Projektbericht

Entwicklung einer graphischen Benutzeroberfläche & Smartphone-Applikation zur Auswertung von Wegmesssensoren

Alexander Riss, Benjamin Brachmann, Florian Fechner, Marcel Wächter, Maximilan Kohler, Tobias Oberbeck

Hochschule Osnabrück
Fakultät Ingenieurwissenschaften und Informatik
Barbarastr. 16, D-49076 Osnabrück

alexander.riss@hs-osnabrueck.de
benjamin.brachmann@hs-osnabrueck.de
florian.fechner@hs-osnabrueck.de
marcel.waechter.de@hs-osnabrueck.de
maximilian.kohler@hs-osnabrueck.de
tobias.oberbeck@hs-osnabrueck.de

Zusammenfassung:
Der folgende Projektbericht beschreibt den Entwicklungsprozess einer Desktop-Anwendung für das Windows-Betriebssystem, wie auch einer Smartphone-Webanwendung zur Fernsteuerung der Desktop-Anwendung.
Eingegangen wird auf das Ziel, die Planung und die Umsetzung, wie auch auf bewältigte Herausforderungen und Hindernisse während der Projektphase. Des Weiteren wird auf die Qualitätskontrolle der entstandenen Software eingegangen.
Auch werden die verwendeten Technologien, Frameworks und genutzte Werkzeuge beschrieben.
Umgesetzte Softwarelösungen werden mit Diagrammen und Screenshots der entstandenen Anwendungen dargestellt.
Zum Schluss folgt ein Fazit des gesamten Software Engineering Teams zur Projekt- und Entwicklungsphase und zur Zusammenarbeit im Team.

Inhaltsverzeichnis

1. Einleitung

1.1 Ziel

Ziel des Projektes „Entwicklung einer graphischen Benutzeroberfläche und einer Smartphone-Applikation zur Auswertung von Wegmesssensoren“ im Rahmen des Moduls „Software Engineering Projekt“ war die Planung, Entwicklung und Implementierung einer neuen graphischen Desktop-Anwendung für Windows, um eine bereits bestehende Anwendung abzulösen.
Die neue Software dient zur Minimierung der Einarbeitungszeit für einen Praktikumsversuch der Studenten im Bereich der Handhabungstechnik und Robotik. Des Weiteren soll der Mehraufwand durch die gleichzeitige Nutzung von zwei Rechnern verhindert werden, wie es zum Beginn des Projektes der Fall war.
Laufwege zwischen Robotern und dem Computer zum Bedienen der Software, sollen ebenfalls mit Hilfe einer zu entwickelnden Smartphone-Anwendung minimiert werden. Diese Anwendung soll die Möglichkeit bieten, die oben erwähnte Software aus der Ferne begrenzt zu steuern.
Somit können die Studenten den Fokus im Praktikum auf die Industrieroboter verlagern.

1.2 Hintergrundinformationen zum Projekt

Im Labor für Handhabungstechnik und Robotik werden in studentischen Praktikums-Versuchen Laser-Wegmessgeräte zur Genauigkeits- und Bewegungsuntersuchung an Industrierobotern benutzt. Die Laser des Herstellers „Keyence“ werden standardmäßig mit der mitgelieferten Software „LK-Navigator“ bedient. Da es sich hierbei um professionelle Software handelt, bietet diese sehr viele Funktionalitäten, die im Praktikumsablauf nie benötigt werden. Dies sorgt für eine lange Einarbeitungszeit und für eine erschwerte Bedienung der Software für Laien.

Während den Praktika, stellte sich schnell heraus, dass die Konfiguration und die Bedienung der Software viel Zeit in Anspruch nahm, welche die Studierenden von den Hauptaufgaben ablenkte.
Ein weiteres Problem war die Nutzung von drei Lasern gleichzeitig. Mit der beigelieferten Software ist die Nutzung von nur zwei Lasern standardmäßig vorgesehen. Auch die Auswerteeinheit der Lasersensoren bietet nur zwei Steckplätze für die jeweiligen Laser an. Im Versuch werden jedoch drei benötigt, ein Laser pro Achse im Raumkoordinatensystem. Somit arbeiten die Studierenden parallel an zwei Rechnern, auf denen jeweils die LK-Navigator Software läuft und eine Auswerteeinheit mit maximal zwei Messgeräten ausliest.

1.3 Aufgabenstellung

Vor der eigentlichen Implementierung der beiden Anwendungen, wurde eine Anforderungsanalyse durchgeführt, welche nach Abschluss ein Pflichtenheft als Produkt lieferte.
Anhand dieser Planung wurde das gesamte Projekt in folgende Arbeitspakete aufgeteilt, um das parallele Arbeiten im Team zu gewährleisten:

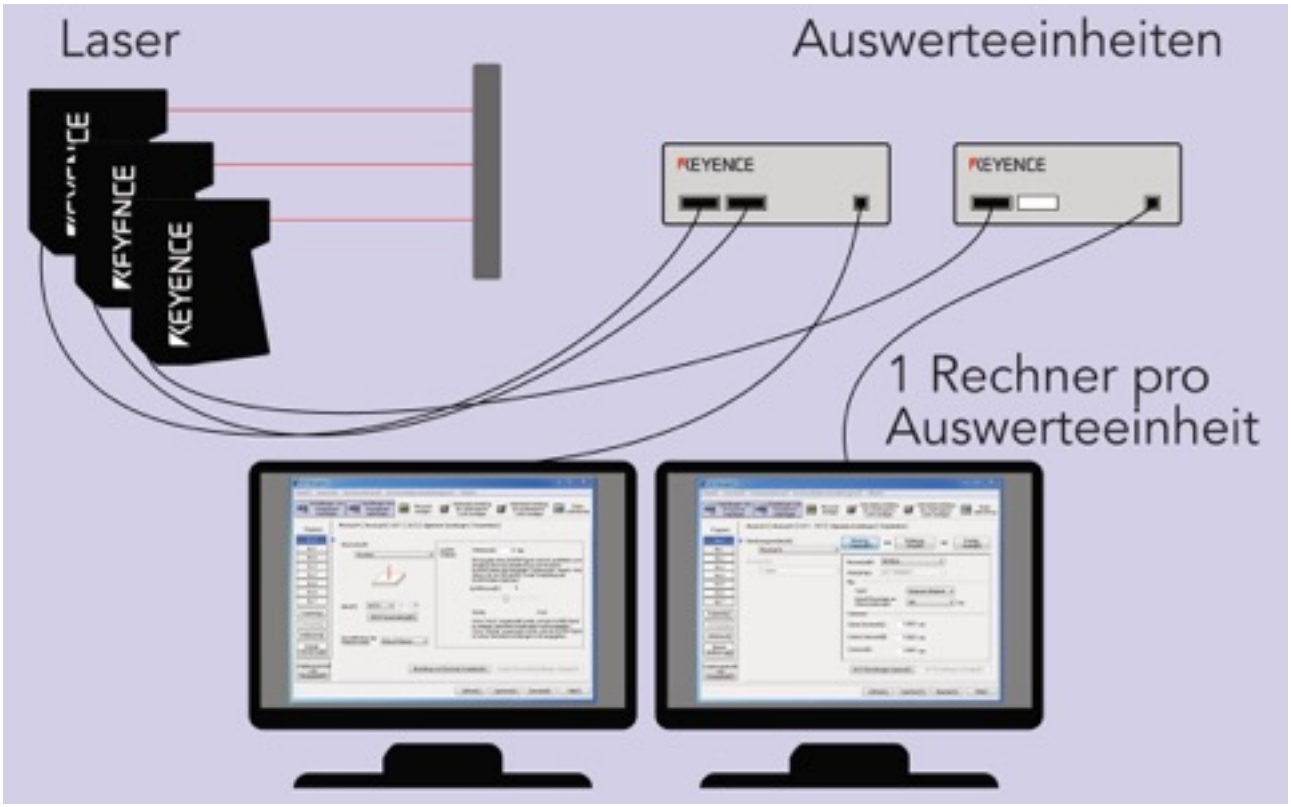
- Entwicklung der grafischen Oberfläche
- Businesslogik
- Kommunikation mit der Hardware

- Smartphone-Anwendung
- Netzwerkkommunikation

Die Aufgaben bestanden darin, eine Desktop-Anwendung zur Auswertung der Keyence Wegmesssensoren zu entwickeln. Wichtige Anforderungen sind die leichte und sich selbsterklärende Bedienung und die Beschränkung auf die nur für den Praktikumsversuch relevanten Funktionalitäten.

Messvorgänge in Form von Weg-Zeit- und Genauigkeitsmessungen sollen durchgeführt und direkt im Programm kontrolliert werden können. Um dies durchführen zu können, ist das Auslesen der Sensoren und das simultane Verarbeiten der erhaltenen Daten der Sensoren nötig.
Es sollen die Optionen bestehen, ausgewählte Messvorgänge nach Excel zu exportieren oder komplett zu verwerfen.
Des Weiteren soll mit Hilfe einer Smartphone Anwendung die Fernsteuerung der Hauptanwendung möglich sein, um direkte Anwesenheit am Rechner zu vermeiden. Hierfür wird eine Verbindung zwischen Smartphone und Desktop-Anwendung benötigt. Die Smartphone Applikation soll zwei Bedieneroptionen zur Verfügung stellen:
Einen einmaligen Auslöser, welcher den Messvorgang durchführt und für jede Messung verantwortlich ist und einen Zuschauer-Modus, in welchem sich alle anderen verbundenen Teilnehmer befinden und den Messvorgang beobachten und kontrollieren können.

Die Umsetzung der genannten Arbeitspakete erfolgte in Kleinteams, welche je nach Präferenz und Fähigkeiten der Mitglieder des Software Engineering Teams entstanden sind.
In den dadurch entstandenen Kleinteams begann die Planung, die Entwicklung und die Kontrolle der Software zu den jeweiligen Arbeitsgebieten.

