Indoor-Lokalisierung mit Hilfe neuronaler Netze

Pascal Oblonczek, Sebastian Zahn

Thema der Bachelorarbeit

Indoor-Lokalisierung mit Hilfe neuronaler Netze

Stichworte

Lokalisierung, Mobile Location, Neuronales Netz, MLP, Backpropagation, Lernbasiert, Location Service, Triangulation, Trilateration, WLAN, UWB

Kurzzusammenfassung

Heutzutage stehen zahlreiche Verfahren zur Lokalisierung in geschlossenen Räumen zu Verfügung, die unterschiedlichste Anforderungen abdecken. Im Rahmen von bewohnten Räumen ist die Verwendung von WLAN-Signalen der bereits vorhandenen Kommunikationsinfrastruktur eine sehr kostengünstige Lösung. Es müssen jedoch starke Schwankungen in der Signalstärke aufgrund von äußeren Einflüssen berücksichtigt werden, die ansonsten zu verfälschten Ortsangaben führen könnten.

In dieser Arbeit wird ein Algorithmus als neuronales Netz (MLP) vorgestellt, der diese Schwankungen auszugleichen versucht. Die Leistungsfähigkeit dieser Lösung wird im Kontext einer kompakten, mobilen Client-Server-Anwendung unter Betrachtung von bereits am Markt vorhandenen Lösungen analysiert und bewertet.

Pascal Oblonczek, Sebastian Zahn

Title of the paper

Indoor-Location with neural networks

Keywords

Location estimation, Mobile location, Neural network, MLP, Backpropagation, Learn based, Location service, Triangulation, Trilateration, WLAN, UWB

Abstract

Today a variety of methods exists to estimate one's indoor location. Each method has very indiviual characteristics that have especially to be taken care of in living places where WLAN based location methods are the first choice in terms of cost effectiveness. Environmental chages affect the WLAN signal strength easily, thus raw measurement data processing may lead to false location information.

In this paper a learn based algorithm is created that tries to level these environmental changes. The effectiveness of the algorithm is being analysed and compared to present solutions in the context of a simple mobile client-server-application.

Inhaltsverzeichnis

1 Einführung

Was noch vor wenigen Jahren in Form von Paketverfolgung in Logistikzentren oder Containerumschlagstrategien in Überseehäfen fast ausschließlich eine Anforderung der Industrie darstellte, hat sich auch längst im Endverbrauchermarkt etabliert; Tragbare Navigationsgeräte leiten über drei Millionen [1] Autofahrer via GPS sicher ans Ziel und während die persönliche Navigation schon zu den Selbstverständlichkeiten des Alltags zählt [2], entwickelt sich um die Information des eigenen, momentanen Standortes ein neues Anwendungsgebiet rasant [3]: „Location Based Services“, also Dienste, deren Inhalte und Leistungen vom Standort des Empfängers abhängen. In einer Vielzahl mobiler Endgeräte finden sich hierfür mittlerweile verschiedenste Technologien zur Lokalisierung und Datenübertragung. Im Outdoor-Bereich sind GPS bzw. das 3G-Mobilfunknetz die stärksten Vertreter. Beide funktionieren innerhalb von geschlossenen Räumen nur unzureichend, sodass sich im Indoor-Bereich andere Techniken etabliert haben.

Das Spektrum ist bei den Indoor-Lokalisierungssystemen viel stärker gefächert als im Outdoor-Bereich. Es gibt verschiedene optische, akustische und elektromagnetische Ortungsverfahren in einer Vielzahl von kommerziellen und auch freien Systemen. Die Anforderungen an die Infrastruktur unterscheiden sich je nach zugrunde liegender Technologie erheblich. Während manche Systeme auf bereits vorhandenen Kommunikationsstrukturen aufbauen, benötigen Andere eine eigenen Infrastruktur aus speziellen Sensoren oder Signalgebern. Dieser Aspekt wirkt sich deutlich auf die Anschaffungskosten eines Systems aus.

1.1 Zielsetzung

Das grundlegende Ziel dieser Arbeit besteht in der Evaluation von zwei verschiedenen Varianten von künstlichen, neuronalen Netzen als Lokalisierungsalgorithmen zur Verwendung in einem Indoor-Lokalisierungsszenario.

Damit die Evaluation an einem realen Szenario durchgeführt werden kann und nicht auf simulierte Umgebungen zurückgegriffen werden muss, soll im Vorfelde der Evaluation ein Konzept für einen einfachen, mobilen Lokalisierungsdienst erstellt werden. Das Konzept soll soweit implementiert werden, dass eine Evaluation sinnvoll

gelingen kann und eine Weiterentwicklung für aufbauende Projekte möglich wird.

1.2 Gliederung

Diese Arbeit gliedert das Thema der Indoor-Lokalisierung in drei Teilbereiche.

Im ersten Bereich geht es darum, die Grundlagen der Indoor-Lokalisierung und ihrer auf dem Markt erhältlichen Lösungen vorzustellen und eine grobe Einschätzung der Leistungsfähigkeit zu geben.

