

1 Software-Entwicklung und Experimente

Konzeptumsetzung, erste Messwerte für die Erstellung eines Modells. Dazu wird die verwendete Hardware und Software beschrieben sowie die App-Entwicklung und die Middleware des Roboters dokumentiert. Der Simulationsumgebung wird aufgrund ihres großen Umfangs das nächste Kapitel gewidmet.

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., weil es weit verbreitet ist und somit ein großer potentieller Nutzerkreis existiert. Zudem erfüllt es alle Hardware- und Software-Anforderungen zum Empfang von BLE-Signalen. 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

Dieses Smartphone dient als Empfangsstation des Beacon-Signale.

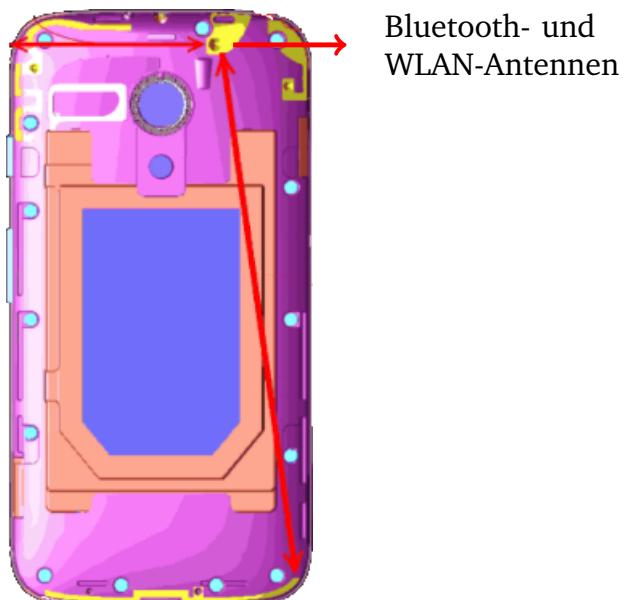


Abb. 1.1: Anordnung der Antennen an der Rückseite eines Motorola Moto G [?]

!Atennencharakteristiken! Idee für die Erklärung der seltsamen Empfangsqualität bei den verschiedenen Ausrichtungen: durch die Verteilung der Antennen ergibt sich auch eine Verteilung der Empfangscharakteristiken. Die dabei aufgenommenen Messwerte sehen bei den

unterschiedlichen Ausrichtungen deshalb immer gleich aus, auch wenn die Signale aus einer anderen Richtung kommen, weil die Charackteristik speziell auf auf die normale Position optimiert wurde. Durch die Komplexität der Charackteristiken lässt sich dies nun schlecht visuell darstellen. Wenn man das Verhalten aber kennt (vorsicht, jedes Smartphone hat eine unterschiedliche Charakteristik), kann man Gegenmaßnahmen treffen. -> Heuristik

1.1.2 Youbot



Abb. 1.2: Youbot von der Kuka AG [?]

1.1.3 Scitos G5



Abb. 1.3: 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 ROS

Hier wird ROS erklärt.

1.2.2 Miracenter

Mit der Software „Miracenter“ der Firma MetraLabs GmbH ist es möglich, mit einem Scitos G5 Roboter den Grundriss eines Raumes zu erstellen. Als Sensorinformationen bezieht die Software Daten von den Inertialsensoren, Schrittmotoren und des Laserscanners und kombiniert diese zu einer Karte. Dabei wird mithilfe der Schrittmotoren der zurückgelegte Weg gemessen und diese Information mit den Inertialsensoren verbessert. Der Laserscanner sucht Erklärung der Funktionsweise des Frameworks dabei nach festen Objekten und misst die Entfernung vom Roboter zu den Objekten. Im nebenstehendem Bild ist eine solche Karte als Beispiel aufgeführt. Die Daten die Miracenter dem Nutzer zur Verfügung stellt, bestehen aus der Position des Roboters und seiner Ausrichtung in der Karte. Die Karte liegt dabei als PNG-Datei vor, wobei ein Pixel einer konstanten metrischen Länge entspricht. Die Farbwerte der Bildpunkte beschreiben zudem, ob an einer Stelle ein Hinderniss oder ein frei befahrbarer Raum vorliegt. In 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 verändert werden. Dabei ist darauf zu achten die Auflösung nicht zu verändern, da ansonsten die Dimensionen nicht mehr übereinstimmen. Nach 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 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 dem Nutzer verwehrt bleibt.

