

1 Software-Entwicklung und Experimente

Aus den anfänglichen Ideen für die Umsetzung des Lighthouse Keeper entstehen in diesem Kapitel die ersten Module für dessen Verwirklichung. Zum Anfang wird die verwendete Hardware vorgestellt und in Verbindung dazu, die Aufgaben der genutzten Software-Komponenten näher erläutert. Durch die geeignete Wahl aller Bestandteile, lassen sich viele Aufgaben auf vorgefertigten Module übertragen. Beispielsweise werden statt einer eigenen Entwicklung, ein bestehendes Kommunikations-Framework und eine bereits existierende Middleware für den Roboter genutzt. Mit der Nutzung dieser fertigen Lösungen wird im großen Maße Arbeitsaufwand eingespart und ermöglicht einen stärkeren Fokus auf das eigentliche Thema. Da dies im Hinblick auf die Smartphone-Applikation nicht immer möglich war, wird deren Implementierung in diesem Kapitel exemplarisch einmal dargestellt.

Die durchgeführten Experimente beziehen sich in diesem Teil der Arbeit auf die Messung der Signalstärke der Beacons in Abhängigkeit zu deren Entfernung. Diese Phase in der Prozessplanung legt den Grundstein für die Modellbildung, welche anschließend im nächsten Kapitel vorgenommen wird. Ferner werden einzelne Einflüsse auf die Messungen näher betrachtet und eine Analyse zu der Batterielaufzeit eines Beacons in Abhängigkeit zu seinen Einstellungen aufgestellt.

1.1 Verwendete Hardware

Da die Beacon-Technologie vorrangig zur Indoor-Lokalisierung von Personen eingesetzt wird und als Peripheriegerät meistens ein Smartphone Verwendung findet, wird auch ein solches für die gesamte Testdauer als Messgerät genutzt. Die Wahl fiel dabei auf ein Android-Smartphone mit dem Namen Motorola Moto G der gleichnamigen Firma Motorola Inc. Es wurde ausgewählt, weil es alle Hardware- und Software-Anforderungen zum Empfang von BLE-Signalen erfüllt und als ein Standard-Smartphone gilt, sodass sich mit den ihm erzielten Ergebnissen auch auf andere Produkte übertragen lässt. Des Weiteren werden zwei Roboter in den Experimenten genutzt, um die Messungen reproduzierbar und standardisiert durchzuführen. Für die reinen Distanz-Signalstärke-Messungen wird der Roboters bzw. lediglich sein Arm namens „Youbot“ der Kuka AG und später für die Validierung einer Beacon-Konfiguration der Roboter „Scitos G5“ der MetraLabs GmbH verwendet.

1.1.1 Motorola Moto G

Das Moto G dient als Empfangsstation der BLE-Signale, dessen grundlegende Spezifikationen ein 1,2 GHz Snapdragon 400 Prozessor mit 1 GB RAM und ein WCN3620 BT/FM/WLAN RF Modul aussmachen ([?]). Seine Abmaße betragen 129.9 mm × 65.9 mm × 11.6 mm bei einem Gewicht von 143 g. Des Weiteren verwendet es standardmäßig ein Android 4.3 als Betriebssystem, welches jedoch für die Experimente auf die Version 4.4 geupdated wurde. Neben der technischen Ausstattung und Software sind für die späteren Messungen die Antennen und deren Charakteristiken von hoher Bedeutung, denn deren Eigenschaften wirken sich direkt auf den Empfang der Signale aus. Um die Einflüsse besser zu verstehen, sind in den Abbildungen 1.1 und 1.2 die Anordnung der Antennen einmal skizziert. Dabei fällt es auf,



Abb. 1.1: Vorderseite des Motorola Moto G [?]