Im zweiten Bereich wird in einer Anforderungsanalyse ein adäquates System benannt, dass dann zur prototypischen Nutzbarkeit entwickelt wird. Dieses selbst entwickelte System wird auf Basis von künstlichen, neuronalen Netzen das Problem der Indoor-Lokalisierung versuchen zu lösen.

Im dritten Bereich wird das vorgestellte System entlang der Anforderungen bewertet und es werden Möglichkeiten für die Weiterentwicklung und Optimierung aufgezeigt.

2 Grundlagen

Die erste Frage, die sich beim Thema Indoor-Lokalisierung stellt ist, wie diese überhaupt funktionieren kann. Ziel dieses Kapitels ist daher die Definition eines formellen Rahmens für diese Arbeit und die Vorstellung einiger, grundlegender Verfahren zur Lokalisierung.

2.1 Definitionen

2.1.1 Raum

Im Rahmen dieser Arbeit soll der Begriff „Indoor“ eingeschränkt werden, da davon zunächst sowohl tausende von Quadratmetern große Lagerhallen wie auch kleine Wohnräume erfasst werden. Im Sinne der vorhandenen Arbeitsumgebung als Lokalisierungs-Szenario (s. Kapitel 4.3) gilt ein Wohnraum von 60m² als Ausgangssituation.

Die Ausdehnung des Raumes wird in der Breite (w) und Höhe (h) in Metern erfasst und auf einen zweidimensionalen Vektorraum A (Area) übertragen, sodass gilt:

$$A = \{ (x, y) \mid x \in \mathbb{R} : 0 \leqslant x \leqslant w, \; y \in \mathbb{R} : 0 \leqslant y \leqslant h \} \quad (1)$$

Für die Anwendung lässt sich somit z.B. ein 60m² großer Raum $A(6,10)$ definieren.

2.1.2 Referenzpunkt

Im Raum oder in der Nähe des Raumes sind mehrere, stationäre Sendeeinheiten (*Referenzpunkte*) verteilt, welche kontinuierlich Funksignale emittieren, die später detektiert und dem jeweiligen Referenzpunkt zugeordnet werden sollen.

Da Referenzpunkte theoretisch auch außerhalb von A liegen können, gilt für den Raum, in dem sich die Referenzpunkte aufhalten können:

$$A' = \{ (x, y) \mid x \in \mathbb{R} : -d_{rmax} \leqslant x \leqslant w + d_{rmax}, \; y \in \mathbb{R} : -d_{rmax} \leqslant y \leqslant h + d_{rmax} \quad (2)$$

Formel 2 beschreibt den vergrößerten Aufenthaltsraum A' für die Menge aller

Referenzpunkte R . d_{rmax} ist dabei die maximal zulässige Distanz eines Referenzpunktes zu A , sodass dieser noch als Referenzpunkt für A gewählt werden darf.

Für jeden Referenzpunkt aus R kann ein Ortsvektor l_{ref} (Location of Reference) angegeben werden, der dessen Position in A' beschreibt:

$$\vec{l_{ref}} = \begin{pmatrix} x \\ y \end{pmatrix} \in A' \tag{3}$$

2.1.3 Endgerät

Das Endgerät ist das Gerät, dessen Position im Raum A ermittelt werden soll. Dafür muss das Gerät in der Lage sein, die Funksignale der Referenzpunkte in R zu empfangen und je nach anzuwendendem Verfahren auszuwerten.

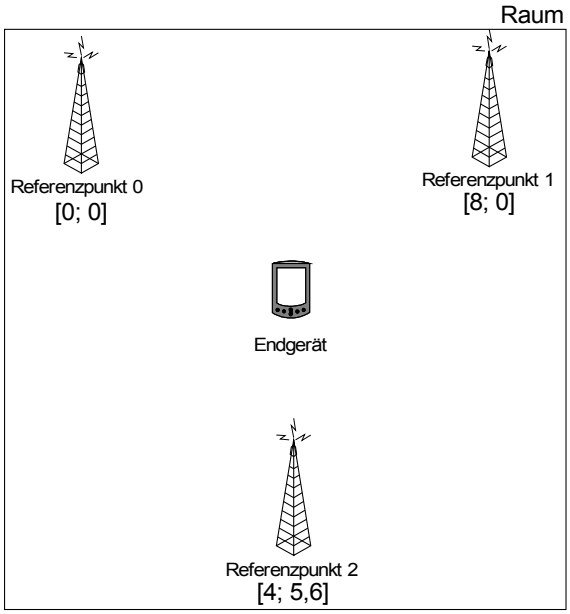


Abbildung 1: Endgerät umgeben von stationären Referenzpunkten

Abbildung 1 zeigt einen Raum mit drei Referenzpunkten mit deren jeweiligen Standorten $l_{ref_{0..2}}$.