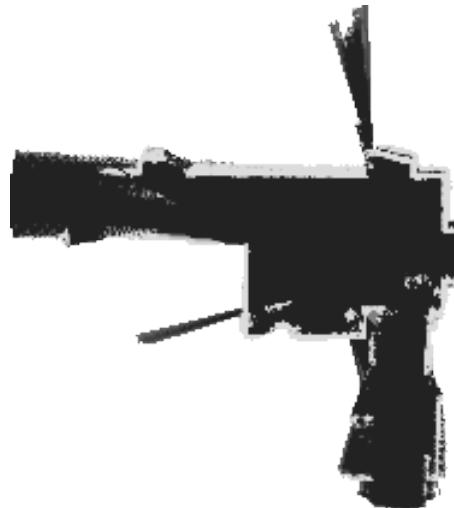


Abb. 1.4: Grundriss eines Flures in Gebäude 29 der OvGU, Stockwerk 3

1.2.3 Android-Studio

Programm zur Erstellung von Apps.

1.2.4 Estimote SDK for Android

Bibliothek für die Einbindung in eine Android-App. Hier kann schon auf Teile in der App eingegangen werden.

1.3 App-Entwicklung

Was muss die App leisten?

1.3.1 ROS-Anbindung

Kurze Beschreibung des Talkers.

1.3.2 Lagemessung

Auslesen des Gyros und der Komplementärfilterung.

1.3.3 Beacon-Detektierung

Probleme ansprechen mit gleichzeitiger Nutzung von WLAN und Bluetooth. Am Ende ein schöner Programmablaufplan.

1.4 Experimente

1.4.1 Versuchsplanung

Nötig?

1.4.2 Messung der BLE-Signalausbreitung

Zuerst werden zwei bis drei Distanzen und dazu die berechnete Distanz aus der App gegenübergestellt. Das wird schlecht sein und aus dem weitren Grund, dass es eine „Black Box“ ist (d.h. man kann nicht einsehen, wie die Distanz aus dem RSSI-Wert berechnet wird), wird hier geplant ein eigenes Modell aufzustellen. Da wurde entschieden für verschiedene Distanzen nur die RSSI-Werte aufzunehmen, um später ein eigenes Modell zu implementieren.



