auch Analog-Digital-Wandler, welcher kontinuierliche Spannungen einliest und diese in diskrete Zahlenwerte umwandelt. Das LaunchPad hat zwei identische ADC Module mit je 12 Bit Auflösung. Dies bedeutet, dass die analoge Spannung, die zwischen 0V und 3,3V liegt, durch eine 12 Bit Zahl dargestellt wird. 4095 (2^{12}-1) entspricht demnach 3,3V. Jedes Modul kann jeden der 12 analogen Eingänge auslesen (siehe Seite 801 des „TivaC Mikrocontroller Datenblatt“). Die Module besitzen je einen Wandler, welcher bis zu einer Millionen Umwandlungen je Sekunde machen kann. Obwohl dies nach viel klingt, ist es im Vergleich zur Taktrate des Prozessors (40 - 80 MHz) ziemlich wenig. Um komplexere Aufgaben zu ermöglichen, besitzt jedes Modul vier Sample Sequencer, deren Aufbau bis auf die Tiefe des FIFOs (und damit der Anzahl der speicherbaren Samples) identisch ist. Folgende Anzahl an Samples können maximal gespeichert werden:

- Sequencer 0: bis zu 8 Pins
- Sequencer 1: bis zu 4 Pins
- Sequencer 2: bis zu 4 Pins
- Sequencer 3: bis zu 1 Pin

Der Trigger (z.B. Software-Aufruf oder Timer) sorgt dafür, dass das ADC Modul alle Pins in einem Sample Sequencer hintereinander ausliest. Wenn dies geschehen ist, führt das Modul seine Endroutine aus, z.B. setzt es einen Flag oder kopiert die gelesenen Werte in eine Variable. Wenn mehrere Sequencer auf einmal getriggert werden, so werden sie nach absteigender Priorität (welche Sie selbst festlegen können) abgearbeitet, da das Modul nur einen AD-Wandler hat.

4.2.3.2 Anforderungen an die Klasse

Die ADC-Module des LaunchPad stellen einen beachtlichen Funktionsumfang und Freiheitsgrad bei der Konfiguration zur Verfügung, der weit über die Anforderungen dieser Klasse hinausreicht.

Jedes Objekt der ADC-Klasse soll genau einen Pin auslesen, und darf dazu einen ganzen Sample Sequencer beanspruchen. Der Sample Sequencer muss also nie mehr als einen Pin auslesen.

4.2.3.3 `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.

  Dem Sample Sequencer muss eine Priorität von 0 bis 3 zugeteilt werden, für den Fall, dass mehr als einer gleichzeitig angefordert wird. Dabei dürfen keine zwei Sequencer die gleiche Priorität haben. Da für diese Klasse die Reihenfolge keine Rolle spielt, überlegen Sie sich, wie Sie sicherstellen können, dass jeder Sequencer sicher eine eindeutige Priorität hat, auch wenn nicht bekannt ist, welche anderen Sequencer noch verwendet werden.

  Zur fehlerfreien Funktion des ADC muss der jeweils benutzte GPIO Port freigeschaltet und der Pin als ADC Eingang konfiguriert werden. Der `analogInput` legt eindeutig fest, welcher GPIO Port und Pin verwendet werden. (siehe „TivaC Mikrocontroller Datenblatt“ Seite 801). Überlegen Sie sich, wie Sie diese Information zur Konfiguration nutzen können.

  ⓘ Die Konstanten `ADC_CTL_CHx` sind keine Zufallswerte. Tatsächlich handelt se sich um fortlaufende Zahlen: `ADC_CTL_CHx = x` mit `0 <= x <= 11` (Siehe `"driverlib/adc.h"` Zeile 85 folgend). Überlegen Sie sich, wie Sie dies nutzen können. Werfen Sie ggf. nochmal einen Blick in die GPIO Klasse, in der etwas ähnliches bereits vorteilhaft genutzt werden konnte.

4.2.3.4 `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).

  ⓘ Sie müssen nicht berücksichtigen, dass diese Methode Auswirkungen auf andere ADC-Objekte auf dem gleichen ADC-Modul hat.

4.2.3.5 `ADC::read`

- Rückgabewert/-typ
  `uint32_t`: Wert von 0 bis 4095 entsprechend einer Spannung von 0 V bis 3,3 V
- Übergabeparameter
  `void`
- Funktion/Umsetzung
  Liefert für den zugehörigen Pin einen Wert zwischen 0 und 4095 zurück, der einer Spannung von 0 V bis 3,3 V entspricht. Die Methode startet die AD-Konvertierung und wartet das Ergebnis ab.

  Achtung! Da die Möglichkeit besteht, dass der selbe Analogeingang sowohl innerhalb des Hauptprogrammes als auch innerhalb einer ISR ausgelesen wird, müssen Interrupts während dem Auslesen mit `IntMasterDisable()` deaktiviert und am Ende der Methode mit `IntMasterEnable()` wieder aktiviert werden.

  Andernfalls kann es zu folgendem Problem kommen:
  - Hauptprogramm startet AD-Konvertierung
  - Interrupt
  - ISR startet AD-Konvertierung, was nutzlos ist, da sie bereits läuft
  - AD-Konvertierung abgeschlossen, das entsprechende Flag wird noch in der ISR zurückgesetzt
  - ISR endet
  - Hauptprogramm wartet auf das Flag, das das Ende der AD-Konvertierung markiert. Da diese aber schon beendet ist, wird das Hauptprogramm die Warteschleife nie verlassen können.

  Die Lösung des Problems ist zu verhindern, dass der Prozessor während der AD-Konvertierung auf Interrupts reagiert. So ist er gezwungen, jede AD-Konvertierung sauber zu beenden, bevor die Möglichkeit besteht eine neue zu starten.

4.2.3.6 `ADC::readVolt`

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

  ① Sie können die `ADC::read` verwenden.

4.2.3.7 Testprogramm

- Mindestens zu prüfende Methoden
  - `ADC::init`
  - `ADC::setHWAveraging`
  - `ADC::read`
  - `ADC::readVolt`
- Programmvorschlag
  Das Testprogramm vergleicht die Spannung in Volt an einem Testpin mit der Spannung an einem Referenzpin. Liegt die Testspannung höher als die Referenzspannung, leuchtet die on-board LED grün, andernfalls rot. Für eine möglichst stabile Referenz soll diese 64-fach gemittelt werden. Testpin und Referenzpin werden unterschiedlichen ADC-Modulen zugeteilt.

4.2.4 PWM-Klasse

4.2.4.1 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 ist allerdings für sehr viele Anwendungen nicht akzeptabel. Sie wollen z.B. mit Ihrem Modellauto nicht immer nur Vollgas fahren, oder stehen bleiben.

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

Das Hin- und Herschalten geschieht mit einer festen Frequenz. Zusätzlich gibt der sog. Tastgrad an, wie lange der Pin an oder ausgeschaltet ist. Im Falle des TivSegs™ bestimmt dieser Wert also 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 Compare-Wert entscheidet, bis zu welchem Zähler-Wert die Ausgänge auf HIGH bleiben. 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.

4.2.4.2 Anforderung 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.

4.2.4.3 PWM::init

- Rückgabewert/-typ
  void
- Übergabeparameter

`System *sys`:	Zeiger auf die laufende Instanz der System-Klasse
`uint32_t portBase`:	Basis-Adresse des GPIO-Ports mit den PWM-Pins, z.B. `GPIO_PORTF_BASE`.
`uint32_t pin1, pin2`:	Die zwei Ausgangspins, z.B. `GPIO_PIN_6` und `GPIO_PIN_7`. Sie müssen zum gleichen PWM-Generator gehören. In welcher Reihenfolge die Pins angegeben werden, spielt keine Rolle.
bool `invert`:	Optionale Angabe ob die Ausgangssignale invertiert werden. Der Standardwert ist **false**.
`uint32_t freq`:	Optionale Angabe der PWM-Frequenz in Hz. Der Standardwert ist 5 kHz.