2.1.4 Lokalisierung

Grundsätzliches Ziel der Anwendung ist, den Ort des Endgerätes im Raum A unter Zuhilfenahme der Informationen über die Referenzpunkte in R zu ermitteln. Dieser

Vorgang wird als Lokalisierung bezeichnet und im Laufe dieser Argumentation wird L als Lokalisierungsfunktion bezeichnet:

$$L : R \rightarrow A \tag{4}$$

Das Ergebnis der Lokalisierung soll ein zweidimensionaler Ortsvektor $\vec{l}$ sein, der auf den Punkt im Raum A zeigt, an dem das Endgerät vermutet wird:

$$\vec{l} = \begin{pmatrix} x \\ y \end{pmatrix} \in A \tag{5}$$

Um später auch Aussagen über die Genauigkeit des Verfahrens treffen zu können, muss auch zumindest bei der Verifikation der Ergebnisse der tatsächliche Standort des Endgerätes bekannt sein. Er wird als $\vec{l_{real}} \in A$ bezeichnet.

Es gibt zahlreiche Verfahren, um L zu realisieren. Alle in den folgenden Kapiteln vorgestellte Verfahren verfolgen hierfür unterschiedliche Ansätze.

2.1.5 Distanzwertermittlung

Die Distanzwertermittlung ist der erste Schritt, der für viele Verfahren für eine Lokalisierung stattfinden muss. Ziel ist für jeden Referenzpunkt r_i einen Wert $d_i \in \mathbb{R}$ zu finden, der die Entfernungen der Referenzpunkte zum messenden Endgerät untereinander in Relation setzt.

Es wird hierfür vorausgesetzt, dass sich die Charakteristik des Funksignals eines Referenzpunktes mit wachsender Messdistanz am Endgerät stetig ändert, z.B. dass der Signalpegel kontinuierlich und messbar mit wachsender Distanz zum Referenzpunkt sinkt.

Sei S die Menge der an $\vec{l_{real}}$ gemessenen Signaleigenschaften eines Referenzpunktes $r_0 \in R$, so sei S die Menge der Messungen von R und die Distanzwertfunktion:

$$\begin{gathered} d : S \rightarrow \mathbb{R} \\ d_0 = d(s_0), \quad s_0 \in S \end{gathered} \tag{6}$$

Die Maßeinheit der Distanzwerte spielt hierbei vorerst eine untergeordnete Rolle.

Wichtiger für die anschließenden Verfahren ist die Relation zwischen den Distanzwerten, damit eine Aussage über den Aufenthaltsraum des Endgerätes zwischen den Referenzpunkten getroffen werden kann.

Auch ob die Abbildung linear oder z.B. quadratisch ist, muss vorerst offen gelassen werden, da je nach Messmethode beides und auch noch anderes möglich sein kann.

2.1.6 Lokalisierungsalgorithmus

Der Lokalisierungsalgorithmus ist der eigentliche Algorithmus, der den Ort des Endgerätes ermittelt. In vielen Verfahren wird hierbei auf die Werte aus der Distanzwertermittlung zurückgegriffen. Es gibt allerdings auch Verfahren, die Distanzwertermittlung und Lokalisierungsalgorithmus vereinen, somit gilt für den Lokalisierungsalgorithmus vorerst:

$$l : \mathbb{R}^n \rightarrow A,\ n = |R| \tag{7}$$

Die Eingabemenge könnte somit eine Menge von Distanzwerten oder auch eine Menge von rohen Signalwerten sein. In jedem Falle soll die Mächtigkeit der Eingabemenge der Anzahl von Referenzpunkten entsprechen.

2.1.6.1 Abweichung

Ein wesentlicher Aspekt für die Definition der Güte eines Lokalisierungssystems ergibt sich aus der Genauigkeit. Im Rahmen dieser Anwendung soll eine Genauigkeit erreicht werden, die die differenzierte Betrachtung des Ortes in einem kleinen Wohnraum erlaubt.

Die Abweichung e soll in Metern angegeben werden und beschreibt die euklidische Distanz zwischen $\vec{l}$ und $\vec{l}_{real}$.

2.1.6.2 Fehlerrate

Die Fehlerrate betrachtet die Zuverlässigkeit eines Lokalisierungsalgorithmus auf einer Skala von 0 bis 1. Für die Beurteilung wir die Abweichung jeder Messung herangezogen. Überschreitet die Abweichung e eine vorgegebene Maximalabweichung g , die ebenfalls in Metern angegeben wird, so gilt die Messung als ungültig.

Die Fehlerrate über n Messungen lässt sich dann wie folgt angeben:

$$E = \frac{\sum_{n} \begin{cases} 1 & falls\ e_n > g \\ 0 & sonst \end{cases}}{n} \tag{8}$$

Eine hohe Fehlerrate spricht somit für eine unzuverlässige Lokalisierung, bei der häufig mit falschen Ergebnissen gerechnet werden muss. Eine Fehlerrate bei 0 spricht für eine zuverlässige Lokalisierung.

