Softwareprojektpraktikum Lego Mindstorms
Informatik 11 Embedded Software

Abschlussbericht Team 2

J. Breyer, F. Friedrichs, C. Kloos, G. Kobsik, R. Kupper

8. Mai 2018

Inhaltsverzeichnis

1. Einleitung

Im Rahmen des Praktikums *Entwicklung NXT-gesteuerter LEGO-Fahrzeuge mit Java* stand unser Team vor der Aufgabe, aus dem *Lego Mindstorm NXT 1.0* Set einen Segway zu bauen und diesen über das *LeJOS* Betriebssystem mit Java funktionsfähig zu machen. Dabei sollte in diesem Projekt die Softwareentwicklung mit Scrum angewendet werden. Das Praktikum bestand dabei aus einer Reihe von Abnahmen und Zwischenvorträgen, um den aktuellen Stand des Projekts zu überprüfen.

Zu Beginn beschäftigte sich das Projekt mit der Einrichtung und Installation von Eclipse und Git, welche die vorgegebenen Werkzeuge für dieses Praktikum darstellen. Als erste große Aufgabe folgte die Implementierung eines PID Reglers, sodass der Segway balancieren kann. Anschließend wurde eine Anforderungsanalyse durchgeführt, die den restlichen Ablauf des Projekts bestimmte. Die nächste Aufgabe stellte schließlich die Entwicklung eines großen zusammenhängenden Systems dar, bestehend aus einer GUI am PC, dem NXT und der Kommunikation zwischen beiden Geräten. Es folgte die Implementierung der Bewegungssteuerung sowie zwei Extras, welche diese Arbeit abschließen sollen.

Dabei ging es nicht nur um das Programmieren, sondern auch um die Organisation innerhalb eines Teams. Der folgende Bericht dokumentiert die Durchführung und das Ergebnis unserer Gruppe in diesem Projekt.

2. Anforderungen

2.1. Hardware

Hier betrachten wir die uns zur Verfügung gestellte Hardware, insbesondere die Sensorik des NXT-Bricks. Der Lego Mindstorms NXT verfügt über einen 48 MHz ARM-Prozessor, 64 KiB Hauptspeicher sowie 256 KiB Flash-Speicher für Programme.

Der Gyro-Sensor ist ein Einachsen-Gyroskop mit einer Auflösung von $1\,^\circ\,\mathrm{s}^{-1}$. Bedingt durch die Diskretisierung werden Winkelgeschwindigkeiten unter $1\,^\circ\,\mathrm{s}^{-1}$ nicht erkannt. Folglich lässt sich durch aufsummieren der Winkelgeschwindigkeiten kein absoluter Winkel berechnen. Eigene Tests ergaben einen Fehler von bis zu 15° pro Umdrehung, trotz regelmäßiger Rekalibrierung.

Für die Ansteuerung der Räder stehen zwei Motoren zur verfügen. Die Motoren selbst verfügen über einen Quadraturencoder mit einer Auflösung von 720 Ticks pro Umdrehung. Softwareseitig wird allerdings nur eine Auflösung von 360 Ticks pro Umdrehung bereitgestellt.

Der Ultraschall-Sensor erkennt Objekte innerhalb eines Kegels mit 30° Öffnungswinkel. Die Distanzauflösung betragt 1 cm. Der theoretische Bereich in welchem Objekte erkannt werden können ist somit 1 cm bis 254 cm, der tatsächliche Bereich liegt allerdings eher bei 7 cm bis 160 cm [1]. Durch den Wert 255 wird signalisiert, dass kein Hindernis erkannt wurde. Leider wird durch den selben Wert ebenfalls ein Fehler bei der Distanzbestimmung signalisiert, was die Hinderniserkennung stark erschwert. Bei eignen Tests traten die Fehler sehr häufig auf. Funktionsbedingt können schalldämpfende Oberfläche nur schlecht erkannt werden.

Auf PC-Seite werden eigene Laptops verwendet, so dass dort keine besonderen Einschränkungen vorliegen.

2.2. Software LeJOS

LeJOS ist eine Java Virtual Machine (JVM) und kann als alternative Firmware für den Lego NXT verwendet werden. Die LeJOS NXT Runtime stellt eine minimale Standardbibliothek zur Verfügung. Lambda-Ausdrücke werden nicht unterstützt.

Der Scheduler arbeitet strikt nach Priorität und verwendet bei Threads gleicher Priorität Round Robin mit einem Intervall von 2 ms.

2.3. Issues

Die von der Betreuern gestellten Anforderungen wurden der Einführungspräsentation entnommen und in GitLab (vgl. Abschnitt 4.2) als Issues strukturiert. Hierzu wurde im Zuge eines Meetings ein Brainstorming durchgeführt, um alle Anforderungen zu erfassen und den entsprechenden von der Leitung vorgegebenen Deadlines zuzuordnen. Diese groben Anforderungen wurden danach weiter spezifiziert und in den Issues in kleinere Teilaufgaben aufgeteilt, um den verantwortlichen Programmierern in jeder Woche einen Leitfaden zu bieten. Alle Issues wurden mit Labeln versehen, die unter anderem die Abnahme, zu der die Anforderung erfüllt sein musste, die voraussichtlich beteiligten Programmierer und eine grobe Einteilung in die Software-Komponente (beispielsweise GUI, PID-Regler oder Kommunikation) kennzeichneten. Die derartige Organisation bot den Programmierern eine schnelle Übersicht über bevorstehende Aufgaben. Dieses Konzept wurde ab Abnahme 3 umgesetzt.

2.3.1. Struktur

Die derart angelegten Issues waren strukturell gleich aufgebaut. Ein Issue bestand aus einer aussagekräftigen Überschrift, die die zu erfüllende Anforderung auf den ersten Blick erkennbar machte, sowie einer kurzen Beschreibung der Anforderung. Der erste Unterpunkt Umsetzung beinhaltete die spezifizierte Zusammensetzung der Anforderung und die Übereinkunft unserer Gruppe über die Art der Implementierung.

Im optionalen Anwendungsfall wurde eine beispielhafte Durchführung der Aufgaben angegeben, die einer Erfüllung der jeweiligen Anforderung entsprach. Im Unterpunkt Tests wurden ein oder mehrere Tests mit eventuellen Vor- und Nachbedingungen angegeben, deren Erfüllung einer Erfüllung der jeweiligen Anforderung entsprach. Zusätzlich wurde jedes Issue mit einer Deadline und dem grob geschätzten Arbeitsaufwand versehen.

Die derart erfassten Anforderungen wurden von der Projektleitung überprüft und bestätigt.

2.3.2. Anforderungen

Abnahme 1

Für die erste Abnahme wurde eine erfolgreiche Installation der vorgegebenen Entwicklungsumgebung vorgenommen. Zusätzlich dazu arbeiteten sich die Beteiligten eigenständig in das verwendete Framework und Git ein. Ein gruppeninternes Wiki zur zentralen Bereitstellung wichtiger Informationen wie Tutorials und Protokolle wurde angelegt.

Die Anforderung an die erste Abnahme bestand zudem aus einem ersten Hardware-Konzept und einigen kürzeren Beispielprogrammen, um das grundsätzliche Verständnis im Umgang mit der Hard- und Software zu demonstrieren. Hierzu wurde ein „Hello World"-Programm, dessen Zweck in der Ausgabe einer einfachen Textzeile auf dem Bildschirm des NXT bestand, ein Programm zur Ansteuerung der einzelnen Motoren sowie eine Kalibrierung und Auslesung des Gyrosensors geschrieben. Zudem befassten sich einzelne Mitglieder bereits mit dem Konzept des Reglers.

Abnahme 2

In der zweiten Abnahme war ein erfolgreiches Balancieren mit nicht notwendigerweise selbstgeschriebenem Code gefordert. Zudem sollte der Aufbau des Segways abgeschlossen sein. Die Projektmanagement-Tools, auf die sich die Gruppe geeinigt hat, wurden ebenfalls vorgestellt.

Abnahme 3

Im Zuge der dritten Abnahme wurden die Anforderungen an das Projekt erhoben und strukturiert in Issues im GitLab eingetragen. Daraus resultierend wurde ein Zeitplan erstellt, an dem sich die Beteiligten orientieren konnten. Als Lastenheft dient die Präsentation der Projektleitung, als Pflichtenheft genügen die angelegten Issues. Da zur Erfüllung dieser Anforderungen nicht alle Mitglieder benötigt werden, beschäftigt sich der Rest der Gruppe mit der Optimierung des Reglers.

Zwischenvortrag 1

Zu diesem Termin musste neben einer kurzen Präsentation des Arbeitsfortschrittes ein erstes Kommunikationsprotokoll entworfen werden. Zur Ausarbeitung dieses Protokolls trafen sich Repräsentanten jeder Gruppe, um eine gruppenübergreifende Kompatibilität zu gewährleisten. Die Anforderungen an die Kommunikation wurden dabei im Voraus erhoben und ein Prototyp von uns entworfen, der in weiten Teilen in das endgültige Protokoll übernommen wurde.

Abnahme 4

Für die vierte Abnahme teilte sich das Team in zwei kleiner Gruppen auf. Die erste Gruppe beschäftigte sich mit dem Entwurf einer GUI (vgl. Abschnitt 7), die eine Nutzung sämtlicher Funktionalitäten ermöglichen sollte, ohne dass diese implementiert waren. Die zweite Gruppe beschäftigte sich mit der konkreten Implementierung der Kommunikation.

Abnahme 5

Im Zuge der fünften Abnahme wurde der Entwurf der GUI vervollständigt und die Kommunikation fertiggestellt. Beide Softwarekomponenten wurden anschließend verknüpft, sodass eine Erfassung wesentlicher Messgrößen des NXT möglich war und einem Nutzer

angezeigt werden konnte.

Abnahme 6

Für die sechste Abnahme war die Navigation mit Buttons und Tastatur über Distanzen sowie absolute und relative Winkel gefordert. Dadurch bedingt wurden sowohl der Regler als auch die Kommunikation und die GUI um entsprechende Parameter und Berechnungen erweitert. Zusätzlich zu den Anforderungen implementierte unsere Gruppe die Möglichkeit, den NXT alternativ über einen Controller zu steuern (vgl. Abschnitt 9.2.

Zwischenvortrag 2

Zu diesem Termin einigte sich das Team auf die zu realisierenden Extras, die neben dem Projektstatus vorgestellt werden sollten. Hierbei wurde sich zum einen für eine Navigation mit Hinderniserkennung und -behandlung und zum anderen für eine algorithmische Optimierung der Regler-Gewichte entschieden. Für diese Extras teilte sich das Team in zwei kleinere Gruppen auf, die die Implementierung je eines Extras weitgehend autonom vornahmen.

Abnahme 7

Für die siebte Abnahme war eine NXT-interne Positionsbestimmung erforderlich. Die ermittelte Position wurde an den PC gesendet und ein Navigationsbefehl zu konkreten Koordinaten dann in eine Drehung und zu fahrende Distanz umgerechnet. Dadurch wurde das Navigieren nach Koordinaten ermöglicht.

Abnahme 8

Die Gruppe entschied sich dazu, an beiden Extras parallel zu arbeiten. Dadurch wurden im Zuge der achten Abnahme Arbeitsfortschritte und Zwischenergebnisse präsentiert, ohne dass die endgültigen Anforderungen an das jeweilige Extra erfüllt waren. Die Navigationsgruppe band einen Ultraschallsensor an den NXT an, um Daten über die Umgebung zu erfassen. Diese Daten wurden an den PC gesendet und in einer Datenstruktur gespeichert, aus der eine Karte gezeichnet werden konnte. Zum Zeitpunkt der Abnahme konnten Hindernisse detektiert und auf der GUI angezeigt werden, wurden jedoch noch nicht behandelt.

Die Optimierungsgruppe hatte zu dem Zeitpunkt erste Testdurchläufe mit linearer Suche gestartet und sich im Wesentlichen mit der Datenerhebung befasst, ohne diese umfassend auszuwerten.

Abnahme 9

Für die neunte Abnahme wurde die Arbeit an beiden Extras beendet. Zur genauen Funktionalität und den Konzepten der Extras siehe Abschnitte 5.1 und 9.

3. Aufbau

Der physische Aufbau unseres Roboters kam maßgeblich durch Ernüchterung über die verfügbaren Bauteile sowie falsches Priorisieren von bestimmten Eigenschaften zustande. Anfänglich überprüften wir verschiedene Konzepte nach denen der Aufbau sich richten sollte, beispielsweise den Brick so weit oben wie möglich zu platzieren um dessen Masse zum Vorteil beim Balancieren zu nutzen. Ähnlich dem Prinzip des Balancierens eines Besens mit einer Hand. Des weiteren vermuteten wir, dass die gesamte Struktur so stabil und verwindungssteif wie nur irgend möglich sein müsse. Dazu kamen selbstverständlich noch die Vorgaben der Kunden, dass der Brick nicht höher als 13cm sitzen darf sowie ausreichend Sturzbügel.

Abbildung 1: Aufbau Segway

Nach nur wenigen Tests mit der LeJOS Segoway-Beispielklasse wurden schnell Komplikationen deutlich. Ein kopflastiger Roboter lässt sich zwar theoretisch einfacher balancieren, allerdings muss dafür mit ausreichender Geschwindigkeit auf Fallbewegungen reagiert werden, was mit den verfügbaren Motoren schlicht nicht möglich ist. Deren Schlupf sowie Ansprechverhalten ist dafür zu schlecht. Eben diese Nachteile waren es auch welche uns zwangen Räder mit lediglich 5,6 cm Durchmesser zu wählen, da dort das Spiel der Motoren durch den geringeren Umfang weniger ins Gewicht fiel. Wir nahmen den resultierenden Nachteil einer geringeren Höchstgeschwindigkeit in Kauf da der gewählte Umfang den besten Kompromiss bot. Des weiteren entschieden wir uns für eine standardmäßige Montage der Räder auf der Innenseite der Achsen. Dies ermöglicht ein deutlich schnelleres Drehen um die eigene Achse. Das Auslesen von korrekten und sinnvollen Daten aus dem, uns zur Verfügung stehenden, Gyrosensor erwies sich als sehr schwierig, weshalb wir wert darauf legten, dass dieser möglichst fest, direkt und schwingungsfrei befestigt wurde. Der letztendliche Aufbau wurde so vor allem nach den Kriterien Einfachheit, Gewichtsreduzierung und Balancierfreudigkeit geschaffen.

4. Projektorganisation

Die Organisation des Projekts wurde hauptsächlich durch äußere Bedingungen bestimmt. Es wurde eine vereinfachte Form des Scrums für die Struktur vorgegeben. Als Scrum-Master wurde Gregor Kobsik bestimmt. Unser Team bestand aus fünf Mitglieder, wobei das Praktikum mit 6 ECTS Punkten angerechnet wird, was einem Arbeitsaufwand von 180h pro Mitglied entsprechen soll. In Anbetracht dessen, dass das Praktikum während des Semesters neben anderen Veranstaltungen nur ein Fünftel der regulären Arbeitszeit

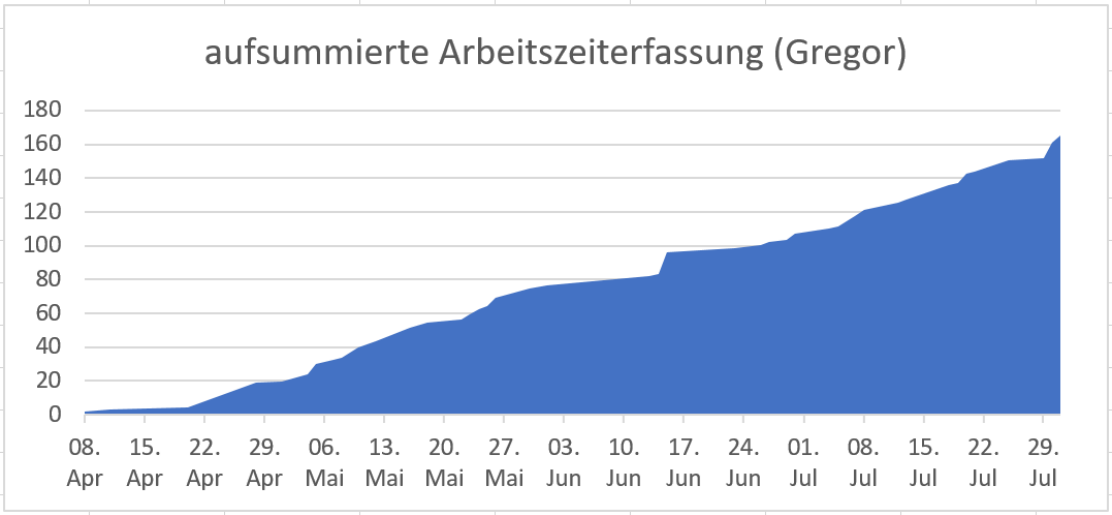


Abbildung 2: Aufsummierte Arbeitszeiterfassung von Gregor Kobsik.

in Anspruch nehmen sollte, entschieden wir uns für eine mittlere Arbeitszeit von 8h pro Woche und Teammitglied. Das entspricht einem Aufwand von 120h im Verlauf des gesamten Praktikums bis zum Abschlussvortrag. Die geforderte und die tatsächlich zur Verfügung stehende Arbeitslast stehen schon am Anfang des Projekts im Widerspruch. Eine additive Arbeitszeiterfassung eines Teammitglieds sieht man in Abbildung 2.