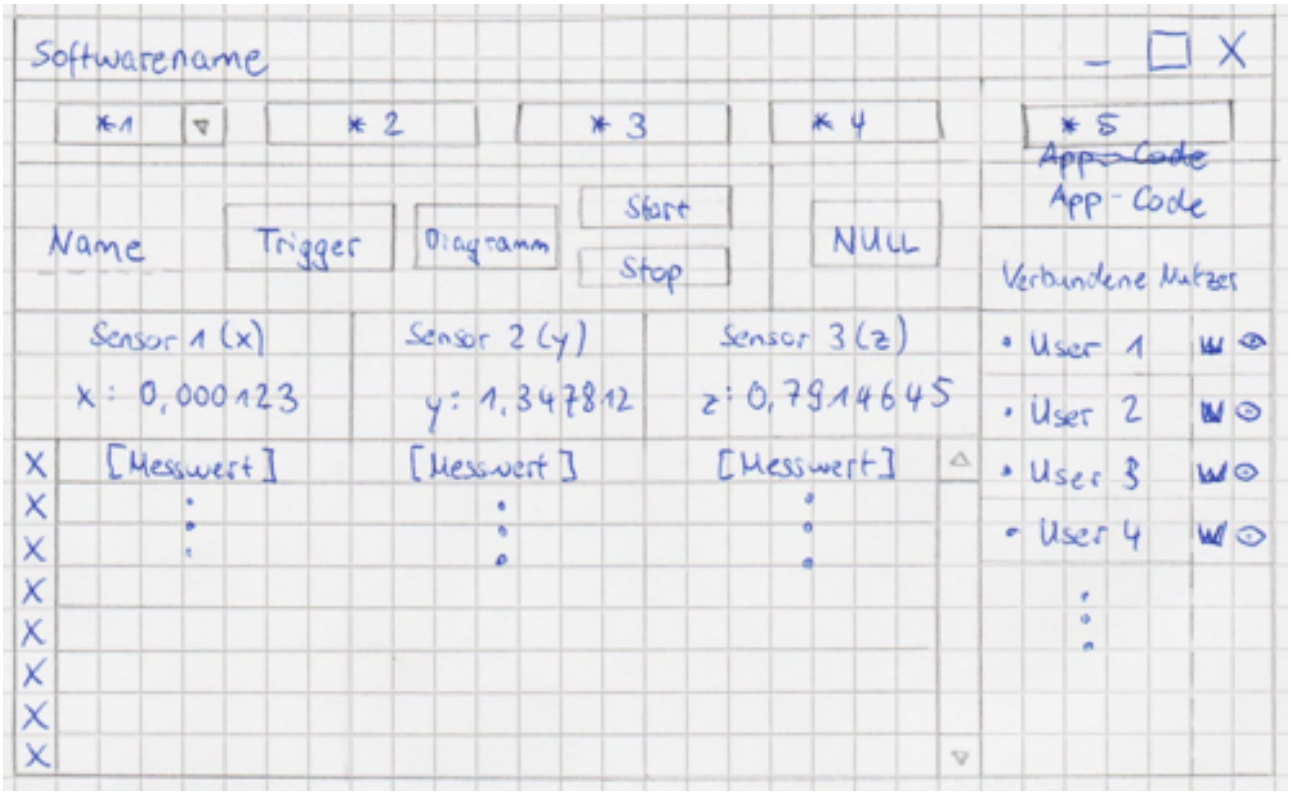


Aufbau des Praktikumversuches mit der bestehenden Keyence Software

2. Konzept

2.1 Potenzielle Lösungsansätze

Benutzeroberfläche
Für die grafische Benutzeroberfläche (GUI) war zunächst die zu verwendende Technologie auszuwählen. Dabei wurden zahlreiche Ansätze betrachtet und auf ihre Eignung untersucht. Ein erstes, skizziertes Mockup lieferte Anhaltspunkte dafür, welche Funktionen und Elemente vorhanden sein müssen. Zudem war der Gestaltungsspielraum im Design ein wichtiger Punkt, da das Design mit unseren Kernkompetenzen übereinstimmt.



Erstes MockUp zur Benutzeroberfläche

Die Aufgabenstellung sah eine Entwicklung in den Programmiersprachen C/C++ bzw. C# vor – Java Bibliotheken wie Swing wurden also nicht berücksichtigt. Für die Umsetzung in C++, die zu Beginn bevorzugt wurde, wurden mehrere Frameworks betrachtet. Dabei wurden unterschiedlich gewichtete Faktoren unter die Lupe genommen. Erstens musste die Verfügbarkeit festgestellt werden, denn viele Bibliotheken sind nicht oder nur eingeschränkt kostenlos zu benutzen. Als prominentes Beispiel ist hier QT zu nennen, welches die umfangreichste Bibliothek in C++ für Benutzeroberflächen ist. QT schien für unsere Zwecke zwar sehr geeignet, allerdings war eine Lizenz nur mit eingeschränkter Funktionalität zu erhalten, was dazu führte, dass wir andere Möglichkeiten vorzogen. Ein weiterer Punkt war die Umsetzbarkeit der GUI mit der entsprechenden Bibliothek. Ein Blick auf die Funktionalitäten und ein Abgleich mit den technischen Anforderungen unserer Software wurden durchgeführt, um die Eignung festzustellen. Nach dieser Analyse standen noch zwei C++-Bibliotheken in der engeren Auswahl: Windows Forms und wxWidgets. Diese Bibliotheken sind frei erhältlich und daher auch im Umfang und ihren Gestaltungsmöglichkeiten QT unterlegen.

Mit wxWidgets wurde nun als Erstes ein Prototyp ohne jegliche Funktionalität erstellt, um die Möglichkeiten in Gestaltung und Funktion zu testen.

Hardware-Kommunikation
Für die Kommunikation mit den Lasermesssensoren wurde beabsichtigt eine Verbindung mittels USB-Kabel vorzunehmen, da auch im Praktikum ein USB-Kabel benutzt wird. Mit libusb wurde sich eine einfache und schnelle Implementierung der Messbefehle erhofft. Die Alternative wäre die Verwendung einer seriellen RS-232-Schnittstelle. Hierfür hätte man jedoch ein spezielles Kabel verlöten müssen und auch die Verwendung der Messbefehle mittels einzelner Byte-Codes schien komplizierter und damit zeitintensiver umzusetzen. Daher entschied man sich ein USB-Kabel mit der C++ Bibliothek libusb zu nutzen.

Smartphone-Applikation
Eine der wichtigsten zu entwickelnden Komponenten ist die gewünschte Fernsteuerung der Desktop-Applikation. Die Komponente soll als mobile Applikation auf Geräten wie Tablets, Smartphones und falls möglich auch auf Notebooks und anderen Rechnern laufen und es den Bedienern des Systems ermöglichen Funktionen der Desktop-Applikation aufzurufen. Es soll möglich sein Messungen und Messreihen anzulegen und diese zu Verwalten, Messungen durchzuführen, Ergebnisse der Messungen und ganze Messreihen einzusehen und das System zu Kalibrieren.

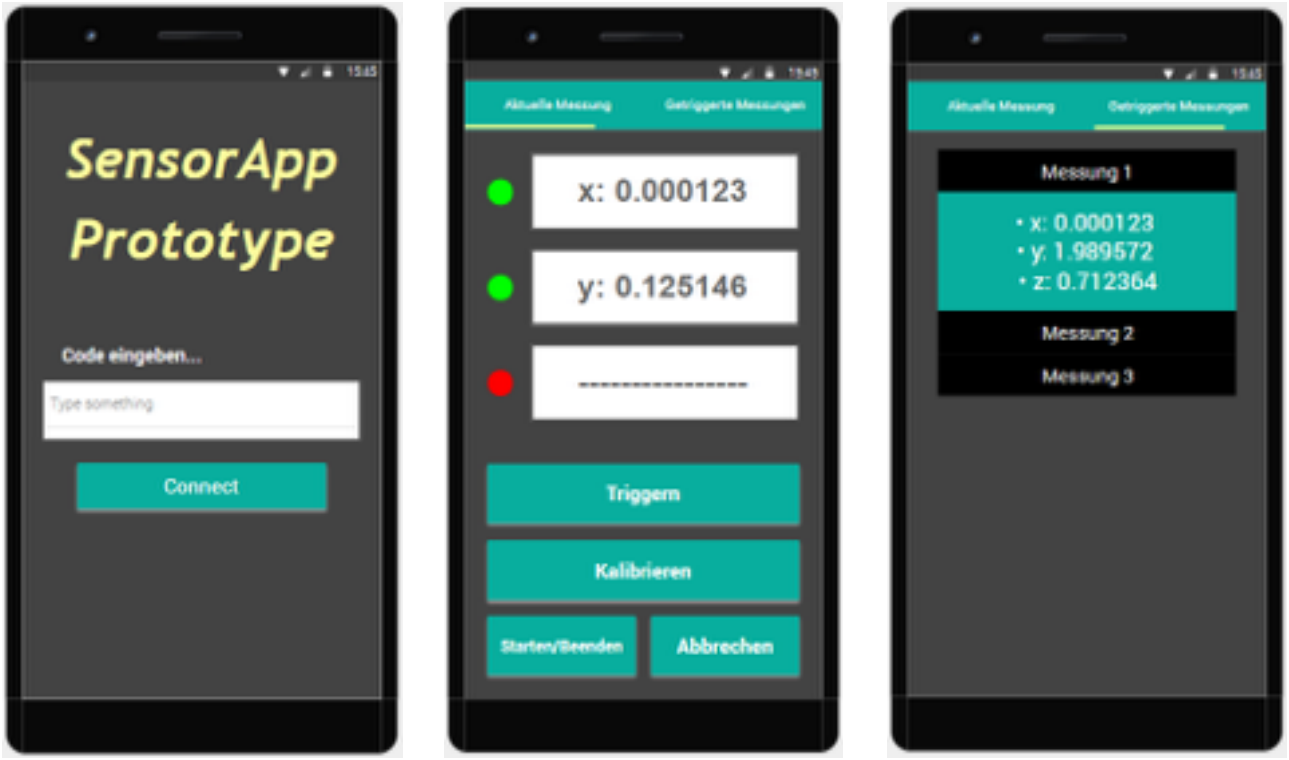
Um die mobile Applikation möglichst universell und plattformübergreifend ausführen zu können, bedarf es eines Frameworks wie Cordova, welches Javascript Code in eine native Applikation übersetzt und somit eine einzige Entwicklung für drei Plattformen (Windows-Phone, Android, iOS) ermöglicht. Eine weitere Möglichkeit ist es, die mobile Applikation als Web-Applikation erreichbar zu machen und somit alle Plattformen zu erreichen, welche einen gängigen Webbrowser besitzen.
Da die zweite Lösung deutlich mehr Plattformen und Systeme abdeckt entschieden wir uns diese in der Entwicklungsphase umzusetzen.

Die zu entwickelnde mobile Applikation soll die Rolle des Smartphone-Bedieners unterscheiden und deutlich visuell darstellen können. Außerdem sollen dem Smartphone-Bediener, welcher die Rolle Zuschauer hat, bestimmte Funktionen, die zum Anlegen von Messreihen und dem eigentlichen Messen entnommen werden, sodass nur der Smartphone-Bediener, welcher die Rolle Auslöser hat, diese Funktionen ausführen kann.

Das Aufrufen der mobilen Applikation soll über den Webbrowser geschehen, indem in die Adresszeile des Browsers die IP-Adresse des Rechner auf dem die Desktop-Anwendung läuft eingegeben wird.
Der nächste Schritt soll das Eingeben eines Sicherheitscodes und des Benutzernamens in vorgesehene Felder sein. Der Sicherheitscode dient der Authentifizierung des Nutzers, wobei der Benutzername hingegen an die Desktop-Applikation weitergegeben wird, um diesen in der Liste mit den Benutzernamen anderer Smartphone-Bediener verwalten zu können.

Der Seitenwechsel in der mobilen Applikation soll strikt und logisch definiert sein, damit der Bediener Funktionen nur in vorgesehener Reihenfolge aufrufen kann. Diese Unterteilung in Schritte dient der Einfachheit der Benutzung und der Fehlervermeidung. Buttons und Bedienelemente sollen klar erkennbar und hervorgehoben sein, Text und Form soll sich von der Farbe des Hintergrunds deutlich abheben und erkennbar sein. Die

Farben in der mobilen Applikation sollen sich denen in der Desktop-Applikation ähneln, jedoch soll die mobile Applikation im Gegensatz zu dieser als Fernsteuerung erkennbar bleiben.



Erste Prototypen der Smartphone-Applikation

Netzwerkkommunikation

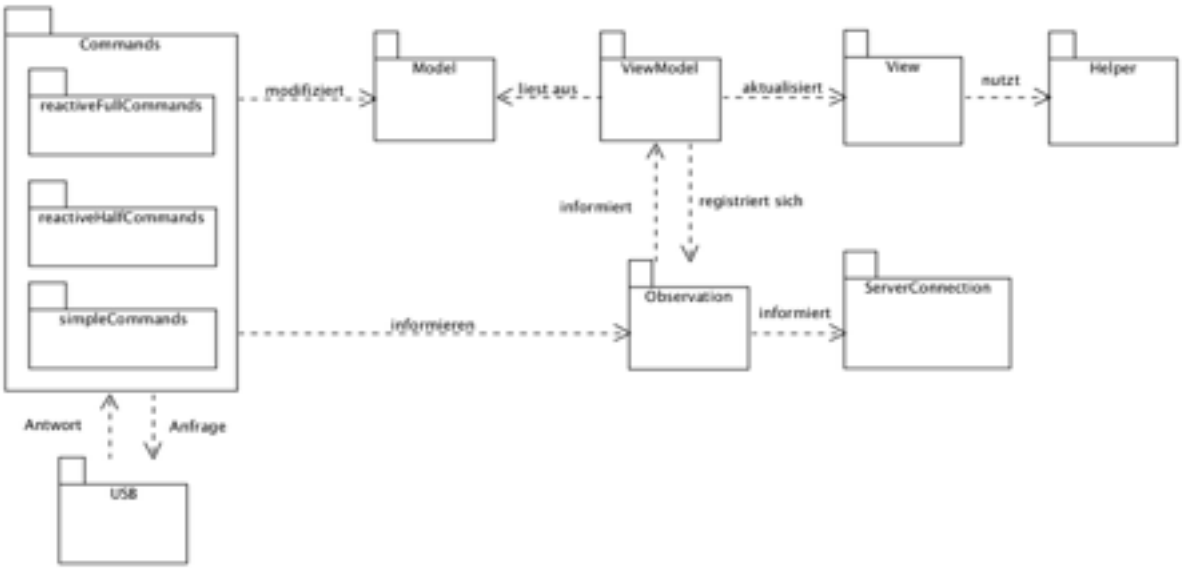
Aufgrund der Idee, dass die Desktop Applikation zur direkten Steuerung der Laser und Verwaltung der Daten zuständig sein soll, die mobile Applikation hingegen nur als eine Fernsteuerung der Desktop Applikation mit der Möglichkeit einige Messdaten einzusehen eingeplant ist, ist eine Kommunikation zwischen den beiden unabhängig ablaufenden Programmen notwendig.
Die Kommunikation muss in Echtzeit ablaufen, und es dem Bediener der mobilen Applikation ermöglichen sich frei zu bewegen. Aus diesem Grund fiel die Entscheidung eine kabellose Netzwerkkommunikation zu nutzen.

