Universität Kassel
Fachbereich 16 - Informatik und Elektrotechnik

Teamarbeit

Abschlussbericht

Autoren: Dennis Knitterscheidt
Robert Meschkat 28227496
Philipp Schenk 33309370
Eric Wagner 32233447

Betreuer: M. Sc. Stephan Opfer

Inhaltsverzeichnis

1 Einleitung

Philipp
REMOVE THIS: Beispiel

Kurze Beschreibung des Themas Teamarbeit. Was ist der Bericht?

2 Technische Arbeit

2.1 Arbeitsauftrag

Im KickOff-Workshop zur Veranstaltung „Teamarbeit“ wurden mehrere Aufgaben vorgestellt. Das Team hat sich dann zusammen gefunden, um folgende Aufgabe zu bearbeiten: Für die vom Team bearbeitete Aufgabe sollten ein Turtlebot und die existierende Software so erweitert werden, dass der Turtlebot als Transportsystem im Fachgebiet eingesetzt werden kann. Der Turtlebot sollte über die Software an einen Ort im Fachgebiet geschickt werden können, an dem er einen Gegenstand entgegen nimmt. Der Gegenstand sollte dann an einen vorher in der Software definierten Zielort gebracht werden.
Für die Umsetzung sollte ein Behältnis für den Transport der Gegenstände auf dem Turtlebot befestigt werden. Der Behälter sollte mit entsprechender Sensorik ausgestattet werden, damit der Turtlebot erkennt, wenn ein Gegenstand auf ihm plaziert wurde. Da der Turtlebot nur Daten verarbeiten kann, die sich in seinem Weltmodell befinden, mussten die Sensordaten außerdem in das bestehende Framework eingepflegt werden.
Um dem Turtlebot den Transportauftrag zu erteilen, sollte eine graphische Benutzeroberfläche entwickelt werden, in der man den Start- und Zielpunkt, sowie den zu transportierenden Gegenstand festlegen können soll.

2.2 Entwurf

2.2.1 Erste Bestandsaufnahme

Da kein Teammitglied zuvor mit den Turtlebots gearbeitet hat, musste sich das Team zunächst einen groben Überblick über die bestehende Hard- und Software verfassen, bevor mit der eigentlichen Planung begonnen werden konnte.
Für die Sensorik des Behälters war der Aufbau des Turtlebots ausschlaggebend: Gesteuert wird der Turtlebot von einem Notebook. Dieses ist über USB sowohl mit der Fahrbasis, als auch mit der Sensorik für die räumliche Erkennung verbunden. Damit wurde klar, dass auch der Gegenstandsdetektor über USB mit dem Notebook verbunden werden muss.
Die Software-Sammlung, die für den Betrieb des Turtlebots benötigt wird, basiert auf dem „Robot Operating System“, kurz ROS genannt, in der Version *Kinetic* und ist in C++ verfasst. Das User Interface musste als mit ROS kommunizieren können und im besten Fall in C++ geschrieben sein.

2.2.2 Recherche

Nachdem die Rahmenbedingungen abgeklärt waren, begann das Team zunächst mit der Suche nach Lösungen für die Aufgabenstellung.
Bei der Sensorik für die Gegenstandserkennung hat sich das Team schnell auf einen Taster festgelegt, der ausgelöst werden soll, wenn ein Objekt in den Behälter gelegt wird.

Kurzzeitig war auch noch ein RFID-Leser im Gespräch, der zusätzlich oder anstelle des Taster angebracht werden sollte. Mit dem Lesegerät hätten zwar die zu transportierenden Gegegenstände identifiziert werden können, allerdings hätten dann auch alle Objekte mit einem entsprechenden Marker versehen werden müssen. Daher entschied sich das Team für den Taster, der beim Transport den größeren Spielraum lässt und außerdem leichter umzusetzen war.
Für die mögliche Umsetztung des Tasters gab es verschiedene Lösungsansätze: Die einfachste Lösung wäre ein bereits fertiger Taster mit USB-Anschluss, der zum Beispiel eine Tastaturtaste simuliert. Die Recherche des Teams zeigte, dass entsprechende „Ein-Knopf-Tastaturen“ tatsächlich existieren, sich jedoch preislich in keinem realistischem Rahmen bewegen. Ein weiterer Vorschlag war die Modifikation eines Peripheriegerätes, wie Maus oder Tastatur. Auch diese Idee wurde verworfen, da sie vom Team als zu aufwändig und als mögliche Fehlerquelle angesehen wurde.
Für die Entwicklung des User Interfaces wurden zwei C++-Bibliotheken evaluiert: Zum einen der Quasi-Standard „Qt“ und zum anderen das „Chromium Embedded Framework“, kurz CEF. Bei der Evaulation stellte sich das CEF als interresante Lösung heraus, wurde aber vom Team aber abgelehnt, da es für dieses Projekt als zu aufwändig erschien.
Als Plattform für die Entwicklung wollte das Team sowohl das im Fachgebiet eingesetzte Ubuntu 16.04 LTS mit ROS Kinetic, als auch das zu diesem Zeitpunkt aktuelle Ubuntu 18.04 LTS mit ROS Melodic erproben.