Um die Anforderungen trotzdem zu erfüllen, entschieden wir uns dazu, Redundanz zu minimieren, Schnittstellen zwischen einzelnen Komponenten klar zu definieren und so eine parallele Entwicklung von Features, trotz ihrer gegenseitigen Abhängigkeit, zu ermöglichen. Infolgedessen spezialisierten sich die Mitglieder des Teams auf einzelne Aspekte ihrer Arbeit, wie GUI, Regler oder die Kommunikation. So konnten einzelne Personen zwei Wochen lang, trotz der üblichen wöchentlichen Fristen, für eine Abnahme arbeiten, indem sie sich schon frühzeitig mit den geforderten Anforderungen auseinandersetzten. Hierdurch konnte der Zeitaufwand auch sehr gut im Voraus approximiert werden und die benötigten Ressourcen bei Bedarf umverteilt werden. Das Konzept fand seinen Höhepunkt in der Implementierung der zusätzlichen Features, der Navigation und dem evolutionären Algorithmus für den Regler, die vollkommen autonom voneinander in der Entwicklung und Funktionalität sind.

Des Weiteren muss man anmerken, dass alle Teammitglieder bei jedem Meeting anwesend waren. Dies erleichterte die Organisation und führte dazu, dass jeder sich an den Diskussionen und Grundsatzentscheidungen beteiligen konnte. Es wurden alle internen Deadlines eingehalten, wodurch die einzelnen Features problemlos ineinander eingebunden werden konnten.

4.1. Zentrale Konzepte

Den Mittelpunkt der Projektorganisation stellten die wöchentlichen Meetings dar. In diesen Meetings wurden nicht nur die vergangenen und zukünftigen Aufgaben besprochen, sondern auch Ideen ausgetauscht und Verbesserungen zur Implementierung kommuniziert. Der Erfolg der Meetings lässt sich auf eine feste Struktur zurückführen, die zunächst eine Agenda vor jedem Meeting voraussetzt. Ein weiterer Kernpunkt ist die Protokollführung. Jedes Treffen wurde abwechselnd von einem Teammitglied protokolliert. Das Protokoll bündelte unsere Entscheidungen in einem Dokument und wurde innerhalb von 24h in unserem internen Wiki veröffentlicht. Eine feste Struktur lässt sich auch innerhalb eines Meetings wiederfinden. Es wurde von dem Scrum-Master vorbereitet und geleitet, wobei es durch wiederkehrende Tagesordnungspunkte gegliedert worden ist. Jedes Treffen fing mit einer Eröffnung an, gefolgt von der Verlesung des letzten Protokolls, den vergangenen Aufgaben, der vergangenen Präsentation, sowie der Aufteilung der zukünftigen Aufgaben und der zukünftigen Präsentation. Es folgte der Tagesordnungspunkt Sonstiges und das Meeting wurde durch die Verlesung des aktuellen Protokolls beendet.

Ein weiteres Kernkonzept der Organisation stellt Pair Programming dar. Es zeichnet sich durch Partnerarbeit aus; hierbei programmiert eine Person, während die andere daneben sitzt und den Programmierer bei der Arbeit beobachtet. Auch wenn sie in erster Linie nicht aktiv programmiert, so dient sie vor allem der Qualitätssicherung. Sie muss sowohl den Code verstehen können, als auch mögliche Fehler direkt entdecken. Dies verkürzt im Anschluss die Fehlersuche und bietet ein simultanes Code Review. Darüber hinaus werden die Kernkonzepte gemeinsam besprochen und es fließen gemeinsame Ideen in die Implementierung. Studien belegen eine tatsächliche Verbesserung der Codequalität, sowie weitere Lerneffekte für beide Beteiligten, da sie ihre Erfahrungen direkt bei der Arbeit austauschen können.

Die treibende Kraft für die Entwicklung des NXT-Roboters stellten die Abnahmen dar. Jede Abnahme hatte einen fest definierten Funktionsumfang, den wir erfüllen mussten. Da die zu erfüllenden Aufgaben ausreichend genau spezifiziert worden sind, konnten wir testorientiert entwickeln. Zu jeder Abnahme haben wir eine Reihe an Tests definiert, die unser Roboter zunächst intern und danach extern in der Abnahme bestehen musste. Die Implementierung orientierte sich hierbei auf der Erfüllung der Funktionalität. Die folgende Behebung der potentieller Fehler orientierte sich hingegen an der Erfüllung der von uns definierten Tests.

4.2. Weitere Konzepte

Ein weiteres Konzept, das am Anfang sehr erfolgreich angewendet wurde, ist das Planing Poker. Hierbei schätzt jedes Teammitglied mittels spezieller Karten den Arbeitsaufwand in Stunden für ein Feature. In unserem Fall verwendeten wir die Karten mit 1, 2, 3, 5 und 8 Augenzahlen. Die Karten werden verdeckt gelegt und dann gleichzeitig umgedreht. Die Personen mit den niedrigsten und höchsten Zahl muss seine Entscheidung motivieren und so eine Diskussion anfangen. Dies ist ein iteratives Verfahren, das in unserem Fall schnell terminierte und gleichzeitig durch die Diskussion ein natürliches Brainstorming

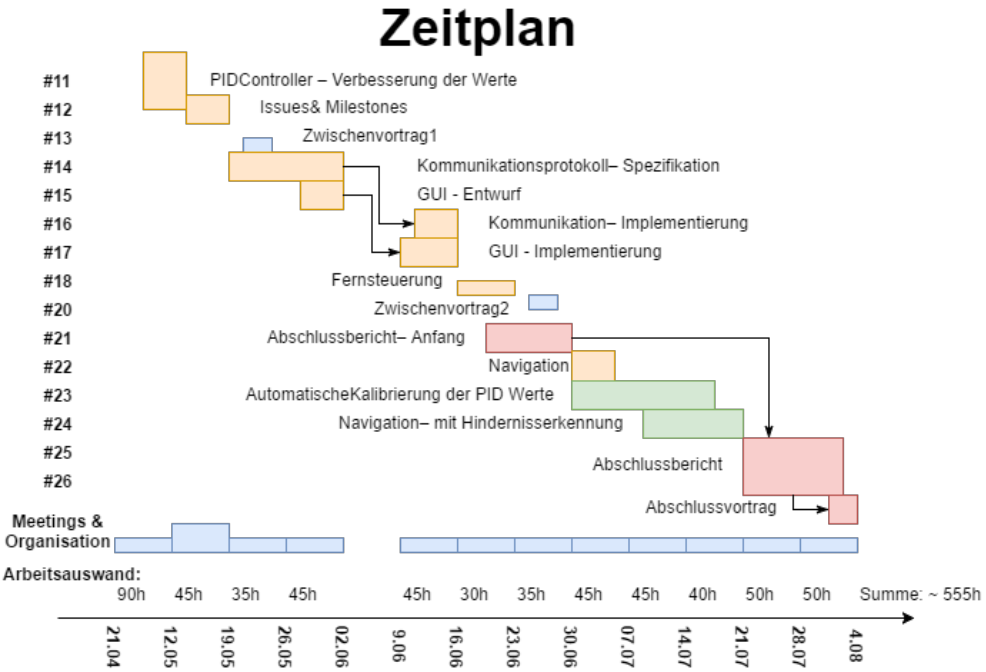


Abbildung 3: Initialer Zeitplan für die Erfüllung der Aufgaben.

hervorbrachte. Das Verfahren fand keine Anwendung mehr, nachdem wir für eine Abnahme den vollständigen Zeitplan erstellten, da wir hierfür eine ausführliche Diskussion zu jedem Thema führen mussten. Hierdurch verlor das Projekt einiges an Dynamik, da wir uns einer statischen Durchführung des vorgegebenen Plans widmeten. Abbildung 3 zeigt den initialen Zeitplan, wobei auf der y-Achse die Issues bzw. der organisatorische Aufwand eingetragen ist. Die x-Achse stellt den zeitlichen Verlauf und die Flächen die geschätzte Verteilung der Zeit auf die einzelnen Aufgaben, wobei die Diskretisierung auf 2,5h pro Flächeneinheit gewählt wurde.

Neben den vorherigen Konzepten gehört auch `GitLab` als Verwaltungs- und Organisationswerkzeug dazu. Durch `GitLab` konnten wir die benötigten Issues erstellen und uns so immer an einer aktuellen Übersicht über die zur Durchführung benötigten Schritte orientieren. Zusätzlich erlaubte es eine Nachverfolgung der aufgewendeten Zeit für die einzelnen Teilaufgaben. Das Wiki ermöglichte es uns, die Meetingprotokolle einfach zu veröffentlichen und allen Mitgliedern zur Verfügung zu stellen. Der größte Vorteil des `Git` ist allerdings die Versionskontrolle und Codeverwaltung. Hierdurch konnten wir durch die Branches an verschiedenen Features gleichzeitig und unabhängig voneinander arbeiten. Insbesondere erlaubte es uns, die Versionskontrolle auf Features auszulagern und erst später in den Hauptcode einzubinden, nachdem diese getestet wurden und ein Review passierten. Als Git-Workflow verwendeten wir Feature-Branches. Die Entwicklung eines Features fand also ausschließlich in einem dazu angelegten Branch statt, der `master`-Branch enthält nur Merge-Commits. Das Arbeitswerkzeug erlaubte zudem, auch

überflüssige Elemente wie beispielsweise ein Dateiverwaltungssystem für den NXT zu implementieren, aber nie einzubinden, da im späteren Verlauf des Projekts keine sinnvolle Verwendung dafür gefunden wurde.

5. Regler

Als Grundlage für den von uns entwickelten Regler diente die `Segoway`-Klasse von LeJOS. Deren Regler besteht im wesentlichen aus einem PD- und einem PI-Regler mit den Einflussgrößen Neigungswinkel `gyroAngle`, Winkelgeschwindigkeit `gyroSpeed`, zurückgelegter Strecke `motorPos`, Geschwindigkeit `motorSpeed` sowie der Zielgeschwindigkeit `motorControlDrive`. Der `Segoway`-Regler weist einige Schwächen auf. Beispielsweise fängt der NXT beim Balancieren an zu „wandern“, also sich über Zeit immer weiter von seiner Startposition zu entfernen, ohne dazu angewiesen worden zu sein. Durch eigene Test konnte dieses Verhalten darin begründet werden, dass `Segoway` die Winkelgeschwindigkeiten des Gyros aufaddiert um den Neigungswinkel des NXT zu berechnen. Wie in Abschnitt 2.1 erläutert, ist dieser Winkel fehlerbehaftet und der Fehler wächst mit der Zeit. Um die Abweichung in der Neigung zu Kompensieren vergrößert der `Segoway`-Regler die Distanzabweichung in die entgegengesetzte Richtung und „wandert“ folglich.

Als Konsequenz entwickeln wir unseren eigenen Regler von Grund auf, übernahmen aber sinnvolle Konzepte des `Segoway`-Reglers wie die Fall-Detection und den grundlegenden Aufbau des Reglers. Die Funktionalität ist in zwei Klassen aufgetrennt worden, nämlich `Sensordata`, welche sich um die Erhebung der Sensordaten kümmert, sowie `MotorController`, welche den eigentlichen Regler beinhaltet. Abbildung 4 zeigt ein vollständiges Block-Diagramm unseres Reglers. Abgesehen von `distanceTarget` und `headingTarget`, welche Soll-Zustände für die Navigation darstellen, gibt es nur noch vier Einflussgrößen für den Regler.

`gyroSpeed` ist die aktuelle Winkelgeschwindigkeit des Gyro-Sensors. Im Gegensatz zum `Segoway`-Regler keine eigene Rekalibrierung des Gyro-Sensors durch, da `GyroSensor.getAngularVelocity` bereits selbst alle fünf Sekunden eine Rekalibrierung vornimmt, um der Drift entgegenzuwirken.

`gyroIntegral` ist kein eigentlicher Winkel mehr, sondern ein gedämpftes Integral. Dies hat den entscheidenden Vorteil, dass die Drift des Winkels in der Dämpfung verschwindet. Der Null-Winkel stellt sich somit an der Neigung ein, in der sich der NXT die meiste Zeit befindet, was die aufrechte Position ist. Eine starke, plötzliche Auslenkung wird dadurch gut erkannt und kann korrigiert werden. Eine andere Methode, die versucht wurde, um mit der Drift umzugehen, war die Verwendung eines zeitlich beschränkten Integrals über die Winkelgeschwindigkeiten der letzten n Ticks. Aufgrund des erhöhten Speicherverbrauches um die letzten n Winkelgeschwindigkeiten vorzuhalten, erwies sich diese Methode weniger effektiv.

`motorSpeed` ist wie beim `Segoway`-Regler der Durchschnitt der Differenz in der `motorDistance` über die letzten vier Ticks.

`motorDistance` wird einerseits verwendet, um den NXT an einer Stelle zu halten und Positionsänderung durch Ausgleichsbewegungen auszugleichen, andererseits

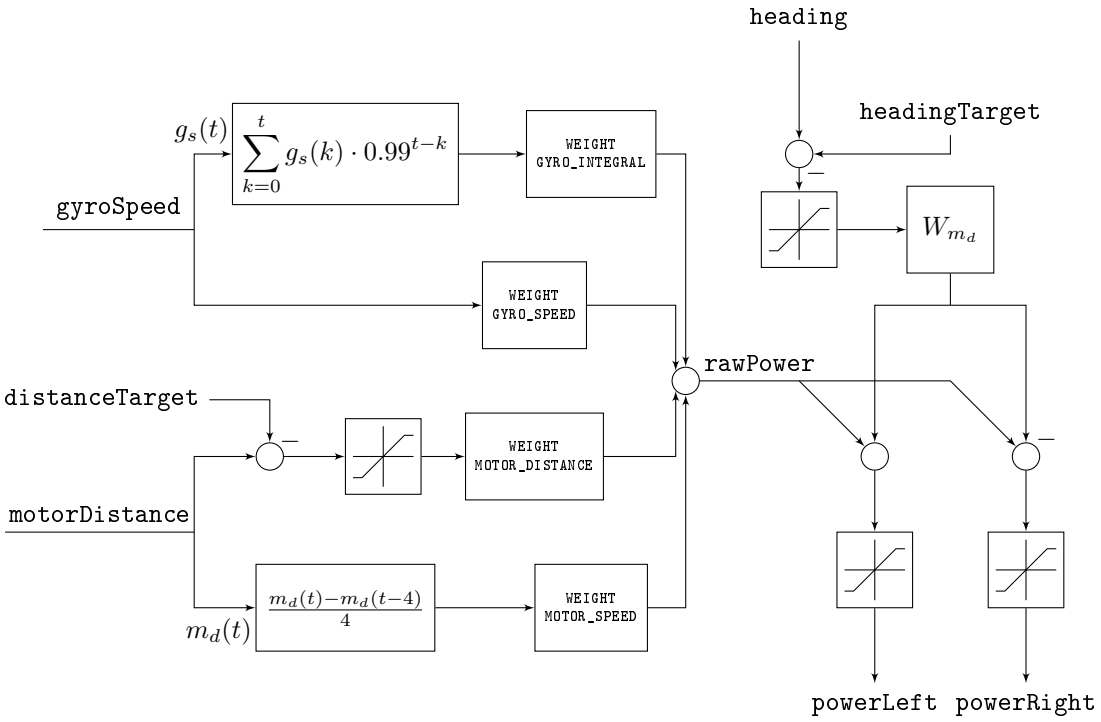


Abbildung 4: Wirkungsplan des PID.

wird hierüber die Bewegung gesteuert. Der Regler versucht die Differenz zwischen `motorDistance` und `distanceTarget` zu minimieren. Durch das Setzen von `distanceTarget` kann somit eine Bewegung erzwungen werden. Die Differenz wird auf das Intervall $[-$`MAX_DISTANCE_INFLUENCE`$,$`MAX_DISTANCE_INFLUENCE`$]$ beschränkt, um zu verhindern, dass ein großer Wert für `distanceTarget` den Regler zu einer entsprechenden Reaktion veranlasst, welche ein Balancieren unmöglichen machen würde. Als ein guter Wert für `MAX_DISTANCE_INFLUENCE` haben sich 15 cm herausgestellt.