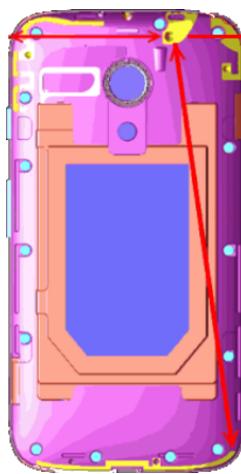


Abb. 1.2: Rückseite vom Moto G mit Antennen-Gerüst [?]

dass sich WLAN- und Bluetooth-Modul die selben Antennen teilen. Dies führt zu der Frage, ob es zu Konflikten in der Funktionsweise des Smartphones kommt, wenn gleichzeitig auf beide Module zugegriffen wird. Jedoch dazu mehr im Abschnitt der App-Entwicklung. Die zweite Frage die sich daraus stellt, ist die Veränderung der Empfangs- und Sendequalität des Moto G in verschiedenen Positionen. Eine Antenne hat je nach Bauform und Funktionsweise Bereiche, in der sie mit voller Leistung sendet und empfängt, aber auch Bereiche in der Funk-Signale sie weder verlassen noch erreichen können. Das hat verschiedene physikalische Gründe, jedoch sind diesen komplexen nichtlinearen Eigenschaften der Antennen des Moto G zumindest nicht öffentlich bekannt und können auch nicht einfach bestimmt werden. Es sei nur anzumerken, dass bei den Messungen auch darauf geachtet werden muss, wie und an welchen Halte-Punkten das Moto G am besten befestigt wird, ohne deren Transceiver-Fähigkeiten negativ zu beeinflussen. Eine gute Annahme dabei ist es das Smartphone so auszurichten, als ob es flach in der Hand eines Menschen liegen würde. Der Hersteller wird schließlich darauf bedacht sein, sein Produkt für einen normalen Betrieb auszulegen und somit auch die Konstruktion und Bau der Antennen danach optimieren. Diese Annahme muss jedoch noch anhand von Messungen verifiziert werden.

1.1.2 Youbot

In vorigen Abschnitt wurde schon angesprochen, dass die Lageposition des Messinstrumentes zu seiner Empfangsleistung überprüft werden muss. Zudem soll das Smartphone so gehalten werden, als wenn es sich in einer flachen Hand befindet und so auch die Experimente durchgeführt werden. Um all dies zu erreichen und auch unter der Anforderung an einen automatisierten und reproduzierbaren Prozess, empfiehlt es sich einen Roboterarm als Mess-Plattform zu benutzen. Aufgrund der Verfügbarkeit wurde das Modell „Youbot“ der Firma „Kuka“ gewählt und für die Messungen mit einer Halterung aus einem 3D-Drucker ergänzt (siehe Abbildung 1.3). Die Vorteile des Systems sind zum einen der montierte hochpräzise Roboterarm mit Greifer auf dem Youbot und zum anderen, dass ein vollwertiger Rechner mit einem Linux Betriebssystem und eine drahtlose WLAN-Schnittstelle im System verbaut sind und so die Kommunikation zum Roboter sehr einfach aufgebaut werden kann. Der künstliche

Arm kann sich dabei um seine fünf Achsen drehen (siehe Abbildung 1.4) und bietet somit genug Möglichkeiten, die Lageposition vom Moto G zu verändern.



Abb. 1.3: Youbot mit Halterung (gelber Aufsatz)