2.2.3 Erster Entwurf

Nach der Recherche entschied sich das Team dazu, den Taster selbst zu bauen. Dafür sollte er mit einem Microcontroller verbunden werden, der die Signale über USB an das Notbook überträgt. Als Microcontroller wurde der „Digispark Rev. 3“ (Abb. 1) gewählt, da er mit seinen kleinen Abmessungen und seinem geringen Preis eine gute Lösung zu sein schien.

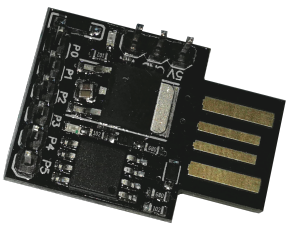

Abbildung 1: Digispark Rev. 3

Für die Software untersuchte das Team den existierenden Code in den verschiedenen Git-Repositories und entwickelte aus den Ergebnissen einen ersten Ansatz für die Software-Architektur (Abb. 2) des Projekts. Für den Client sollte es dabei ein Frontend

auf Basis von Qt geben, das die Benutzerobefläche zur Verfügung stellt. Das Frontend schickt die Benutzereingaben an das Backend, welches die Daten verarbeitet und an ROS (bzw. die ROS-Kernanwendung) weiter reicht. Der existierende ROS-UDP-Proxy soll die Daten als ROS-Message über das WLAN an den Turtlebot schicken. Auf dem Turtlebot soll es eine ROS-Komponente geben, die die Daten zu Kommandos für den Turtlebot umwandelt.