- 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 aufgrund der übergebenen Werte für `portBase`, `pin1` und `pin2` bestimmen.

  ① 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 folgend 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.

  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).

4.2.4.4 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.

  Ⓑ 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 entsprechende 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 Übergabeparameter `adjustPWMClockDiv` hinzufügen, der den default Wert **`false`** hat. Nur wenn dieser explizit auf **`true`** gesetzt wird, darf die Methode den Teiler ändern.

4.2.4.5 `PWM::setDuty`

- Rückgabewert/-typ
  `void`

- Übergabeparameter
  `float` `duty`: Tastgrad als Gleitkommazahl zwischen `-1.0f` und `1.0f`. Das Vorzeichen bestimmt die Drehrichtung.

- Funktion/Umsetzung
  Ist das PWM-Objekt konfiguriert, lässt sich der angeschlossene Motor mit dieser Methode vollständig steuern. Je nach Richtung, also je nachdem ob der übergebene Tastgrad positiv oder negativ ist, liegt das entsprechende PWM-Signal nur an dem einen oder nur an dem anderen Ausgangspin des PWM-Generators an. Der jeweils andere Pin liegt auf Masse. Als Konvention gilt, dass ein Tastgrad größer Null „vorwärts“ bedeutet, und in dem Fall das PWM-Signal am „niedrigwertigen“ Pin anliegt. Sollen also z.B. die Pins PB6 und PB7 angesteuert werden, so liegt das Signal für Werte größer Null an PB6 an.

  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.

4.2.4.6 Testprogramm

- Mindestens zu prüfende Methoden
  - `PWM::init`
  - `PWM::setFreq`
  - `PWM::setDuty`

- Programmvorschlag
  Das PWM-Testprogramm ermöglicht es Ihnen, Ihre zwei Motoren und deren Ansteuerung mit dem Treiber SN754410 zu testen. Die Eingabe erfolgt über vier GPIO Pins, zwei pro Motor. Sie dienen dazu, für jeden Motor unabhängig den Tastgrad um 10% erhöhen und die Drehrichtung der Motoren wechseln zu können. Beachten Sie, dass Tastgrade von über 100% nicht auftreten dürfen; setzen Sie den Tastgrad in dem Fall zurück auf 0%.

  Einer der Motoren soll mit der Standardfrequenz der PWM-Klasse betrieben werden, der andere mit einer Frequenz von 2500 Hz.

  Sie können die vier steuernden GPIO-Pins so legen, dass es keine Konflikte mit dem Aufbauvorschlag fürs komplette MiniSeg™ (siehe Kapitel Aufbau des MiniSeg™) gibt. Da Ihr Launchpad nur zwei Taster hat, können Sie analog zum Fußschalter Drahtbrücken für die zwei anderen Taster nehmen. Denken Sie daran, dass Sie – wie im Segway-Programm auch – 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.

4.3 Segway-Programm

4.3.1 Einleitung

Dieser zweite Teil der Aufgabe unterscheidet sich deutlich von den grundlegenden Klassen. Letztere haben Sie vertraut mit der Hardware und den Möglichkeiten des Mikrocontrollers gemacht. Jetzt geht es darum, mit diesem „Werkzeug“ zu arbeiten, wobei Ihre Spielräume bei der Umsetzung bedeutend größer werden. Es gibt viele gute Lösungen!

Da Sie die Praxistauglichkeit Ihres Codes erst bei der Probefahrt selbst überprüfen können, raten wir Ihnen dringend, Ihre Ansätze mit den Tutoren zu besprechen. Diese können Sie bereits vorher auf eventuelle Probleme aufmerksam machen.

Einen Überblick über den Aufbau und die Funktionsweise des Segway-Programmes finden Sie in den Unterlagen.

Denken Sie daran, die verwendeten Konstanten und Einstellungen in der Konfiguration einzutragen.

4.3.2 Lenkung

4.3.2.1 Motivation und Anforderungen

Die Lenkung besteht aus einem Potentiometer, das durch die Lenkstange (beim TivSeg™) leicht in die eine bzw. in die andere Richtung gedreht wird. Durch diese Drehung wird eine leichte Spannungsänderung an einem der analogen Eingänge des Mikrocontrollers verursacht, die mittels ADC ausgelesen wird.

Beim TivSeg™ ist der Verstellbereich des Potentiometers mechanisch begrenzt, schließlich können Sie die Lenkstange nur um wenige Grad in beide Richtungen schwenken. Auf dem MiniSeg™ gibt es eine solche Begrenzung nicht. Um diese nachzuahmen, sind auf dem MiniSeg™ zwei Widerstände in Reihe zum Potentiometer eingebaut (R5 und R6 in Abbildung 30).

Von dem Messwert zum normierten Ausschlag der Lenkung zu gelangen, ist Aufgabe der Steering-Klasse. Neben der Pflicht-Methode `Steering::getValue` werden Sie wahrscheinlich noch weitere Methoden implementieren müssen, z.B. zur Kalibrierung der Lenkung. Selbstverständlich müssen diese Methoden, falls nötig, auch an den passenden Stellen im Quellcode eingefügt werden.

Der Regler des Segways erwartet einen Wert zwischen -1.0 und 1.0, entsprechend einem linken oder rechten Vollausschlag der Lenkung. Da die Achse des Potentiometers keine Markierung hat, kann beim Zusammenbau nicht garantiert werden, dass das Potentiometer in jedem TivSeg™ die gleiche Nullstellung hat. Auch genaue Amplitude kennen Sie nicht im Voraus.

Um zu testen ob Ihre Klasse mit diesen unvermeidbaren Schwankungen von TivSeg™ zu TivSeg™ klar kommt, können Sie unterschiedliche Widerstände R5 und R6 in Ihrem MiniSeg™ einbauen. Passende Werte sind im Schaltplan hinterlegt (siehe Aufbau des MiniSeg™). Stellen Sie sicher, dass die `Steering::getValue` Methode unabhängig von den verwendeten Widerständen stets den Bereich von -1.0 bis +1.0 korrekt abdeckt. Hierzu empfiehlt es sich, den Rückgabewert mittels `System::setDebugVal` und `System::sendDebugVals` in der Arduino IDE zu plotten.

Ⓑ Tatsächlich lässt sich die Lenkstange des TivSegs weiter in eine Richtung als in die andere Richtung bewegen, da die Anschläge nicht exakt symmetrisch positioniert sind. Die Mittelposition ist also nicht genau der Mittelwert aus linkem und rechtem Vollausschlag. Wird der Mittelwert als Nullposition verwendet, so muss die Lenkstange für eine Geradeausfahrt minimal schief gehalten werden. Wird diese Abweichung kompensiert, kann es je nach Ansatz sein, dass Sie als Konsequenz weiter oder schneller in eine Richtung lenken als in die andere. Die optionale Bonusaufgabe lautet somit, eine Steering-Klasse zu schreiben, die diese Eigenschaften berücksichtigt, die Mittelposition korrekt erkennt, und sowohl fürs Links- als auch fürs Rechtslenken die gleiche Empfindlichkeit und den gleichen Höchstwert hat.

4.3.2.2 `Steering::getValue`

- Rückgabewert/-typ
  `float`: Wert zwischen `-1.0f` und `1.0f` entsprechend einer vollen Lenkung nach rechts (`-1.0f`) bzw. links (`1.0f`). `0.0f` entspricht einer Geradeausfahrt.
- Übergabeparameter
  `void`

4.3.2.3 Beispiel

Folgende Messung wurde auf einem TivSeg™ gemacht. In diesem Fall lag die Lenkspannung zwischen ca. 2,5V und ca. 3,0V. Anhand Abbildung 24 können Sie sehen, wie die Schwankung der Spannung in diesem Bereich auf -1 bis 1 übersetzt wird.