Die Art der Übertragung, muss die Übertragung über WLAN oder Bluetooth sein, da aus den Anforderungen hervorgeht, dass eine kabellose Kommunikation gewünscht ist. Außerdem unterstützen die meisten gängigen Smartphones zum Austausch von Daten die beiden genannten, kabellosen Übertragungsarten.

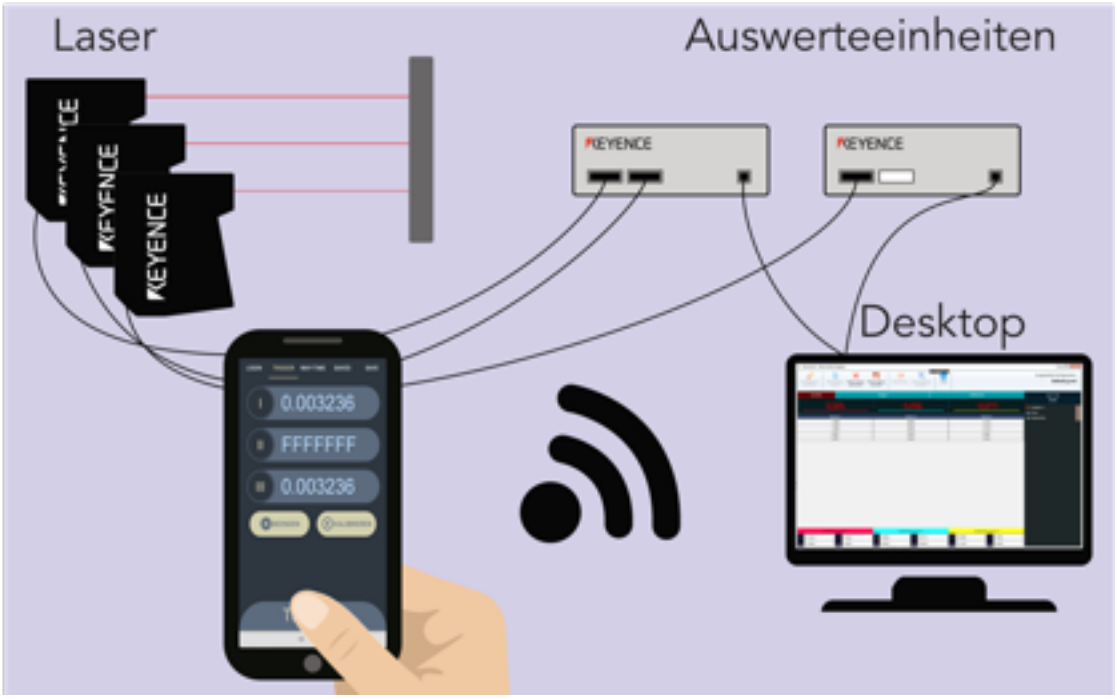
Damit eine hohe Datenübertragungsrate und Reichweite garantiert werden kann, entschieden wir uns für die Verbindung über WLAN. Im Gegensatz zu der maximalen Reichweite eines Bluetooth Moduls (10 Meter) reicht das Signal eines WLAN Moduls deutlich weiter. Die Übertragungsrate eines Bluetooth Moduls entspricht ungefähr 1Mbit/s wohingegen ein WLAN Modul bis zu 108Mbit/s (bei entsprechendem Gerät) erreichen kann. Die Kommunikation wird somit mit durch das Internet-Protokoll stattfinden. Ein weiterer Vorteil bei der Verbindung über WLAN ist, dass Web-Techniken, bekannte Frameworks und gesammelte Erfahrung bei der Entwicklung eingesetzt werden können.

Eine Aufgabe für die Umsetzung der Netzwerkkommunikation ist es nun zu entschieden ob Frameworks verwendet werden sollen und falls ja, welche Frameworks verwendet werden sollen.
Zu beachten ist, dass das zu verwendende Framework oder die zu verwendende Technik plattformübergreifend kommunizieren können muss. Außerdem muss die Technik eine Verbindung zu mehreren Clients zu selben Zeit aufbauen können und die Möglichkeit bieten bidirektional zu kommunizieren. Des Weiteren ist es wichtig herauszufinden, ob ein Server zwischen den kommunizierenden Endgeräten benötigt wird oder eine Peer-to-Peer Verbindung möglich ist.

2.3 Darstellung der der Implementierung



Package-Diagramm der Desktop-Anwendung



Geplanter Aufbau des Versuches

3. Umsetzung und Herausforderungen

3.1 Benutzeroberfläche

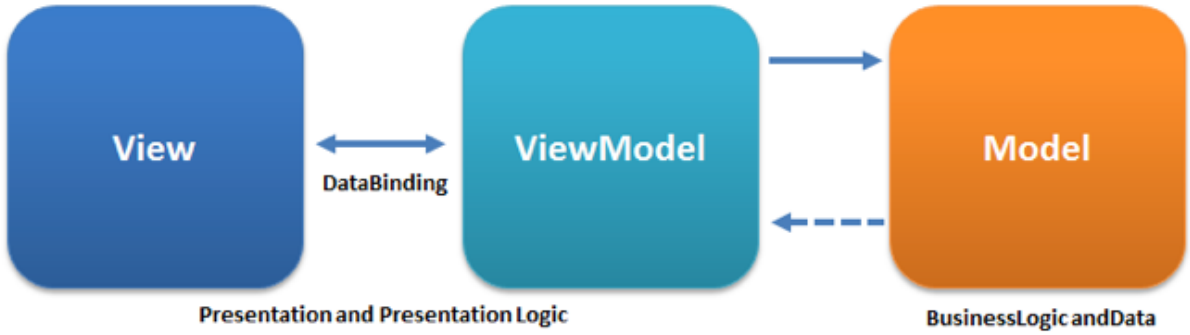
Der für die grafische Benutzeroberfläche mit wxWidgets entwickelte Prototyp wurde schnell wieder verworfen, da die Möglichkeiten der Bibliothek nicht den gesetzten Anforderungen entsprachen. Vor allem die Gestaltung war hier von Bedeutung, da sowohl Windows Forms als auch wxWidgets einen veralteten Style (Windows XP / 7) und wenige Anpassungsmöglichkeiten boten. Die Entwicklung einer eigens gestalteten Oberfläche (Design von sämtlichen Elementen in Form und Farbe) war hier nicht möglich und entsprach damit nicht den Anforderungen, die wir als Medieninformatiker stellten.

Unter Besprechung mit dem Team wurde nun auch eine Umsetzung in der Programmiersprache C#, welche wir bis dahin noch nicht benutzt hatten, in Betracht gezogen. C# bietet mit der Windows Presentation Foundation (WPF) ein Microsoft Framework, welches standardmäßig Teil des .NET-Frameworks und somit direkt in der Microsoft Entwicklungsumgebung Visual Studio verfügbar ist. Über das Dreamspark-Programm der Hochschule ist diese Software für alle Studenten frei erhältlich. Da das Ziel eine Desktop-Anwendung unter Windows war, stellte sich WPF als vielversprechende Lösung für diese Anforderungen dar.

Windows Presentation Foundation bietet freie Gestaltung in jeder Sicht. Jedes Oberflächenelement kann separat über die eigene Auszeichnungssprache XAML (basiert auf XML) angepasst und gestaltet werden. Zudem sind zahlreiche Elemente von Buttons bis Datenlisten verfügbar. Dabei wird außerdem die Präsentation in XAML von der Geschäftslogik in C# getrennt, was eine lose Kopplung fördert und eine getrennte Entwicklung der beiden Komponenten ermöglichte.

XAML ist so mächtig, dass die gesamte Oberfläche über die Auszeichnungssprache erstellt werden kann. Dabei wird die Gliederung der einzelnen Elemente vorzugsweise über ein Raster durchgeführt, was zu einem responsiven Verhalten bei Veränderung der Fenstergröße führt. Lediglich Werte, die sich zur Laufzeit ändern und die Kontrolle von Button-Click-Events wird programmatisch mit C# verwaltet.

Der primitive Ansatz dabei ist, alle Operationen in der vorgesehenen Code-Behind-Datei zu behandeln. Dadurch entsteht allerdings eine enge Abhängigkeit zwischen der XAML-Datei und der zuvor genannten Code-Behind-Datei, sodass Änderungen in einer der beiden Dateien oft Änderungen in der Anderen verlangen. Aufgrund dessen wird in der Praxis häufig auf das sogenannte MVVM-Pattern (Model-View-ViewModel) zurückgegriffen. Mit Hilfe dieses Patterns werden Model und View klar voneinander getrennt, was das parallele Arbeiten an beiden Bereichen ermöglicht, ohne dass Änderungen vorgenommen werden müssen.



Darstellung des MVVM-Patterns

Das Model repräsentiert die Daten, die in der Anwendung benötigt werden. In diesem Fall wurden hier einfache Klassen wie User oder Configuration erstellt, von denen später Objekte erzeugt werden sollten. Die View beschreibt alles, was der Benutzer sehen kann und beinhaltet keine Programmlogik. Dafür ist das ViewModel zuständig, welches viele Gemeinsamkeiten mit dem Controller aus dem bekannten MVC-Pattern besitzt. Das ViewModel nimmt Daten aus der View entgegen und verarbeitet diese weiter. Außerdem hält es die View über Commands auf dem neuesten Stand.

Um veränderliche Daten für das ViewModel verfügbar zu machen, werden sogenannte Data-Bindings benutzt. Diese werden im ViewModel wie einfache Variablen behandelt, nur dass eine Änderung des Variablenwertes direkt in der View dargestellt wird. So lassen sich ganze Listen mit Werten aus Modelobjekten erstellen und erweitern, wie es in dieser Anwendung z.B. für Messwerte oder Benutzer der mobilen Applikation benötigt wurde.

Aufgrund der vielen Vorteile, die das MVVM-Pattern bietet, wurde die gesamte Benutzeroberfläche damit erstellt. Zunächst ist ein voll funktionsfähiger Prototyp unabhängig von anderen Arbeitspaketen entwickelt worden, welcher bereits die komplette Interaktion mit der Anwendung ermöglichte, ohne dass echte Daten im Hintergrund aufgenommen oder verarbeitet wurden.