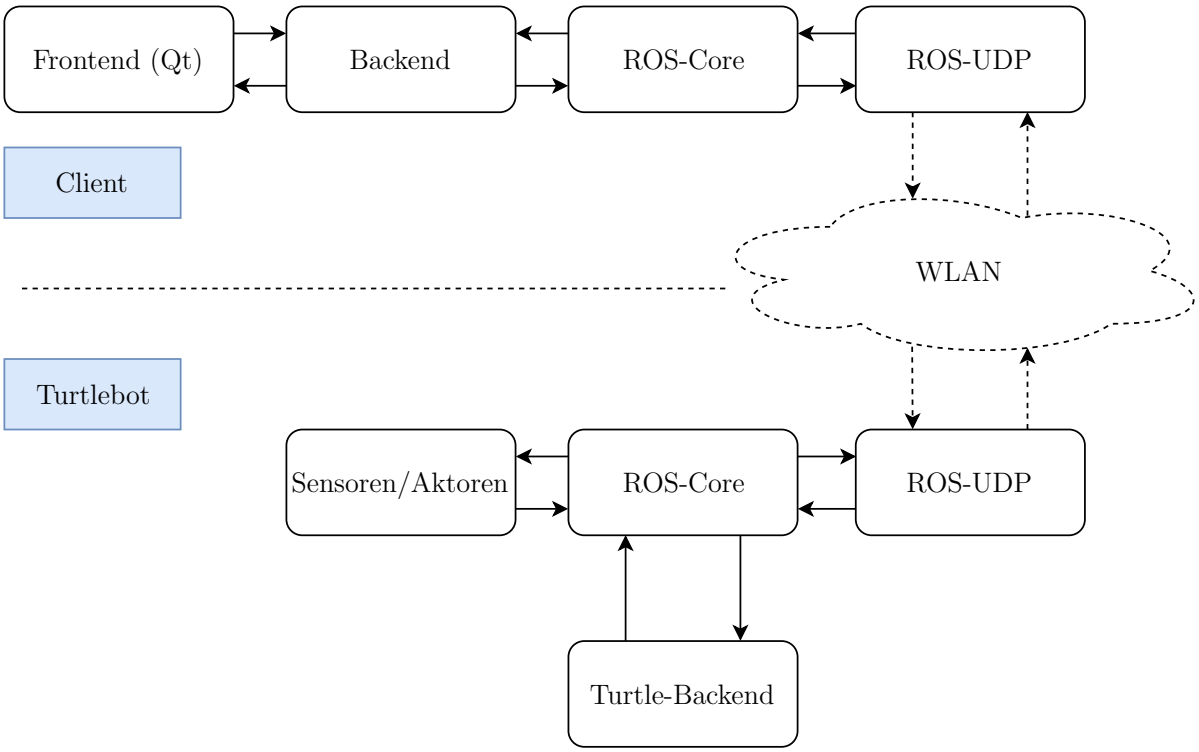

Abbildung 2: Erster Architektur-Entwurf

2.2.4 Zweiter Entwurf

Nach Rücksprache mit dem Betreuer des Teams, Stephan Opfer, zeigte sich, dass die Software auf dem Turtlebot komplexer war, als zunächst angenommen: Wie in der Abbildung 3 zu erkennen ist, gibt es neben dem UDP-Proxy, Sensoren und Aktoren, auch noch ein Weltmodell und eine Komponente namens „Alica“. Im Weltmodell wird der von den Sensoren erfasste Zustand, sowie weitere Informationen, festgehalten. Alica kann sich den aktuellen Zustand aus dem Weltmodell holen, auf der Basis entsprechender Pläne das weitere Vorgehen ermittlen und das neue Verhalten an die Aktoren mitteilen. Für die Software auf dem Turtlebot musste das Team also keine eigene Komponente entwicklen, sondern den Taster in die Sensorik, das Weltmodell und die Pläne einpflegen, Außerdem musste eine Nachricht des Clients mit Start-, Endpunkt und Gegegenstand ebenfalls in das Weltmodell eingetragen werden, damit diese Informationen ebenfalls in die Planung

mit einbezogen werden können.
Auf der Client-Seite erhielt das Team zusätzlich den Hinweis, dass es eine spezielle Version von Qt gibt, die direkt mit ROS kommunzieren kann, sodass die Aufteilung in Frontend und Backend nicht nötig ist.