2.1.7 Selbstortung

Für die folgenden Ausführungen wird stets davon ausgegangen, dass das Endgerät die Messdaten erhebt und seine Position selbst bestimmt. Dieser Ansatz wird allgemein auch als Selbstortung bezeichnet, welche im Gegensatz zur Fremdortung steht, bei der das Endgerät von der Infrastruktur lokalisiert wird [4].

2.2 Algorithmen zur Distanzwertermittlung

Die Funksignale der Referenzpunkte können auf verschiedene Weisen verarbeitet werden. Gängige Verfahren werten Laufzeiten, Signalpegel oder Empfangsrichtungen aus, um Rückschlüsse auf die Distanz vom Endgerät zum jeweiligen Referenzpunkt zu ermitteln.

Schwierigkeiten können sich aus den ständig schwankenden Signalpegeln der einzelnen Referenzpunkte ergeben. Gerade im Indoor-Bereich ist mit starken Störeinflüssen zu rechnen, die sich durch die zahlreichen, eng stehenden Wände ergeben. Diese bieten ausgedehnte, reflektierende Flächen, von denen die Funksignale in den Raum zurückgeworfen werden können. Die reflektierten Signale weisen dann in der Regel einen niedrigeren Pegel auf, als die direkt empfangenen Signale, sodass an einem Ort durchaus für einen Referenzpunkt verschiedene Signalstärken ermittelt werden können.

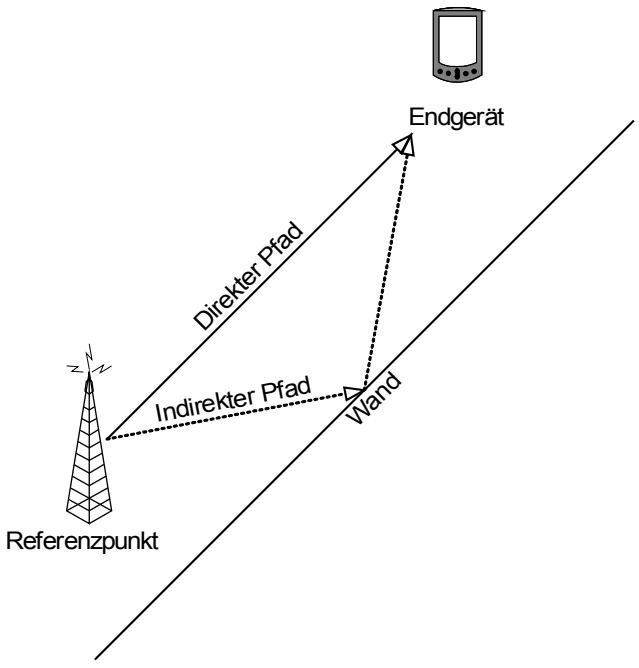


Abbildung 2: Multipath Fading

Dieser Effekt ist auch als Multipath Fading bekannt und stellt eines der Hauptprobleme im Hochfrequenzfunk für die gewünschte Anwendung dar, da nicht nur der Pegel sondern auch die Laufzeit des Signals verfälscht wird, sodass nahezu alle Verfahren betroffen sind [5][6].

Weiterhin sind Faktoren wie die Luftfeuchtigkeit und der Aufenthalt von Personen im Raum als Störfaktoren zu berücksichtigen, welche die Funksignale abschwächen und somit ebenfalls den empfangenen Signalpegel am Messgerät herabsetzen. Dieser Effekt wird als Shadowing bezeichnet [5][6].

Um eine möglichst präzise Ortung zu erreichen, sollte der reinen Messwertermittlung also eine Ausgleichsrechnung nachgeschaltet sein, welche die störenden Einflüsse erkennt und den gemessenen Wert korrigiert, bevor dieser in den Algorithmus einfließen kann .

2.2.1 ToA (Time of Arrival)

Der Ansatz des ToA-Algorithmus besteht darin, Laufzeitunterschiede bei der Funkübertragung zueinander in Relation zu setzen [7]. Um diese Laufzeitunterschiede messbar erfassen zu können, muss unabhängig vom Algorithmus sichergestellt sein, dass alle Teilnehmer des Aufbaus über eine einheitliche Uhrzeit verfügen. Der Ablauf sieht vor, dass alle Referenzpunkte zu einem festgelegten Zeitpunkt ein Impulssignal aussenden. Dieses wird abhängig vom Abstand am Endgerät zu verschiedenen Zeitpunkten wahrgenommen.