Neben den Standardelementen, die WPF bereitstellt, bietet Visual Studio mit dem Nuget-Paketmanager die Möglichkeit, zahllose, hilfreiche Plugins mit in die Anwendung einzubinden. Dies wurde an vielen Stellen benutzt, um Programmcode zu kürzen oder neue Funktionen benutzen zu können. Hierzu zählen z.B. folgende Plugins:

- MVVM Light & MVVM Dialogs zum Erstellen und Verwalten von Popup-Fenstern (vorgegebene Fenster wie Fehlermeldungen oder Dateibrowser wurden verwendet)
- WPF Toolkit zum Erstellen von Graphen aus Datensätzen
- Fody zum automatischen Einbinden von INotifyPropertyChanged-Funktionalität in allen Properties

Es wurde außerdem ein Ribbon (Menüleiste) verwendet, welches man aus Microsoft-Programmen wie Paint oder Office seit Windows 7 kennt. Damit sollte die Oberfläche durch bekannte Elemente intuitiv gestaltet werden.

Zum Behandeln von Events bei Buttonklicks wurden sogenannte Relay-Commands benutzt. Diese Lösen beim Klicken auf den Button eine Funktion aus und haben den

Vorteil, dass sie in Execute- und CanExecute-Funktionen unterteilt sind. Executes sind eben jene Funktionen, die ausgelöst werden sollen während CanExecutes beschreiben, ob ein Button auslösbar ist (oder deaktiviert werden soll). Mithilfe eines Enums wurden mehrere Applikationsstatus erstellt, welche in den CanExecute-Funktionen benutzt werden und einen geregelten Ablauf garantieren, sodass kein Button zur falschen Zeit ausgelöst werden kann.

Der entstandene Prototyp wurde dann weiter genutzt und mit den anderen Komponenten der Applikation verbunden. Dies erforderte mehrere Umstellungen, vor allem auf Modellebene. Dies wurde durch eine von der Planung abweichenden Entwicklung des Modells auf GUI-Seite notwendig und hätte durch bessere Planung und Kommunikation verhindert werden können.

Im Laufe der Entwicklung wurden an der Benutzeroberfläche zahlreiche Usertests durchgeführt, um die Usability der Anwendung zu erhöhen und Feedback zum Design zu erhalten. So wurden farbliche Anpassungen vorgenommen und Elemente für bessere Übersichtlichkeit verschoben bzw. hinzugefügt.

Das Design der Oberfläche erhielt eine geringere Priorität und wurde dementsprechend erst gegen Ende vorgenommen. Die Anwendung wurde mit einem Flatdesign entwickelt und über aussagekräftige Farben und Strukturen intuitiv bedienbar gemacht. Bilder und Farben lassen sich einfach in XAML einfügen bzw. ändern und auch das Einbinden einer eigenen Schriftart war möglich.

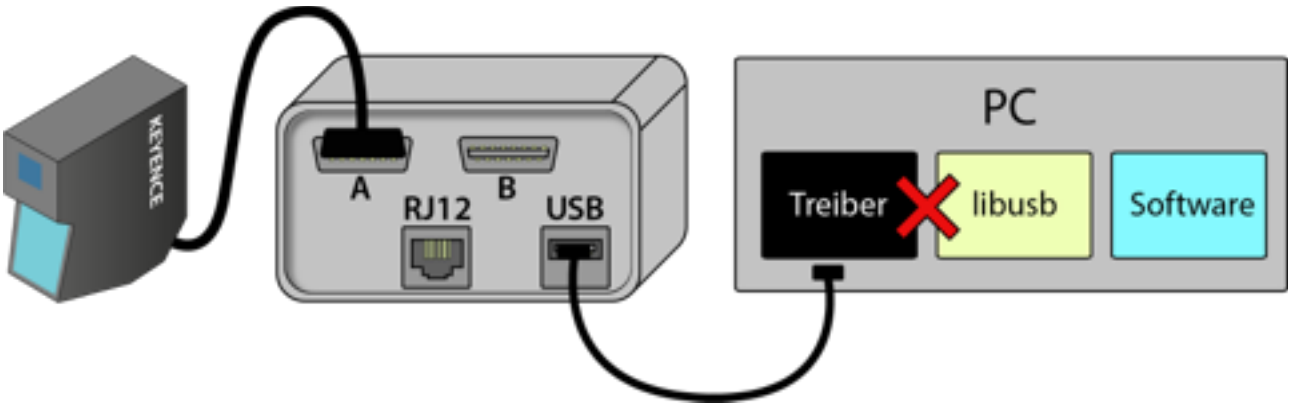
Letztendlich war Windows Presentation Foundation eine gute Wahl für das grafische User Interface. WPF war relativ einfach zu erlernen und bietet eine gute Trennung zwischen Design und Programmierung, was mit anderen Bibliotheken so nicht möglich gewesen wäre.

3.2 Hardwarekommunikation

In diesem Abschnitt wird auf die Kommunikation zwischen der Software und den Laser-Wegmessgeräten (Messköpfe) eingegangen. Dieser Aspekt der Software ist äußerst kritisch, da bei nicht bestehender Kommunikation die gesamte Software nicht funktional und damit nutzlos wäre.

Nicht zufriedenstellende Lösungsansätze

Zunächst wird der generelle Aufbau beschrieben:
Während der Anfangszeit der Entwicklung wurde mit einem einzelnen Messkopf gearbeitet. Der Messkopf wurde an einer Steuereinheit angeschlossen. Diese wiederum wurde mit einem USB-Kabel an dem Computer angeschlossen.



Sensor, Steuereinheit, PC

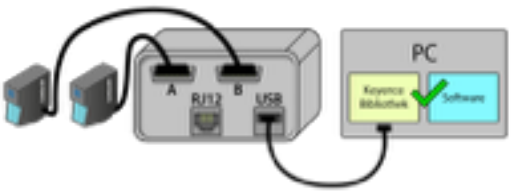
Der erste Lösungsansatz wurde mit Hilfe der Bibliothek *libusb* durchgeführt. Bei *libusb* handelt es sich um eine Sammlung von Befehlen, die einen einfachen Zugriff auf USB-Peripherie ermöglicht. So konnte problemlos mit einem ersten Programm von allen am Computer angeschlossenen USB-Geräten die jeweilige Vendor ID und Product ID ausgelesen werden. Jedem Hersteller von USB-Geräten ist eine eindeutige Vendor ID zugeordnet. Jedem Gerätetyp wiederum ist eine eindeutige Product ID zugeordnet. Aus Kombination der beiden IDs kann ein Gerät genau identifiziert werden. Die beiden IDs der Steuereinheit wird benötigt, um eine Verbindung per *libusb.*

Durch das Programm wurden zusätzlich zu den wirklich per USB angeschlossenen Geräten die IDs weiterer nicht nachvollziehbarer Geräte angezeigt. Neben der Steuereinheit wurden keine weiteren USB-Geräte angeschlossen, aber dennoch weitere angezeigt, weshalb keine der Vendor ID und Product ID Paare unmittelbar der Steuereinheit zugeordnet werden konnte. Die IDs der Steuereinheit wurden ermittelt, indem diese einmal entfernt, das Programm gestartet, die Steuereinheit wieder angeschlossen und das Programm erneut gestartet wurde. Bei dem zweiten Start konnten die beiden neu hinzukommenden IDs der Steuereinheit zugeordnet werden.

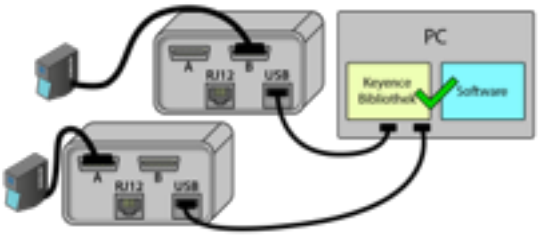
Mit der so gefundenen Vendor ID und Product ID wurde in einem zweiten *libusb*-Programm versucht eine Verbindung zur Steuereinheit aufzubauen. Nach zahlreichen Versuchen stellte sich heraus, dass der Treiber vom Hersteller Keyence nicht mit *libusb* kompatibel ist. Da es sich bei dem Treiber von Keyence um eine Blackbox handelt, man also nicht einsehen kann wie er funktioniert, hätte man einen eigenen Treiber von Grund auf neu entwickeln müssen. Die Entwicklung eines eigenen Treibers würde sowohl unsere Fähigkeiten als auch den zeitlichen Rahmen sprengen.

Herrn Höckmann wurden die Probleme geschildert, welcher wiederum die Firma Keyence kontaktierte. Aus dem Kontakt resultierend besorgte Herr Höckmann eine vollständige Bibliothek, die alle vorgefertigten Befehle beinhaltet, die die Hardware laut Betriebsanleitung unterstützt. Die Benutzung der Bibliothek gestaltete sich als problemfrei, sodass erstmals Messwerte mit Hilfe eines C++ und auch C# Programms ausgelesen und via Konsole ausgegeben werden konnten.

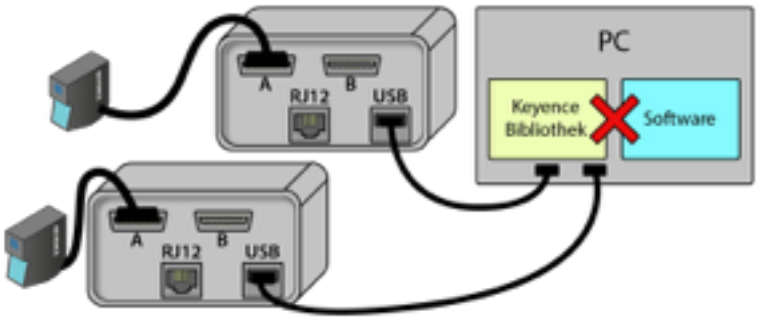
Aufgrund von laufenden Praktikumsversuchen während des Semesters, war die benötigte Hardware nur eingeschränkt verfügbar. Es wurde mit nur einem Messkopf und einer Steuereinheit gearbeitet. Dem SWE-Team wurde daraufhin ein weiterer Messkopf und eine weitere Steuereinheit zur Verfügung gestellt.



Beide Messköpfe in einer Steuereinheit



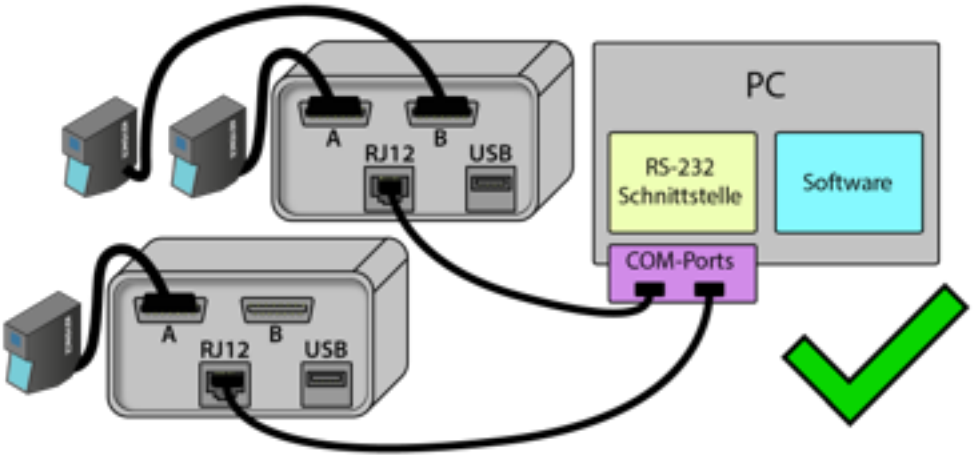
Messkopf jeweils in A und B



Messkopf jeweils in A und A. Messwerte überdecken sich

Mit Hilfe einer Steuereinheit und zwei angeschlossenen Messköpfen (Anschluss an Buchse A und B) konnten fehlerfrei Messwerte ausgelesen werden. Beim anschließen von zwei Steuereinheiten und einem Messkopf pro Steuereinheit (jeweils in Buchse A) hat der Messwert eines Messkopfes den Messwert des anderen Messkopfes überdeckt. Gleiches passierte, wenn die Messköpfe jeweils in Buchse B angeschlossen wurden. Ebenfalls führte der Einsatz von drei Messköpfen zu fehlerhaften Ergebnissen. Es musste also festgestellt werden, dass die Keyence Bibliothek – genau wie die bisher verwendete LK-Navigator Software – nur eine simultane Verwendung von maximal zwei Messköpfen unterstützt. Es musste eine andere Lösung gefunden werden.