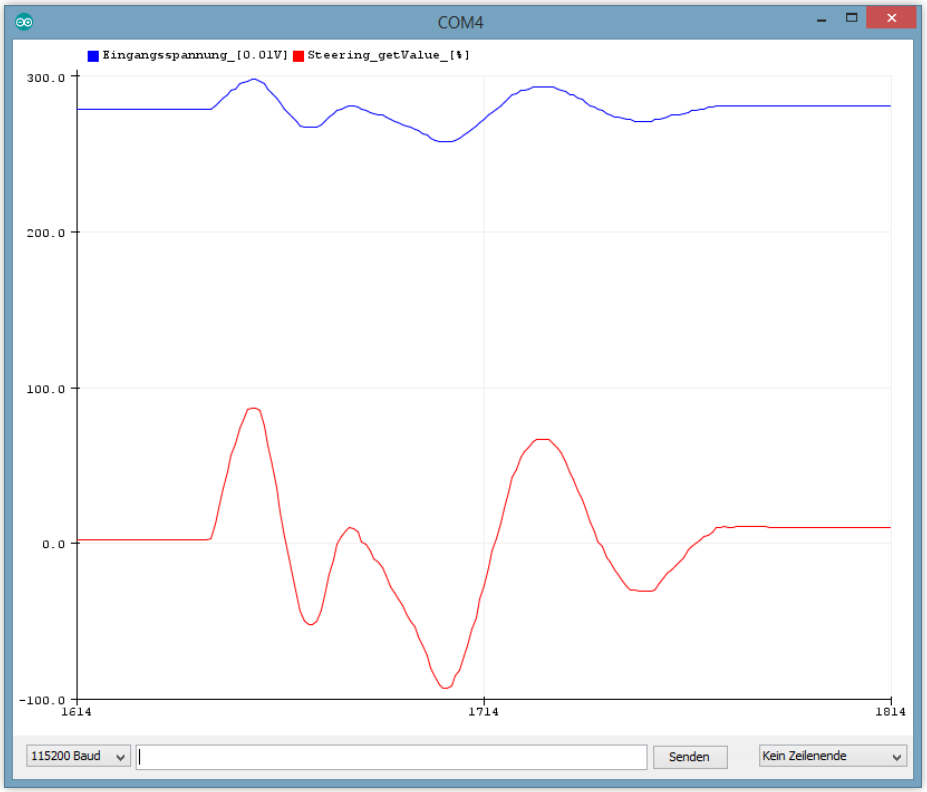

Abbildung 24: Eingangsspannung in 10 mV und verarbeiteter Wert für Controller in % im Arduino Serial Plotter

4.3.3 Akkuspannungsüberwachung

4.3.3.1 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 TivSegs™ liegt diese Grenze bei 21V. Obwohl mit fortschreitender Entladung auch die Leistungsfähigkeit nachlässt, so ist es für den Fahrer nicht möglich zu erkennen, ab wann der Akku Schaden nimmt. Es bedarf daher einer genauen Spannungsüberwachung. Der entsprechende ADC-Pin ist auf dem TivSeg™ mit einem einfachen 1:10 Spannungsteiler an die Akkuspannung angeschlossen. Sie messen auf dem TivSeg™ also zu jedem Zeitpunkt 1/10 der aktuellen Batteriespannung. Da das MiniSeg™ keine Akkus hat, können Sie die Spannung mittels des Potentiometers (RV2 im Schaltplan, siehe Abbildung 30) beliebig einstellen und so Ihre Spannungsüberwachung testen.

Integrieren Sie eine Spannungsüberwachung in die `Segway::backgroundTasks`-Methode, die das TivSeg™ stoppt, 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`).

4.3.4 Segway-Steuerung

4.3.4.1 Aufgepasst!

Diese Aufgabe ist nur für die Studierende vorgesehen, die drei statt zwei Leistungspunkte für dieses Praktikum erhalten. Andernfalls finden Sie diese Aufgabe bereits fertig als Teil Ihrer Code-Vorlage und müssen dieses Kapitel nicht beachten.

4.3.4.2 Motivation und Anforderungen

Während Sie den Regelalgorithmus als Klasse gestellt bekommen (siehe Controller-Klasse), ist das Einbauen dieser Klasse ins Segway-Programm Ihre Aufgabe. Ebenfalls Teil der Aufgabe ist das Überwachen des Fußschalters.

Eine Erklärung der Funktion sowie das Flussdiagramm eines nicht verpflichtenden Lösungsansatzes finden Sie in den Unterlagen. Passen Sie die `Segway::update`- und ggf. die `Segway::backgroundTasks`-Methode an.

5 Test und Debugging

Nachdem Ihnen in den vorherigen Kapiteln die Aufgabenstellung und alle benötigten Softwaretools nähergebracht wurden, sind Sie in der Lage, alle geforderten Klassen zu bearbeiten. Leider kommt es beim Programmieren nur äußerst selten vor, dass der geschriebene Code auf Anhieb fehlerfrei funktioniert. Aus diesem Grund ist das Debuggen und Testen eines bestehenden Programms eine ebenso wichtige Kompetenz eines Softwareentwicklers wie das Implementieren neuer Funktionen. Um Ihnen das Leben zu erleichtern, finden Sie in diesem Kapitel einige Tipps beim Benutzen des CCS Debuggers sowie zusätzliche Infos die fürs Testen auf dem MiniSeg™ relevant sind.

Um zu verhindern, dass Sie von den vielen Infos erschlagen werden, empfehlen wir, dieses Kapitel erst durchzuarbeiten, wenn Sie bereits mindestens eine Klasse selbst geschrieben haben – ohne fertigen Code können Sie schließlich nicht viel testen.

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 Fault ISR 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 pausieren, 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, in der eine TivaWare™ API Methode aufgerufen wird. Da Sie davon ausgehen 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 Sie 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 nichts 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 expression“). Für ganz harte Fälle ließen sich noch die einzelnen Register im entsprechenden Fenster auslesen, dies ist jedoch normalerweise nicht nötig und erfordert einiges an 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 beschriebenen FaultISR kann das Debuggen sehr umständlich sein. Um dieses Problem teilweise zu umgehen, wurde in der System-Klasse die `System::error`-Methode implementiert. Das Aufrufen dieser Funktion stoppt alle Interrupts und setzt den Code in 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` zu arbeiten, müssen Sie diese an den kritischen Stellen in Ihren Klassen einbauen. Ziel ist es, alle problematischen Fälle (Overflow und 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

Sämtliche „Hardware Klassen“, die Sie bisher kennengelernt haben (GPIO, ADC, Timer, PWM) dienen schlussendlich nur dazu, das Segway-Programm, das das TivSeg™ (bzw. MiniSeg™) steuert, mit Daten zu versorgen (siehe Programmstruktur). Daher versteht sich, dass es wenig Sinn macht, an der Segway-Klasse zu arbeiten (Lenkung und Akkuspannungsüberwachung), ohne alle „Hardware Klassen“ zuvor vollständig implementiert und getestet zu haben.

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 an der Segway-Klasse arbeiten zu können, dürfen Sie für die nicht bearbeiteten Klassen unsere Musterlösung verwenden. Weitere Informationen diesbezüglich erhalten Sie zeitnah auf GitLab als Issue. 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 Timer-, ADC- und PWM-Klasse 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 MiniSeg™

Nachdem Sie alle Klassen vollständig implementiert und die grundlegenden Klassen (siehe oben) mit den Testprogrammen überprüft haben, können Sie nun das MiniSeg™ zum weiteren Test verwenden. Der folgende Abschnitt lehrt Sie die Handhabung des MiniSegs™ und zeigt wie Sie dessen Funktionalität überprüfen.