Um wechselnde Batteriespannungen zu kompensieren, wurde ein Skalierungsfaktor von 7.5 V/`Battery.getVoltage()` eingeführt. Der Wert von 7.5 V wurde gewählt, da unter 7 V der NXT in vorherigen Tests träge reagierte und nicht in der Lage war, über einen längeren Zeitraum aufrecht zu stehen.

Für die Lenkung existiert analog zu `distanceTarget` ein `headingTarget`. Der Regler minimiert ebenso die Differenz von `heading` zum `headingTarget` und durch Setzen von `headingTarget` wird eine entsprechende Drehung ausgelöst.

5.1. Automatische Kalibrierung der Gewichte

Das sich unser Regler von dem `Segoway`-Regler unterscheidet und andere Eingangsgrößen verwendet, mussten wir neue Gewichte für die jeweiligen Werte finden. Eine Simulation kam nicht infrage, da keiner von uns besondere Erfahrung mit Regelungstechnik hatte und das System viele Unbekannte wie den Schlupf der Reifen, die Leistung der Motoren in Ab-

hängigkeit der Akkuspannung oder die Genauigkeit des Gyro-Sensors aufweist. Das händische Einstellen der Regler-Gewichte stellte sich als mühsam und umständlich heraus. Daher wählten wir die automatische Kalibrierung der Gewichte als eines unserer Extras.

Zeit	Aktion
-00:05 - 00:00	stillstehen (standard PID)
00:00 - 00:01	stillstehen (neue Werte)
00:01 - 00:06	fahre 20 cm nach vorne
00:06 - 00:08	drehe um 180°
00:08 - 00:13	fahre 20 cm nach vorne
00:13 - 00:15	drehe um −180°
00:15 - 00:18	fahre 10 cm nach vorne
00:18 - 00:20	drehe um −180°
00:20 - 00:23	fahre 10 cm nach vorne
00:23 - 00:25	drehe um 180°

Abbildung 5: Ablaufplan der automatisierten Tests.

Ein großes Problem der händischen Kalibrierung war, dass es kein Unterscheidungsmerkmal gab, woran sich entscheiden ließe, welches von zwei Gewichten besser ist. Darum entwarfen wir einen festen Testablauf von 25 s Dauer, welcher in Abbildung 5 zu sehen ist. Dieser Testablauf kann automatisch ausgeführt werden. Während des Tests sammelt der NXT Daten und übermittelt nach Ende des Tests an den PC. Ein Test endet entweder nach 25 s oder sobald der NXT umgefallen ist. Die gesammelten Daten sind die durchschnittliche Batteriespannung, durchschnittliche Abweichung von `distanceTarget` und `headingTarget`, sowie die aufrecht gestandenen Zeit. Diese Daten zusammen mit den vier Gewichten für den Testlauf bilden ein 8-Tupel, welches in einer CSV-Datei für die spätere Auswertung gespeichert wird. Zur Auswahl der durchzuführenden Testläufe wurden zwei Algorithmen implementiert.

5.1.1. Binäre Suche

Bei der Optimierung über binäre Suche wird jedes Gewicht separat maximiert. Da der Einfluss der einzelnen Gewichte nicht unabhängig ist, ist es notwendig alle Gewichte wiederholt zu optimieren, bis nur noch wenig Änderung eintritt.

Um ein Gewicht zu optimieren, wird der bisherige Wert x_{old}, sowie ein ε_i betrachtet. Es werden drei Testläufe x_i, $x_i - \varepsilon$ und $x_i + \varepsilon$ durchgeführt. Der Wert mit dem besten Testergebnis wird übernommen und ε_i halbiert, woraus sich die Iterationsvorschift

$$\begin{aligned} x_0 &= x_{old} \\ \varepsilon_0 &= x_{old} \\ x_{i+1} &= \operatorname*{arg\,max}_{x \in \{x_i, x_i - \varepsilon_i, x_i + \varepsilon_i\}} fitness(x) \\ \varepsilon_{i+1} &= \frac{\varepsilon_i}{2} \end{aligned}$$

ergibt. Die Konvergenz dieses Verfahrens ist quadratisch, wodurch sich eine geringe Anzahl an Testläufen ergibt.

In der Praxis stellte sich heraus, dass die Testergebnisse sehr stark streuen und einzeln für sich nicht gut reproduzierbar sind. Dadurch läuft die binäre Gefahr, sich sehr schnell

einmal für die falsche Seite zu entscheiden und anschließend das Maximum nicht mehr finden zu können.

5.1.2. Evolutionärer Algorithmus

Im folgenden wird ein Set von PID-Gewichten als Individuum bezeichnet. Eine Population setzt sich aus einer Menge von Individuen zusammen.

Der Ablauf des evolutionären Algorithmus ist folgender: Zu Beginn wird eine Population erzeugt, welche sich zu 20 % aus den besten Individuen aus der Datenbank und zu 80 % aus zufälligen Mutationen der bereits in der Population enthaltenen Individuen zusammen setzt. Anschließend wird jedes Individuum getestet.

Für die nächste Generation werden die besten 20 % der Population beibehalten, 40 % werden durch Kreuzung der verbliebenen Individuen wieder aufgefüllt. Die restlichen 40 % stammen aus zufälligen Mutationen. Der Evolutionsschritt wird für einige Generationen wiederholt.

Der evolutionärer Algorithmus zeigte sich im Gegensatz zur binären Suche weniger anfällig für die hohe Streuung der Testergebnisse und lieferte wesentlich diversere Testläufe. Dadurch konnte die Experimentalzeit effektiver genutzt werden und mehr verwertbare Daten entstanden.

5.1.3. Auswertung

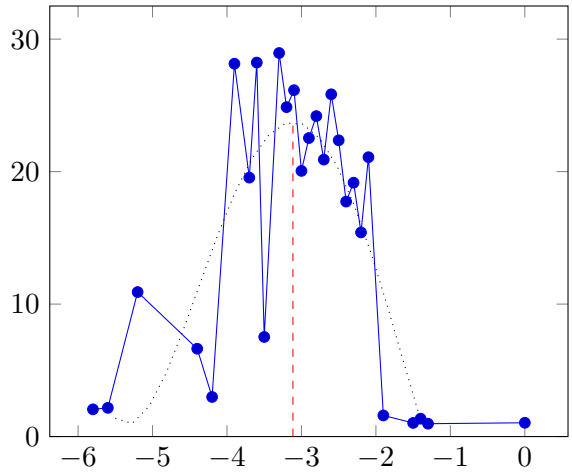
Insgesamt wurden 811 Tests durchgeführt. Von diesen wurden 17 verworfen, deren durchschnittliche Akkuspannung unter 6.6 V lag. Unterhalb dieser Spannung war der NXT, trotz Normalisierung der Spannung im Regler auf 7.5 V, in keinem Testlauf in der Lage, aufrecht zu bleiben. Diese Daten stammen zur Hälfte aus der ersten Testserie mit der binären Suche und zur anderen Hälfte aus der zweiten Serie mit dem evolutionärer Algorithmus. Die Daten können gemeinsam ausgewertet werden, da beide Verfahren nur eine Auswahl der Testläufe treffen, aber auf deren Ergebnis keinen Einfluss haben.

Bei der Auswertung wurde einheitlich Gleichung (1) als Fitness-Funktion verwendet. Diese legt ein hohes Gewicht auf ruhiges Balancieren, wobei Umfallen in jedem Fall schlechter ist. Durch den linearen Einfluss der Zeit beim Umfallen kann auch bei schlechten Werten noch ein Aussage über deren relative Güte getroffen werden.

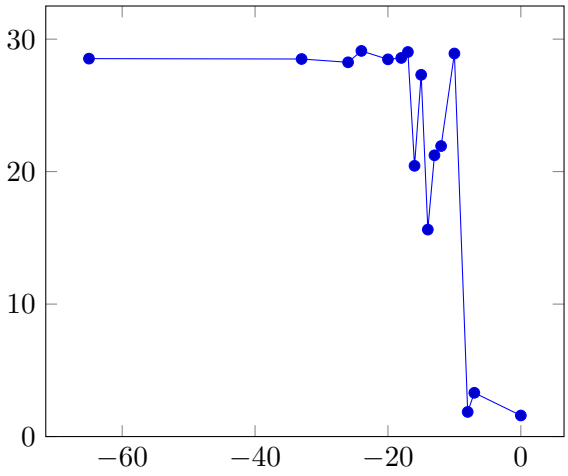
$$fitness(t, \bar{U}, \bar{\Delta}_{distance}, \bar{\Delta}_{heading}) = \begin{cases} t & : t < 25 \\ 25 + \frac{75}{1+\bar{\Delta}_{distance}+\bar{\Delta}_{heading}} & : \text{sonst} \end{cases} \quad (1)$$

Das Ergebnis der Auswertung, aufgeteilt nach jedem Gewicht ist in Abb. 6 zu sehen. Die Fitness-Werte wurden jeweils nach Gewicht gruppiert und der Durchschnitt je Gruppe betrachtet. Per polynomischer Regression wurde für jedes Gewicht ein Optimum bestimmt. Für `WEIGHT_GYRO_INTEGRAL` ist diese Regression wenig sinnvoll, da während der Tests die linke Flanke des Optimums nicht gefunden wurde. Es lässt sich trotzdem ein obere Grenze von -20 für `WEIGHT_GYRO_INTEGRAL` in den Daten erkennen.

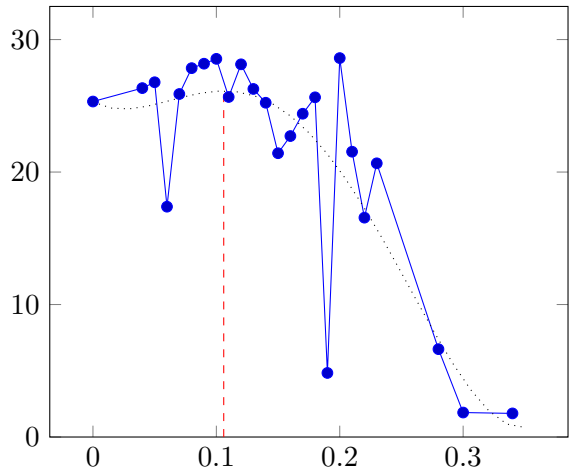
Die als Ergebnis der Gewichtsoptimierung gefundenen sind in Tabelle 1 dargestellt.



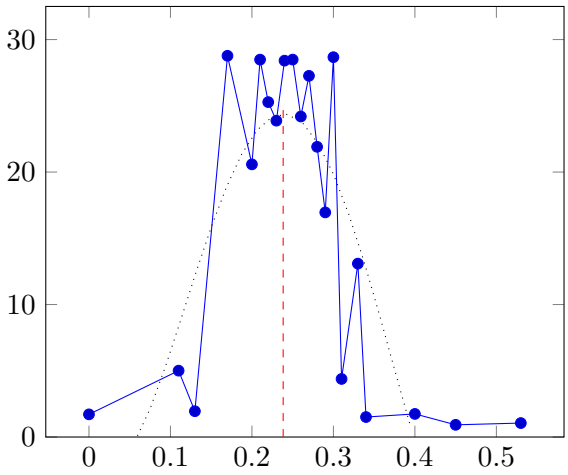
(a) `WEIGHT_GYRO_SPEED`: Maximum der polynomischen Regression bei 3,11437.



(b) `WEIGHT_GYRO_INTEGRAL`: Regression nicht sinnvoll.



(c) `WEIGHT_MOTOR_DISTANCE`: Maximum der polynomischen Regression bei 0,105979.



(d) `WEIGHT_MOTOR_SPEED`: Maximum der polynomischen Regression bei 0,238327.

Abbildung 6: Fitness pro Gewicht.

Gewicht	**Alt**	**Neu**
`WEIGHT_GYRO_SPEED`	-2.8	-3.11437
`WEIGHT_GYRO_INTEGRAL`	-13	-20
`WEIGHT_MOTOR_DISTANCE`	0,15	0.105979
`WEIGHT_MOTOR_SPEED`	0,225	0.238327

Tabelle 1: Optimale Gewichte nach der Auswertung.

6. Architektur

Das Gesamtprojekt gliedert sich in zwei Teilprojekte, das PC-Projekt auf PC-Seite und das NXT-Projekt auf NXT-Seite. Diese Unterscheidung wurde bereits sehr früh getroffen, da sich die Hardware und insbesondere die zur Verfügung stehende Software auf beider Seiten sehr unterscheidet (vgl. Abschnitt 2.1). Die Brücke zwischen den beiden Projekten bildet die abstrakte Kommunikationsklasse, von der beide Implementierungen der Kommunikation erben. Die entsprechenden UML-Diagramme sind im Appendix unter den Abbildungen 9 bis 18 zu finden.

6.1. NXT

Für das NXT-Projekt wurde das prozedurale Paradigma gewählt. Bis auf Ausnahmen sind alle Methoden und Variablen als `static` deklariert. Klassen dienen nur zur logischen Strukturierung des Quellcodes und werden nicht instanziiert. Dies erlaubt der JVM, mehr Optimierungen vorzunehmen. In Java können Methoden, die nicht `final` oder `static` sind, von Unterklassen überschrieben werden (Laufzeit-Polymorphie). Aus diesem Grund kann die Bindung der Methoden nicht durch den Compiler erfolgen, sonder geschieht zur Laufzeit, was zeitgleich einen Overhead erzeugt. Aufgrund der limitierten Rechenkapazität des NXT und Echtzeitanforderung des Balancierens ist es wichtig, unnötigen Overhead zu vermeiden. Der Verzicht auf Objektorientierung auf NXT-Seite brachte wenig Probleme mit sich, da durch die LeJOS JVM insgesamt nur wenige nutzbare Bibliotheks-Klassen zur Verfügung stehen und Lambda-Ausdrücke generell nicht unterstützt werden. Zudem kann jede Klasse einem Hardwarebestandteil eindeutig zugeordnet werden. Es ist nicht sinnvoll mehrere Objekte einer Klasse besitzen zu können, da die Erfüllbarkeit der Funktionalität an äußere Gegebenheiten gebunden ist.

Da auf dem NXT mehrere Threads gleichzeitig laufen, die sich gegenseitig jedoch nicht blockieren dürfen (z.B.: Kommunikation und Balancieren) wurde darauf geachtet, dass alle Threads die gleiche Priorität erhalten, da sonst ein höher priorisierter Thread die anderen blockieren kann (vgl. Abschnitt 2.1).

6.2. PC

Für das PC-Projekt wurde auf die objektorientierte Programmierung gesetzt. Im Gegensatz zu dem NXT ist man auf der PC-Seite nicht durch die Hardware limitiert und kann einen Overhead zur Laufzeit in Kauf nehmen, wenn man hierdurch schneller und fehlerfreier das gewünschte Ergebnis erzielen kann. Schon die Implementierung der GUI benutzt Objekte, die wie z.B. Buttons mehrfach vorzufinden sind (vgl. Abschnitt 7). Auch die Verwendung von Interfaces ermöglichte uns eine intuitive Anbindung der Klassen an das vorhandene Eventsystem. Für eine gleichzeitige Ausführung verschiedener Aufgaben, wie der Kommunikation, der GUI oder der Navigation, wurden jeweils eigene Threads erstellt, die unabhängig voneinander arbeiten. Hierbei wurde darauf geachtet, alle Aufgaben, die rechenintensiv sind und nicht vom NXT übernommen werden müssen, auf den PC auszulagern. So wird die Navigation auf dem Rechner berechnet und lediglich ein

Befehl für die Bewegung an den NXT übermittelt (vgl. Abschnitt 9). Das PC-Projekt dient als ein logischer Master, der den NXT steuert.

7. GUI

Erste Konzepte für eine grafische Benutzeroberfläche wurden bereits frühzeitig im Verlauf dieses Projekt besprochen. Das Aufstellen des Lasten- sowie Pflichtenhefts war dabei von entscheidender Bedeutung, da hier der Umfang der notwendigen Funktionen deutlich wurde. Die drei wesentlichen Bereiche Überwachungsgrößen, Kommunikation und Funktionen wurden früh definiert sowie erste Lösungsansätze erarbeitet.