Bei dem finalen Lösungsansatz wurde an eine Steuereinheit zwei Messköpfe angeschlossen, an die andere nur ein Messkopf in Buchse A. Die beiden Steuereinheiten wurden jeweils über eine serielle RS-232-Schnittstelle mit dem Computer verbunden. Auf der Seite der Steuereinheit kam ein RJ12 Stecker zum Einsatz, welcher bekanntermaßen auch in Telefonen verbaut wird. Auf der anderen Seite wurde ein 9-poliger D-Sub Stecker verwendet. Die D-Sub Stecker sind über einen COM-Port zu USB Adapter mit dem Computer verbunden worden. Herr Höckmann war so freundlich das Kabel zusammen zu löten, wobei die Belegung der Kontakte der Betriebsanleitung entnommen werden konnte.

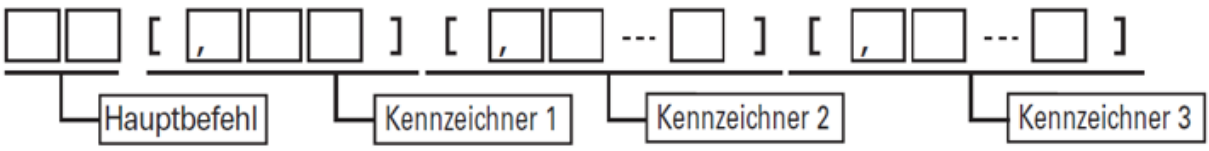


Alle 3 Messköpfe über RS-232 verbunden

Finale Umsetzung

Mit dem neuen Versuchsaufbau sah man im Geräte-Manager des Windows Rechners die beiden Steuergeräte als zwei verschiedene COM-Ports. Das Terminalprogramm HTerm ermöglichte es, erste Befehle nach Vorlage aus der Betriebsanleitung an die Steuereinheiten zu senden und Rückgabebefehle – mit beispielsweise Messwerten – zu empfangen.

Ein Befehl besteht aus einer Folge von Bytes, die an die Steuereinheit geschickt bzw. von ihr empfangen werden. Im Allgemeinen wird mit den ersten Bytes eines Befehls die Art des Befehls festgelegt. Optional können je nach Befehl durch ein Komma (0x2c in Hexadezimal) getrennt, weitere Parameter folgen. Beendet wird ein Befehl immer durch ein Carriage Return CR (0x0D).



Aufbau eines Befehls

Als Beispiel für das Befehlsformat soll der Befehl zur Messwertausgabe dienen: Er beginnt mit einem ‘M’ (0x4d) für Messen, gefolgt von einer ‘0’, ‘1’ oder ‘2’ je nachdem, ob an Messkopf A & B, Messkopf A oder Messkopf B eine Messung durchgeführt werden soll. Da bei diesem Befehl keine weiteren Parameterwerte angegeben werden können, endet er direkt mit einem Carriage Return.

Betrieb	Eingangsbefehl	Antwortbefehl
Messwertaus-gabe	M a CR	M a , h h h h h h h h [, h h h h h h h h] CR

Beispiel für einen Befehl

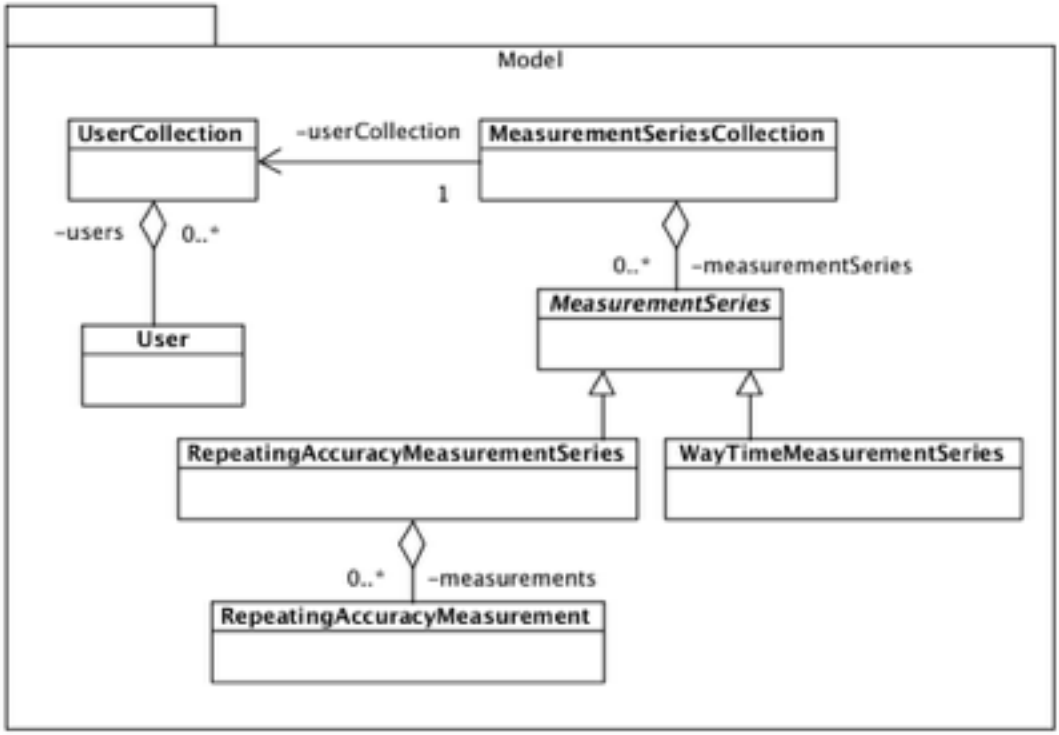
Nachdem erste Erfolge mit HTerm verzeichnet werden konnten, begann die Recherche wie man mit C#-Code eine serielle Verbindung aufbaut, da ein Wechsel der Programmiersprache von C++ C# aufgrund der Benutzeroberfläche stattfand. Nach langer Suche und vielem Herumprobieren konnte ein Beispiel-Programm gefunden werden mit dem erstmals ein Messbefehl an die Steuereinheit gesendet und ein Messwert empfangen wurde. Das Programm war aber fehlerbehaftet. Messwerte wurden nicht als ganzer Befehl empfangen, sondern nur Bruchstückhaft. Die Bytes des Antwortbefehls kamen also immer wieder in Häppchen nicht nachvollziehbarer Größe an. Durch dieses Verhalten wäre es schwierig den Befehl zu interpretieren, wenn er beispielsweise noch nicht vollständig empfangen wurde. Um trotzdem zuverlässig einen kompletten Befehl zu erhalten wurde ein Zwischenspeicher vom Typ Schlange (Queue) eingeführt.
Beim Empfangen neuer Daten werden diese an die Schlange angehängt. Anschließend wird über die Schlange iteriert, um zu überprüfen, ob ein oder mehrere Befehle bereits vollständig in der Schlange liegen. Dies geschieht, indem geprüft wird, ob ein oder mehrere Carriage Returns (im Code '\r') in der Schlange liegen, da diese das Ende eines Befehls und damit seine Vollständigkeit zeigen.
Trotz der langsameren Datenübertragung per serieller Schnittstelle gegenüber USB konnten einzelne Messwerte schnell genug übertragen werden, um sie praktisch live anzuzeigen. Auch dass eine Lösung gefunden werden konnte mit der die Software nur noch an einem PC zu bedienen ist, ist eine große Verbesserung gegenüber dem LK-Navigator. Ebenfalls zu bedenken ist, dass es sich bei der USB-Kommunikation um einen äußerst kritischen Teil der Software handelt. Hätten die Probleme erst deutlich später oder gar nicht gelöst werden können, hätte dies zu einem starken Verzug im Zeitplan oder sogar zum Scheitern des gesamten Projektes führen können. Insgesamt konnten die benötigten Anforderungen jedoch äußerst zufriedenstellend und im Zeitrahmen gelöst werden.

3.3 Model

Innerhalb der Software besteht die Anforderung, dass Messreihen zwischengespeichert, in Excel exportiert und auf der Benutzeroberfläche dargestellt werden können. Zudem müssen auch die Nutzer, welche am verteilten System mit ihren Smartphone teilnehmen zwischengespeichert werden können.

Um dies zu ermöglichen, müssen die Messdaten und Nutzer solange im RAM-Speicher abgelegt werden können, bis der Nutzer der Desktop-Applikation das Programm schließt. Daraus geht hervor, dass die Software ein Modul benötigt, welches sich um die Datenhaltung der Software kümmert. Dieses Modul ist das „Model“.

Das Model hat keine notwendigen Abhängigkeiten zu anderen Modulen. Es bildet ein geschlossenes System, welches eine Schnittstelle zum Lesen, Modifizieren und Entfernen von Daten für die anderen Module bietet.



Klassendiagramm des Models

Messreihen werden in einer Sammlung von Messreihen (MeasurementSeriesCollection) gehalten. Da es zwei unterschiedliche Typen von Messreihen gibt, nämlich die Wiederholungsgenauigkeit-Messreihe (RepeatingAccuracyMeasurementSeries) und die Weg-Zeit- Messreihe (WayTimeMeasurementSeries), entstand der Bedarf die Messreihen mittels Einsatz einer abstrakten Klasse Messreihe (MeasurementSeries) zu abstrahieren. Objekte der zwei Typen von Messreihen erben von der abstrakten Klasse und mittels Polymorphie können alle Typen von Messreihen in der Messreihensammlung gehalten werden. Sollte also der Bedarf nach einer weiteren Art von Messreihe entstehen, müsste man innerhalb des Models eine Klasse hinzufügen, die von der abstrakten Klasse „MeasurementSeries“ erbt.

Wiederholungsgenauigkeit-Messreihen bestehen aus Wiederholungsgenauigkeit-Messungen (RepeatingAccuracyMeasurement). Wiederholungsgenauigkeit-Messungen bestehen aus drei Werten, welche die drei Werte repräsentieren, die die 3 Sensoren zu dem Zeitpunkt der Messung lieferten. Weg-Zeit- Messreihen bestehen aus einer Liste von Zeit und Weg-Wertepaaren, umgesetzt durch Zwei-Tupel. Neben den Messreihen werden auch die Smartphone-Nutzer des verteilten Systems im Model gehalten. Zu ihren Informationen gehört ihr Name und ob sie die Berechtigung zum Auslösen einer Messung haben.

Jede Klasse innerhalb des Model-Moduls bietet Methoden zum Lesen und Modifizieren ihres Zustands an. Diese Methoden können von anderen Modulen verwendet werden um das Modul auszulesen oder anzupassen.

In der Benutzeroberflächenlogik (sowohl vom Smartphone, wie auch von der Desktop-Applikation) spielt das Model eine wichtige Rolle. Die Daten, welche im Model vorliegen müssen dem Nutzer angezeigt werden können, damit dieser über den Zustand seiner Messungen Bescheid weiß. Hierbei entsteht die Anforderung, dass die

Benutzeroberflächenlogiken über Änderungen innerhalb des Models informiert werden müssen. Dies wird realisiert durch das Observer-Entwurfsmuster realisiert. Hierzu mehr im folgendem Kapitel „Befehlsschicht“.

3.4 Befehlsschicht

Nebenläufigkeit
Unsere Software muss viele Funktionalitäten bezüglich der Änderung von Daten anbieten. Viele dieser Funktionalitäten sind Abhängig von den Antworten, die uns die Auswerteeinheiten liefern. Eine Herausforderung stellt hierbei der Umgang mit den zu der Hauptapplikation asynchron verlaufendem Antwortverhalten der Auswerteeinheiten dar. Hauptapplikation, Auswerteeinheit 1 und Auswerteeinheit 2 entsprechen drei parallel ablaufenden Prozessen.

Da das Model von den anderen Modulen verwendet wird, muss an dieser Stelle paralleler Zugriff vermieden werden. Hinzukommt, dass die Oberflächenlogik keinen direkten Aufruf auf das Model vollziehen sollte, da das Modul ViewModel sonst viel zu komplex und dementsprechend schlechter zu warten machen würde, was die folgende Entwicklung der Software in die Länge gezogen hätte. Um diese beiden Anforderungen realisieren zu können, entschieden wir uns entschieden an dieser Stelle das Command-Entwurfsmuster zu verwenden.