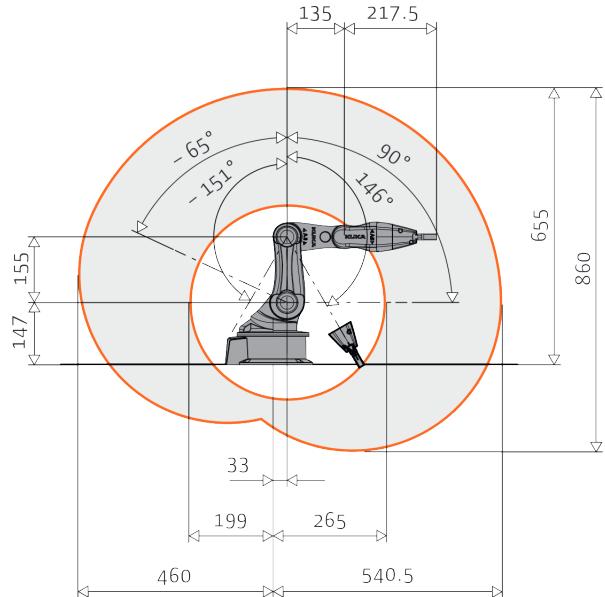


Abb. 1.4: Zeichnung eines Youbot-Arms und seiner fünf Rotationsachsen [?]

1.1.3 Scitos G5

Die Durchführung der Evaluation einer Beacon-Konfiguration wäre grundsätzlich auch mit dem modifizierten Youbot aus Abschnitt 1.1.2 möglich. Jedoch wurde es in der Aufgabenstellung gefordert, den Roboter „Scitos G5“ von der MetraLabs GmbH zu verwenden. Der Scitos G5 ist mit den Maßen 55 cm × 60 cm × 60 cm etwas größer als der Youbot und durch seinen Dreirad-Lenkung weniger wendig (vgl. Abbildung 1.5). Die Vorteile gegenüber dem Youbot bestechen jedoch durch seine bessere Ausstattung und den größeren Software-Umfang, weswegen es keiner zusätzlichen Entwicklungen bedarf und somit den Arbeitsaufwand für das Lighthouse Keeper-Konzept verringert wird. Der Scitos G5 verfügt über Schrittmotoren, einem Laserscanner und Inertialsensoren zur Navigation und er besitzt zudem einen leistungsfähigen Intel Core i7-Prozessor, WLAN und ein Linux-Betriebssystem, sodass auch hier genug Ressourcen für eine Kommunikation vorhanden sind. Die Machbarkeit des gesamten Kontroll-Prozesses von Beacon-Konfigurationen beruht dabei auf der Leistungsfähigkeit von diesem Roboter, denn er muss in der Lage sein, seine Position unabhängig von der Trilateration von Beacon-Signalen zu bestimmen und diese als Referenzquelle zur Verfügung zu stellen. Für seine Positionsbestimmung verwendet er die Software-Umgebung „Miracenter“ (siehe 1.2.2 im nächsten Absatz), welche die Sensordaten aus der Wahrnehmung (Motoren, IMU, Laserscanner) so verwertet, dass sie eine Karte der Umgebung erstellen kann und somit im Abgleich von Sensorinformationen und Karte der Roboter stets seine Position kennt. Der Hersteller wirbt dabei mit einer Nutzlast von bis zu 50 kg und einer Maximalgeschwindigkeit von $1,4 \frac{m}{s}$ bei einer batteriebetriebenen Laufzeit von ca. 20 Stunden [?], sodass ein künstlicher Roboterarm auf dem Scitos G5 ebenfalls denkbar wäre und somit auch die Aufgaben vom Youbot zukünftig übernimmt.



Abb. 1.5: Scitos G5 von der MetraLabs GmbH

1.2 Verwendete Software

Um die Hardware zu nutzen und das geplante Konzept umzusetzen, benötigt es einer Kommunikation zwischen den Geräten und weiterer Werkzeuge zur Aufnahme von Messungen und deren Verarbeitung. Bei der Umsetzung wurde besonders auf Konfirmität der verschiedenen Systeme und deren reibungslosen Zusammenspiels geachtet. Um eine gemeinsame Basis zu schaffen, wurde das Software-Framework „Robots Operating System“(ROS) verwendet. Mit einem gemeinsamen Standard lassen sich die Messungen besser vergleichen, wodurch ihre Qualität und Aussagekraft zunimmt. Die Messungen müssen dabei auf der Smartphone-Plattform und den Roboter-Plattformen aufgenommen und diese synchronisiert werden. Während ROS die übergeordnete Schnittstelle darstellt, müssen auf den einzelnen Hardware-Elementen die Messungen eigenständig durchgeführt werden. Die Messung auf dem Scitos-Roboter entfällt dabei auf die Software „Miracenter“ und funktioniert ohne weiteres. Die Software für die Messungen auf dem Smartphone ist hingegen nicht vorgefertigt und muss mithilfe eines Editors für Android-Applikationen und einer speziellen Bibliothek für die Kommunikation Smartphone ↔ Estimote Beacon entwickelt werden.