Die Namen sind trivial gewählt: "Überwachungsgrößen"(oben im Bild) zeigt Informa-

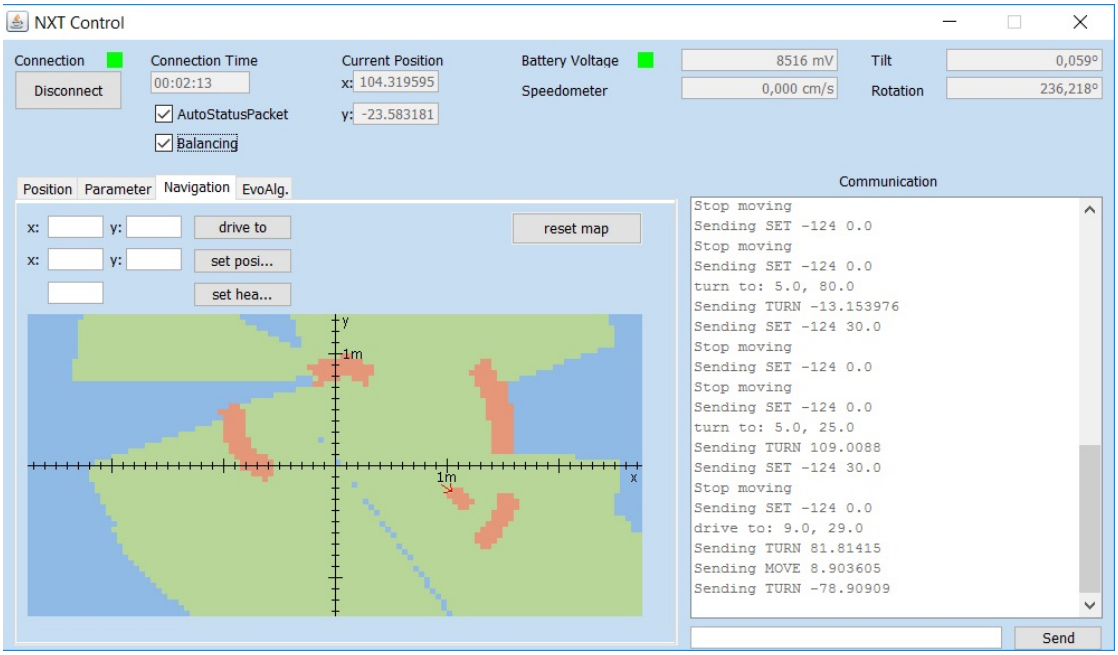


Abbildung 7: GUI NXT Control

tionen, beispielsweise Batteriespannung, Neigung, Ausrichtung, Geschwindigkeit, Verbindungsdauer u.Ä., sowie alle nötigen Bedienelemente zum Verbindungsaufbau. "Kommunikation"(rechts) beschreibt schlichtweg eine Konsole mit Ein- und Ausgabefeldern die zum einen eine Bestätigung für getätigte Befehle und empfangene Daten ist, sowie die Möglichkeit bietet in Textform alle Bedienelemente der GUI manuell einzugeben. "Funktionen"(links) ist ein Reiterfeld das vor allem durch seine Modularität einen großen Umfang bietet wie etwa die Parametrierung(u.a. der PID-Werte) aber auch alle nötigen Felder und Bedienelemente zur Navigation und zum Fahren. Die Verwendung eines solchen Tab-Views erlaubt das Hinzufügen und Entfernen von Funktionen und Extras ohne das grundlegende Bedienkonzept der graphisches Benutzeroberfläche ändern zu müssen. Die Oberfläche ist einheitlich in englischer Sprache geschrieben. Initial sind alle But-

tons, außer dem "Connect"-Button, deaktiviert und werden erst bei erfolgreichem Verbindungsaufbau zu einem NXT freigegeben. Ab diesem Zeitpunkt beginnt die Uhr unter "Connection Time" zu laufen und da der NXT von sich aus Status Pakete beginnt zu verschicken werden auch die restlichen Informationsfelder befüllt. Die Beschriftung der Buttons in Kombination mit passenden Labels ermöglicht eine Verwendung der GUI ohne eine Einweisung des Nutzers. Es wurde an sinnvollen Stellen Checkboxen statt Buttons verwendet oder Dropdown-Menüs statt Freitextfeldern. Der Zugriff auf nicht erlaubte Felder wie beispielsweise dem Ausgabefeld der Konsole sind deaktiviert und der Nutzer wird bei unzulässigen Eingaben auf seinen Fehler hingewiesen. Dem Reiter "Navigation"kommt beim Arbeiten am NXT mit der GUI besondere Bedeutung zuteil. Die dargestellte Karte bietet eine Übersicht der Umgebung sowie die aktuelle Position. Der Ursprung des Koordinatensystems bildet der Punkt (0,0), welcher dem Startpunkt des NXT zum Beginn des Balancieren entspricht. Der NXT selbst wird durch einen roten Pfeil symbolisiert dessen Richtung der Ausrichtung des Roboters entspricht. Hierbei wird initial eine Ausrichtung in positiver Y-Achse angenommen. Des weiteren bietet der Reiter Bedienelemente zur Steuerung (z.B. drive to) oder auch zum manuellen Setzen der Position und der Ausrichtung. Interessant wird das Fenster vor allem in Kombination mit der Navigation mit Hinderniserkennung (vgl. Abschnitt 9.3), denn die erfassten Daten des Ultraschallsensors werden direkt in die Karte gespeist und farblich dargestellt. Auch hier wurde bei der Implementierung besonderes Augenmerk auf Intuitivität gelegt, so erlaubt die Karte eine interaktive Nutzung, indem der Benutzer über ein Mausklick die Position definieren kann, die der NXT anfahren soll.

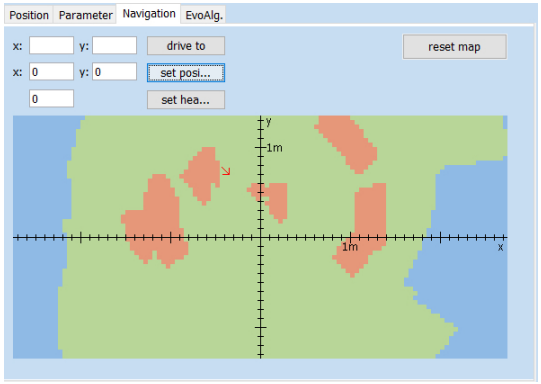


Abbildung 8: Reiter Navigation

8. Kommunikation

Um die Kommunikation zwischen NXT und PC zu realisieren, wurde gruppenübergreifend ein allgemeingültiges Kommunikationsprotokoll entworfen, das Möglichkeiten zu gruppeninternen Erweiterungen offenlässt. Da die endgültige Fassung dieses Protokolls zu großen Teilen auf unserem zuvor ausgearbeiteten Prototypen basiert, wird sie hier umfangreich vorgestellt. Das Protokoll spezifiziert dabei die Art der Datenübertragung und die allgemeine Codierung von Befehlen. Der Grundgedanke dieses Protokolls ist die Ermöglichung einfacher Kommunikation zwischen PCs und NXTs, die von verschiedenen Gruppen stammen.

8.1. Allgemein

Das allgemeine Kommunikationsprotokoll definiert die Datenübertragung als Stream-basiert über das `NXTComm`-Interface. Das Interface bietet mit dem Übertragungsmodus `PACKET` eine sichere Datenübertragung mit Header-Daten und umfangreiche Funktionalitäten zum Verbindungsauf- und -abbau sowie zum Datenaustausch. Empfohlen wird hierbei die Verwendung von DataInput- und DataOutput-Streams für die Handler. Eine Implementierung soll eine Übertragungsrate von fünf Commands pro Sekunde in beide Richtungen gewährleisten können. Paketverluste sind nur bei hoher Auslastung auf der Seite des Slaves möglich und werden abhängig von der Implementierung entweder ignoriert oder quittiert und dem PC angezeigt.

8.1.1. Commands

Wesentlicher Bestandteil des Kommunikationsprotokolls ist die allgemeingültige Codierung von Nachrichten. Eine Nachricht beginnt immer mit einem Command, gefolgt von eventuellem Payload, der beliebige Länge haben kann, aber für jeden Command fest definiert ist. Der Handler eines Commands ist dafür verantwortlich, die richtige Anzahl an Bytes zu lesen, um eine korrekte Interpretation der Daten zu gewährleisten. Jeder Command wird in Form einer CommandID vom Typ byte codiert und einer Nachricht vorangestellt.

Die genaue Codierung der Commands, ihre Struktur und die Kommunikationsrichtung ist Tabelle 2 im Anhang zu entnehmen.

Set

Der erste allgemeingültige Command ist Set. Set wird vom PC and den NXT gesendet, um den Wert eines Parameters neu zu setzen und wird gefolgt von der Codierung eines zulässigen Parameters sowie den zu setzenden Werten.

Get

Der Get-Command wird vom PC an den NXT gesendet, um den aktuellen Wert eines Parameters abzufragen, und wird ebenfalls von der Codierung eines zulässigen Parameters gefolgt.

GetReturn

Der GetReturn-Command kennzeichnet die Antwort des NXT auf eine Get-Anfrage des PCs. Der GetReturn-Command wird vom NXT an den PC gesendet und wird von der Codierung des gefragten Parameters sowie den Werten gefolgt.

Move

Der Move-Command wird vom PC an den NXT gesendet und wird von einer Distanz in Zentimetern gefolgt, um die sich der NXT in gerader Richtung vorwärtsbewegen soll.

Turn

Der Turn-Command wird vom PC an den NXT gesendet und wird von einem Winkel in Grad gefolgt, um den sich der NXT nach links (mathematischer Drehsinn) drehen soll.

MoveTo
Der MoveTo-Command ist reserviert. Er war ursprünglich dazu gedacht, den NXT dazu zu veranlassen, sich zu einer Koordinate zu bewegen. Das Navigieren zu einer Koordinate soll durch PC-interne Umrechnung über Move- und Turn-Commands realisiert werden. Als Konsequenz ist das Senden dieses Commands zurzeit ein Fehler.

Balancing
Dieser Command wird von einem Boolean gefolgt, der den NXT anweist, das Balancieren zu starten oder zu stoppen. True wird hierbei als Anweisung zum Starten interpretiert.

LogInfo
Der Command wird vom NXT zum PC gesendet, um kurze Nachrichten übermitteln zu können, die nicht über andere Commands realisierbar sind wie beispielsweise Debug-Nachrichten. Der String, der diesem Command folgt, wird in UTF-8 kodiert.

ErrorCode
Dieser Command überträgt spezifizierte Errors des NXT an den PC.

Disconnect
Dieser Command wird vom PC an den NXT gesendet, um einen sicheren Verbindungsabbau zu initialisieren. Der Verbindungsabbau ist in diesem Fall eindeutig erwünscht.

ProtocolVersion
Dieser Command wird initial vom NXT zum PC gesendet und von einem byte gefolgt, welches die Protokollversion des NXT kodiert. Die Protokollversionen sind dabei intuitiv mit den Gruppennummern kodiert. Das allgemeine Kommunikationsprotokoll wird mit 0 kodiert. Der PC entscheidet nach Empfang dieses Commands, ob er nur die allgemeinen oder zusätzlich die gruppenintern spezifizierten Commands und Parameter verwendet. Da gruppenfremde NXTs die internen Spezifikationen nicht kennen, führt ein Senden nicht-allgemeiner Commands und Parameter zu Fehlern in der Verarbeitung und ist daher illegal.

Die CommandIDs im Bereich von 128-255 sind für gruppeninterne Definitionen offen. Dies ermöglicht eine problemlose Erweiterung der allgemeingültigen Commands bei Bedarf, ohne Kollisionen mit internen Definitionen zu riskieren. Interne Erweiterungen der Commands können bei speziellen Extras notwendig sein, da in diesem Fall eine Erweiterung der allgemeingültigen Commands nicht sinnvoll ist.

8.1.2. Error-Codes

Ein ErrorCode-Command wird immer von einem codierten Error gefolgt. Allgemeingültige Error-Codes existieren zwei. Auch bei den Error-Codes ist der Bereich von 128-255 für gruppeninterne Erweiterungen offengelassen.

NXTFallen
Der NXT sendet diesen Error, wenn das Balancieren aufgehört hat, ohne dass ein entsprechender Command gesendet wurde.

PacketLoss
Dieser Error kann vom NXT gesendet werden, wenn ein Paketverlust erkannt wurde und quittiert werden soll.

8.1.3. Parameter

Um grundlegende Kommunikation mit gruppenfremden NXTs zu ermöglichen, muss eine Schnittmenge an Parametern allgemeingültig kodiert werden. Dadurch können wesentliche Informationen des NXT wie Batteriespannung und Geschwindigkeit versionsunabhängig abgefragt werden. Spezifischere Parameter, die beispielsweise bei der Gewichtung der Werte des Reglers eine Rolle spielen, werden hingegen gruppenintern im Bereich von 128-255 kodiert. Die genaue Kodierung der Parameter sowie Typ und Einheit ihrer Werte ist Tabelle 3 im Anhang zu entnehmen.

BatteryVoltage/GyroAngle/TachoLeft/TachoRight
Diese Parameter kennzeichnen die aktuellen Messgrößen der verwendeten Hardware. Da diese Hardware von jeder Gruppe verwendet wird, können die Parameter allgemeingültig definiert werden.

Heading
Dieser Parameter bezeichnet die aktuelle Ausrichtung des NXT in Grad. Seine Startposition ist hierbei als 0 definiert und die Ausrichtung im mathematischen Drehsinn nach links berechnet.

Position
Dieser Parameter kennzeichnet die aktuelle vom NXT berechnete Position als Koordinaten. Als Ursprung wird hierbei initial die Startposition festgelegt.

MovementSpeed
Die aktuelle, aus der Geschwindigkeit aller Motoren gemittelte Geschwindigkeit des NXT.

StatusPacket

Eine Bündelung mehrerer, häufig abgefragter Parameter, die Heading, Position und MovementSpeed beinhaltet.

AutoStatusPacket
Dieser Parameter ermöglicht die Aktivierung und Deaktivierung automatischen Sendens von StatusPackets auf Seiten des NXT.

PID-Gewichte
Da die Anzahl und Art der PID-Gewichte stark implementierungsabhängig ist, wurde für diese nur ein Intervall bereitgestellt, ohne die Gewichte konkret zu benennen.

8.2. Gruppeninterne Erweiterung

Die gruppeninternen Erweiterungen der Commands beschränken sich auf das Definieren der CommandID 0 als invalide. Auch die Error-Codes wurden nicht erweitert.

8.2.1. Parameter

GyroSpeed/GyroIntegral/MotorDistance/MotorSpeed
Diese Parameter kennzeichnen die Gewichtung der PID-Werte. Zur konkreten Verwendung und dem Einfluss der PID-Werte siehe Kapitel Regler.

WeightAll
Dieser Parameter bündelt, ähnlich wie das StatusPacket, alle PID-Werte, um mit weniger Commands mehr Daten übertragen zu können.

ConstantRotation/ConstantSpeed
Diese Parameter ermöglichen das Setzen und Abfragen einer konstanten Drehgeschwindigkeit oder Bewegungsgeschwindigkeit des NXT. Diese Option wird in der PC-internen Navigation nach Koordinaten genutzt, kann aber auch manuell verwendet werden.

WheelDiameter/Track
Diese Parameter ermöglichen das Anpassen der NXT-Software an Hardwaremodifikationen.

UltraSensor
Dieser Parameter kennzeichnet Daten, die vom Ultraschallsensor ermittelt und übertragen werden.

CollectTestData/Measurements
Diese Parameter werden bei der automatischen Kalibrierung des PID-Reglers verwendet. CollectTestData startet dabei einen neuen Testdurchlauf und löscht die alten Daten, Measurements kennzeichnet gesammelte Daten, die vom NXT an den PC gesendet werden.

9. Navigation

Die Navigation des NXT basiert auf der Steuerung. Für die Steuerung des NXT wurden vier Funktionen implementiert: `move`, `turn`, `constantSpeed`, `constantRotation`. Die Bewegung wurde über Manipulation des PID Reglers realisiert (vgl. Abschnitt 5).

Der Befehl `move` bewegt den NXT um eine zuvor bekannte Distanz in einer geraden Linie. `turn` dreht den NXT um einen zuvor bekannten Winkel. `constantSpeed` beschleunigt den NXT auf eine bestimmte Geschwindigkeit, allerdings ohne mit der Bewegung aufzuhören. Diese Aktion wird über einen weiteren `constantSpeed` Befehl mit der Geschwindigkeit `0` realisiert. `constantRotation` funktioniert analog zu `constantSpeed`, verursacht aber eine Drehung des NXT.

9.1. Navigation mit der GUI

In der GUI können die zuvor genannten Varianten der Bewegungsfernsteuerung ausgelöst werden. Für das Senden von `move` und `turn` Befehlen stehen im Tab `Position` zwei Buttons zur Verfügung, die die `move` und `turn` Befehle mit Parametern aus den nebenstehenden Textfeldern versenden. Ein dritter Knopf kann den NXT außerdem auf eine absolute Gradzahl ausrichten. Dies ist durch das Speichern und Aktualisieren von Daten, die der NXT an den PC sendet, realisiert. Wichtig für die Navigation sind dabei das Heading, also die aktuelle Ausrichtung, und die Position. Mit Hilfe der aktuellen Aurichtung wird die schnellste Drehung auf die gewünschte Gradzahl berechnet und durch einen `turn` Befehl ausgeführt.