Nachdem Sie Ihr MiniSeg™ wie im Kapitel Aufbau des MiniSeg™ beschrieben aufgebaut und das Steckbrett auf eine ebene Fläche gestellt haben, können Sie Ihren Code auf das LaunchPad flashen. Zu Beginn wird das Potentiometer an Pin PE2 auf Maximum gestellt („Akku geladen“) und das Kabel an Pin PB5 mit Masse verbunden („Fußschalter nicht gedrückt“). Die Motoren dürfen sich jetzt nicht drehen. Kalibrieren Sie bei Bedarf Ihre Lenkung (Potentiometer an Pin PE1).

Im nächsten Schritt stellen Sie die Lenkung in ihre Mittenposition und trennen den Pin PB5 von der Masse („Fußschalter gedrückt“). Nun sollten die Motoren beginnen, sich zu drehen. Durch leichtes Neigen des Steckbrettes abwechselnd in beide Richtungen sollten Sie eine Ruhelage finden, bei der die Motoren sich nicht oder nur sehr langsam drehen. Sobald Sie diese Position verlassen, werden die Motoren wieder arbeiten, damit sich das Fahrzeug selbstständig wieder in seine Ruhelage bringt (was nur beim TivSeg™ klappt). Durch Betätigen der Lenkung sollten Sie beobachten können, dass jeweils einer der Motoren sich schneller dreht als der andere oder, je nach Lage, beide Motoren in unterschiedliche Richtungen drehen.

Zuletzt können Sie noch Ihre Akkuspannungsüberwachung testen. Stellen Sie das entsprechende Potentiometer in die andere Extremstellung („Akku vollständig entladen“). Nach fünf Sekunden müssen die Motoren zum Stillstand kommen. Erst nach zurückstellen des Potentiometers und einem Reset des LaunchPad ist das MiniSeg™ wieder betriebsbereit. Die Lenkung muss dann erneut kalibriert werden. Im Anschluss können Sie noch die Position des Potentiometers ermitteln, bei dem der Akkuschutz gerade so anspricht und das MiniSeg™ abschaltet. Prüfen Sie, ob dies der geforderten Spannungsgrenze entspricht.

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 (Siehe auch Empfehlung zu Bezeichner).

- 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 die Hinweise in den Programmierrichtlinien zu diesem Thema beachten (siehe oben).
- Teilt man einen Integer (Ganzzahl) durch einen anderen Integer, so wird eine Ganzzahldivision durchgeführt (bspw. ist `7/2 = 3;  5/3 = 1;  1/4 = 0`). Dies kann vermieden werden, indem Sie explizit angeben, dass die Variablen als `float`, also eine Gleitkommazahl behandelt werden sollen (Stichwort Type casting).

  Da jede Konvertierung zusätzlichen Rechenaufwand 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 den gewährten 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` berechnet, und 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 keinen Nachlass von z.B. 10%, 20%, oder 30% 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 die Multiplikation durchführen, wird die Zahl deutlich größer als 100 und der Genauigkeitsverlust bei der Division 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 eine 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 `uint8_t`. Ebenfalls gleich ist das Verhalten an den Enden der Skala: `-120 - 10 = 126` und `100 + 100 = -56`.

- Mehrfachvergleiche 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 der 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 kompilieren, 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 Fault ISR landen: Haben Sie daran 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 MiniSeg™

- Falls sich das MiniSeg™ beim Testen nicht wie erwartet verhält, lohnt sich ein Blick in die `Config.h`, ob dort auch die Präprozessorvariable für das MiniSeg gesetzt ist (siehe Spezifische Einstellungen für TivSeg™ und MiniSeg™).
- Falls es Probleme mit dem MPU6050 gibt (Keine Werte werden empfangen, `MPUCommunicationError`, etc.) überprüfen Sie, ob Sie den Sensor richtig angeschlossen haben. Häufig geschieht es, dass Teile (auch andere als der MPU6050) falsch angeschlossen werden. Sie können zudem die `main.cpp` der Accel_Class_Test verwenden, um die Funktion Ihres MPU6050 zu überprüfen.
- Nochmal der Hinweis darauf, dass Sie nicht davon ausgehen können, dass die Mitte des Spannungsbereiches des Potentiometers tatsächlich auch die Nullstellung der Lenkstange ist.

5.5.4 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 in Ihrem Repository darum bitten, es wird allerdings nicht seine höchste Priorität sein, das zu erledigen.
- Bitte posten Sie ihre Fragen nur im Fragen-Projekt. Wenn Sie in Dokumente und Infos posten, werden alle Studenten eine Mail dafür bekommen. So gehen wichtige Informationen und Ankündigungen schnell 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 das TivSeg™ zum Fahren zu bringen. Die Grundarchitektur ist Ihnen bereits gegeben. Die Abstraktionsebenen sind in Abbildung 25 dargestellt. Nachfolgend finden Sie auch eine ausführliche Dokumentation der einzelnen gegebenen Klassen.

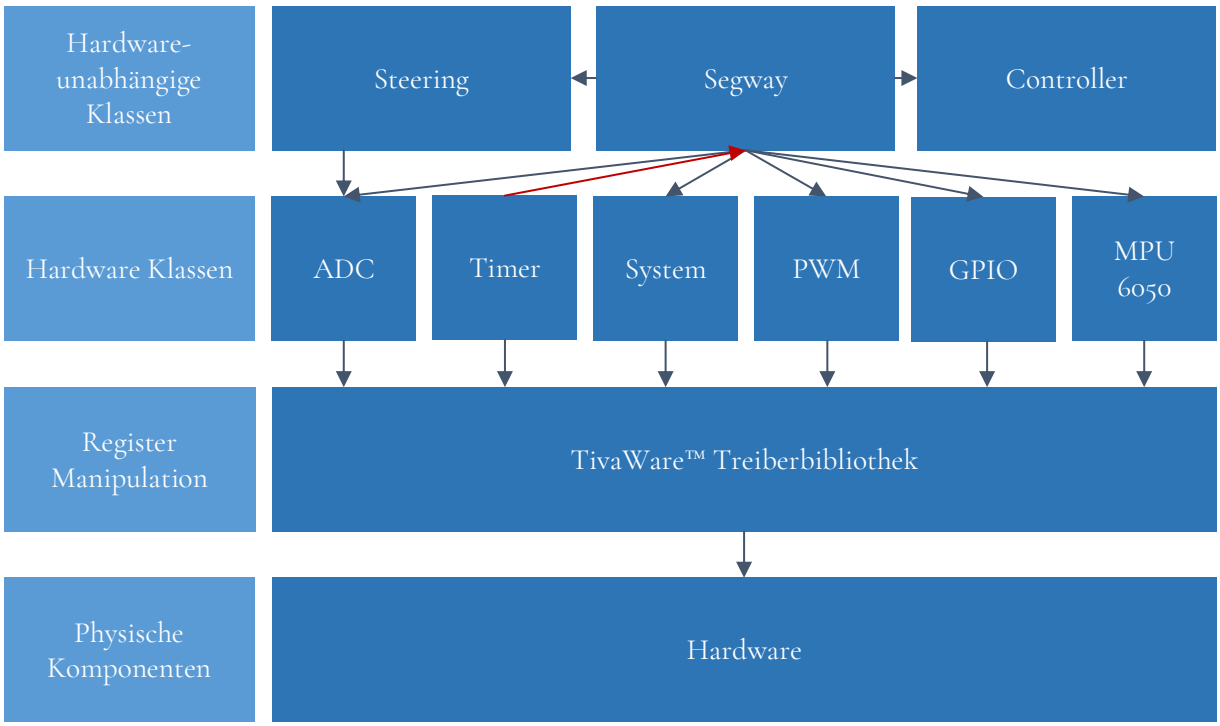

Abbildung 25 Abstraktionsebenen

Zu beachten ist hierbei, dass zwar in der Segway- und der Controller-Klasse die „Magie“ geschieht, jedoch die in der `main()` initialisierten Timer dafür sorgen, dass das Programm läuft!