Abb. 1.5: Distanz-Signalstärke-Messung Bild 1

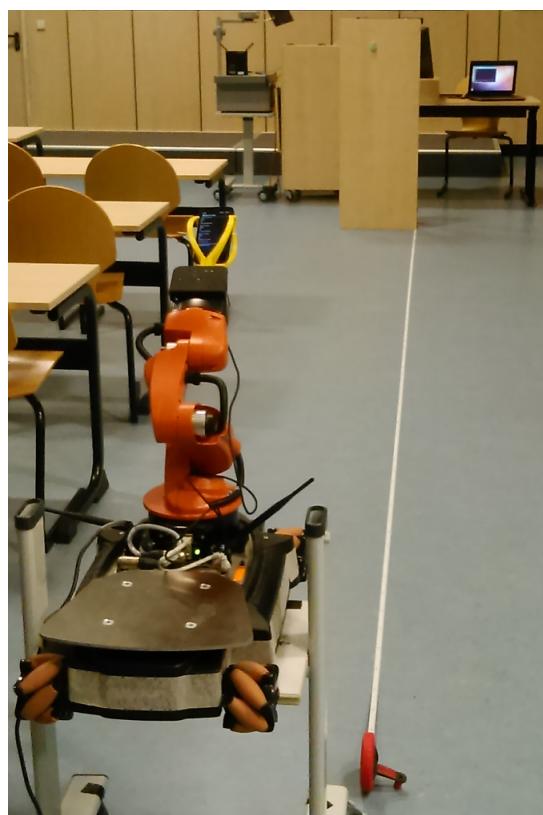


Abb. 1.6: Distanz-Signalstärke-Messung Bild 2



Abb. 1.7: Distanz-Signalstärke-Messung Bild 3

Einfluss der Intervalllänge

Wird das Rauschen vermindern.

Einfluss der Signalstärke

Dies soll auch in Abhängigkeit zur Signalsstärke passieren, um dessen Ausbreitung zu erforschen.

1.4.3 Auswirkung der Smartphone-Ausrichtung

Hier kommen mal alle Messungen rein. Auch die mit den Beacons an allen drei Himmelsrichtungen.

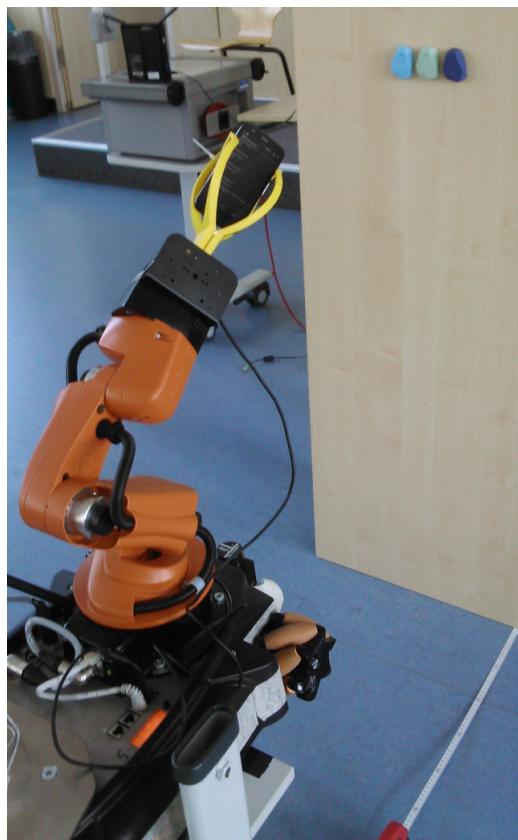


Abb. 1.8: Distanz-Signalstärke-Messung bei unterschiedlichen Smartphone-Ausrichtungen

1.4.4 Einfluss anderer Signalquellen

Für die Erklärung aus dem zweiten Kapitel (2,4 Ghz) benötige ich auch die Störung und Interferenzen gleicher Signale. Ich muss ja schließlich erklären, warum ich nicht viele viele iBeacons nebeneinander klatschen kann, um eine möglichst genaue Standortbestimmung zu erhalten (denn die Signale stören sich untereinander). Es ist z.B. aufgefallen, dass die Beacons sich gegenseitig in der Sendestärke beeinflussen. Deswegen musste ich Einzelmessungen vornehmen und hier ließe sich auch begründen nicht zu viele Beacons in der realen Anwendung zu plazieren, um durch Elektrosmog nicht zu viele Einflussfaktoren in die Lokalisierung miteinfließen zu lassen. Viel hilft hier nicht immer viel, weniger ist manchmal besser, etc.