Ebenfalls im Tab `Position` befinden sich vier Knöpfe in Anordnung der Pfeiltasten einer Tastatur. Ist dieser Tab geöffnet reagiert die GUI auf die `W`, `A`, `S` und `D` der Tastatur und ordnet sie sinngemäß den Knöpfen der GUI zu. Diese Knöpfe steuern den NXT über die `constantSpeed` und `constantRotation` Funktionen. Die Geschwindigkeiten für Drehung und Fahren entsprechen dabei Konstanten, die so gewählt wurden, dass der NXT stabil bei seinen Bewegungen ist. Außerdem können diese Parameter über den Tab `Parameter` manuell gesetzt werden, wobei die Werte direkt den Geschwindigkeiten der Drehung und des Fahrens entsprechen. Daher wird nicht empfohlen von dieser Methode Gebrauch zu machen.

9.2. Navigation mit dem Controller

Um das manuelle Steuern des NXT für den Benutzer handlicher zu machen wurde zusätzlich eine Steuerung mit einem Gamepad implementiert. Diese wird durch die JInput Library möglich gemacht, die die Inputs eines Controllers abrufbar macht. Dazu werden `constantSpeed` und `constatRotation` verwendet. Die Daten des linken Analogsticks sowie der beiden Trigger werden in einem regelmäßigen Takt abgerufen. Der rechte Trigger löst eine Bewegung nach vorne aus, während der linke Trigger eine Bewegung nach hinten auslöst. Die Daten des Analogsticks werden in eine Drehung nach links oder rechts umgesetzt, je nachdem in welche Richtung der Stick bewegt wird. Dabei wird sowohl bei den Triggern als auch bei dem Analogstick die Intensität der Aktion beachtet. Das bedeutet,

dass ein schwächeres Drücken eines Triggers eine ebenso langsamere Bewegung an den NXT sendet. Um unnötige Befehle bei kleinen Intensitäten zu Filtern, werden Befehle nur bei einer Intensität von mindestens zehn Prozent gesendet. Die einzige Ausnahme bildet der Befehl zum stoppen einer Drehung oder einer Bewegung, der beim Unterschreiten dieser zehn Prozent einmalig gesendet wird. Erst bei einem erneuten Überschreiten werden neue Befehle bezüglich Bewegung gesendet. Es findet zudem eine Diskretisierung der abgetasteten werde in 20 Stufen statt, um dem analogen Rauschen entgegen zu wirken und so überflüssige Kommunikation zu vermeiden.

9.3. Navigation über die Karte mit Hinderniserkennung

Zunächst bestand die Navigation nach Koordinaten über eine Umrechnung, in der aus den aktuellen Daten und der gewünschten Position ein `turn` und ein `move` Befehl berechnet wurde, der diese Position direkt anfährt. Diese Funktion wurde durch eine komplexere Version ersetzt (vgl. Abschnitt 9.4), wird allerdings von dieser intern verwendet und ist daher nicht mehr direkt aufrufbar. Die GUI verfügt außerdem über eine Karte, in der die Position und die Ausrichtung des NXT angezeigt wird. Mit einem Klick auf die Karte wird ein Befehl zum Anfahren der entsprechenden Koordinate gesendet. Der im NXT verbaute Ultraschallsensor sendet außerdem regelmäßig seine Daten an den PC, der aus diesen Daten die Positionen von Hindernissen ausließt und in einer internen Datenstruktur speichert. Das interne Setzen eines Hindernisses erfolgt über eine Lernrate. So ist eine dynamische Umgebung erfassbar und fehlerhafte `frei` Signale (vgl. Abschnitt 2.1) lassen den NXT nicht direkt in ein Hindernis fahren. Zum Speichern wird die Karte in Quadrate mit 5 cm Kantenlänge unterteilt, da die Ungenauigkeiten der Hardware keine genauere Diskretisierung nötig machen. Der Inhalt dieser Karte wird in regelmäßigem Abstand komplett auf die Karte der GUI gezeichnet, wobei zwischen bekannten und unbekannten Feldern unterschieden wird. Über der Karte befinden sich Knöpfe zum setzen von Position und Ausrichtung, die die neuen Parameter an den NXT senden und die Karte in der GUI resetten.

9.4. Navigation mit Hindernisumfahrung

Um Hindernisse zu umfahren wurde ein A*-Algorithmus implementiert. Unbekannte Elemente der internen Karte werden dazu als frei betrachtet. Die Karte wird zur Realisierung des A*-Algorithmus in einen Graph umgerechnet, wobei jede diskretisierte Position in einem Radius von zehn Metern ein Knoten ist, und je zwei (auch diagonal) benachbarte Positionen eine Kante erhalten. Dieser Algorithmus ermöglicht eine bessere Navigation, welche die ursprüngliche Navigation nach Koordinaten abgelöst hat. Die neue Version wird in einem eigenen Thread gestartet. Diese berechnet mit A* zunächst den Weg. Anschließend wird für alle Knoten auf dem Weg überprüft, ob diese direkt ansteuerbar sind, also kein Hindernis zwischen Startknoten und aktuellem Knoten ist. Dies ist notwendig, da die Karte in Quadrate diskretisiert wurde, und eine diagonal verlaufende Gerade sonst durch viele einzelne 5 cm lange Bewegungen unterteilt werden würde. Diese Abfolge an Bewegungen würde den NXT nur unnötig aus dem Gleichgewicht bringen.Begonnen wird

beim Startknoten.Sobald der erste Knoten nicht mehr direkt ansteuerbar ist, wird der vorherige Knoten direkt angesteuert. Der Drehwinkel wird mit der alten Methode zum Anfahren von Koordinaten berechnet (vgl. Abschnitt 9.3),während die Fahrt anschließend mit einem `constantSpeed` Befehl gestartet wird. Ist der aktuelle Knoten erreicht, oder ein Hindernis auf dem Weg detektiert worden, wird der Weg neu berechnet. Das Durchführen einer regelmäßigen Neuberechnung ist ebenfalls möglich. Wird ein Hindernis detektiert, leuchtet in der GUI außerdem ein `blocked way` Zeichen auf. Dieser Vorgang wiederholt sich so lange, bis das Ziel erreicht wurde, oder die Berechnung des Weges fehlschlägt. Wird das Ziel erreicht, wird ein abschließender Befehl gesendet, der den NXT nochmal auf das Ziel steuern lässt, um Ungenauigkeiten auszugleichen.

Ein beispielhafter Ablauf ist in Abb. 19 zu sehen. In diesem Beispiel wird durch einen Klick auf die Karte ein Befehl zur Navigation mit Hindernisumfahrung ausgelöst. Die „Autostatuspakete“ werden hierbei nach vorgegebenem Takt empfangen. In diesem konkreten Fall wird bereits in der ersten Iteration das Zwischenziel erreicht, welches gleichzeitig das gewünschte Ziel ist. Deswegen finden keine weiteren Überprüfungen und Neuberechnungen statt, und abschließend wird ein weiterer Befehl zum Fahren gesendet.

10. Reflexion

10.1. J. Breyer

Das Softwareprojektpraktikum hat mir eine wichtige Erfahrung während des Studiums geboten. Konzepte zur Organisation und Strukturierung in größeren Projekten, die mir zuvor nur in der Theorie vermittelt wurden, konnte ich jetzt praktisch nachvollziehen. Von Vorteil war für die Arbeit an dem Projekt sicher, dass sich unser Scrum-Master im Vorfeld schon ausgiebig mit den zu erwartenden Anforderungen und Problemen auseinandergesetzt hatte. Dadurch konnten grobe Fehler wie unzureichende Planung vermieden werden.

Die Erhebung der Anforderungen, die in der Theorie recht simpel erschien, stellte sich für mich als überraschend schwierig heraus, da die Kunden nur vage Vorstellungen der tatsächlichen Funktionalitäten hatten. So musste unsere Gruppe eigenständig präzisieren, inwieweit welche Anforderung als erfüllt betrachtet werden konnte.

Die Gruppengröße von fünf Personen war meiner Ansicht nach passend. Wir legten gleich zu Beginn des Praktikums fest, dass nicht die gesamte Gruppe an einer Anforderung arbeiten würde, sondern mehrere Anforderungen in jeder Woche auf kleinere Gruppen aufgeteilt wurden. Die Bearbeitung der Aufgaben wurde in wöchentlichen Meetings festgelegt und abgeglichen. Bei fünf Personen das Konzept des Pair-Programmings umzusetzen erscheint auf den ersten Blick durchaus unmöglich, da bei der Aufteilung eine Person übrigbleibt. Daher weichten wir das Pair-Programming etwas auf und ließen auch Arbeit einer einzelnen Person oder Coding mit drei Personen über längere Sitzungen zu.

Bei der Aufteilung der Aufgaben bildeten sich mehr oder weniger feste Paare von Programmierern, die sich auf bestimmte Teile der Software konzentrierten. Ich hatte das Glück, die fünfte Person, aber nicht das sprichwörtliche fünfte Rad am Wagen, zu sein. So wurde ich oft je nach Bedarf und Arbeitsaufwand in den wöchentlichen Treffen derjenigen

Gruppe zugeordnet, die es nötiger hatte. Dadurch habe ich einen sehr guten Überblick über den gesamten Code, dessen Funktionalität und das Zusammenwirken der einzelnen Klassen erhalten. Hierbei lernte ich, wie wichtig Code-Transparenz bei der Bearbeitung der Aufgaben ist. Da verschiedene Aufgaben parallel mithilfe von Git-Branches bearbeitet wurden, war die Annahme, dass eine Methode zuverlässig das bot, was sie bieten sollte, grundlegend bei klassenübergreifenden und -internen Aufrufen.

Der Arbeitsaufwand für dieses Praktikum war meiner Meinung nach angemessen. So gab es Wochen, in denen weniger zu tun war, und Wochen, in denen mehrere Probleme kurzfristig viel Zeit beanspruchten. Das Aufteilen der Gruppe in kleinere Teilgruppen war nicht nur sinnvoll sondern auch notwendig, da sich die Terminkalender der einzelnen Mitglieder teils stark unterschieden. Problematisch war meiner Erfahrung nach die Kommunikation in den Kleingruppen. Da unsere Gruppe zu großen Teilen über WhatsApp in Kontakt blieb, wurden auftretende Probleme oder Ergebnisse, die spezifische Anforderungen betrafen, gleich der gesamten Gruppe mitgeteilt. Dies erschwerte den Überblick vor allem dann, wenn beide Kleingruppen gleichzeitig über verschiedene Probleme diskutierten, was allerdings nur selten der Fall war. Gleichzeitig bot diese Art der Kommunikation aber den Vorteil, dass in manchen Situationen Personen, die nicht mit der Aufgabe vertraut gewesen waren, die Lösung eines Problems bereits kannten oder fanden, an dem die anderen Mitglieder möglicherweise sonst viel Zeit verbracht hätten.

Abschließend lässt sich für mich sagen, dass dieses Praktikum mir enorm geholfen hat, nicht nur theoretische Konzepte in die Praxis umzusetzen, sondern auch organisiert an Projekten zu arbeiten.

10.2. F. Friedrichs

Bei der Auswahl des Praktikums hatte ich mir zunächst nur wenige Informationen über das Praktikum besorgt. Der Hauptgrund für dieses Praktikum war die Programmiersprache, da ich außer Java noch mit keiner größeren Sprache in Kontakt gekommen bin. Außerdem habe ich zuvor in einer AG des Gymnasiums mit einer älteren Version der Lego Roboter gearbeitet, wenn auch nur mit der beiliegenden Software. Mir war auch bewusst, dass das Praktikum deutlich mehr Arbeit sein wird, als einfach nur einen Roboter aus Lego zusammenzubauen.

Durch einige Anekdoten eines Professors war mir außerdem bewusst, dass Softwareprojekte wie dieses Praktikum einen sehr großen Verwaltungs- und Planungsaufwand haben. Dies wurde dann durch das Praktikum bestätigt, denn die Organisation nahm vergleichsweise viel Zeit in Anspruch. Neben den ein bis zwei wöchentlichen Meetings, kam unter anderem noch die Anforderungsanalyse dazu, die in einer Woche zusätzlich zu den normalen Meetings weitere zwei Stunden am Abend belegt hat und ich am Ende froh war, dass wir dies hinter uns gebracht haben. Generell konnte ich sehr viel über die Koordination eines solchen Projekts in einem Team lernen. Außerdem hat mir das Projekt geholfen, unpräzise Anforderungen so umzusetzen, dass der Kunde nachher zufrieden ist. Denn meistens bestanden die Aufgaben aus sehr allgemein formulierten Anforderungen, die die meisten Kunden in der Realität aber auch nicht präziser hätten formulieren können.

Die Meetings haben mir sehr gut gefallen, hauptsächlich wegen unserer Methode des `Planning Poker` . So konnte derjenige, der über ein Thema am meisten Bescheid wusste den anderen vermitteln wie ein Problem am einfachsten zu lösen ist. Da ich von mir selbst aus sehr selten programmiere, habe ich viel über die Techniken, die über die Vorlesung Programmierung hinaus gehen, gelernt. Aber ich habe auch gelernt, wie ich mich im Zweifel selbst über ein Thema informieren kann. Darüber hinaus habe ich auch viel über Git gelernt, von dem ich am Anfang noch kein großer Freund war. Mir war auch klar, dass ein solches Werkzeug nötig ist, um Gruppenarbeit überhaupt erst möglich zu machen. Als ich dann aber erfahren habe, dass die meiste Funktionalität von Git ebenfalls in unserer IDE implementiert war, fiel mir das arbeiten mit Git deutlich leichter.

Das beste an dem Praktikum war aber, dass man am Ende relativ viele Ergebnisse sehen konnte. Sowohl bei den einzelnen Modulen als auch im Großen und Ganzen konnte man immer vergleichsweise viel Fortschritt sehen. Am Ende einen fernsteuerbaren Roboter zu haben, der nach einem Klick auf die Karte in der GUI die entsprechende Position anfährt fühlte sich schon nach einer Errungenschaft an.

Das wohl einzig negative an dem Praktikum war die Hardware selbst. Weil die Motoren und Sensoren sehr ungenau sind, haben sie mehr oder weniger alleine das Aussehen des Roboters bestimmt. Trotzdem brachte der Lego Bausatz auch Vorteile, unter anderem eine einfache Möglichkeit den AUfbau des NXT zu verändern. Wäre dies nicht möglich hätte dies einen enorm großen zusätzlichen Aufwand verursacht. Generell war der Arbeitsaufwand angemessen, wenn auch nicht immer gut verteilt. So hatte jeder im Team mindestens einmal die Ehre um die 20 Stunden in der Woche an dem Projekt zu arbeiten, im Durchschnitt entsprach die Arbeit aber ungefähr den ECTS Credits.

Alles in allem hat mir das Praktikum sehr gut gefallen und ich würde es jedem weiterempfehlen. Besonders das Lernen von Techniken im Bereich Programmierung und Organisation haben mich außerdem dazu motiviert, mich in Zukunft auch selbst an ein Projekt zu wagen.

10.3. C. Kloos

Meine Entscheidung bei der Praktikawahl ging maßgeblich auf eine Erinnerung aus dem ersten Semester zurück, als ich eine Gruppe Studenten in den Räumlichkeiten der Hochschule sah, die eben dieses Softwarepraktikum belegten und an ihrem Segway arbeiteten. Ich war begeistert, dass das Wissen aus drei bzw. vier Semestern ausreichend war um ein solches Projekt umsetzen zu können. Als Erstsemester hatte ich damals Schwierigkeiten konkrete Anwendungsbeispiele für die gelernten Vorlesungsinhalte zu erkennen. Bei der Wahl zum Praktikum legte ich dann vor allem Wert darauf ein Projekt zu finden bei dem man möglichst bei null beginnend und aufeinander aufbauend eine konkretes Produkt entwickelt. Rückblickend wurden meine Erwartungen an den Lernerfolg und Spaß deutlich übertroffen. Anfänglich war mir der Umgang mit einem Git-Repository nicht geläufig, auch hier konnte das Praktikum für mich einen Lernerfolg erzielen. Der Schwierigkeitsgrad und die Anforderungen für die wöchentlichen Abnahmen war passend gewählt und mein Kenntnisstand in Java und allen notwendigen Programmierkonzepten aus gymnasialen Oberstufe und drei Semester des Studiums waren ausreichend, sodass ich

mich angemessen gefordert fühlte jedoch nie überfordert. Ich lernte meine Teammitglieder erst beim Scrum-Meeting kennen, dennoch war der Umgang schnell freundschaftlich und unkompliziert. Wir konnten von einander lernen und unser Wissen ergänzte sich, sodass wir kein Problem vorfanden welches wir nachhaltig nicht lösen konnten. Rückblickend war das Praktikum trotz einer guten Organisation und Verteilung der Aufgaben dennoch sehr zeitaufwändig. Da jedoch ausreichend Lernerfolg vorhanden war und mit jeder Abnahme unser Projekt mehr Gestalt annahm konnte ich mich gut motivieren. Als Verbesserung würde ich vorschlagen mehr Zeit für Extras einzuräumen, da zumindest in unserer Gruppe noch viele Ideen und Konzepte erarbeitet wurden die durchaus einen Mehrwert am Endprodukt erwirkt hätten. Dies hängt wahrscheinlich auch damit zusammen, dass ich den Eindruck hatte, dass meine Gruppe insgesamt sehr motiviert an das Projekt ging und darin nicht nur ein abzulegendes Modul sah. Ich schließe mich da gerne mit eine ein und sehe das als positives Attribut, welches nur für die Qualität des Praktikums spricht. Als einziger Kritikpunkt hätte ich mir von den Kunden zwischendurch mehr Rückmeldung über ihre Zufriedenheit mit den Zwischenergebnissen gewünscht.
Ich bin mit meiner Wahl sehr zufrieden und würde das Praktikum einem Kommilitonen weiterempfehlen.