6.2 Segway-Programm

6.2.1 Grundsätzliche Funktion

Solange niemand auf dem TivSeg™ steht, wirkt es für den Fahrer inaktiv; es werden nur Hintergrundaufgaben ausgeführt. Steigt eine Person auf das TivSeg™ und drückt dabei den Fußschalter, so beginnt die Regelung des TivSegs™ mit der Stabilisierung. Steigt der Fahrer ab, oder fällt runter, stoppt das TivSeg™.

6.2.2 Programmablauf

6.2.2.1 Hauptprogramm

Die `main()` Funktion innerhalb der Segway_Test/main.cpp initialisiert das System und das TivSeg™. Anschließend führt es die Hintergrundaufgaben in der Dauerschleife aus (Abbildung 26). Die Interrupt Service Routinen der beiden Timer innerhalb der main.cpp werden in Abbildung 27 dargestellt.

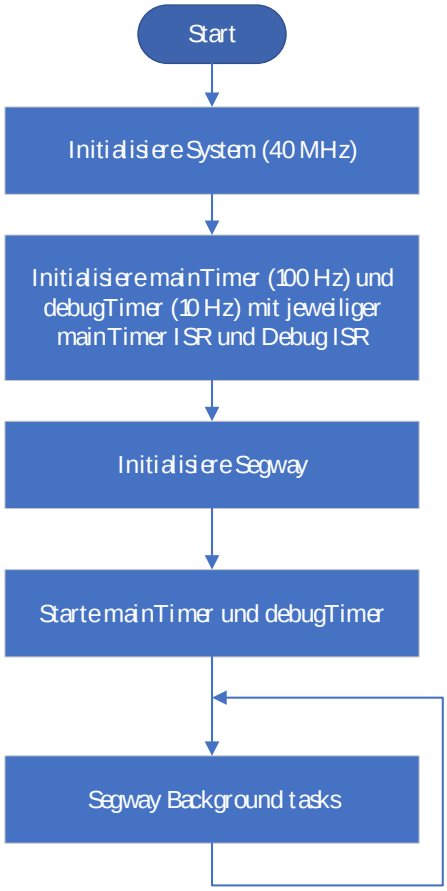

Abbildung 26: Flussdiagramm der `main()` des Segway Hauptprogrammes

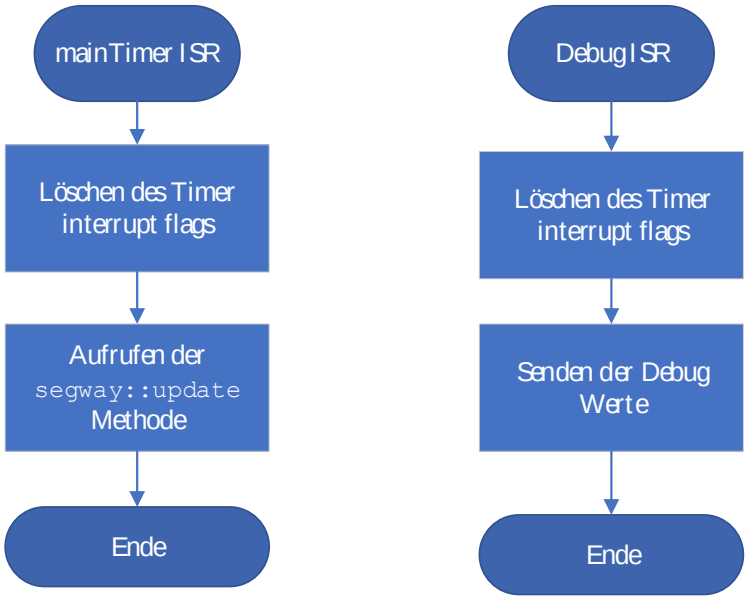

Abbildung 28 Flussdiagramm der Interrupt Service Routinen

6.2.2.2 `Segway::update`

Diese Methode steuert das Verhalten des TivSeg™. Sie liest die Sensordaten ein und aktualisiert die Motorsteuerung. Da diese Methode als periodischer Interrupt durchgeführt wird, enthält sie nur Aufgaben, die zeitkritisch sind. Eine Interrupt Service Routine (ISR) enthält nur so viel Code wie nötig (Abbildung 27).

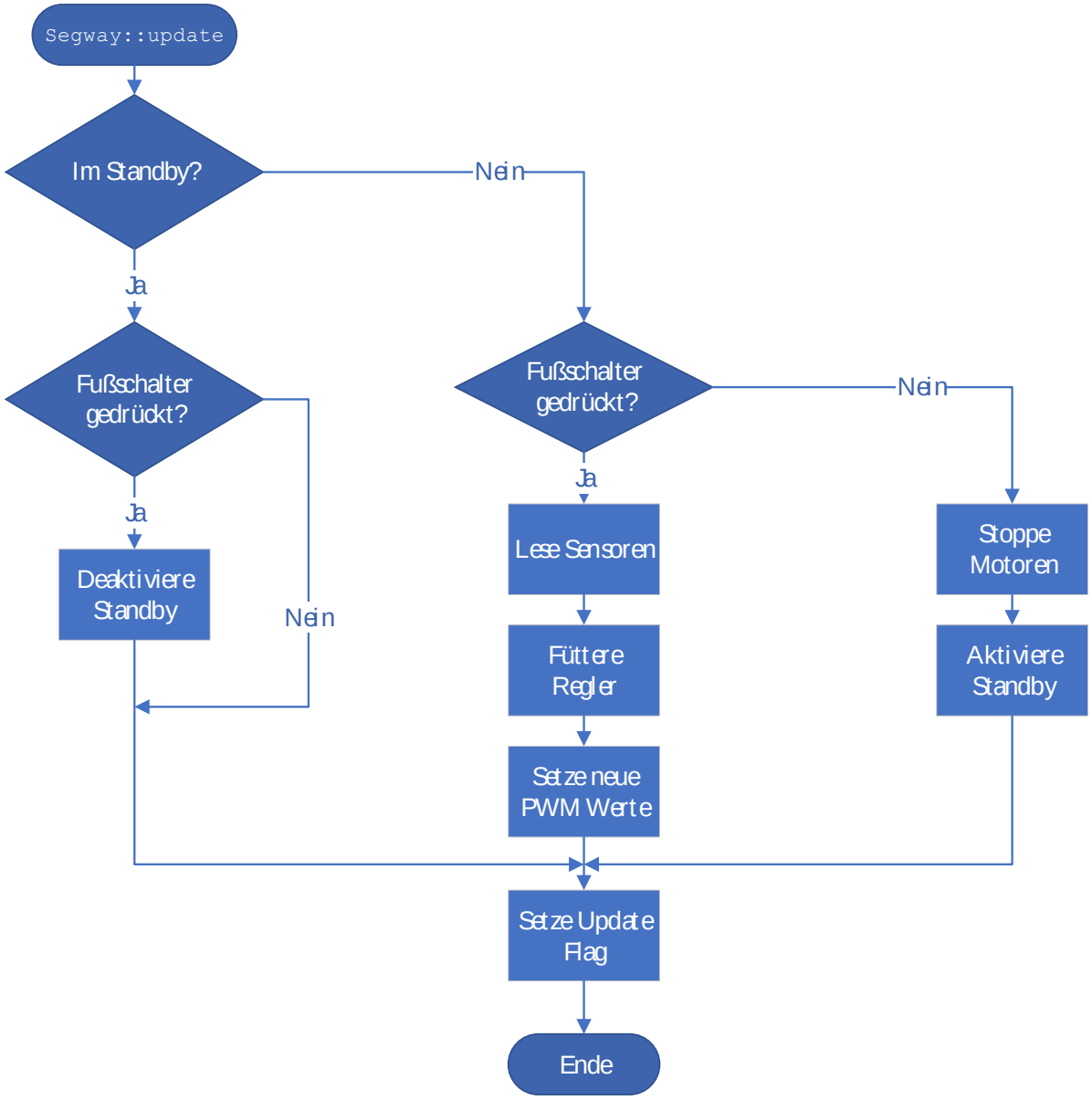

Abbildung 27: Flussdiagramm der `Segway::update` Methode

6.2.2.3 `Segway::backgroundTasks`

Neben der Steuerung des TivSegs™ müssen auch noch Aufgaben im Hintergrund durchgeführt werden. Diese Methode unterteilt sich in zwei Bereiche: der erste Bereich wird nur abgerufen, wenn davor die `Segway::update` Methode durchlaufen wurde. Der zweite Bereich wird bei jedem Durchlauf ausgeführt (Abbildung 28). In Ihrer GitLab Vorlage sind beide Bereiche leer, deswegen sind sie im nachfolgenden Diagramm mit „ToDo“ gekennzeichnet. Diese Methode bietet Ihnen nur den nötigen Rahmen für die Umsetzung Ihrer Aufgaben.