Abb. 1.9: Gleichzeitige Messung dreier naheliegender Beacon-Signale

1.4.5 Energieverbrauch eines Beacon

Der Wartungsaufwand für ein Beacon hängt maßgeblich an der Laufzeit der in jedem Beacon eingebauten Batterie ab. Die in Kapitel 1 und 2 erwähnten Einstellmöglichkeiten von Sendeleistung und der Häufigkeit von Sendeintervallen beeinflussen dabei den Stromverbrauch der Beacon und somit auch die Betriebsdauer der Batterien. Dieser Abschnitt soll sich damit beschäftigen, inwieweit sich die Parameter auf die Wartungszyklen auswirken. Dies soll später dabei helfen, zwischen der Auslegung der beiden Parameter und des gewünschten Wartungsaufwandes abwägen zu können. Denn in großen Projekten mit mehreren hundert Beacons steigt der Wartungsaufwand proportional und wenn einzelne Beacons andere Einstellungen als die Mehrheit aufweisen (um beispielsweise wichtige Gebiete besser abzudecken), wird auch der Überblick über die nötigen Wartungszyklen verloren gehen. Um die Abhängigkeiten zu veranschaulichen, wird anhand vom Datenblatt des verwendeten Hardware-Chip nRF51822[18] in den genutzten Beacons eine Simulation entworfen, die die Einstellmöglichkeiten als Variablen betrachtet. Als Ausgangsbasis für den Energiespeicher wird eine CR2450-Batterie mit 1,8 Wh[?] angenommen. Hierbei wurde nicht vom Idealzustand ausgegangen, sondern ein üblicher Faktor von 0,7 hinzu multipliziert, um der Alterung der Zelle und weiterer Effekte Rechnung zu tragen. Durch die gering fließenden Ströme und der minimalen Bauweise der Beacons, können auch keine direkten Messungen vorgenommen werden. Zudem kann auch nicht verifiziert werden, wie lange ein Sendeaufrag oder

ein Zustand vom Prozessor dauert. Um trotzdem ein gutes Simulationsergebnis zu erreichen, muss tiefer in die Funktionsweise der Beacons geschaut werden. Als größte Unbekannte tritt dabei die Sendedauer auf. Bei einer angenommenen Datenrate von 250Kbit/s und bei einer Größe eines Datenpaketes von 38 Bytes [?] (siehe ??), dauert eine Sendesequenz rund 1 ms. Die benötigte elektrische Leistung für eine vorher definierte Sendeleistung beträgt dabei ein Vielfaches dessen und ist davon stark nichtlinear abhängig. Die dazu nötigen Beziehungen können anhand des Datenblattes des nRF51822 Chips hergeleitet werden. In diesem Fall wurde ein Polynom 4. Grades als Funktion der einstellbaren Sendeleistung aufgestellt, welches annähernd diese Abhängigkeiten beschreibt. Da die Kommunikation bidirektional ist, existiert auch ein Empfangsmodus der ebenfalls mit 1 ms Empfangsdauer angenommen wird, aber dessen Modus hingegen einen konstanten Stromverbrauch aufweist.

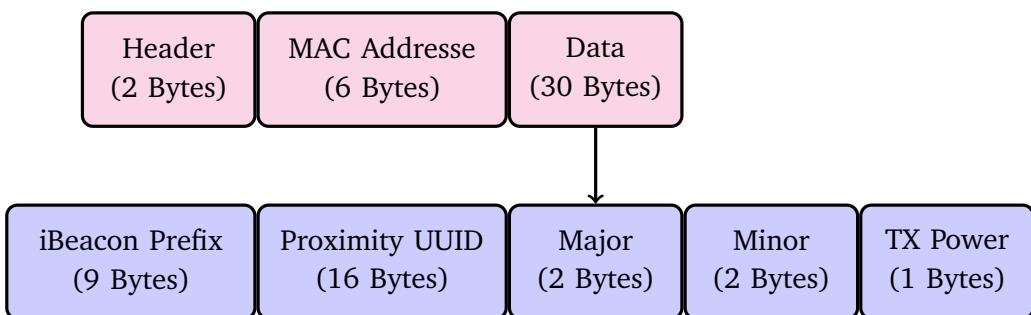


Abb. 1.10: Anteile eines Datenpaketes in der Beacon-Kommunikation auf der MAC-Ebene