10.4. G. Kobsik

Ich habe mich im Vorfeld über das Projekt bei Kommilitonen, die das Praktikum schon in letzten Jahren absolviert haben, über die Anforderungen und die Umsetzung des Projekts informiert. Daraus wurde mir direkt klar, dass ich die Rolle des Scrum-Masters in diesem Projekt übernehmen möchte, um praktische Erfahrungswerte aus der Leitung eines kleinen Teams zu ziehen. Andere Veranstaltungen vermitteln diese Schlüsselkompetenzen nicht, die im Berufsleben einen schnellen Aufstieg zu Managementpositionen ermöglichen können. Des Weiteren konnte ich durch das Vorwissen die Planung schon an die kommenden Herausforderungen anpassen und hatte einen Überblick über die Konzepte, die sich bewährt haben, und die, welche zu Problemen geführt haben. Damit musste unser Team, bis auf die Umsetzung einer asynchronen Kommunikation, keine Entscheidung revidieren und keine Arbeit verwerfen. Ein besonderer Dank gilt hier meinem Team, das durch Disziplin in der Implementation und die Einhaltung der Deadlines meine Arbeit als Scrum-Master enorm erleichterte.

Die Arbeit mit einem iterativen und agilen Verfahren hat sich bewährt, welches es uns insbesondere erlaubte, wöchentlich Fortschritte zu präsentieren und so einen messbaren und kontinuierlichen Fortschritt zu erreichen, der sich positiv auf die Motivation auswirkt und mögliche konzeptuelle Fehler frühzeitig erkennt und beseitigt. Ein stärkerer Fokus auf eine testbasierte Implementierung des Codes mit `JTest` würde zur gleichbleibenden, unteren Schranke für die Codequalität führen. Dies würde in unserem Projekt jedoch zu viele Kapazitäten verbrauchen, sodass eine Umsetzung dessen nur für Projekte umsetzbar ist, in denen man die Eingaben mit erwarteten Ausgaben für eine Funktion ausreichend schnell definieren kann. Die Nutzung von Pair Programming anstatt wöchentlicher Codereviews zieht mit sich genauso viele Vorteile wie Nachteile. Ein wöchentlich wechselndes Konzept würde die Nachteile beider Verfahren besser dämpfen, indem alle im kollektiven

Review aus den Fehlern aller Teammitglieder lernen und man immer noch direkt bei der Implementierung ein Feedback bekommt.

Die Organisation des Projekts durch die Meetings stellte sich als Erfolg heraus, auch wenn diese zunächst unnötigen Ballast darstellten. Hierbei muss ich noch lernen, nicht immer alle Beteiligten ausreden zu lassen, sondern diese zu unterbrechen und den Fokus stärker zu lenken. Für größere Projekte würde es bedeuten, dass ein Vormeeting in einer kleinen Gruppe von drei ausgewählten Mitarbeitern durchgeführt wird, in dem die Konzepte zunächst intern diskutiert werden und so eine Agenda erstellt wird, die das externe Meeting mit dem gesamten Team begleitet. In unserem Fall habe ich das Vormeeting alleine durchgeführt, hierbei fehlte natürlich externer Input.

Die Erstellung eines festen Zeitplans hat uns am Anfang viel Zeit gekostet und für den späteren Verlauf die Dynamik der Meetings gesenkt. In weiteren Projekten würde ich die Organisation vor den Mitarbeitern kapseln, sodass der Zeitplan nur von dem Organisationsteam aufgestellt wird und die Mitarbeiter ihre Gedanken und Vorschläge dynamisch äußern können. Planing Poker ist hierbei ein sehr gutes Werkzeug.

Zunächst erschienen mir die Anforderungen an die Umsetzung des Projekts für fünf bis sechs Mitglieder als zu viel. Ein kleines Team von zwei bis drei Menschen könnte auf viel organisatorischen Overhead verzichten und das gleiche Produkt liefern, jedoch in der Retroperspektive würde ich das nicht mehr behaupten. Ich würde insbesondere die Extras auf ein Produkt einschränken und dort weitere Konzepte zur Qualitätssicherung und Teamleitung ausprobieren. Dieses Lernkonzept rückte im Verlauf des Praktikums in den Hintergrund. Hier könnten die Betreuer die Teilnehmer dazu ermutigen, sich bei der Umsetzung der Extras weniger auf die Funktionalitäten und mehr auf die Art der Umsetzung als Lernziel zu fokussieren.

10.5. R. Kupper

Meine Wahl fiel hauptsächlich auf das NXT-Praktikum, da ich bei meiner Arbeit als HiWi im Rahmen der Potentialanalysen bereits mit NXT-Robotern zu tun hatte.

Das NXT-Praktikum war meine erste Erfahrung mit einem Entwicklerteam von mehr als drei Leuten. Daher war die Arbeit im Team sehr interessant und lehrreich.

Im Team stellte sich schnell eine Aufteilung der verschieden Aufgabenbereiche ein. Dadurch gab es zu jedem Bereich einen oder zwei Spezialisten, was die Kommunikation im Team sehr vereinfacht hat. Da die Implementierung der Funktionalität für jede Abgabe meist in verschiedene Bereiche fiel, konnten die Aufgaben so gut aufgeteilt werden.

Die Verwendung von Feature Branches erfordert regelmäßiges rebasen, vorallem dann, wenn ein Feature Branch von einem anderen abhängt. Es gab vor allem zu Beginn ein unübersichtliches hin-und-her zwischen verschiedenen Branches. Hier zeigt sich, wie wichtig es ist, verschiedenen Kenntnisstände im Team zu berücksichtigen und gegebenenfalls Schulungen anzubieten um sicherzustellen dass alle nicht nur mit den Werkzeugen, sonder auch mit den Verfahrensweisen vertraut sind.

Wie sich im späteren Verlauf des Projekts herausstellte, wären regelmäßige Code-Reviews angebracht gewesen. Es traten im Verlauf des Projekt mehrfach Probleme auf, deren Ursache darin lag, das eine Funktionalität redundant vorlag. Spätere Änderun-

gen führten dann zu Inkompatibilität und nur schwer zu findenden Bugs. Ebenso wurde viel Funktionalität selbst implementiert, statt auf existierenden Lösungen aus der Java-Standardbibliothek zurückzugreifen. Insgesamt bin ich mit der Qualität unserer Codebasis nicht zufrieden, allerdings fehlt gegen Ende des Projekts sowohl die Zeit als auch die Notwendigkeit technische Schulden abzubauen. Das stellt wohl Projektalltag dar.

Viel Frustration während des Projekts entstand durch die sehr unpräzise Sensorik des NXT. Dabei ist insbesondere der Einachsen-Gyro hervorzuheben. Mit einem Integrierten 3-Achsen-Gyroskop und Beschleunigungssensor wären wir für die Positionsbestimmung nicht auf die Rad-Encoder angewiesen gewesen. Bedingt durch den Schlupf der Räder die Die Positionsbestimmung darum sehr unzuverlässig. Dies brachte insbesondere Probleme für die Navigationskarte mit sich. In einem richtigen Projekt wäre dies Anlass gewesen, eine Kosten-Nutzen-Rechnung zu erstellen um den Kunden zur Anschaffung besserer Hardware zu bewegen.

Trotz allem, oder gerade deswegen, bin ich beeindruckt von der Funktionalität, die wir schlussendlich umsetzen konnten. Zu sehen, wie der NXT durch einen Klick auf die Karte automatisch zum Ziel und sogar um Hindernisse herum navigiert finde ich sehr eindrucksvoll.

Das NXT-Praktikum ist eine gute Gelegenheit, um einmal eine Rolle in einem Entwickler-Team auszuprobieren und Projekterfahrung zu sammeln.

11. Zusammenfassung

Die Quintessenz unserer Erfahrungen stellt die folgende Zusammenfassung dar. Unerwarteterweise lag die Lernkompetenz dieses Praktikums nicht in der Programmierung des NXT-Roboters mit Java, sondern vermittelte uns vielmehr die praktischen Erfahrungswerte in der Arbeit mit einem Entwicklerteam. Insbesondere haben wir positive Erfahrungen mit Scrum gesammelt, das als iteratives und agiles Verfahren gut in ein dynamisch gesteuertes Team passte. Weitere Organisationswerkzeuge, die wir in zukünftigen Projekten nutzen würden, sind Planing Poker, das eine dynamische Arbeitskraftverteilung erlaubt, sowie Pair Programming, das interaktive Zusammenarbeit fördert. Jedoch dürfe man nicht der Illusion verfallen, dass man hierdurch auf Code Reviews verzichten könne. Dieses Projekt wurde zusätzlich durch die Versionskontrolle über `Git` unterstützt, indem einzelne Features unabhängig voneinander entwickelt werden konnten, ohne die Übersicht über den bestehenden Code zu verlieren. Dies ermöglichte eine effiziente Arbeitsteilung, ohne zusätzliche Redundanz in der Verwaltung verschiedener Versionen zu verursachen. Auch wenn die Organisationsstruktur den Hauptteil unserer neu gewonnenen Erfahrungen bildete, so stellte sich heraus, dass wir öfter auf `Standard Librarys` zurückgreifen müssten und immer mit Interfaces arbeiten sollten, die die Implementierung stärker von der bereitgestellten Funktionalität kapseln würden. Eine Kapselung von dem Organisationsteam und dem Implementierungsteam hat sich als sinnvoll erwiesen, da dies dem kreativen Wertschöpfungsprozess nicht im Weg stand und eine dynamische Bewertung der Anforderungen erlaubte, ohne dass sich das Team durch statische Pläne, die mit fragilem Vorwissen erstellt worden sind, lenken lässt.

Insgesamt ist das Praktikum in unserem Team eine willkommene Abwechslung zu dem theoretisch orientierten Studium, auch wenn es mit einem erhöhten Arbeitsaufwand innerhalb des Semesters im Vergleich zu anderen Veranstaltungen verbunden ist.

Literatur

[1] "The ultrasonic sensor." https://nxttime.wordpress.com/2012/09/12/the-ultrasonic-sensor/, accessed 2017-07-29.

A. UML Diagramme

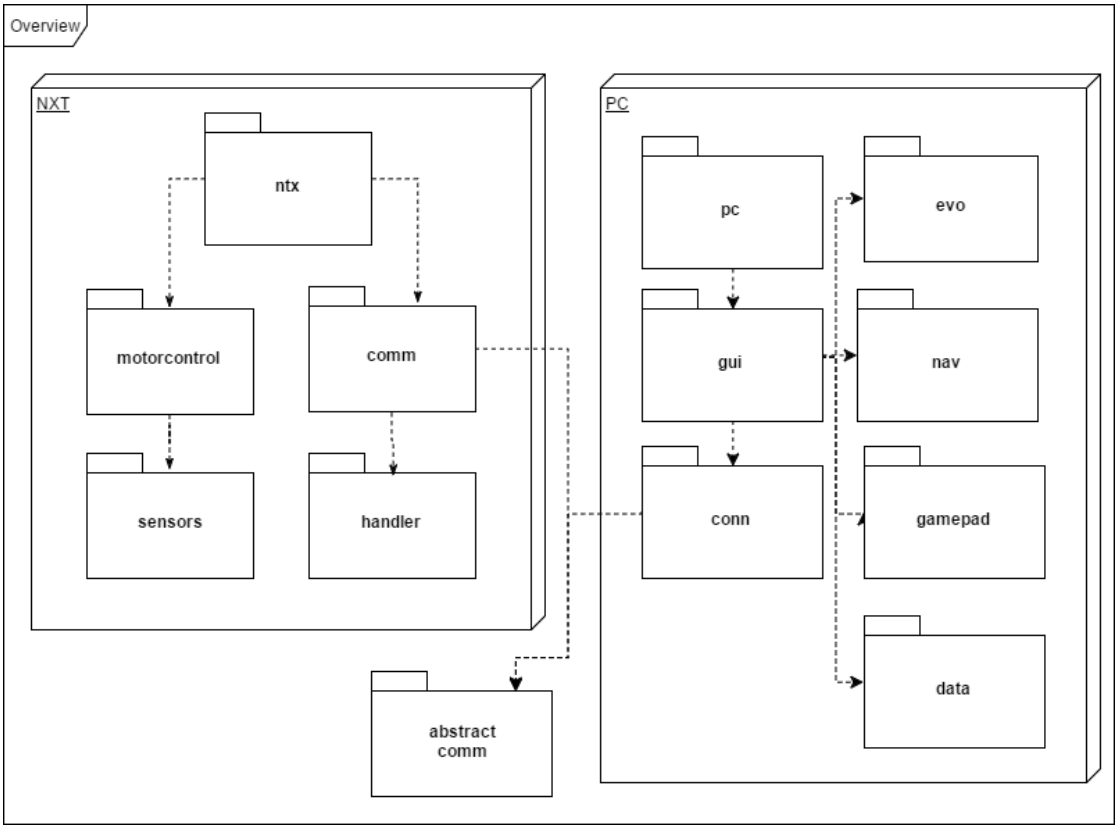


Abbildung 9: Übersicht der einzelnen Pakete in beiden Projekten.

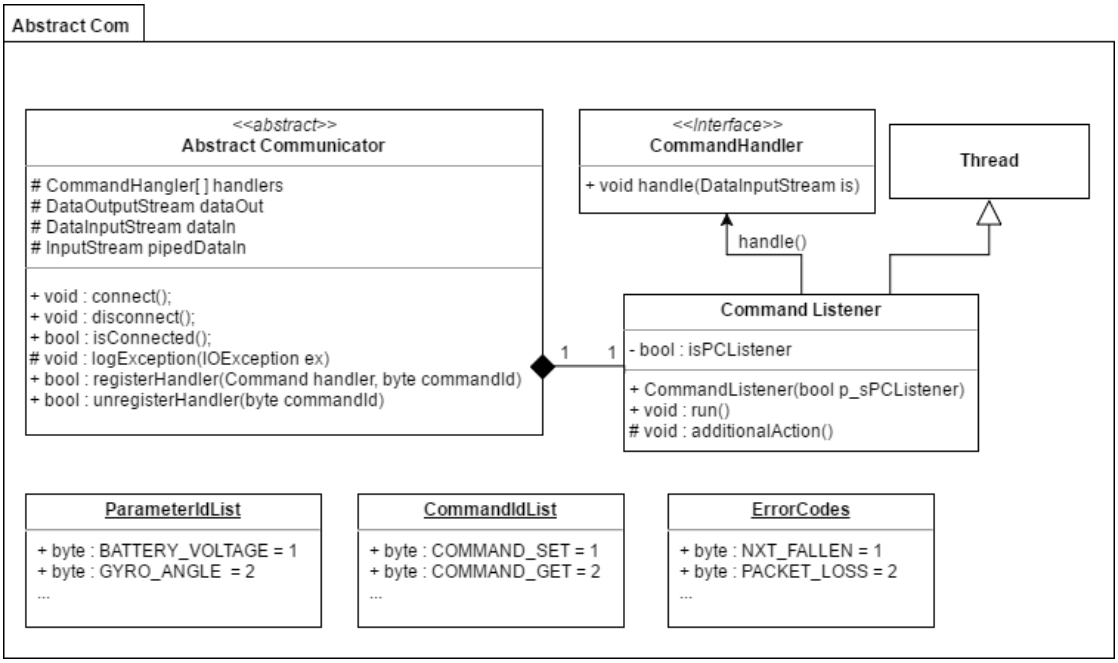


Abbildung 10: Die abstrakte Kommunikationsklasse verbindet beide Projekte, indem sowohl das PC- als auch NXT-Projekt von dieser Klasse erben.