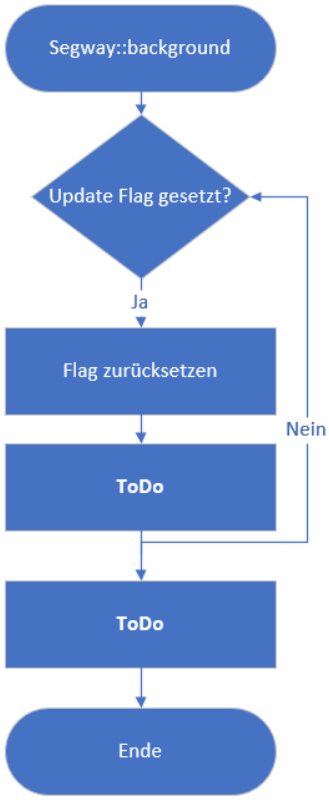

Abbildung 28: Flussdiagramm der `Segway::backgroundTasks` Methode

6.2.3 Konfiguration

6.2.3.1 Zweck

Die gesamten Einstellungen, wie z.B. das Pinout oder die PWM-Frequenz, liegen in der Datei `Config.h`. Sie wird in das Hauptprogramm, in die Controller-Klasse und in die Segway-Klasse eingebunden. Falls Sie Ihr MiniSeg™ wie im Schaltplan vorgeschlagen aufbauen, dann können Sie diese Datei auch in Ihre Testprogramme einbinden. In den Grundlegende Klassen darf sie allerdings nicht eingebunden werden!

6.2.3.2 Spezifische Einstellungen für TivSeg™ und MiniSeg™

TivSeg™ und MiniSeg™ nutzen bekanntlich den gleichen Programmcode und einen fast identischen Aufbau. Trotzdem sind beide Geräte in ihren Eigenschaften sehr unterschiedlich. Während dies für die Hardware Klassen keinen Unterschied macht, so ist es beispielsweise für den Regler von großer Bedeutung, ob er gerade ein MiniSeg™ oder ein TivSeg™ samt Fahrer ausbalancieren muss.

Um diese Unterschiede zu berücksichtigen, müssen einige Konfigurationsgrößen angepasst werden. In der `Config.h` sind die passenden Größen sowohl fürs TivSeg™ also auch fürs MiniSeg™ hinterlegt. Welche Werte genutzt werden, können Sie über die zwei `#define` zu Beginn der Datei festlegen. Standardmäßig ist die `Config.h` auf das MiniSeg™ ausgelegt, was nur für die Probefahrt am Praktikumsende geändert werden muss.

6.2.3.3 Arbeiten mit der `Config.h`

Wenn Sie beim Arbeiten am Segway-Programm Konstanten und Einstellungen benötigen, prüfen Sie zuerst, 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.

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 sich 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 sie selbst programmieren und mit der Musterlösung abgleichen können. Dies bereitet Sie optimal auf die restlichen Aufgaben vor. Die MPU6050- und die Controller-Klassen brauchen Sie jedoch für die erfolgreiche Bearbeitung des Praktikums nicht zu verstehen.

6.3.2 GPIO-Klasse

6.3.2.1 Zweck

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.

6.3.2.2 `GPIO::init`

- Rückgabewert/-typ
  `void`
- Übergabeparameter

`System *sys`:	Zeiger auf die laufende Instanz der System-Klasse
`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 boolesche 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_2MA`
- `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.

6.3.2.3 `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
  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` (oder `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 wird.
- 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 wird.
- 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 onbord-LED auf dem LaunchPad um, wenn der Taster SW1 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 System-Klasse

6.3.3.1 Einleitung

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.

6.3.3.2 `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 gewünschte Frequenz konfiguriert wird. Zusätzlich werden der PWM Unit Clock Divisor gesetzt, alle gesperrten GPIO-Pins entsperrt, die Debugging-Schnittstelle konfiguriert und die Interrupts freigeschaltet.

6.3.3.3 `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
  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 das TivSeg™ abschaltet. Landen Sie nämlich in der `FaultISR`, werden Sie merken, dass die Motoren einfach weiterdrehen.

  Sie könnten weitere Funktionen hinzufügen (z.B. Blinken der onboard-LED). Beachten Sie nur, dass Sie wegen eines Fehlers hier gelandet sind, und eventuell nicht mehr alles zu 100% funktioniert. Halten Sie daher den Code innerhalb der Methode so knapp wie möglich.

  Siehe auch Verwenden der `System::error`-Methode.

6.3.3.4 `System::getClockFreq`

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

6.3.3.5 `System::enableFPU`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `void`
- Funktion
  Aktiviert die Gleitkommaeinheit des Mikrocontrollers. Andernfalls muss die Rechnung mit Gleitkommazahlen auf den Ganzzahl-Rechenwerken der CPU durchgeführt werden, was erheblich mehr Zeit in Anspruch nimmt.

6.3.3.6 `System::setPWMClockDiv`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `uint32_t`: Wert für den PWM Unit Clock Divisors. Eine Zweierpotenz von 1 bis 64.
- Funktion
  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.

6.3.3.7 `System::getPWMClockDiv`

- Rückgabewert/-typ
  `uint32_t`: Wert des PWM Unit Clock Divisors. Eine Zweierpotenz von 1 bis 64.
- Übergabeparameter
  `void`
- Funktion
  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 aktuelle Teiler kann mit dieser Funktion ausgelesen werden.

6.3.3.8 `System::delayCycles`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `uint32_t cycles`: Gewünschte Verzögerung in Taktzyklen
- Funktion
  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.

6.3.3.9 `System::delayUS`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `uint32_t us`: Gewünschte Verzögerung in µs
- Funktion
  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.

6.3.3.10 `System::setDebugging`

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

6.3.3.11 `System::setDebugVal`

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

  `int_32_t value`: Wert als Integer, der im Arduino Plotter angezeigt werden soll. Bei Fließkommavariablen empfiehlt es sich u.U. sie mit einem Faktor 100 zu multiplizieren, um die gewünschte Auflösung zu erreichen.
- Funktion
  Ermöglicht das Speichern von Werten zum Übermitteln mit `System::sendDebugVals`. Beachten Sie bitte, dass momentan nur acht Werte parallel übermittelt werden können, da der Aduino Plotter nur acht verschiedene Farben zur Verfügung stellt. Um mehr Variablen zu übermitteln, erhöhen sie einfach die Variable `System::maxDebugVals`. Detailliertere Erklärungen inkl. Verwendungsbeispiel sind im Quellcode der Methode gegeben.