1.2.1 Robot Operating System – ROS

Das Robot Operating System oder kurz ROS, ist ein Projekt für die Kommunikation verschiedener Hardware-Plattformen untereinander. Anhand der Namensgebung leitet sich auch schnell der Einsatzzweck – nämlich für Robotersysteme – her. Dabei liegen die Schwerpunkte des Projektes auf [?]:

- „Peer to Peer“ (P2P)-Verbindungen
- Modularer Aufbau
- Unterstützung mehrerer Programmiersprachen
- freier Nutzung und Open-Source

Ein System das auf ROS aufbaut, besteht dabei aus mehreren miteinander verbundenen Rechnern (sog. „Hosts“) die zur Laufzeit über P2P miteinander kommunizieren. Bei der

P2P-Verbindung können alle Teilnehmer ihre „Dienste“ bzw. ihre Informationen gleichermaßen einander anbieten und nutzen, indem sie Daten im Netzwerk gleichzeitig empfangen und senden können. Dabei läuft auf einem zentralen Server der eigentliche Kern des Frameworks, über den der sämtliche Datenverkehr geleitet wird. Und auf den Host-Systemen laufen die eigentlichen Anwendungen („Nodes“), die über Schnittstellen („Sockets“) des Betriebssystems ihre Daten auf das Netzwerk und schließlich an ROS verteilen. Ausgehend von den Nodes werden den gesendeten Nachrichten gesondert Bezeichnungen („Topics“) zugeordnet und mit einem Datentyp versehen. Diese Informationen sind allen Teilnehmer des P2P-Netzwerkes bekannt und können von ihnen angefordert werden. Der Kern organisiert dabei eine einheitliche Uhrzeit, die dadurch für alle Nachrichten bzw. deren Zeitstempel in den verteilten Systemen konsistent bleibt und so eine Synchronisierung der Messwerte nicht mehr nötig wird. Somit ermöglicht die Verwendung von ROS den Aufbau des gesamten Kommunikations-Frameworks, wie es in der Konzept-Planung in ?? festgehalten wurde.

1.2.2 Miracenter

Das Paket „CogniDrive“, als Bestandteil der Software „Miracenter“ von der Firma Metralabs GmbH, dient als Navigator des Scitos G5 und ermöglicht es mit ihm den Grundriss eines Raumes zu erstellen (siehe Abbildung 1.6) und zusätzlich die Lokalisierung des Roboters in diesem durchzuführen. Während beispielsweise der Roboter entlang der Wände fährt, messen seine Schrittmotoren den zurückgelegten Weg, die Inertialsensoren verfeinern die Informationen und erweitern sie um die Ausrichtung des Roboters. Dabei misst zusätzlich der Laserscanner die Distanz zu den Wänden und allen anderen festen Objekten in Reichweite. Aus der Fusion aller Messwerte lässt sich so der Grundsriß des vermessenden Raumes generieren. Im nebenstehendem Bild ist eine solche Karte als Beispiel aufgeführt. Die Daten die Miracenter dem Nutzer zur Verfügung stellt, bestehen dabei aus der Position des Roboters und seiner Ausrichtung in der Karte. Die Karte liegt dabei als „Portable Network Graphics“ (PNG)-Datei vor, wobei ein Pixel einer konstanten metrischen Länge entspricht, deren Verhältnis in einer „Extensible Markup Language“ (XML)-Datei definiert ist. Die Farbwerte der Bildpunkte beschreiben zudem, ob an einer Stelle ein Hinderniss oder ein frei befahrbarer Raum vorliegt. In der Abbildung ist zu erkennen, dass die erstellte Karte teilweise verrauscht ist, Wände nicht gerade verlaufen, oder offene Türen und herumlaufende Personen die Messungen verfälschen. Die Karte kann dafür nach der Erstellung von jedem beliebigen Bildbearbeitungsprogramm geöffnet und bearbeitet werden. Dabei ist darauf zu achten die Auflösung nicht zu verändern, da ansonsten die Dimensionen nicht mehr übereinstimmen oder gegebenenfalls die XML-Datei anpasst werden. Nach