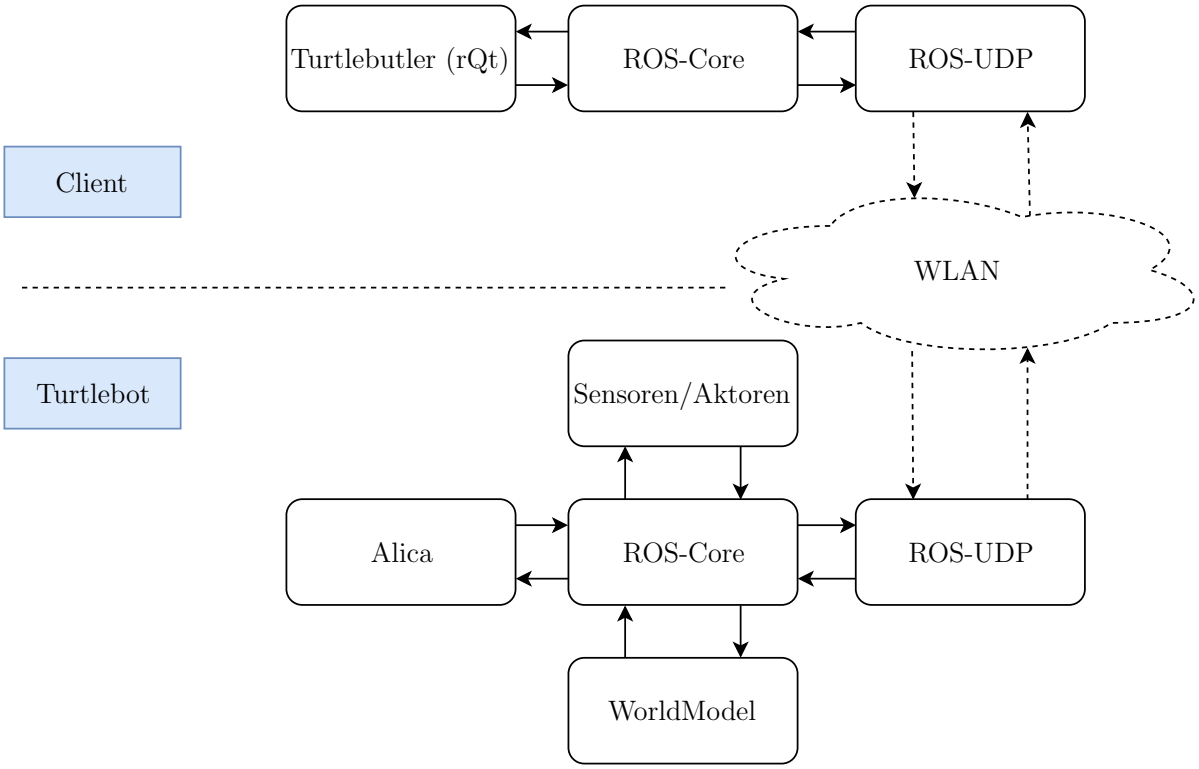

Abbildung 3: Zweiter Architektur-Entwurf

2.3 Umsetzung

Nach dem Kickoff-Meeting wurde vom Team beschlossen, sich einmal pro Woche im Labor des Fachgebiets zu treffen. Zu Beginn hat sich jedes Teammitglied mit Ubuntu und ROS beschäftigt und dieses auf seinem Rechnern installiert. Als Betriebssystem wurde Ubuntu 16.04 verwendet, da die Turtlebots mit ROS Kinetic arbeiten. Da der Turtlebot-Code aus mehreren GitHub-Repositories besteht, mussten diese zusätzlich heruntergeladen werden.
Zuerst wurde in den Treffen mit einem Laptop mit Ubuntu-Version 18.04 versucht, den Code der Repositiories zu kompilieren und auszuführen. Hierbei stieß man auf das Problem, dass einige Bibliotheken fehlten oder Code zum Teil nicht mit der neuen ROS-Version Melodic kompatibel war. Aufgrund dieses Problems wurde nach mehreren Kompilierungsversuchen auf einen Rechner im Labor des Fachgebietes umgeschwenkt, auf dem gemeinsam weiter programmiert wurde. Da auf dem Rechner trotz der korrekten

Ubuntu-Version immer noch Dateien fehlten, wurde der Betreuer zu Rate gezogen. Neben einer Konkretisierung des Arbeitsauftrages (siehe Abschnitt 2.2.4) stellte sich heraus, dass die verschiedenen Repositories unterschiedliche Branches benötigen um richtig zu funktionieren und zusammenzuarbeiten. Dieses Problem betraf nicht den Laptop, der mit dem Turtlebot verbunden war, weswegen der Roboter bereits mit Hilfe eines rviz-Plugins fahren konnte.
Nachdem die Repositories auf die korrekten Branches umgestellt wurden, konnte der Code kompiliert werden und das Team ließ den Turtlebot mit Hilfe des rviz-Plugins vom Rechner aus über den UDP-Proxy fahren. Während und nach der Arbeit an dem Problem des nicht kompilierenden Codes wurde gleichzeitig an dem Interface auf dem Rechner und dem Taster für den Turtlebot gearbeitet.

2.3.1 UI

Das User Interface (UI) wurde mit Hilfe der Software QtCreator in Qt erstellt. In dieser kann der Nutzer leicht Bedienelemente in ein Fenster hereinziehen. Aus dem zusammengebauten Fenster wird von dem Programm eine „.ui“-Datei erstellt. Für diese UI-Datei wurde danach in C++ eine Header- und Implementations-Datei mit den Namen „rqt_turtlebutler“ geschrieben.
Das Interface wurde zuerst von Eric zu Hause erstellt und danach gemeinsam in den Team-Treffen fertig gestellt. Die erste Version der UI enthielt noch Fehler im Programmcode, weswegen sie gebugfixt werden musste. Diese wurden zusammen behoben und die UI konnte zuerst am 9. Juli korrekt kompiliert werden.