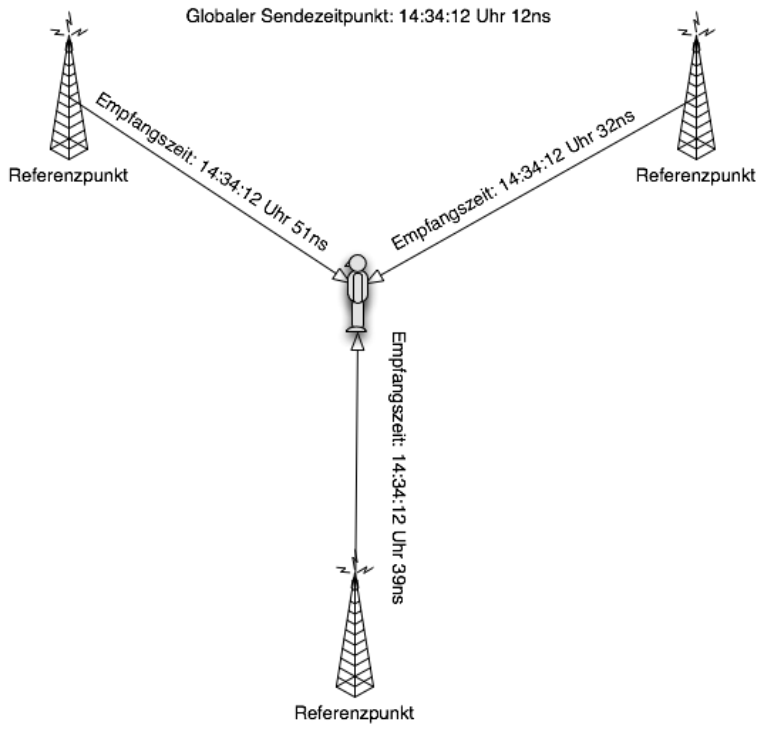


Abbildung 3: ToA

Anhand der Übertragungsgeschwindigkeit und der zeitlichen Differenzen zwischen dem Sende- und Empfangszeitpunkt zwischen dem Endgerät und den Referenzpunkten lassen sich nun auswertbare Distanzwerte ermitteln. In Abbildung 7 wird der Kommunikationsfluss verdeutlicht.

2.2.1.1 Bewertung

Die zu erwartende Präzision des Algorithmus unter Laborbedingungen ist ziemlich hoch. Allerdings leidet der Algorithmus unter Verwendung von Hochfrequenzfunk unter dem Multipath Fading sowie dem Shadowing erheblich, da die Signallaufzeiten in der Luft extrem kurz sind. Die Auswirkungen bei einem verhältnismäßig geringen Messfehler im Bereich von Nanosekunden haben extreme Abweichungen zur Folge. Ganz konkret hat ein Messfehler von einer Nanosekunde bereits eine räumliche Abweichung von 30 cm zur Folge.

Zudem stellt es sich als Problem dar, die Uhren aller Teilnehmer so genau zu synchronisieren, dass das versandte Impulssignal an allen Stationen gleichzeitig ausgesandt wird. Die hier benötigten Anforderungen an die Zeitsynchronisation übersteigen die eines normalen Betriebssystems oder die des allseits bekannten NTP-Protokolls zur Uhrensynchronisation erheblich.

2.2.2 TDoA (Time Difference of Arrival)

Der TDoA-Algorithmus basiert wie der ToA-Algorithmus auf dem Ansatz, Laufzeitunterschiede zwischen mehreren Referenzpunkten zueinander in Relation zu stellen [8]. Das Zeitsynchronisationsproblem vom ToA-Algorithmus wird hier jedoch durch die Verwendung einer zusätzlichen Übertragungstechnologie mit abweichender Übertragungsgeschwindigkeit umgangen. So gibt es beim TDoA-Algorithmus keinen globalen Sendezeitpunkt mehr.

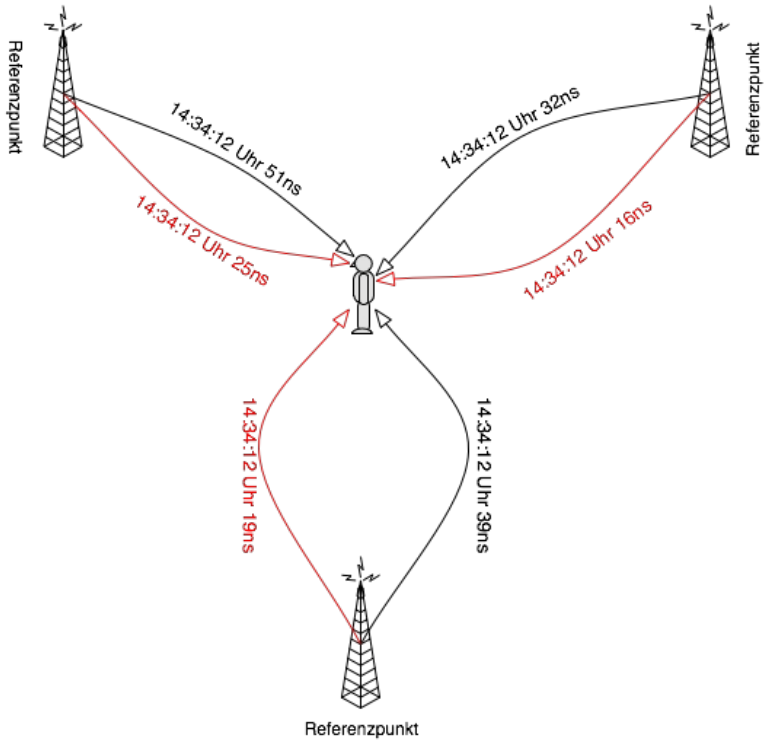


Abbildung 4: TDoA

Im Endgerät werden lediglich die Empfangszeitpunkte der verschiedenen Übertragungstechnologien pro Referenzpunkt miteinander in Relation gesetzt.