Durch den Einsatz des Command-Entwurfmusters ist es möglich, dass Befehle synchron abgearbeitet werden können. Bei der Übergabe an den Command-Executer werden die Befehle in eine Warteschlange aus Befehlen eingefügt und abgearbeitet, sobald die Antworten der Auswerteeinheiten erhalten wurden. Hierbei ließen sich drei unterschiedliche Typen von Befehlen identifizieren.

1**. Einfache Befehle**, die direkt synchron abgearbeitet werden können. Beispiele hierfür sind der Excel-Export oder das Anlegen neuer Messreihe.
2. Reaktive „halbe“ Befehle, welche einen Befehl an genau eine Auswerteeinheit senden. Diese Befehle bestehen aus einem „Anfrage-teil“, in welchem der Befehl an die Sensoreinheiten übermittelt werden kann und einen „Reaktionsteil“, in welchem auf Basis der erhaltenen Antwort von den Auswerteeinheiten reagiert werden kann. Beispiele hierfür sind die Befehle für die Weg-Zeit-Messung und die Konfigurations-Befehle für die Auswerteeinheiten.
3. Reaktive „ganze“ Befehle, welche aus zwei reaktiven halbe Befehle bestehen und dabei sicherstellen, dass beide als ein zugehöriger ganzer Befehl interpretiert wird. Beispiele hierfür sind das Triggern und Kalibrieren, da hier Antworten von beiden Auswerteeinheiten erwartet werden.

Für jede Auswerteeinheit wurde eine Befehls-Warteschlange aus reaktiven Befehlen angelegt. Dadurch sollte sichergestellt werden, dass für jede Auswerteeinheit immer erst die Antwort abgewartet werden würde, bevor die nächste Anfrage gestellt werden darf.

Aus diesem Entwurf entstand die Notwendigkeit eine abgeschlossene Antwort als solche identifizieren zu können. Um dies zu erreichen wurden zwei Zeichen-Schlangen angelegt. Jede Schlange diente dabei zur Überwachung einer Auswerteeinheit. Immer, wenn eine Auswerteeinheit ein Zeichen als Antwort lieferte, wurde dieses in die zugehörige Schlange eingefügt. Wann immer ein „Carriage Return“ (\r) gelesen wurde, wurde der bisherige Inhalt der jeweiligen Schlange zusammengefügt und als komplette Antwort einer

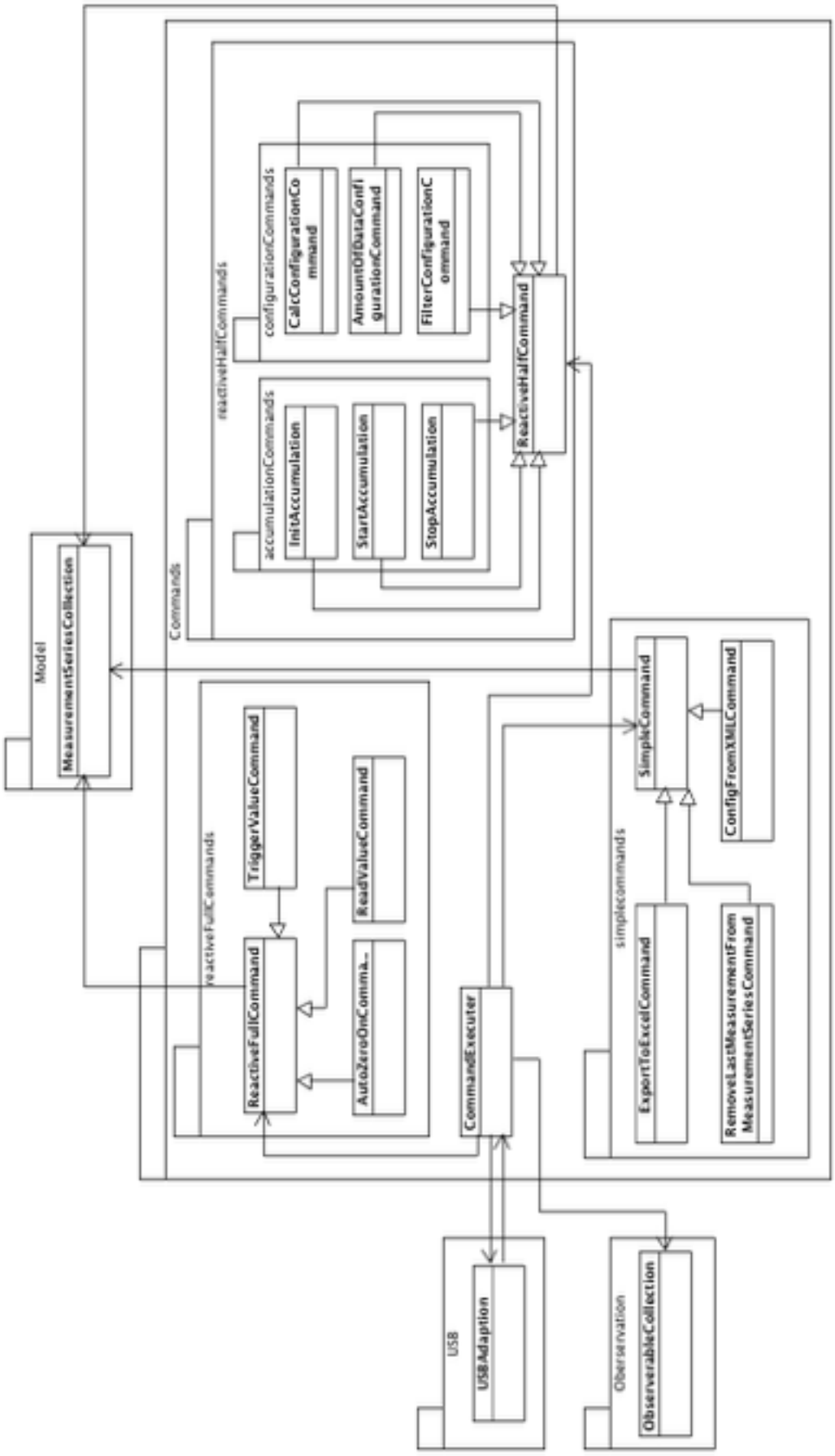
Auswerteeinheit interpretiert. Mit dieser Antwort wurde dann der jeweilige „Reaktionsteil“ des zugehörigen Befehls aufgerufen. Wurde dieser abgearbeitet, wird der Befehl anschließend aus der Befehls-Warteschlange entfernt und der nächste Befehl wird bearbeitet.

Das Benachrichtigungssystem

Außerdem besteht die Anforderung, dass auf der Benutzeroberfläche immer die aktuellen Daten aus dem Model angezeigt werden müssen. Die Benutzeroberflächenlogiken müssen informiert werden, sobald im Model eine Änderung vorgenommen wurde. Dies wurde realisiert durch den Einsatz des Observer-Entwurfmusters.

Das „Subject“ innerhalb des Observer-Entwurfmusters stellt hierbei das Model dar. Sein Zustand ist relevant für die Darstellung auf der Benutzeroberfläche. Die „Observer“ hierbei sind zum einen das ViewModel der Desktop-Applikation, aber auch die Benutzeroberflächenlogik der Smartphones, die am verteilten System teilnehmen. Die Observer registrieren sich initial in einer ObserverCollection. Sobald eine Änderung innerhalb des Models vollzogen wurde, wird die ObserverCollection informiert, welche daraufhin seinerseits alle Observer informiert, die sich bei der ObserverCollection zuvor registriert haben. Auf diese Art und Weise kann die Benutzeroberfläche immer die aktuellen Daten darstellen.

Um die Laufzeit optimieren, aber gleichzeitig die Unabhängigkeit des Models zu anderen Modulen zu bewahren und damit Zyklen zu vermeiden, die die Software schwerer zu warten machen, überließen wir die Aufgabe der Benachrichtigungen bezüglich Änderungen am Model den Befehlen, welche das Model verändern, selbst. Dies erschien uns Entwurfs-technisch der beste Kompromiss zu sein.



Aufbau der Desktop-Anwendung

3.5 Smartphone-Applikation

Zur Umsetzung der Applikation benutzten wir zu Beginn das Web-App-Framework Ember. Nach vielen Versuchen das Framework Socket.io mit Ember zu verbinden wurde festgestellt, dass Kompatibilitätsprobleme zwischen beiden Technologien bestehen.

Nach langer Recherche wurde klar, dass nur wenige Web-App-Frameworks Websockets unterstützen und dass für diese eine anhaltende Einarbeitungszeit einzuplanen ist. Nach solchem Fazit wurde beschlossen, die mobile Applikation ohne Web-App-Frameworks zu implementieren. Stattdessen benutzten wurde das „lightweight“ Frontend-Framework Materialize verwendet, welches den vorgesehenen Zweck erfüllte und für eine gelungene Gestaltung sorgte.

Mithilfe des Javascript Programms wurden folgende Anforderungen umgesetzt:
Verschiedene Modi für Zuschauer und Auslöser
Kommunikation und Authentifizierung bei dem Node.js Server und der Desktop-Applikation
Eine in Schritte unterteilte Bedienung
Auslösen einer Messung
Erstellen und Abbrechen einer Messreihe
Möglichkeit Messergebnisse einzusehen erfüllt

Die mobile Applikation ist mithilfe eines modernen Browsers erreichbar. Das Öffnen und Bedienen der mobilen Applikation kann somit durch die meisten Endgeräte mit einem WLAN Modul und aktuellem Browser erfolgen.

Aktionen wie das Schließen des Browser-Fensters, dem Öffnen der Website oder Verbindungsverlust werden bei dem Node.js Server und anschließende der Desktop-Applikation registriert und behandelt. Wird die Verbindung beabsichtigt oder unbeabsichtigt beendet, registriert der Server den Verbindungsabbruch und informiert den Desktop-Anwender entsprechend.

3.6 Netzwerkkommunikation

Bei der Umsetzung der Netzwerkkommunikation zwischen mobiler Applikation und Desktop-Applikation entschieden wir uns für eine bidirektionale Kommunikation via Websockets. Diese ermöglicht es uns schnell und mit vielen Bedienern zur selben Zeit zu kommunizieren.

Als erste Herausforderung stellte sich die Infrastruktur des Hochschulnetz welches im Vorfeld angedacht war um es für die Datenübertragung zu nutzen. Aufgrund von Konfigurationen des Netzes, die der allgemeinen Sicherheit der Nutzer gilt, ist eine direkte Datenübertragung nicht möglich. Mit der erneuten Mithilfe von Herrn Höckmann wurde ein Accesspoint aufgestellt der ein lokales Netzwerk aufstellt, mit welchem sich die beteiligten Geräte verbinden.

Da die Desktop-Applikation in der Programmiersprache C# geschrieben ist, die mobile Applikation hingegen eine HTML5 Website ist, die mit der Skriptsprache Javascript agiert, musste ein Framework gefunden werden, welches Clients für beide Programme anbietet und diese kommunizieren lässt.

Ein erster Versuch fand mit der Bibliothek SignalR statt, welche intern Websockets nutzt.

SignalR bietet als Host innerhalb der Desktop-Applikation die Möglichkeit eine direkte Verbindung von einem Endgerät zu sich selbst aufzubauen. Ein Vorteil dieser Architektur ist die einfache Strukturierung des Programmcodes, da kein zusätzlicher Knoten (Hop) benötigt wird.
Aufgrund von Abhängigkeitsproblemen durch den NUGGET-Paketmanager von Visual-Studio ließ sich die Technologie nicht ausreichend im gegebenen Zeitrahmen testen um zu einem zufriedenstellenden Ergebnis zu führen. Entsprechend wurde dieser Ansatz verworfen um zeitnah effizient weiter arbeiten zu können.