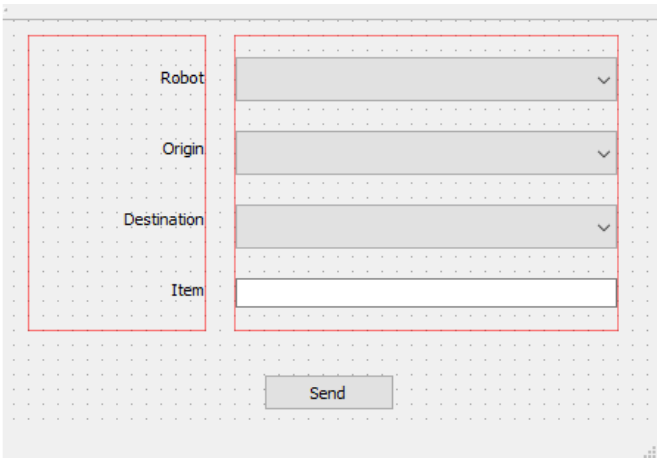

Abbildung 4: Alte Version des User-Interfaces in QTCreator

Wie auf Abb. 4 zu erkennen ist bestand die erste Version der UI aus einer Reihe an Dropdown-Menüs mit einem Knopf. Im ersten Dropdown-Menü kann der Nutzer zwischen einem der drei Roboter des Fachgebiets, Leonardo, Donatello und Raphael,

wählen. Die nächsten zwei Menüs dienen zur Auswahl des Abhol- und Zielpunktes. Zuletzt kann mit einer Texteingabe der gewünschte Gegenstand angegeben werden. Mit Hilfe des Drückens auf den Sendeknopf sollten diese Informationen nun in einer Nachricht verpackt werden, die an den gewählten Turtlebot geschickt wird.
In der ersten Version wurde zunächst eine Konfigurationsdatei namens „turtlebots.txt“ angelegt, die Informationen zu den Turtlebots und festgelegten Punkten auf der von einem Turtlebot aufgenommenen Karte des Fachgebiets enthält. Die Daten in der Datei wurden mit Hilfe von Semikolons und Zeilenumbrüchen getrennt. Zuerst wurden hier die Roboternamen mit ihren ROS-Topics gelesen und danach die Standortnamen mit deren X- und Y-Koordinaten. Die gelesenen Daten wurden danach in die jeweiligen Dropdown-Menüs geschrieben. Der Nutzer konnte hier nun die Namen von Turtlebots und Standorten sehen.
Da die Nachrichten ohne ein passendes Planungs- oder Behaviour-Skript auf der Roboter-Seite nicht funktionierten, wurde zuerst eine einfachere Nachricht verschickt. Hierfür wurde aus dem existierenden rviz-Plugin für den Turtlebot die jeweiligen ROS-Topics der Roboter und der Typ der Nachricht entnommen. Zuerst sollte dementsprechend eine „Pose_Stamped“-Nachricht basierend auf dem gewählten Abholpunkt verschickt werden. Erhält der Turtlebot mit den bereits existierenden Plänen die Nachricht, so fährt er zu der gegebenen Position auf der Karte.
Bei dem Erstellen der Konfigurationsdatei hat sich herausgestellt, dass für jede Position in der Datei von Hand eine Nachricht mit dem rviz-Plugin verschickt und deren Position notiert werden muss. Auf Grund dessen wurde nach einem Gespräch mit dem Betreuer entschieden, die bereits existierende Datei namens „TopologicalModel.conf“ zu verwenden. In dieser befindet sich eine Sammlung aus Points of Interest (POIs) der Räume des Fachgebiets basierend auf einer optimierten Karte.
Zum Einlesen der Datei konnte auf das existierende Paket „SystemConfig“ zurückgegriffen werden. Es liest die Datei ein und stellt die Daten geordnet über entsprende Methoden zur Verfügung. Die eingelesenen Daten konnte dann in eine Map gespeichert werden, die jedem Raum eine Liste von Points of Interests zuordnet.