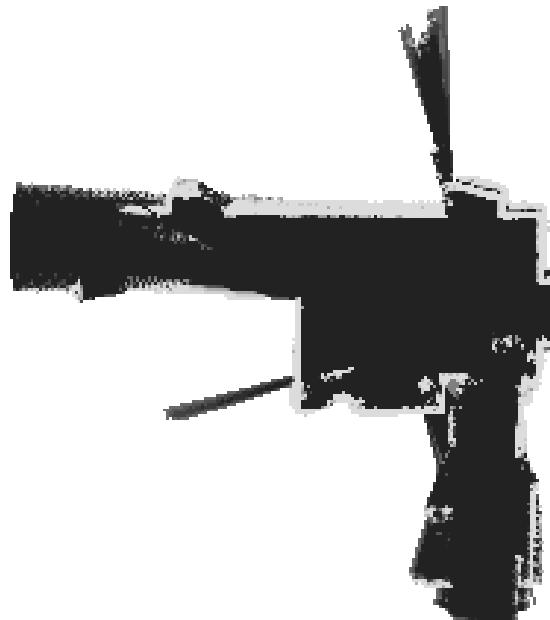


Abb. 1.6: Erstellter Grundriss eines Flures in Gebäude 29 der OvGU, Stockwerk 3

der Bearbeitung der Bilddatei kann sie anschließend im Miracenter geladen werden. Mittels Mausklick in die Karte wird die ungefähre Position des Roboters bestimmt. Danach orientiert sich der Scitos G5 automatisch und lokalisiert sich im weiteren Verlauf selbst anhand seiner Sensordaten. Nun können ihm Positionen und Ausrichtung vorgegeben werden und durch seiner internen Pfadplanung steuert er sie autonom an. Durch ein Miracenter-ROS-Interface kann dabei die interne Lokalisierung und die Vorgabe von Position und Ausrichtung vom Scitos extern übermittelt werden. Die gesamte Software ist jedoch proprietär, d.h. nicht quelloffen, sodass ein tieferer Blick in die Funktionsweise der Software hier verwehrt wird.

1.2.3 Android-Studio, Estimote SDK und ROSjava

Als letzten Baustein in der Software-Architektur fehlt die App für das Moto G, deren Entwicklung auf dem Zusammenwirken dreier Grundpfeiler aufgebaut sein wird:

1. Das Android-Studio [?] ist eine Entwicklungsumgebung für Programme speziell von Android-Smartphones. Es bietet die Mittel eine Applikation, basierend auf der Programmiersprache Java, auf ein Android-Gerät zu portieren und dort auch zu testen. Aufgrund der Implementierung der „Integrated Development Environment“ (IDE) in Java, lässt es sich auf beinahe jedem Betriebssystem verwenden und bietet dadurch eine hohe Flexibilität in der Anwendung.
2. Damit die Applikation die Beacon-Signale verarbeiten kann, müssen die gesendeten Nachrichtenpakete und das verwendete BLE-Protokoll in das Programm eingebettet werden. Vom Hersteller der Beacons wird dafür eigens eine Bibliothek unter dem Namen „Estimote Software Development Kit (SDK)“ [?] bereitgestellt, die sich problemlos integrieren lässt und einfach zu bedienen ist.
3. Um die empfangenen Signale an den Server via P2P zu senden, muss eigens eine Schnittstelle von der Java-basierten Android-App zum in C/C++ gehaltenen ROS implementiert werden. Dafür wird das Paket „Rosjava“ [?] benötigt, um die Unterstützung von ROS-Funktionen für Java-Programme zu realisieren. Dies erlaubt es ROS-Pakete in eine Android-App zu integrieren und dadurch die zu übermittelnden Nachrichtenformate darin auch zu verwenden.