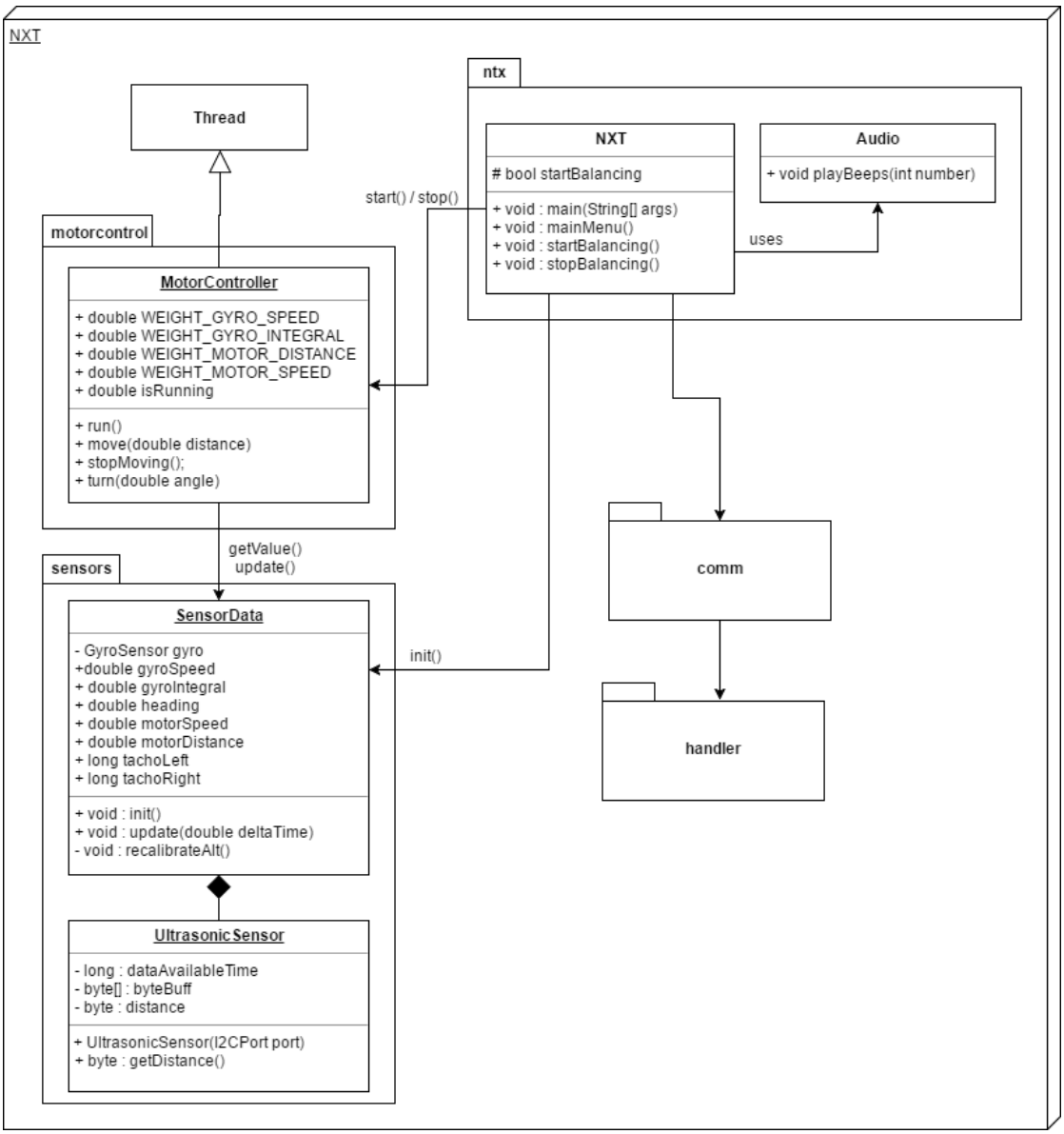


Abbildung 11: Der NXT besteht zur Hälfte aus dem MotorController in dem der Regler emuliert wird, dieser nutzt die Sensordaten, die in eigenen Klassen gehalten werden. Die andere Hälfte des NXT stellt die Kommunikationsklasse dar.

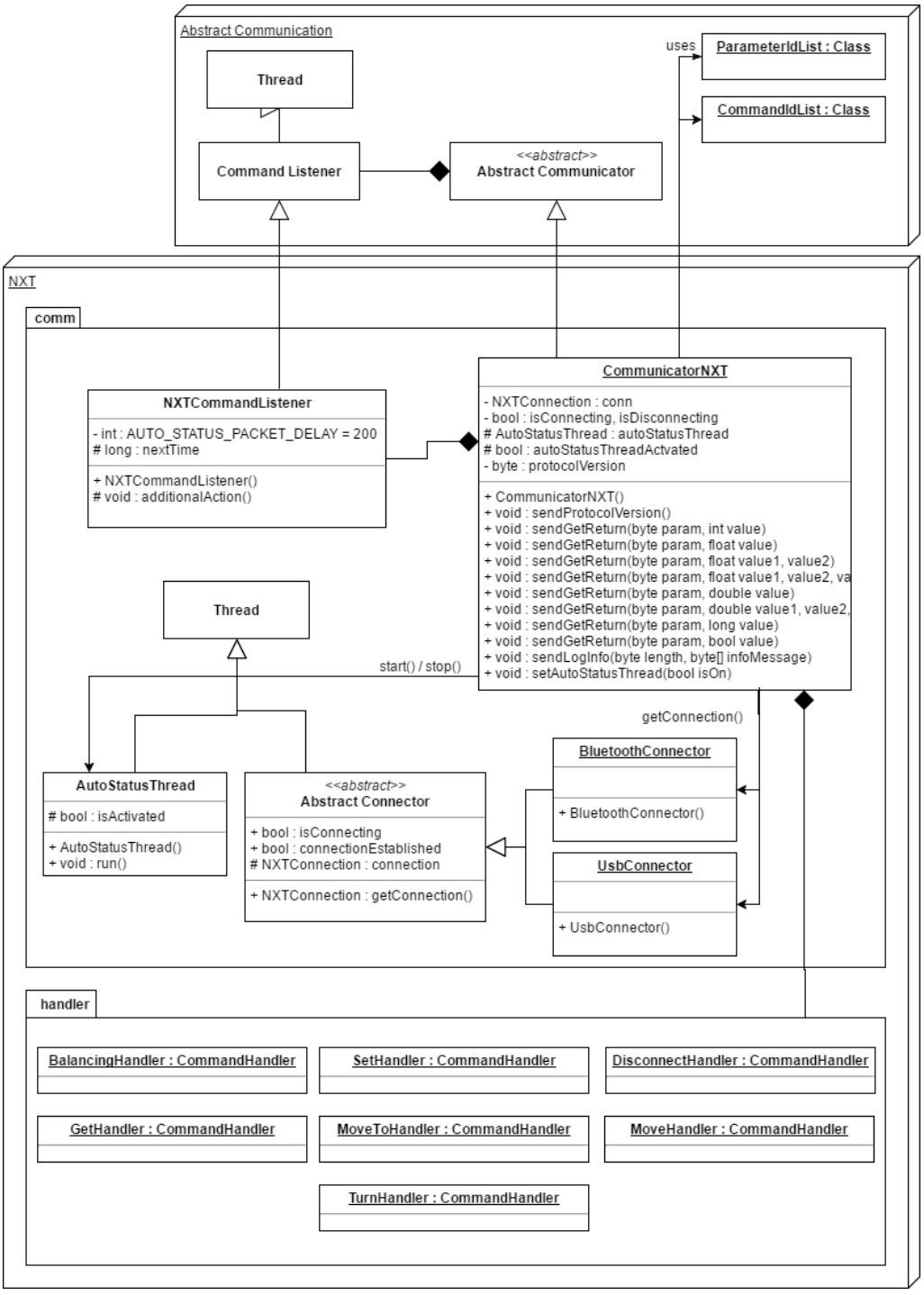


Abbildung 12: Die Kommunikationsklasse des NXT leitet sich vom abstrakten Kommunikator ab. Durch diesen können die Handler registriert werden und der NXT so auf Befehle reagieren.

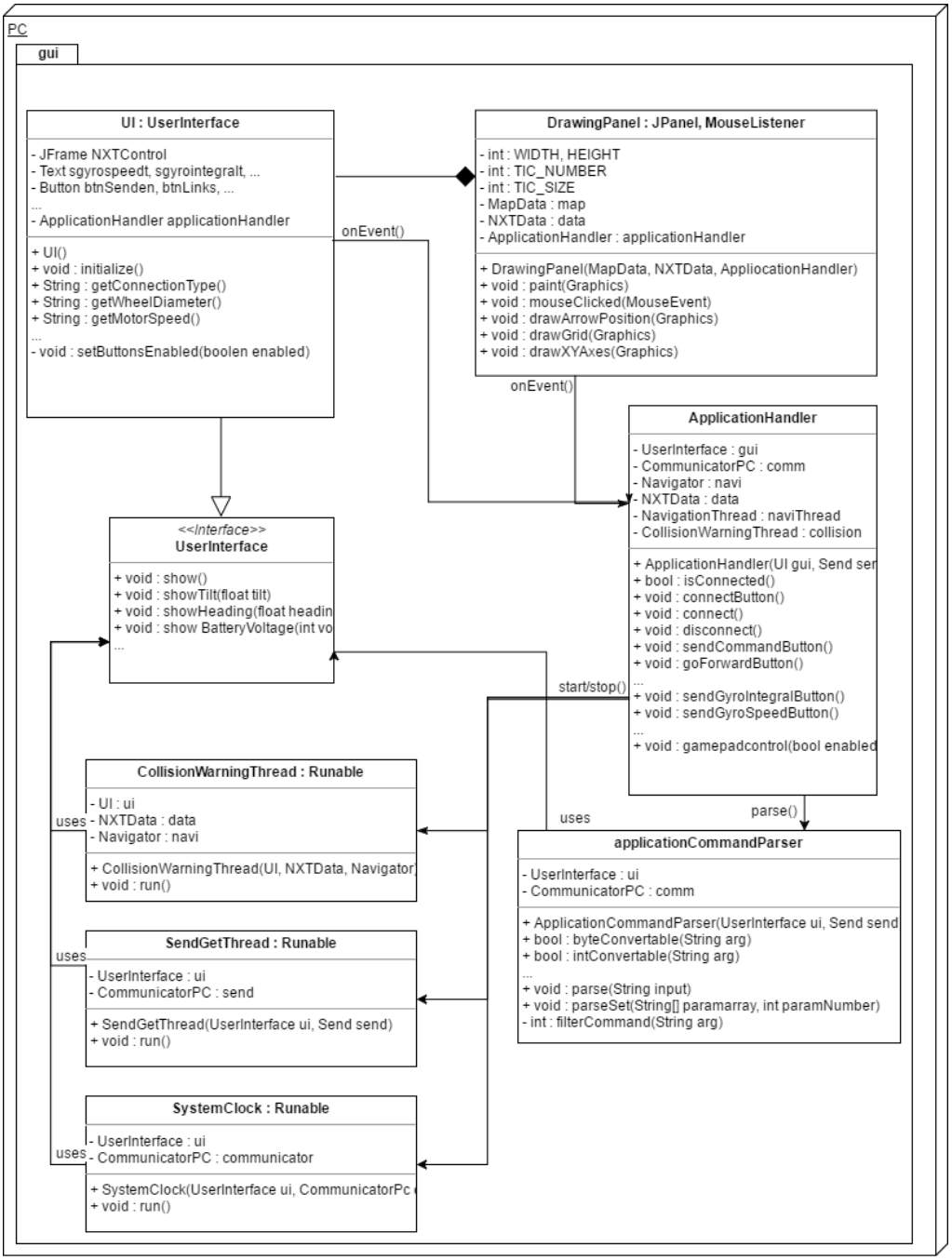


Abbildung 13: Die GUI bildet das Zentrum der PC Architektur. von ihr aus werden alle anderen Komponenten erstellt und verwaltet. Durch die Verwendung des Interfaces und des Handlers wird die Anzeige von der Funktionalität entkoppelt.

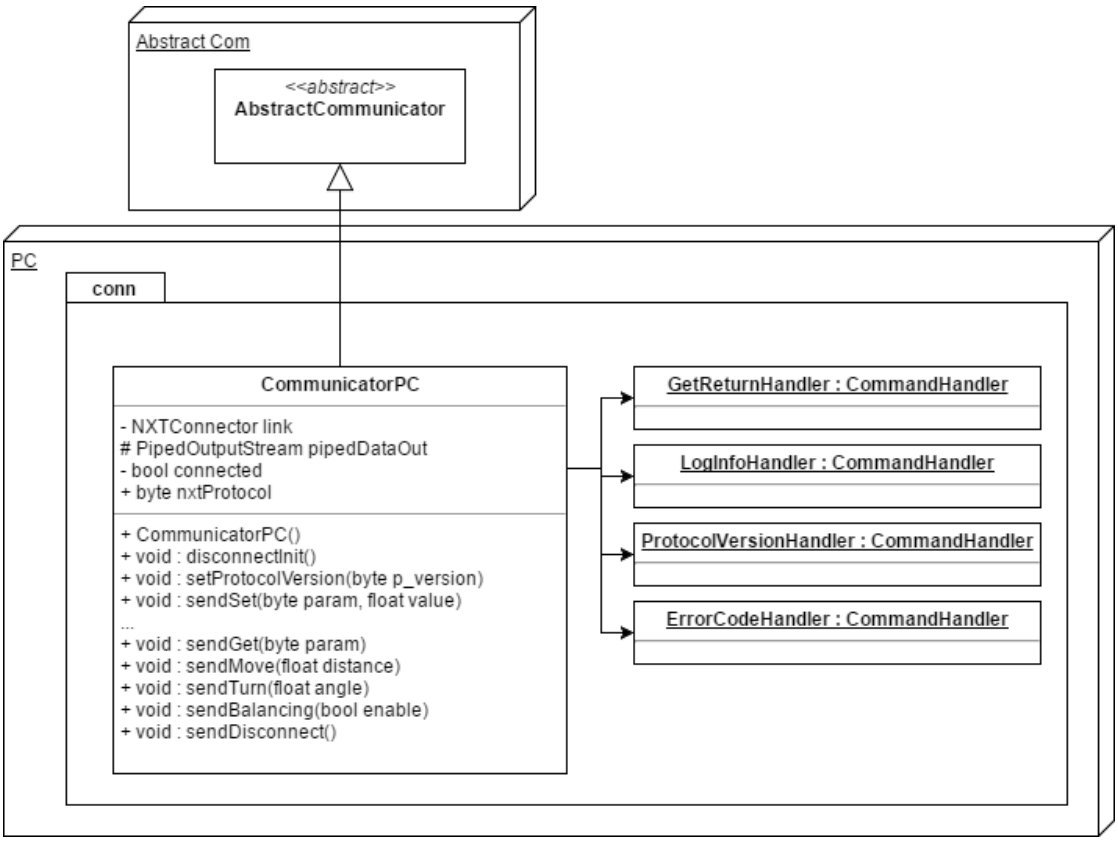


Abbildung 14: Die Kommunikation auf der PC Seite wird vom CommunicatorPC durchgeführt. Dieser kann eine Verbindungsanfrage annehmen, durch diese Verbindung Nachrichten verschicken und auch Empfangen. Die Verarbeitung der Nachrichten übernehmen Handler, die zur Laufzeit registriert oder unregistriert werden können.