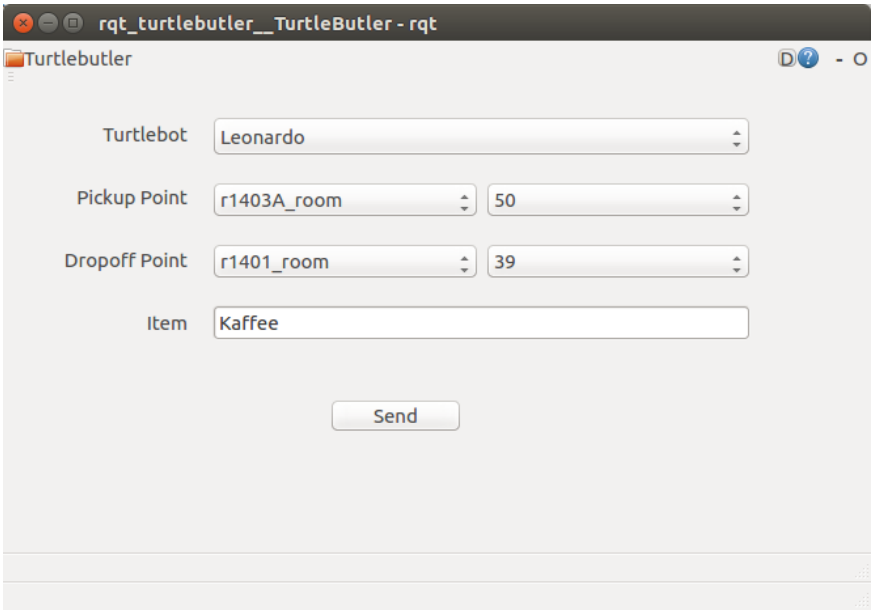

Abbildung 5: Aktuelle Version des User-Interfaces

Auf Grund der Datenstruktur der neuen POIs musste auch die UI verändert werden. Wie man an Abb. 5 und Abb. 6 sehen kann wurden die Dropdown-Menüs der Positionen in zwei Teile aufgeteilt. In dem linken Menü kann der Nutzer den Raum bestimmen und in dem rechten die Position im Raum. Wählt der Nutzer einen Raum, so ändert sich dadurch die Liste an verfügbaren Positionen. Für das Verschicken der Nachricht ist hierbei nur die Position relevant, da diese mit den X- und Y-Koordinaten verknüpft ist, die der Turtlebot benötigt. Da das Behaviour-Skript leider nicht fertig gestellt werden konnte, verschickt die neue UI weiterhin nur die Abholposition als „Pose_Stamped“-Nachricht.

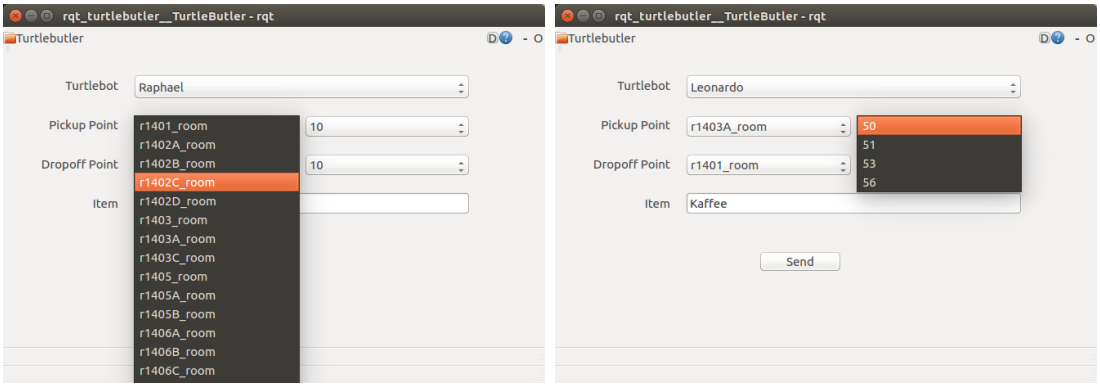

Abbildung 6: Dropdown-Menüs des neuen User-Interfaces

2.3.2 Taster

Dennis

- Rosserial Arduino zur Umwandlung in Nachrichten
- Entscheidung für Arduino
- Schreiben des Codes und Bauen des Tasters
- Einbauen des Tasters in das Weltmodell (Philipp)
- Verbesserung des Tasters mit Korb aus Haribo