6.3.3.12 `System::sendDebugVals`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `void`
- Funktion
  Übermittelt bis zu acht gespeicherte Werte mithilfe der seriellen Schnittstelle über USB an einen Computer. Die Werte können mit `System::sendDebugVals` 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 10 Hz-Timer-Interrupt. In dem gegebenen Quellcode des Segway-Programms ist dies bereits der Fall.

6.3.4 MPU6050

6.3.4.1 Einleitung

Die MPU6050-Klasse bietet eine einfache Handhabung des gleichnamigen 6-Achsen Beschleunigungs- und Gyrosensors. Sie ist auf die Verwendung in einem TivSeg™ zugeschnitten, auf dem nicht alle 6 Messwerte benötigt werden. Die Klasse bietet Methoden, um die Einbaulage zu konfigurieren. Anschließend kann auf die horizontale und die vertikale Beschleunigung, sowie auf die Winkelgeschwindigkeit entlang der Radachse zugegriffen werden.

Der Sensor kommuniziert über ein Protokoll namens I²C. Es benötigt praktischerweise nur 2 Pins. Jedes darüber angesteuerte Gerät hat eine Adresse, mit der es angesprochen wird. Manche bieten die Möglichkeit, diese Adresse zu ändern, so dass mehrere Geräte zusammen genutzt werden können. Beim MPU6050 gibt es einen Pin, namens `AD0`, mit dem zwischen zwei Adressen gewechselt werden kann.

6.3.4.2 `MPU6050::init`

- Rückgabewert/-typ
  `void`
- Übergabeparameter

`System *sys`:	Zeiger auf die laufende Instanz der System-Klasse
`uint32_t I2CBase`:	Basisadresse des verwendeten I²C-Moduls
`bool` `addressBit`:	Gibt an, ob der Adresse-Pin `AD0` high oder low ist.
`char` `wheelAxis`:	Optionale Angabe der Sensorachse, die parallel zur Radachse ist. Kann `'x'`, `'y'`, oder `'z'` sein.
`char` `horAxis`:	Optionale Angabe der horizontalen Achse. Kann `'x'`, `'y'`, oder `'z'` sein.

- Funktion
  Initialisiert den I²C-Bus sowie den Sensor selbst. Optional wird auch die Ausrichtung des Sensors festgelegt. Dazu reicht die Angabe von zwei Achsen. Die dritte Achse wird per Ausschlussverfahren bestimmt.
  Falls die Kommunikation mit dem Sensor fehlschlägt, so springt die Methode in die `System::error` mit dem Fehlerparameter `MPUCommunicationError` (Siehe auch: Häufige Fehler beim MiniSeg™).

6.3.4.3 `MPU6050::setWheelAxis`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `char` `wheelAxis`: Sensorachse, die parallel zur Radachse ist. Kann `'x'`, `'y'`, oder `'z'` sein.
- Funktion
  Konfiguriert die Ausrichtung des Sensors. Muss vor `MPU6050::setHorAxis` aufgerufen werden.

6.3.4.4 `MPU6050::setHorAxis`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `char` `horAxis`: Horizontale Achse. Kann `'x'`, `'y'`, oder `'z'` sein.
- Funktion
  Konfiguriert die Ausrichtung des Sensors. Muss nach `MPU6050::setWheelAxis` aufgerufen werden, und mit einer anderen Achse als Übergabeparameter.

6.3.4.5 `MPU6050::angleRateInvertSign`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `bool` `invertSign`: Legt fest, ob das Vorzeichen der Winkelgeschwindigkeit umgekehrt werden soll.
- Funktion
  Je nach Ausrichtung des Sensors muss das Vorzeichen der Messwerte umgekehrt werden.

6.3.4.6 `MPU6050::accelHorInvertSign`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `bool` `invertSign`: Legt fest, ob das Vorzeichen der horizontalen Beschleunigung umgekehrt werden soll.
- Funktion
  Je nach Ausrichtung des Sensors muss das Vorzeichen der Messwerte umgekehrt werden.

6.3.4.7 `MPU6050::accelVerInvertSign`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `bool` `invertSign`: Legt fest, ob das Vorzeichen der vertikalen Beschleunigung umgekehrt werden soll.
- Funktion
  Je nach Ausrichtung des Sensors muss das Vorzeichen der Messwerte umgekehrt werden.

6.3.4.8 `MPU6050::getAngleRate`

- Rückgabewert/-typ
  `float`: Winkelgeschwindigkeit in °/s
- Übergabeparameter
  `void`
- Funktion
  Liest die aktuelle Winkelgeschwindigkeit aus dem Sensor aus und gibt sie in Grad pro Sekunde zurück.

6.3.4.9 `MPU6050::getAccelHor`

- Rückgabewert/-typ
  `float`: Horizontale Beschleunigung in g
- Übergabeparameter
  `void`
- Funktion
  Liest die aktuelle horizontale Beschleunigung aus dem Sensor aus und gibt sie in g zurück.

6.3.4.10 `MPU6050::getAccelVer`

- Rückgabewert/-typ
  `float`: Vertikale Beschleunigung in g
- Übergabeparameter
  `void`
- Funktion
  Liest die aktuelle vertikale Beschleunigung aus dem Sensor aus und gibt sie in g zurück.

6.3.5 Controller-Klasse

6.3.5.1 Einleitung

Die Controller-Klasse enthält den Regelalgorithmus, der das Segway stabil hält und lenkt.

6.3.5.2 `Controller::init`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `System *sys`: Zeiger auf die laufende Instanz der System-Klasse
  `float` `maxSpeed`: Erlaubte Höchstgeschwindig als Tastgrad zwischen `-1.0f` und `1.0f`. I.d.R. in der Konfiguration definiert.
- Funktion
  Initialisiert den Regler und setzt die Höchstgeschwindigkeit. Überschreitet der Fahrer diese, bremst der Regler den Fahrer ab. Diese Grenze kann später noch mit `Controller::setMaxSpeed` geändert werden.

6.3.5.3 `Controller::resetSpeeds`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `void`
- Funktion
  Setzt alle Geschwindigkeitsvariablen der Regler-Klasse auf Null.

6.3.5.4 `Controller::updateValuesRad`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `float` `steeringValue`: Wert zwischen `-1.0f` und `1.0f` entsprechend einer vollen Lenkung nach links oder rechts. `0.0f` entspricht einer Geradeausfahrt.
  `float` `angleRateRad`: Winkelgeschwindigkeit in rad/s.
  `float` `accelHor`: Horizontale Beschleunigung in g
  `float` `accelVer`: Vertikale Beschleunigung in g
- Funktion
  Berechnet die korrekten PWM-Werte für die beiden Motoren entsprechend den übergebenen Lenk- und Lageinformationen.

6.3.5.5 `Controller::getLeftSpeed`

- Rückgabewert/-typ
  `float`: Tastgrad für den linken Motor als Wert zwischen `-1.0f` und `1.0f`, wobei negative Werte für Rückwärts stehen.
- Übergabeparameter
  `void`

6.3.5.6 `Controller::getRightSpeed`

- Rückgabewert/-typ
  `float`: Tastgrad für den rechten Motor als Wert zwischen `-1.0f` und `1.0f`, wobei negative Werte für Rückwärts stehen.
- Übergabeparameter
  `void`

6.3.5.7 `Controller::getMaxSpeed`

- Rückgabewert/-typ
  `float`: Aktuelle Höchstgeschwindigkeit als Tastgrad zwischen `-1.0f` und `1.0f`.
- Übergabeparameter
  `void`

6.3.5.8 `Controller::setMaxSpeed`

- Rückgabewert/-typ
  `void`
- Übergabeparameter
  `float`: Neue Höchstgeschwindigkeit als Tastgrad zwischen `-1.0f` und `1.0f`.