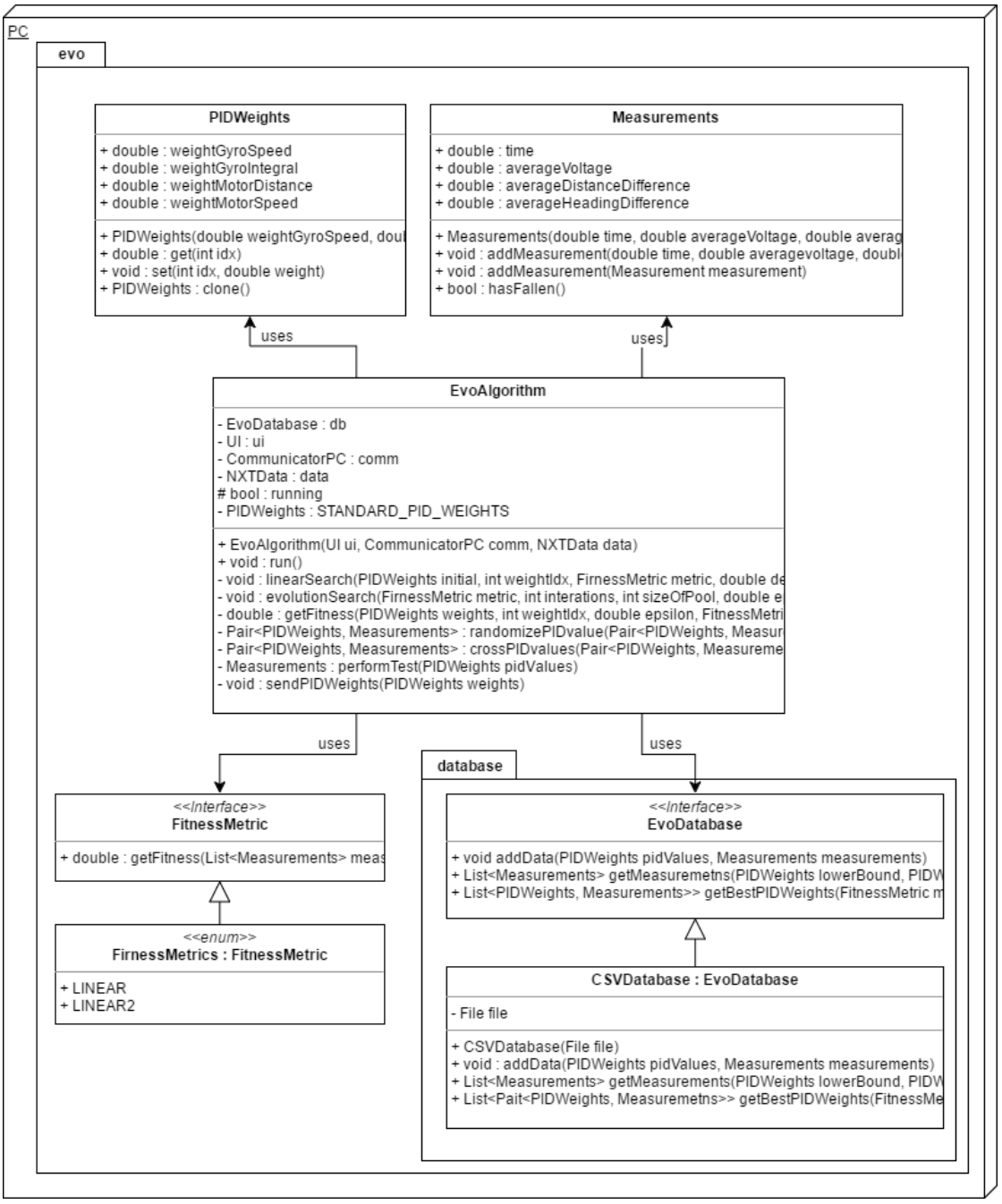


Abbildung 15: Das Evo Paket kapselt zwei Algorithmen zur Kalibrierung der PID Werte, sowie das Datenbanksystem. Das allgemeine Interface wurde durch eine CSV-Datenbank implementiert. Des Weiteren sind zwei Klassen für das Bündeln von Daten vorhanden, die PID-Gewichte und die Messungen verwalten. Die Metrik wird über ein Interface bereitgestellt, die Implementierung findet sich in einem Enum, das alle verfügbaren Metriken bündelt.

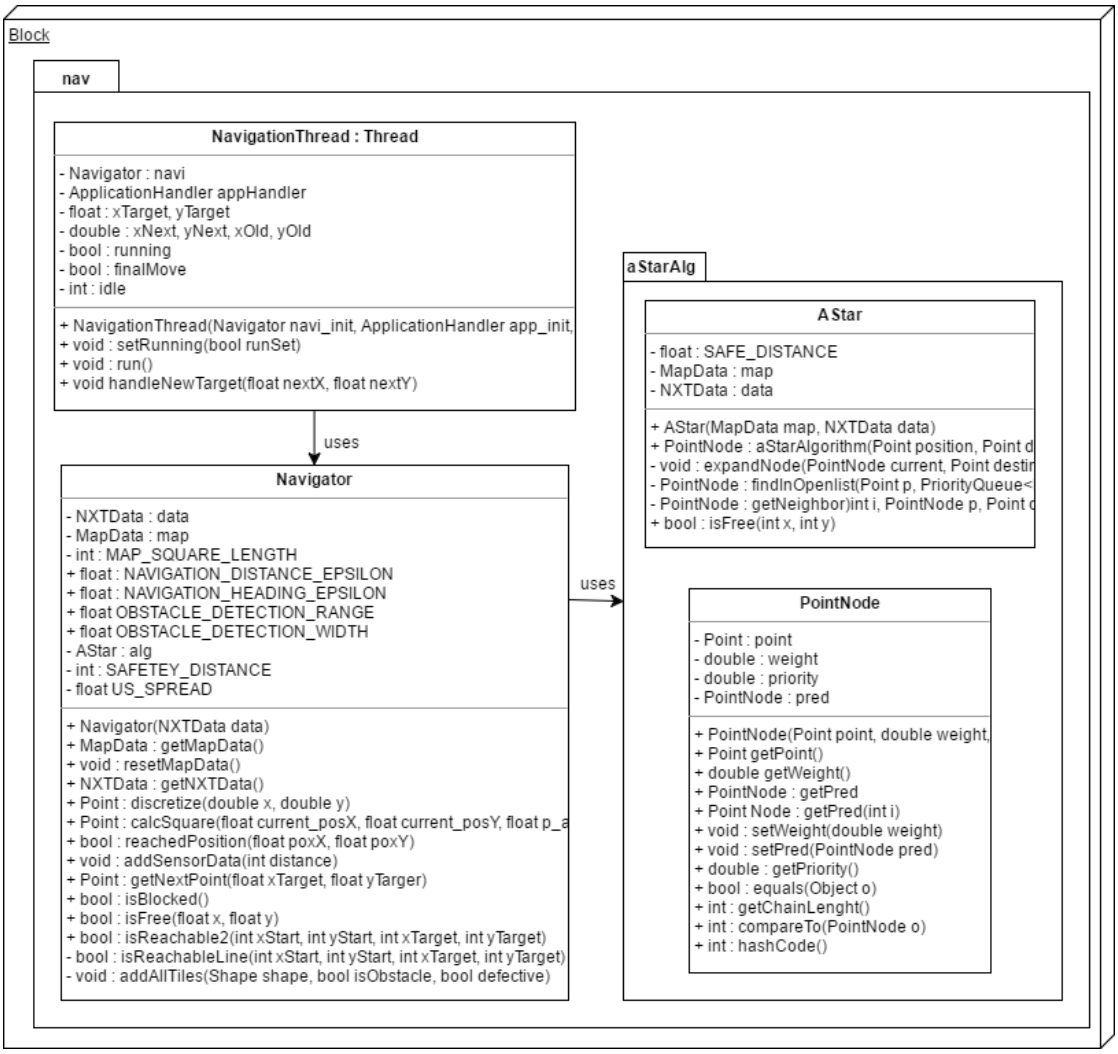


Abbildung 16: Die Navigation wird in einem eigenem Thread ausgeführt, der den Navigator nutzt. Dieser benutzt die Datenklassen NXTData und MapData zur Bestimmung der Route und den A*-Algorithmus, der in einem eigenem Paket implementiert wurde.

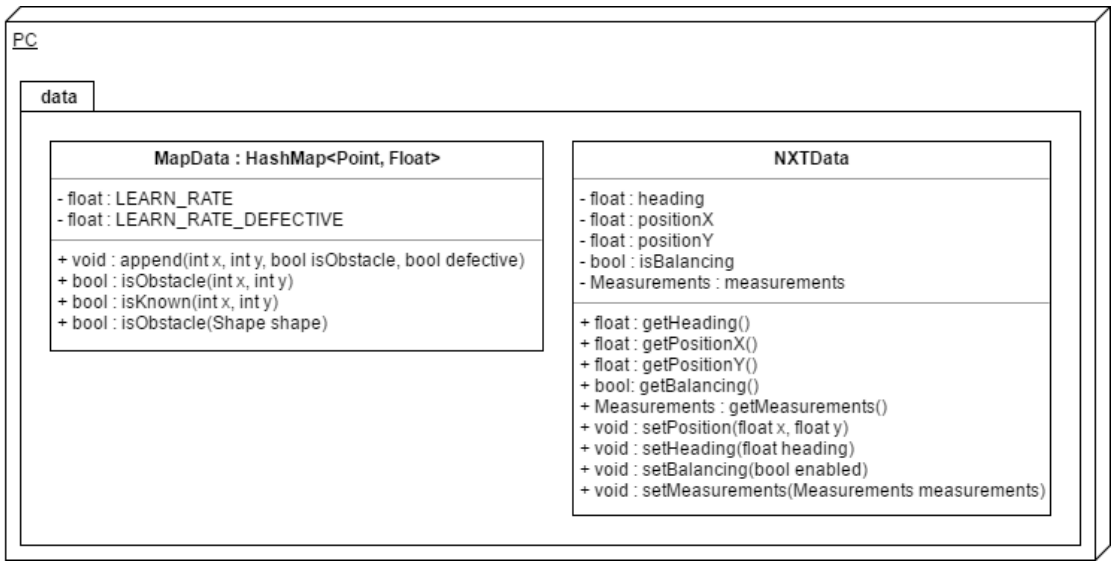


Abbildung 17: Die Daten der Karte wurden für einen effizienten Zugriff über eine HashMap implementiert. Die NXTData stellen eine virtuelle Abbildung des Zustands des NXT auf dem PC dar.

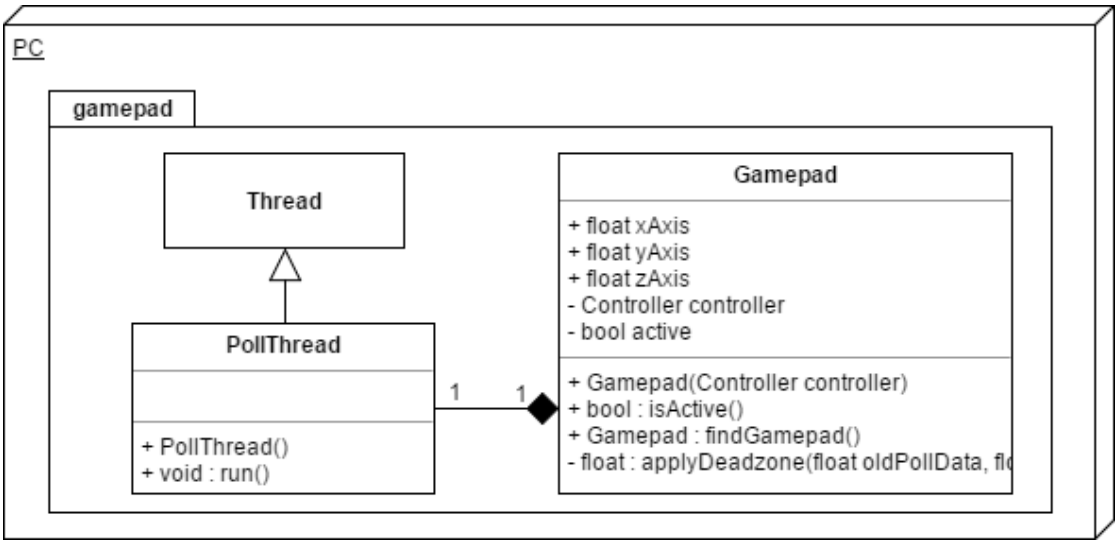


Abbildung 18: Die Gamepad-Klasse verwaltet externe Controller und sendet Steuerungsbefehle entsprechend der Eingabe an die Kommunikationsklasse weiter. Diese läuft in einem eigenem Thread.

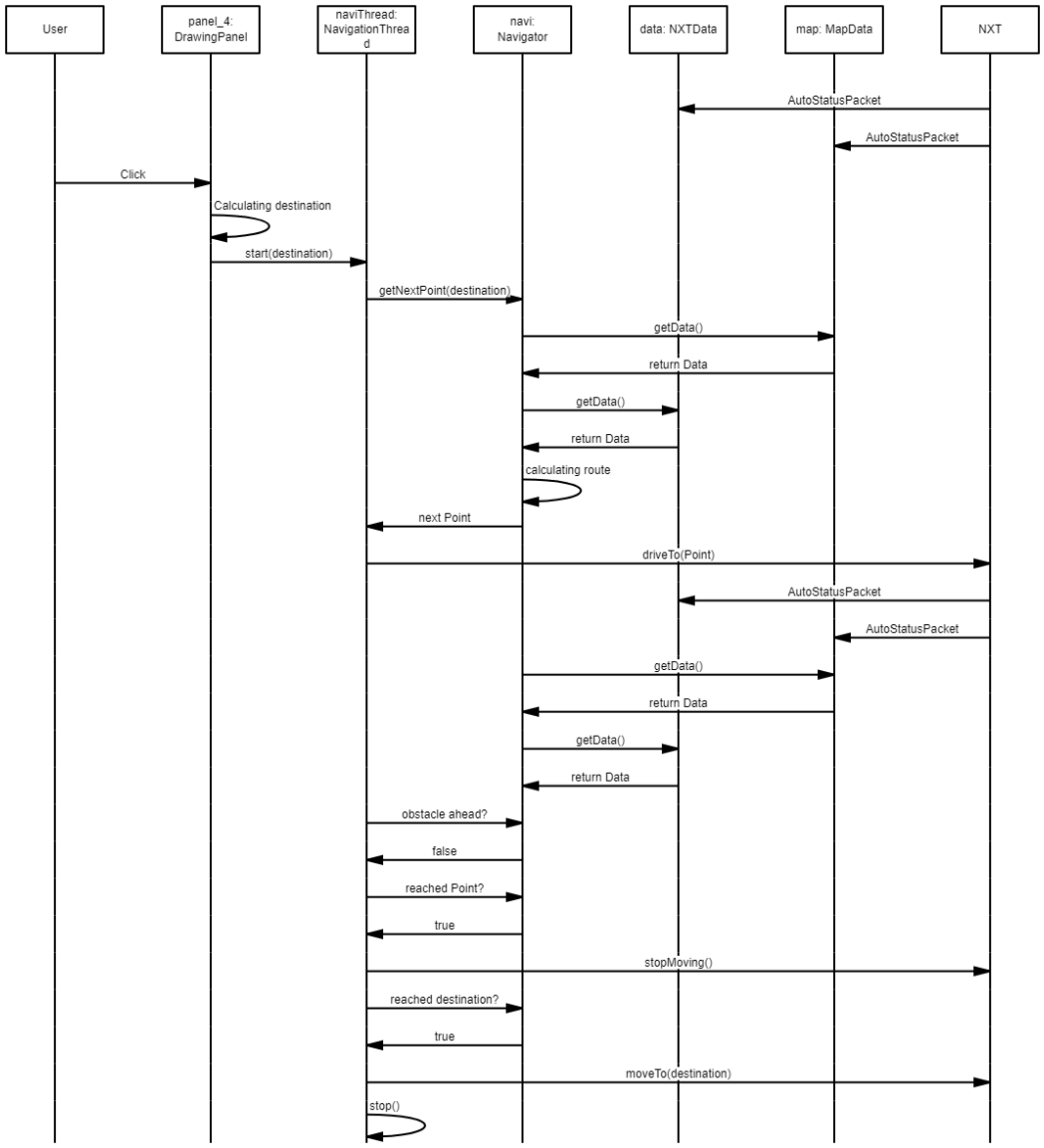


Abbildung 19: Ein Beispiel des Ablaufs der Navigation mit Hindernisumfahrung

B. Kommunikationsprotokoll

Tabelle 2: Command-Tabelle

Command	**CommandID (0-255)**	**Richtung**	**Struktur**
Set	1	PC -> NXT	[commandID:byte][paramID:byte] [value:varying]
Get	2	PC -> NXT	[commandID:byte][paramID:byte]
GetReturn	3	NXT -> PC	[commandID:byte][paramID:byte] [value:varying]
Move	4	PC -> NXT	[commandID:byte][distance:float]
Turn	5	PC -> NXT	[commandID:byte][angle:float]
MoveTo <reserved>	6	PC -> NXT	-
Balancing	7	PC -> NXT	[commandID:byte][enabled:boolean]
LogInfo	8	NXT -> PC	[commandID:byte][length:byte] [message:byte[]]
ErrorCode	9	NXT -> PC	[commandID:byte][errorcode:byte]
Disconnect	10	PC -> NXT	[commandID:byte]
ProtocolVersion	11	NXT -> PC	[commandID:byte][protocolVersion:byte]

Tabelle 3: Parameter-Tabelle

Parameter	**ParamID (0-255)**	**Get**	**Set**	**Typ**	**Einheit**
BatteryVoltage	1	Ja	Nein	int	mV
GyroAngle	2	Ja	Nein	float	°
TachoLeft	3	Ja	Nein	long	°
TachoRight	4	Ja	Nein	long	°
Heading	5	Ja	Ja	float	°
Position	6	Ja	Ja	2x float	cm
MovementSpeed	7	Ja	Nein	float	cm/s
StatusPacket	8	Ja	Nein	4x float	-
AutoStatusPacket	9	Ja	Ja	boolean	-
PID-Gewichte <internal>	21-30	Ja	Ja	varying	-
GyroSpeed	21	Ja	Ja	double	-
GyroIntegral	22	Ja	Ja	double	-
MotorDistance	23	Ja	Ja	double	-
MotorSpeed	24	Ja	Ja	double	-
Weightall	128	Ja	Ja	4x double	-
ConstantRotation	131	Ja	Ja	float	°/s
ConstantSpeed	132	Ja	Ja	float	cm/s
WheelDiameter	133	Ja	Ja	float	cm
Track	134	Ja	Ja	float	cm
UltraSensor	135	Nein	Nein	int	cm
CollectTestData	140	Nein	Ja	boolean	-
Measurements	141	Nein	Nein	4x double	-