2.4 Ausblick

Leider konnte der ursprüngliche Arbeitsauftrag aufgrund von technischen Komplikationen am Anfang und fehlender Zeit nicht zu Ende geführt werden. Das User Interface zum Verschicken der Nachrichten und der Taster zum Überprüfen von Gegenständen konnten zu einem Großteil fertiggestellt werden.
Das Interface lädt im Moment Kartendaten einer idealen Karte, die nicht den Schlupf der Räder des Turtlebots mit einbeziehen. Dadurch kann es sein, dass der Roboter bei längeren Fahrten seine Position verliert oder nicht an seinem Ziel ankommt. Deswegen müssen die momentan verwendeten Points of Interest mit Hilfe von Roboter-Informationen angepasst werden. Neben den idealisierten Positionen schickt die UI nur für den Abholpunkt eine „Pose_Stamped“-Nachricht. Diese sollte durch ein anderes Nachrichtenformat ersetzt werden, das beide Zielorte und den aufzunehmenden Gegenstand zusammen mit dem Absender enthält.
Der Taster wurde bereits in das Weltmodell eingebaut und aktualisiert seinen Wert jede Sekunde. Diese Daten müssen noch in einem Behaviour verarbeitet werden. Das fehlende Planungsbehaviour soll die Nachricht mit dem neuen Format verarbeiten und den Turtlebot zuerst zu dem Abholpunkt fahren lassen. Wird der Taster gedrückt, so soll sich der Roboter kurz danach zu dem Zielpunkt bewegen. Ist ein Turtlebot unterwegs zu einem Abhol- oder Zielpunkt, so soll dieser nicht verfügbar sein, und dem Rechner eine entsprechende Antwort schicken.
Während der Entwicklungszeit gab es zusätzliche Ideen, mit denen das Transportsystem erweitert werden kann. Zum einen könnte man auf dem Laptop ein Text-to-Speech-Programm verwenden, dass den in der Nachricht beschriebenen Gegenstand über die Lautsprecher ausgibt. Somit wüssten auch Unbeteiligte, was der Turtlebot abholen soll. Neben einer Sprachausgabe könnte man auch einen RFID-Leser in den Tragekorb einbauen, der beispielsweise auf RFID-Tags in Büchern oder Tassen reagiert. Dadurch kann der Turtlebot nicht nur überprüfen, dass etwas in dem Korb liegt, sondern auch welcher Gegenstand genau sich dort befindet.

3 Teamarbeit

3.1 Teamrollen

In dem Kickoff-Meeting wurden zwei Aufteilungen von Teamrollen vorgestellt: Basadur und Belbin.

Abbildung 7: Teamrollen nach Basadur

Auf Abb. 7 sind die Teamrollen nach Basadur zu sehen. Diese wurden in Generator, Conceptualizer, Optimizer und Implementer aufgeteilt. Der **Generator** ist jemand, der Dinge ins Rollen bringt und viele Ideen zur Problemlösung sucht. Hierbei ist es schwer, ihn auf eine Idee festzunageln. Der **Conceptualizer** nimmt die Ideen vom Generator und versucht diese zusammen mit eigenen zu verwenden, um nach Lösungen zu suchen. Er möchte dabei das Problem vollständig begreifen. Der **Optimizer** nutzt die abstrakten Ideen von Generator und Conceptualizer zur Umsetzung. Meist fokussiert er sich auf ein Problem und vertraut auf seine eigenen Fähigkeiten bei der Suche nach Lösungen. Der **Implementer** probiert Dinge lieber praktisch aus und verwirft Theorien, die nicht zu dieser Praxis passen. Er ist enthusiastisch und kann gut mit Menschen arbeiten. Dadurch wirkt er manchmal ungeduldig und aufdringlich.

Abbildung 8: Persönlichkeiten im Team nach Belbin

Auf Abb. 8 kann man die Team-Persönlichkeiten nach Belbin sehen, diese sind in drei Kategorien unterteilt, sodass ein Teammitglied meist mehr als einen Typ aufweist. Im Folgenden werden die Rollen mit jeweils einem Satz beschrieben.
Der **Weichensteller** kommuniziert innerhalb und außerhalb des Teams.
Der **Teamarbeiter** ist anpassungsfähig und arbeitet flexibel mit den anderen zusammen.
Der **Koordinator** versucht, alle zum gemeinsamen Ziel zu bewegen.
Der **Spezialist** hat besondere Fähigkeiten, die er zum Lösen der Aufgabe gebrauchen kann.
Der **Erfinder** findet Ideen, die als Grundlage für die Arbeit verwendet werden können.
Der **Beobachter** überdenkt Aufgaben kritisch und sucht nach Lösungen.
Der **Perfektionist** achtet auf Details und verfolgt Dinge bis zum Ende.
Der **Macher** arbeitet hochmotiviert und möchte etwas erreichen.
Der **Umsetzer** hat einen Sinn fürs Praktische und geht systematisch vor.

In den folgenden Abschnitten haben sich die Teammitglieder selber und gegenseitig in den Teamrollen nach Basadur und Belbin eingeschätzt.