Der zweite, erfolgreiche Versuch wurde mithilfe des Frameworks Socket.io umgesetzt.
Socket.io bietet selbst einen Web-Client an und verwaltet Websockets auf einem Node.js Server.
Das Framework gibt klare Richtlinien zur Strukturierung des Programms und Verwaltung der Sessions vor, was einschränkend sein kann, auf der anderen Seite aber sehr zur Fehlervermeidung beiträgt. Für die Desktop-Applikation wurde auf der Plattform GitHub ein implementierter C# Client angeboten, welcher anschließend integriert wurde.

Der Nachrichtenaustausch zwischen der Desktop-Applikation, dem Node.js Server und der mobilen Applikation findet im JSON Format statt. Das JSON Format ist ein etabliertes und übersichtlich strukturiertes Format, welches in Web-Anwendungen oft für ähnliche Zwecke verwendet wird. Der Node.js Server läuft als separates Programm parallel zur Desktop-Applikation ab und wird aus der Desktop-Applikation heraus gestartet. Alle Clients müssen sich vorerst bei dem Node.js Server anmelden und authentifizieren werden bevor ein Nachrichtenaustausch stattfinden kann.

Die Geschwindigkeit, Verlässlichkeit und die Struktur des Servers, sowie das Ansprechen des Clients sind sehr zufriedenstellend. Alle Anforderungen an das Netzwerk wurden erfüllt, sodass eine reibungslose Kommunikation zwischen den beteiligten Endgeräten möglich ist.

Zusammenfassend lässt sich festhalten dass auch bei der Umsetzung der mobilen Anwendung Probleme auftraten die im Vorfeld nicht kalkulierbar waren. Da der Anforderungsteil der sich mit der mobilen Anwendung befasst hat jedoch recht lose definiert war und viel Freiraum bei der Auswahl der Technologien ließ, konnten die Problemen zeitnah angegangen werden. Insgesamt ist das Ergebnis zufriedenstellen da die meisten Anforderungen umgesetzt werden konnten.

4. Softwareergebnisse

Im Folgenden Kapitel sollen die entstandenen Entwicklungsergebnisse präsentiert und ihre Funktionen erläutert werden. Dabei tritt der Programmcode in den Hintergrund und der Fokus liegt auf der Interaktion und Bedienung der Desktop-Anwendung und der mobilen Applikation.

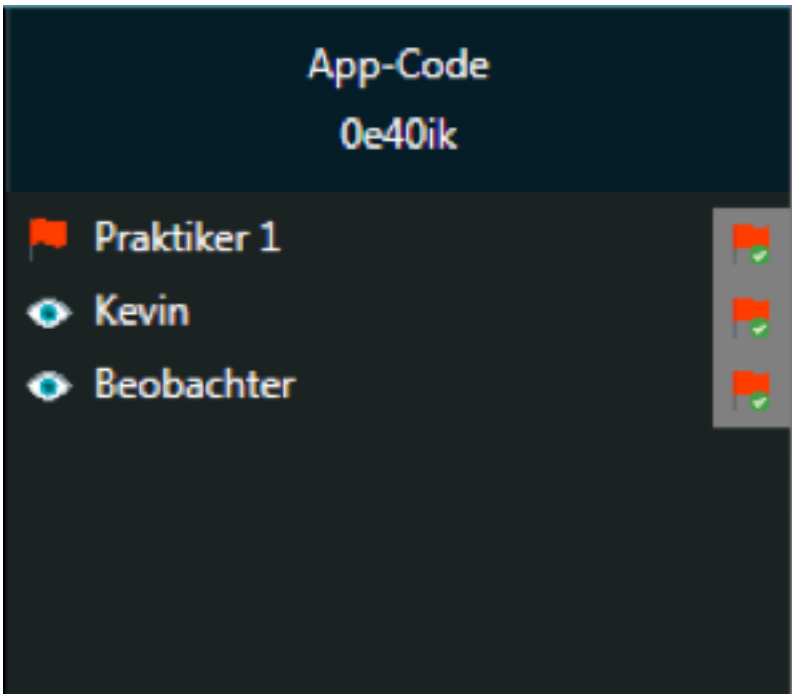
4.1 Desktop-Anwendung

Die Desktop-Anwendung ist eine Windows Applikation, welche standardmäßig in einem Fenster geöffnet wird. Sie besitzt jedoch responsives Verhalten, sodass die Fenstergröße beliebig angepasst werden kann. Die Anwendung besteht aus drei elementaren Bereichen: Einer horizontalen Menüleiste (Ribbon) am oberen Rand, einer Verwaltung der Nutzer der mobilen Applikation am rechten Rand sowie einem konfigurationsabhängigen Hauptteil, der den restlichen Platz einnimmt.
Die Software ist so aufgebaut, dass die Bedienung einem starren Ablauf folgt, d.h. Buttons sind ausschließlich aktiviert und nutzbar, wenn die jeweilige Funktion zum Zeitpunkt sinnvoll ist. Dadurch werden Fehler bei der Bedienung minimiert und die Applikation ist sehr leicht erlernbar. So ist beim Start der Anwendung ausschließlich der Button “Konfiguration auswählen” verfügbar. Die einzelnen Buttons der Menüleiste werden nun stichpunktartig beschrieben:

Ribbon Menüleiste

- “Konfiguration auswählen” öffnet einen Dateibrowser, mit welchem eine .xml-Datei geladen werden kann. Dabei existieren zwei verschiedene Typen, eine Konfiguration für Genauigkeitsmessungen und eine für Weg-Zeit-Messungen. Diese werden im Dateinamen mit “.g” bzw. “.wz” gekennzeichnet. Die Konfigurationen nehmen voreingestellte Einstellungen an den Sensoren vor und ändern zudem die Ansicht des Hauptteils der Desktop-Anwendung. Die ausgewählte Konfiguration wird am rechten Rand der Menüleiste dargestellt.
- “Messvorgang starten” ist nach Auswahl einer Konfiguration aktiviert und startet einen neuen Messvorgang in der gewählten Konfiguration. Nach dem Start eines Messvorgangs ist es nun möglich Messwerte aufzunehmen, zu löschen oder die Sensoren auf 0 zu kalibrieren.
- “Messvorgang abbrechen” ist nach dem Starten eines Messvorgangs verfügbar und verwirft alle im aktuellen Messvorgang aufgenommenen Werte. Beim Abbrechen wird ein Popup-Fenster zur Bestätigung aufgerufen, um ein versehentliches Löschen der Messwerte zu verhindern.
- “Messvorgang beenden” speichert einen gestarteten Messvorgang unter Angabe eines Namens. Gespeicherte Messvorgänge können später erneut aufgerufen und betrachtet werden. Das nachträgliche Aufnehmen weiterer Messungen soll allerdings nicht möglich sein. Der Name des aktuell ausgewählten Messvorgangs ist neben dem Softwarenamen in der Titelleiste zu sehen.

- “Exportieren” ruft ein Popup-Fenster mit einer Liste aller bisher aufgenommenen Messvorgänge auf. Diese können mithilfe von Checkboxen ausgewählt werden. Alle markierten Messvorgänge werden bei Bestätigung in eine Excel-Datei exportiert. Dabei entspricht ein Tabellenblatt jeweils einem Messvorgang.
- “Messvorgänge ansehen” ist ein Drop-Down-Menü mit einer Liste aller gespeicherten Messvorgänge. Bei Auswahl eines Messvorgangs wird dieser erneut geladen und man kann alle aufgenommenen Werte noch einmal betrachten. Die Ansicht im Hauptteil wird entsprechend der Konfiguration, in welcher der Messvorgang erstellt wurde, angepasst.

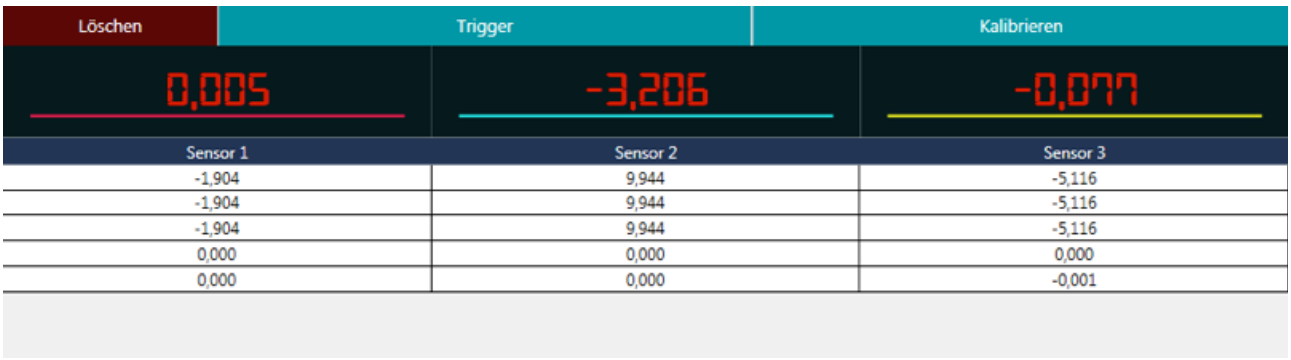


Verwaltung der Benutzer

Auf der rechten Seite der Anwendung befindet sich eine Verwaltung für die Benutzer der Smartphone-Applikation. Es wird ein Code dargestellt, den die Nutzer auf der mobilen Applikation eingeben müssen, um sich mit der Anwendung zu verbinden. Nach der Anmeldung werden alle Nutzer in einer Liste angezeigt und können eine von zwei Rollen annehmen: Auslöser oder Zuschauer. Auslöser können die Smartphone-Applikation als Fernbedienung der Anwendung verwenden - die Details dazu werden im nächsten Kapitel erläutert. Zuschauer haben eingeschränkte Rechte und können ausschließlich Werte einsehen. Über einen Button können in der Desktop-Anwendung Nutzer zum Auslöser befördert werden. Dies ist nur zwischen aktiven Messvorgängen möglich. Alle Nutzer sind nach der Anmeldung standardmäßig Zuschauer.

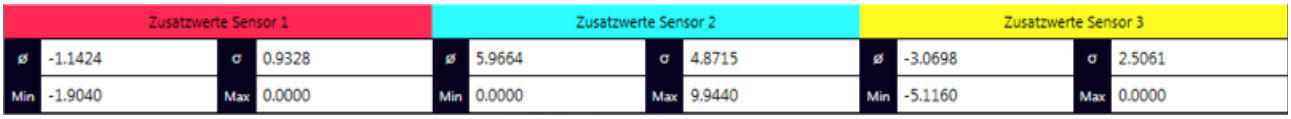
Genauigkeitsmessung Ansicht

Bei der Auswahl der Genauigkeitsmessung Konfiguration wird die Ansicht angepasst. Im Hauptteil sind nun folgende vier Bereiche zu sehen: Eine Buttonleiste mit Funktionen, eine Anzeige der Live-Werte aller Sensoren, eine Liste ausgelöster Messungen sowie eine Tabelle mit hilfreichen Zusatzwerten.



Genauigkeitsmessung Ansicht

Die Buttonleiste beinhaltet elementare Funktionen wie das Löschen einzelner aufgenommener Messwerte, das Auslösen (Triggern) einer Messung und das Kalibrieren aller Sensoren auf 0. Die Live-Werte werden mehrmals die Sekunde aktualisiert und liefern stets den aktuellen Messwert. Sollte ein Sensor außer Reichweite sein, wird dem Benutzer die Ausgabe “FFFFFF” angezeigt. Beim Auslösen einer Messung mit diesem Wert wird außerdem eine Fehlermeldung geworfen und der fehlerhafte Wert in der Liste rot markiert.