1.3 App-Entwicklung

Um das geplante Programm auf dem Smartphone zu verwirklichen, werden die aus ?? beschriebenen Hilfen und zudem eigener Programmcode benötigt. In diesem Abschnitt werden die einzelnen Elemente der App erläutert und ihr Zusammenspiel in der fertigen Applikation anschließend betrachtet.

1.3.1 Beacon-Detektierung

Der kritischste, aber der auch am essentiell wichtigste Programmteil ist die Aufnahme von Beacon-Signalen. Hierfür muss die SDK von Estimote in die App über das Android-Studio als Bibliothek importiert werden. Daraufhin stehen neue Klassen und Objekte der IDE zur Verfügung, die speziell auf die Verwendung von Beacons zugeschnitten sind. Zum einen muss dabei die App bei jedem Start für die Nutzung mit Beacons mit Abfragen neu initialisiert

werden, das wären zum Beispiel die Überprüfung, ob Bluetooth angeschalten ist oder ob BLE überhaupt auf dem Gerät unterstützt wird. Des Weiteren muss der sogenannte Beacon-Manager, welcher im Hintergrund der App läuft, mit Zeiten für die Pause zwischen zwei Abtastungen und der Länge eines Scans, eingestellt werden. Die Problematik der Abstastraten wird hierbei später im Programmablaufplan noch erörtert. Zum anderen muss eine Art Empfangs-Funktion geschrieben werden, die bis zur Beendigung des Programms auf Meldungen der Leuchtfeuer wartet und diese für die Verarbeitung im Programm aufbereitet. Die Anwendung von Filtern oder dergleichen entfällt hier, da schließlich das Verhalten der Signalausbreitung studiert werden soll und zusätzliche Einflüsse die Ergebnisse verfälschen könnten. Interessanter wird es bei der Frage, wie häufig die Abfragen der Eingänge nach Beacon-Signalen stattfinden sollen. Hierbei stellt sich noch eine viel wichtigere Frage und zwar der nach der Funktionsweise der SDK. Also wie die Datenströme, die von den Beacons in Intervallen gesendet, darauf vom Beacon-Manager aufgenommen werden. Denn die Sende-Intervalle der Beacons können variieren, während die Applikation mit festen Werten für alle Beacons initialisiert werden muss. Die Antwort nach den Abläufen im Beacon-Manager kann leider nicht beantwortet werden, da die Entwickler keinen Einblick in ihren Quellcode gewähren und auch sonst keine Angaben oder Dokumentationen veröffentlichen. Eine bessere Veranschaulichung dieser Problematik findet sich im Abschnitt über den Programmablauf.

1.3.2 Lagemessung

Parallel zur Beacon-Detektierung soll gleichzeitig die Lage des Smartphones gemessen werden, um die Antennen-Charakteristiken besser zu verstehen und ihre Einflüsse auf die Empfangsqualität zu untersuchen. Für diese Aufgabe müssen die Inertialsensoren aus Gyroskop, Beschleunigungssensor und Magnetometer vom Smartphone ausgelesen werden, was jedoch einfach umzusetzen ist, da die IDE nativ dafür Bibliotheken bereitstellt.

Während das Auslesen der Information über die Lage des Gerätes ... Überdenken, ob ich das mit rein nehmen will. Ich habe zwar den Komplimentärfilter für den Gyro benutzt, doch die Frage bleibt, ob es das Wert war bzw. erwähnenswert ist.

1.3.3 ROS-Anbindung