Da dem Endgerät zu jeder Technologie die Übertragungsgeschwindigkeit bekannt sein muss, lassen sich hierdurch Rückschlüsse auf die jeweilige Übertragungszeit machen. Abbildung 4 verdeutlicht den Kommunikationsablauf, die Zeitdifferenz zwischen dem schnelleren (rot) und langsameren Übertragungsweg (schwarz) gibt Auskunft über die Signallaufzeit.

2.2.2.1 Bewertung

Der TDoA-Algorithmus gilt als sehr zuverlässig und genau, da er in der Lage ist, mit Hilfe der zwei voneinander unabhängigen Übertragungstechnologien viele Störeinflüsse zu erkennen und zu berücksichtigen [9]. Die Hardwareanforderungen an den Algorithmus sind relativ hoch, da das Timing während der Messwerterfassung eine entscheidende Rolle bei der Genauigkeit im Ergebnis der Ortung hat. Diese Eigenschaft ist ein wesentlicher Nachteil des Algorithmus. Handelsübliche Endgeräte erfüllen leider selten mehrere Übertragungsstandards, welche sich für den Einsatz in diesem Aufbau eignen. Häufig werden beim TDoA-Algorithmus optische bzw. auf Ultraschall basierende Verfahren zur Messwerterfassung verwendet, um neben der Funkübertragung ein System mit möglichst anderem Verhalten zur Verfügung zu haben. Die Verwendung dieser beiden Übertragungsstandards bringt jedoch die Einschränkung mit sich, nur bei direktem Sichtkontakt zuverlässig zu funktionieren. Ein System, welches mittels TDoA eine Lokalisierung zur Verfügung stellt ist das an der HAW entwickelte IMAPS [9].

2.2.3 AoA (Angle of Arrival)

Anders als die bisher genannten Verfahren basiert das AoA-Verfahren nicht auf Zeitmessungen. Es wird der Eintrittswinkel der Funksignale ermittelt und über diese Winkelinformationen anguliert [8]. Oftmals wird der AoA-Algorithmus mit

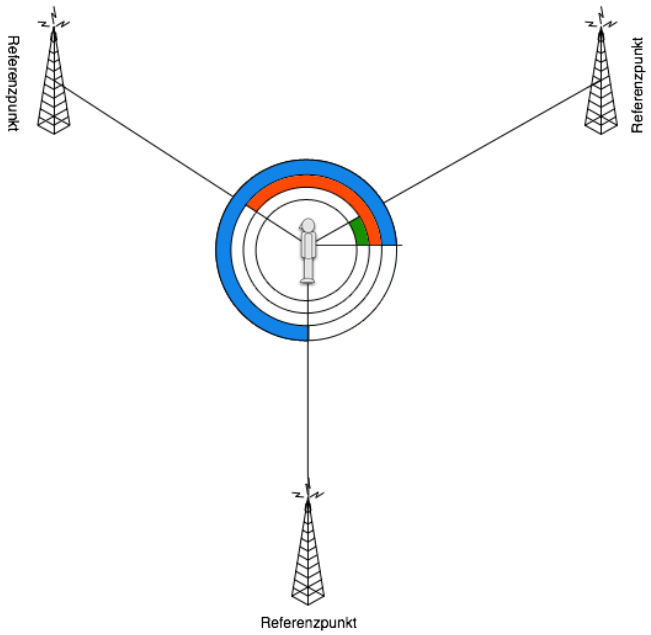


Abbildung 5: AoA

anderen Algorithmen wie TDoA kombiniert. Eine schematische Darstellung des Verfahrens ist in Abbildung 5 zu finden, der Eintrittswinkel der Signale wird am Empfänger registriert (grüner Teilkreis, roter Teilkreis, blauer Teilkreis für die jeweiligen Referenzpunkte).

2.2.3.1 Bewertung

Das AoA-Verfahren stellt eine sinnvolle Erweiterung für technische Umgebungen dar, in denen es möglich ist, auf den Eintrittswinkel der Signale zu schließen. Die Genauigkeit hängt hierbei direkt von der Auflösung der Eintrittswinkel ab. Leider ergeben sich wie bei allen zuvor genannten Algorithmen ebenfalls die Probleme wie Multipath und Shadowing, wodurch fälschlicherweise die Eintrittswinkel von Reflexionen erfasst werden könnten. Die Anforderung an die Hardware, Eintrittswinkel von Signalen zu erfassen, kann in der Regel nicht mit handelsüblicher Consumer-Hardware erfüllt werden.