Bei den tatsächlichen Laufzeiten des Sende- und Empfangsmodus muss jedoch ein Unsicherheitsfaktor mit eingerechnet werden, sodass nicht der berechnete Wert für eine Übertragung genommen wurde, sondern das Zweifache dessen, um ein weniger idealisiertes Bild für die einzelnen Vorgänge zu gestalten.

$$t_{TX} = 2 \text{ ms} \cdot \text{Intervalle [Hz]}$$

$$t_{RX} = 2 \text{ ms} \cdot 1 \text{ Hz}$$

$$t_{CPU} = t_{TX} + t_{RX}$$

$$t_{IDLE} = 1 - t_{CPU}$$

Da für den Sendevorgang noch zusätzlich der Cortex M0 Prozessor aufgeweckt werden muss und dieser das Paket vorbereitet und bis zum Ende der Sendung abwartet, muss hierfür zusätzlich der Energieverbrauch berechnet werden. Da wie erwähnt eine direkte Leistungsmessung an den Beacons durch die gering fließenden Ströme nicht möglich ist, werden aus dem Datenblatt des Prozessors diese rechnerisch ermittelt. Die aktive Phase des Prozessors für alle Operationen vor und nach einer Sendung werden dabei mit 2 ms Sekunden angenommen, welche exakt der Sende- und Empfangsdauer entspricht. Der Energiebedarf der Recheneinheit wurde im Datenblatt mit 4,1 mA bei einem Takt von 16 MHZ angegeben. So ergibt sich für den Prozessor eine Leistung von 12,3 mW. Zusätzlich wird der Stromverbrauch im IDLE-Modus (Schlafphase des Prozessors) mit 2,6 μ A angegeben. Aus den genannten Gegebenheiten lassen sich nachfolgende Gleichungen erstellen und damit eine Simulation eines

Lebenszyklus von einer Batterie näherungsweise bestimmen:

$$\begin{aligned}
 E_{Bat} &= & 1,8 \text{ Wh} \cdot 0,7 &= & 1,26 \text{ Wh} \\
 L_{CPU} &= & 4,1 \text{ mA} \cdot 3\text{V} &= & 12,3 \text{ mW} \\
 L_{TX} &= \text{Polynom 4. Grades (von } 4,7 \text{ mA } \sim 11,8 \text{ mA) } \cdot 3\text{V} &= & 14,1 \text{ mW} \sim 35,4 \text{ mW} \\
 L_{RX} &= & 6,1 \text{ mA} \cdot 3\text{V} &= & 18,3 \text{ mW} \\
 L_{IDLE} &= & 2,6 \mu\text{A} \cdot 3\text{V} &= & 7,8 \mu\text{W}
 \end{aligned}$$

ergeben den Zusammenhang

$$t_{Laufzeit} = \frac{E_{Bat}}{L_{CPU} \cdot t_{CPU} + L_{TX} \cdot t_{TX} + L_{RX} \cdot t_{RX} + L_{IDLE} \cdot t_{IDLE}}$$

und dies ergibt folgendes Simulationsergebniss: Aufgrund der getroffenen Annahmen würde eine Batterie in den niedrigsten Einstellungen für rund 2 Jahre den Betrieb eines Beacon ermöglichen. Im realen Betrieb hatte sich gezeigt, dass eine Batterie schon nach 4 bis 5 Monaten gewechselt werden musste. Während dieser Zeit wurden die Einstellungen nicht geändert und bei einer Intervalllänge von 5 Hz und Sendeleistung von -12 dBm belassen. Beides stimmt mit der Simulation ungefähr überein, was noch kein Beweis für die Gultigkeit der Annahmen bedeutet, jedoch wird der qualitative Verlauf der wechselseitigen Beziehungen von Einstellparameter und Batterilaufzeit skizziert, welcher als Orientierung zur Auslegung der Parameter benutzt werden kann.