6.4 Aufbau des MiniSeg™

Alle Bauteile für den Aufbau eines eigenen MiniSeg™ sollten Sie bereits erhalten haben. Tabelle 1 listet die wesentlichen Bauteile auf. Zusätzlich zu diesen Bauteilen benötigen Sie zudem eine Steckplatine und passende Kabel. Einen möglichen Aufbau des MiniSeg™ finden Sie in Abbildung 29. Das weiße Kabel an Pin PB5 stellt den Fußschalter dar. Wie im Schaltplan (Abbildung 30) zu sehen ist, ist es ein Öffner, d.h. der Pin wird von der Masse getrennt, wenn der Schalter gedrückt wird. Demnach ist in Abbildung 29 der Fußschalter gedrückt.

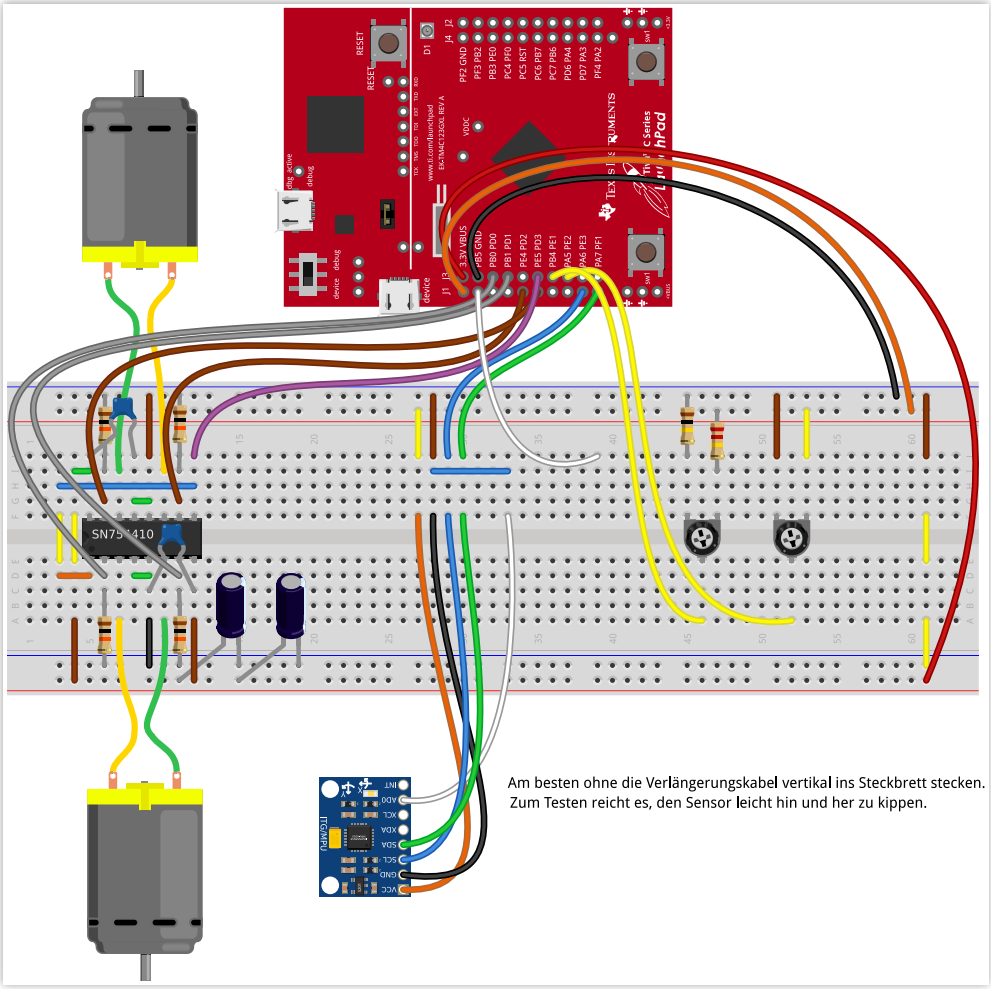

Abbildung 29: Aufbau des MiniSeg™

Ein Schaltplan des MiniSeg™ ist in Abbildung 30 dargestellt. Halten Sie sich an diesen Schaltplan, so sollte Ihrem Aufbau eines eigenen MiniSeg™ nichts im Weg stehen. Den Schaltplan und auch den Aufbau des MiniSeg™ finden Sie auch im Dokumente und Infos Repository.

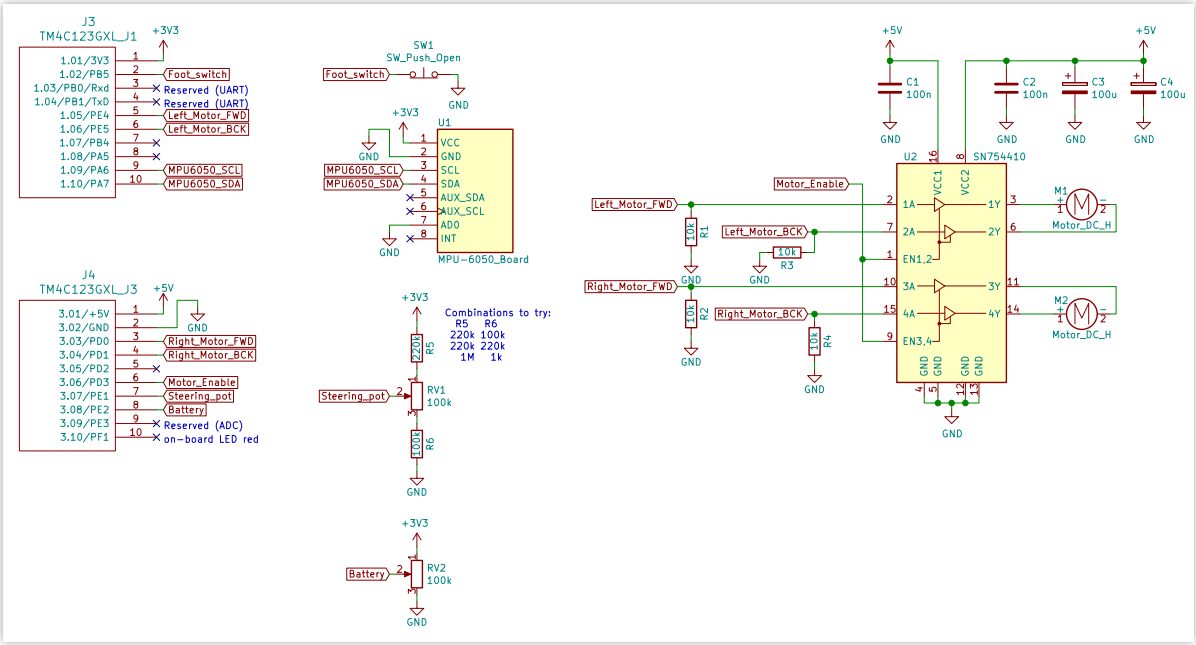

Abbildung 30: Schaltplan MiniSeg™

Anzahl	Bezeichnung	Wert
2 Stück	Elektrolytkondensator	100 µF
2 Stück	Keramikkondensatoren	100 nF
2 Stück	Gleichstrommotoren	3V, max. 1,8 W Motor
1 Stück	Widerstand	1 kΩ
4 Stück	Widerstand	10 kΩ
1 Stück	Widerstand	100 kΩ
2 Stück	Widerstand	220 kΩ
1 Stück	Widerstand	1 MΩ
2 Stück	Potentiometer	100 kΩ Einstellpotentiometer liegend
1 Stück	Bewegungssensor	MPU 6050 Board, 6-Achsen-Bewegungssensor
1 Stück	Halbbrücke	SN754410 4-fach Halbbrücke

Tabelle 1: Bauteile für den Aufbau des MiniSeg

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 zu 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 Mikrocontrollers 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 es 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 an 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 eher 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 Webseite 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 Webseite http://www.learncpp.com/ ausführliche 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
- 9: Arrays, Strings, Pointers, and References
- 11: Basic object-oriented programming

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