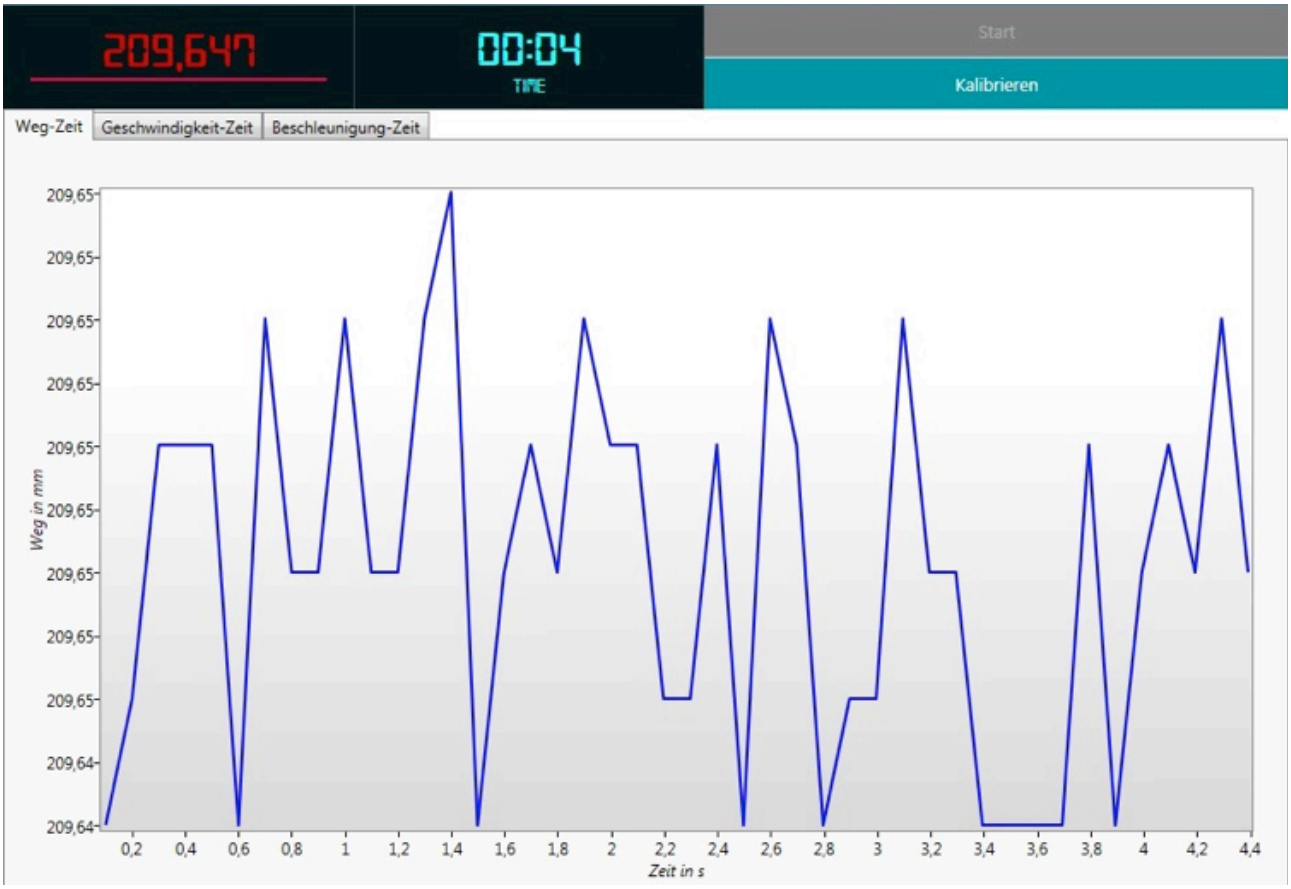


Zusatzwerte aller Sensoren

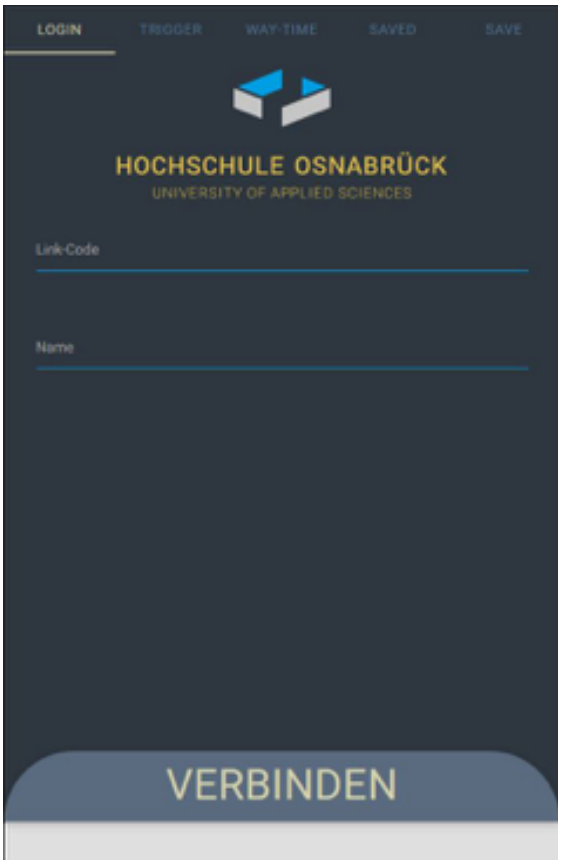
Die Liste zeigt alle aufgenommenen Messwerte an. Einzelne Zeilen können ausgewählt werden, um mit dem Löschen-Button Messwerte zu entfernen. Die Tabelle mit Zusatzwerten wird pro Sensor automatisch aus den aufgenommenen Messwerten erstellt und ist wie die Live-Werte farblich hervorgehoben.

Weg-Zeit-Messung Ansicht

Die Ansicht der Weg-Zeit-Messung beinhaltet ähnliche Elemente wie die der Genauigkeitsmessung, allerdings wird hier nur ein einziger Sensor benötigt, weswegen es neben einem Timer nur eine Liveanzeige gibt. In der Buttonleiste ist nun neben dem Kalibrieren-Button der Button Start hinzugekommen, welcher eine neue Weg-Zeit-Messung startet und dann zum “Stop”-Button wird. Im restlichen Abschnitt werden drei unterschiedliche Graphen nach dem Stoppen einer Messung angezeigt: Weg-Zeit, Geschwindigkeit-Zeit und Beschleunigung-Zeit. Einzelne Werte werden erst beim Exportieren nach Excel sichtbar.



Weg-Zeit-Messung Ansicht



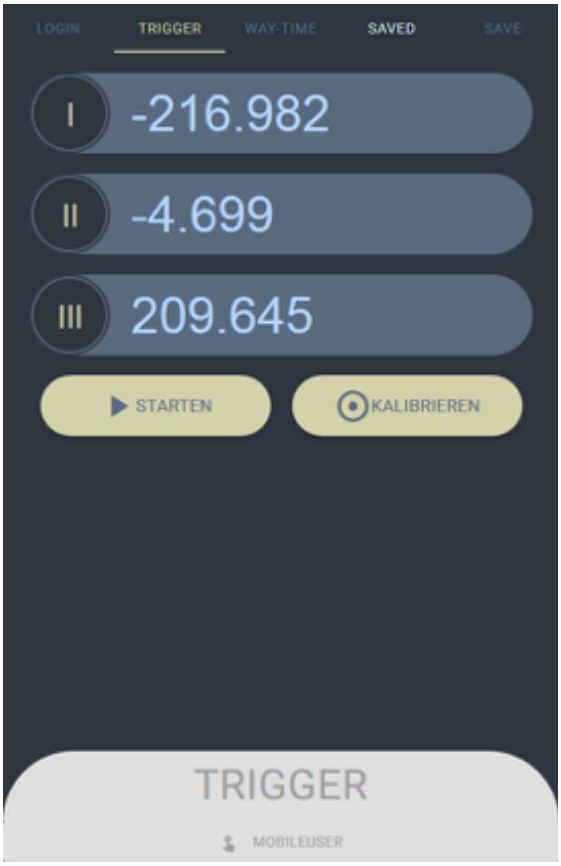
Login-Seite der Applikation

4.2 Smartphone-Applikation

Die Smartphone-Applikation dient als Fernbedienung der Desktop-Anwendung und hat dementsprechend identische, jedoch reduzierte Funktionen. Sie besteht aus drei wesentlichen Bereichen: Einer Tableiste zur Navigation, einem großen Bereich für den Inhalt sowie einer kleinen Leiste am unteren Rand, wo Verbindungsinformationen Platz finden.
Nach dem Aufrufen der Anwendung wird ein Login-Fenster angezeigt. Hier muss der Link-Code, welcher in der Desktop-Anwendung angezeigt wird eingegeben werden, zusammen mit einem einzigartigen Namen, welcher den Benutzer identifiziert. Nach dem Bestätigen wird nun auf das Laden einer Konfiguration gewartet - welches ausschließlich mit der Desktop-Anwendung möglich ist. Die danach geladene Ansicht wird analog zur Desktop-Anwendung über die Konfiguration bestimmt. Zusammen mit den

Live-Werten sind alle aufrufbaren Funktion identisch, nur das Löschen von Messwerten ist nicht möglich.

Beim Speichern wird die Ansicht automatisch gewechselt, damit ein Name eingegeben werden kann. Beim Starten einer Weg-Zeit-Messung wird sowohl ein Messvorgang als auch die Messung selbst gestartet.
Die gesamte Funktionalität des Aufnehmens von Messungen und deren Verwaltung ist dem Auslöser überlassen, um ein Durcheinander bei der Ausführung von Funktionen zu verhindern. Zuschauer können lediglich Live-Werte und gespeicherte Werte ansehen.



Genauigkeitsmessung in der Smartphone-Anwendung

5. Fazit

Es wurde deutlich, dass die Entwicklung einer Desktop-Anwendung mit mobiler Steuerung sich als nicht so trivial herausgestellt hat, wie zum Zeitpunkt der Themenausschreibung von den Beteiligten vermutet wurde. Zu erwähnen ist die Tatsache, dass das Entwicklungsteam ausschließlich aus Studierenden der Medieninformatik bestand, deren Kernkompetenzen nicht die Arbeit mit hardwarenaher Programmierung umfasst. Da jedoch alle Teilnehmer volle Bereitschaft zur Weiterbildung, Diskussion sowie Kompromissbereitschaft gezeigt haben, war das Projekt dennoch ein Erfolg. Das Endergebnis entspricht den Erwartungen die im Vorfeld zwischen Auftraggeber und Entwicklungsteam vereinbart wurden und im Lasten/Pflichtenheft festgehalten wurden.
Das Ziel, die Praktikumsversuche im Robotiklabor durch Software, die auf die Ansprüche maßgeschneidert ist, zu erleichtern, wurde erreicht.
Unabhängig von den technischen Herausforderungen haben sich alle Teammitglieder auch dahingehend persönlich weiterentwickelt, als dass die Erfahrung gemacht wurde zu lernen was es bedeutet in einem sechsköpfigen Team zu arbeiten. Während Meinungsverschiedenheiten und individuelle Vorgehensweisen besonders zu Beginn der Entwicklung den Gesamtfortschritt beeinflusst haben, so wurde im Verlauf der Arbeiten ein gemeinsamer Konsens gefunden, um möglichst effizient und reibungslos dem Ziel entgegen zu streben. Auch der Umgang mit Deadlines, speziell vor dem Hintergrund der gemeinsamen Terminfindung, war für das Team eine Hürde die erfolgreich überwunden wurde.
Während die Implementierung die Studierenden natürlich selbst vornahmen, so wurde an manchen Engpässen bereitwillig Hilfestellung von den Betreuenden geleistet um nicht unnötig Zeit zu verlieren bei dem ohnehin schon engen Zeitplan.
Anfangs festgelegte Rollen innerhalb des Teams wurden im Verlauf des Projekts teilweise gewechselt oder feiner granuliert da sich der volle Umfang der einzelnen Module erst während der Arbeiten überblicken ließ.
Abschließend lässt sich festhalten, dass neben den technischen Kompetenzen, die jeder Teilnehmer zu Beginn mitbrachte, maßgeblich die Organisation, Betreuung sowie Kommunikation, nach Innen und Außen, für den Erfolg verantwortlich ist und somit zu diesem Ergebnis geführt mit dem alle zufrieden sind.