2.2.4 RSSI-Based (Received Signal Strength Indication – Based)

Das RSSI-Based-Ortungsverfahren ist ein weit verbreitetes Verfahren zur Berechnung des momentanen Aufenthaltsraumes eines mobilen Gerätes. Es ist eine reine Softwarelösung und basiert auf einer geschickten Auswertung von ermittelten RSSI-Werten. Ein RSSI-Wert ist eine in fast allen handelsüblichen Geräten mit drahtloser Datenübertragung verfügbare Größe, welche eine Aussage über die empfangene Signalstärke im jeweiligen Endpunkt gibt [8]. Da die Signalstärke in einem direkten Zusammenhang mit der Distanz steht, ist es hierdurch möglich, Wegstrecken zwischen zwei Punkten, dem Sender und dem Empfänger, zu errechnen. Details zur Berechnung des RSSI-Wertes sind im Kapitel 2.2.4.2 nachzulesen. Genau wie die vorhergehenden Verfahren benötigt auch das RSSI-Based-Verfahren mehrere Referenzpunkte, um einen als wahrscheinlich geltenden Aufenthaltsraum oder, im Optimalfall, einen Schnittpunkt aller ermittelten Distanzen zu bestimmen. Die Berechnung dieses Aufenthaltsraumes bzw. des Schnittpunktes wird durch das Laterationsverfahren (siehe Kapitel 2.3.2) bewerkstelligt. Die Probleme des RSSI-Based-Verfahrens liegen in der Genauigkeit und Aussagekraft der RSSI-Signalpegel.

Rein theoretisch hat die Ableitung von der Signalstärke auf die Distanz eine hohe Aussagekraft. Die Distanz lässt sich nach der „Frii'schen Wellengleichung“ [10] sehr

leicht aus den Sende- und Empfangsstärken herleiten:

$$P_{rx} = P_{tx}\left(\frac{\lambda}{4\Pi d}\right)^2$$, umgestellt nach d:

$$d = \frac{\sqrt{(P_{tx})}\lambda}{4\Pi\sqrt{(P_{rx})}}$$

P_{tx} = Sendeleistung vom Sender

P_{rx} = Empfangene Leistung

λ = Wellenlänge

d = Abstand zwischen Sender und Empfänger

Wie sich die Signalstärke hingegen in der Praxis verhält und wo die Probleme des RSSI-Based-Verfahrens liegen, wird im Kapitel 2.2.4.2 näher erläutert.

2.2.4.1 Funktionsweise

RSSI ist ein Messwert, der eine Aussage über die Energie eines empfangenen Funksignals macht. Es handelt sich hierbei nicht direkt um die Signalstärke. Da die Einheit des RSSI durch kein Konsortium festgelegt wurde, ist es dazu gekommen, dass der RSSI abhängig vom Hersteller interpretiert werden muss [11]. Es handelt sich um einen Integer-Wert zwischen -255 und 255. Der wahre Maximalwert des RSSI ist jedoch nicht standardisiert. So hat Cisco beispielsweise für ihre Produkte die RSSI-Werte auf einen Bereich zwischen 0 und 100 beschränkt [11]. Andere Geräte lösen den RSSI zwischen 0 und 128 auf. Der jeweilige Maximalwert des Systems lässt sich jedoch immer über das sogenannte RSSI_MAX-Register ermitteln. Dieser Missstand führt leider dazu, dass es nicht möglich ist, für alle Geräte anhand des RSSI-Wertes direkt auf die Eingangsleistung (dB oder mW) zu schließen. Die Abbildung zwischen diesen beiden Messgrößen muss hier herstellerabhängig implementiert werden. Rein technisch gesehen ist der RSSI-Wert eine am Empfangsgerät gemessene Spannung, welche in ein Verhältnis zu einer Referenzspannung gesetzt wird und dieser neu ermittelte Wert in ein dafür reserviertes Register abgelegt wird.

Laut IEEE ist der RSSI folgendermaßen definiert:

> ***14.2.3.2 RXVECTOR RSSI***
> The receive signal strength indicator (RSSI) is an optional parameter that has a value of 0 through RSSI Max. This parameter is a measure by the PHY sublayer of the energy observed at the antenna used to receive the current PPDU. RSSI shall be measured between the beginning of the start frame delimiter (SFD) and the end of the PLCP header error check (HEC). RSSI is intended to be used in a relative manner. Absolute accuracy of the RSSI reading is not specified. [12] (21.05.09)

2.2.4.2 Verhalten in der Praxis

Um herauszufinden, wie sich der RSSI-Wert in der Praxis verhält, wurde im Rahmen dieser Bachelorarbeit ein Datenlogger geschrieben, welcher es ermöglicht, Messwerte über einen längeren Zeitraum zu erfassen und auswertbar darzustellen (s. Kapitel 4.4.2). Die Messwerte aus Abbildung 6 wurden mit mehreren baugleichen „Apple Airport Express“-Accesspoints gemessen. Allen Accesspoints wurden feste Kanäle zugewiesen, welche sich nicht mit den Kanälen der anderen Accesspoints überschnitten. Als Empfänger wurde ein Apple MacBook an einem festen Ort installiert und der Datenlogger für ca. 1,5 Stunden gestartet. Wie Abbildung 6 zu entnehmen ist, lagen die Schwankungen des RSSI-Signalpegels innerhalb dieses Zeitraumes bei bis zu 15 RSSI-Punkten.