3.1.1 Dennis

Basadur:
Belbin:

3.1.2 Robert

Im KickOff-Workshop hatte sich Robert im Teamrollen-Modell von Basadur als **Conceptualizer** eingeordnet, da er für Probleme gerne erst einen Lösungsansatz entwickelt, bevor er diese angeht. Diese Einschätzung hat sich gerade zu Beginn des Projekts

bestätigt, da er zum Beispiel die ersten Grafiken für eine mögliche Architektur entwickelt hat. Im Laufe des Projekts zeigte sich jedoch auch eine Tendenz zum **Implementer**, da nach Roberts Empfinden je nach Problem durch Ausprobieren und Testen schneller Erfolge erzeilt wurden. Weniger sieht sich Robert als **Optimizer**. Ideen in Lösungen umzusetzen fällt ihm ohne weiteren Input schwer. Das trifft vor allem zu, wenn die Ideen oder Konzepte nicht von ihm selbst stammen.
Bei Belbin sah sich Robert vor dem Projekt hauptsächlich in der der Rolle des **Perfektionisten**, da er sich für nachhaltige Problemlösungen einsetzt und sich gerne auch mal in kleine Teilprobleme verbeißt. Auch diese Einordnung lässt sich nach dem Projekt bestätigen, da er zum Beispiel bei einigen Problemen auf sauberere Lösungen bestanden hat, die jedoch mehr Zeit in Anspruch genommen haben. Außerdem passt in diesem Projekt die Rolle des **Spzialisten** gut zu Robert, da er im Team am meisten mit Linux gearbeitet hat.

3.1.3 Philipp

Basadur:
Belbin:

3.1.4 Eric

In den Teamrollen nach Basadur schätzt sich Eric als **Conceptualizer** und **Optimizer** ein. Als Conceptualizer hat er zu Beginn des Projektes Ideen gesammelt und das Problem in Unterprobleme aufgeteilt. Er versuchte die gesammelten Ideen umzusetzen und zu überdenken. Beim Umsetzen und Problemlösen hat er sich mit den anderen Teammitgliedern zusammengefunden und gemeinsam nach Lösungen und Lösungswegen zu suchen. Als Generator schätzt sich Eric weniger ein, da er sich meist auf eine Idee festlegt und selber selten zu Ideen kommt. Zusätzlich sieht er sich nicht als Implementer, da er meist eher theoretisch über Ideen nachdenkt und Theorien aufstellt. Trotzdem kann er, wie beim Implementer beschrieben, gut mit Menschen arbeiten.
Bei den Teamtypen nach Belbin sieht sich Eric als **Weichensteller** und **Koordinator**. Im Laufe der Teamarbeit hat er sich als Teamleiter etabliert, indem er die Arbeit bei den wöchentlichen Team-Treffen protokolliert hat. Dazu koordinierte er die Teammitglieder und versuchte bei Besprechungen jeden mit einzubeziehen. War ein Teammitglied nicht anwesend bei den Treffen, so wurde dieses von Eric entweder direkt danach durch die WhatsApp-Chatgruppe oder im nachfolgenden Treffen informiert, was es verpasst hatte. Eric lässt sich außerdem noch als **Spezialist** bezeichnen, da er vor dem Projekt bereits mit ROS, Arduino und UIs gearbeitet hat.

3.1.5 Vorwissen der Teammitglieder

Brauchen wir das hier???

- Robert: Erfahrungen in Linux
- Eric: Erfahrungen in UI-Programmierung
- Dennis: Erfahrungen mit Tastern und Hardware

3.2 Teamphasen

Dennis
Grafik der Phasen einbauen.

3.2.1 Forming

Bis wohin wurde nix geschafft? Teamfindung.

3.2.2 Storming

Aushilfe vom Betreuer. Sachen kompilieren und funktionieren.

3.2.3 Norming

Kickoff-Meeting.

3.2.4 Performing

Abschluss. Dokumentation.

3.3 Probleme und Lösungen

Philipp
Komplikationen mit dem Turtlebot

- Falsche Branches (Mit Betreuer gelöst)
- Akku kaputt (Tausch des Akkus durch Betreuer)
- Kommunikation nicht möglich (Deaktivieren eines Netzwerks)
- Kartendaten sind nicht akkurat auf den Turtlebot zugeschnitten

Am Protokol orientieren.

4 Fazit

In einer Gruppe bestehend aus zwei Informatik-Masterstudenten, einem Informatik-Bachelorstudenten und einem Elektrotechnik-Bachelorstudenten wurde gemeinsam ein Nutzer-Interface und ein Korb mit Taster gebaut.
(Gemeinsam, jeder ein Absatz?) Wie hat die Arbeit im Team funktioniert? Anwendung der Workshop-Sachen auf reale Teamarbeit.

5 Quellen

- ROS - http://www.ros.org/
- Turtlebot - https://www.turtlebot.com/
- QT und QTCreator - https://www.qt.io/
- RQT (ROS-QT) - http://wiki.ros.org/rqt