Um zu testen, wie sich der RSSI-Wert auf minimale Schwankungen im Raum verhält, wurden innerhalb der Messungen für die Abbildung 6 um ca. 11:50 Uhr die Fenster im Raum geöffnet und es wurde quer-gelüftet. Schon diese für den Menschen kaum wahrnehmbaren Schwankungen sorgten im Messergebnis für eine Beeinträchtigung des RSSI-Signalpegels.

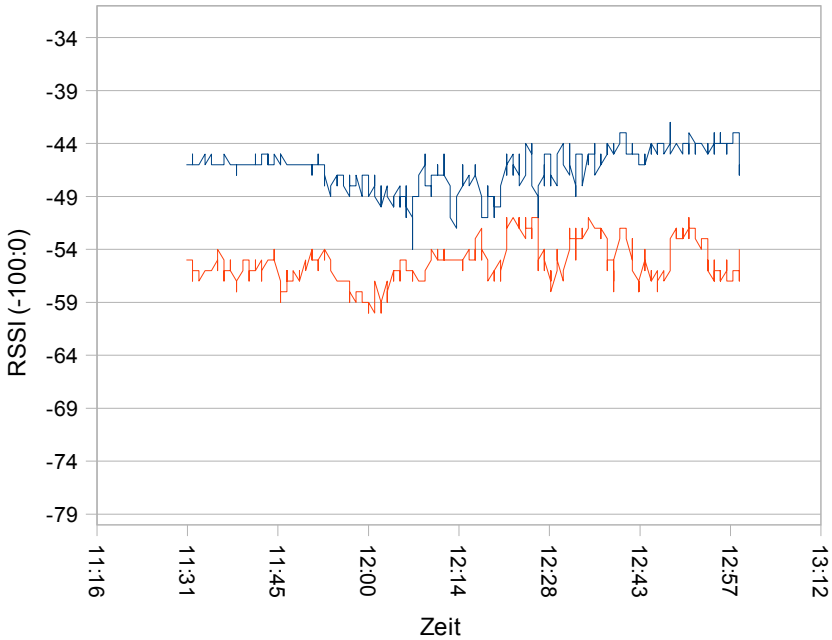


Abbildung 6: Verlauf zweier RSSI-Signalpegel über ca 1,5 Stunden

Es zeigt sich, dass selbst Faktoren wie die Luftfeuchtigkeit oder die Bewegung von Personen im Raum als Störfaktoren zu berücksichtigen sind, da sie die Funksignale abschwächen oder verfälschen und somit auch den empfangenen Signalpegel am Empfänger beeinträchtigen. Diese Erkenntnis über das Verhalten des RSSI deckt sich

mit den Untersuchungen von Frunzke [13] (Kapitel 5.3).

2.2.4.3 Bewertung

Die Vorteile des RSSI-Verfahrens liegen in der Möglichkeit, die Ortung mit Hilfe einer Technik zu realisieren, welche nebenbei als Kommunikationsprotokoll verwendet werden kann. Abgesehen von einer WLAN-Installation werden keine speziellen Hardwareinstallationen benötigt.

Die Intelligenz des Algorithmus liegt im mobilen Endgerät. Somit wird keine externe Infrastruktur benötigt, welche einen erhöhten Wartungsaufwand benötigt oder eine Einschränkung bei der Übertragungsrate darstellen könnte. Aufgrund der Schwankungen des RSSI-Signalpegels scheint es allerdings einer intensiven Vorverarbeitung zu bedürfen, um ungewollte Schwankungen auf ein möglichst geringes Maß zu reduzieren.

2.3 Algorithmen zur Lokalisierung

Im folgenden Kapitel wird sich mit der Frage beschäftigt, wie die aus den vorhergehenden Kapitel gesammelten Messwerte dazu genutzt werden können, eine Position daraus zu ermitteln. Die hier vorgestellten Algorithmen haben oftmals schon zu früheren Zeiten in der Seefahrt Anwendung gefunden und basieren häufig darauf, mit Hilfe mehrerer Positionen, deren Ort bekannt ist, eine Position mit unbekanntem Ort zu ermitteln.

2.3.1 Angulation

Der Algorithmus zur Angulation [14] basiert auf der Winkelmessung zwischen den als bekannt geltenden Positionen und einem unbekannten Punkt. In früherer Zeit benutze man für die Winkelmessung meist einen Theodoliten.

Das Verfahren findet bis heute seinen Einsatzbereich bei den optischen Messverfahren, da diese in der Lage sind, ein sehr präzises Ergebnis bei der Winkelmessung zu liefern. Im Bereich der Funktechnologie stellt sich der Einsatz von Angulation als schwieriger dar, da dort oftmals technisch bedingt keine Informationen über den Eingangswinkel der elektromagnetischen Wellen zur Verfügung stehen.

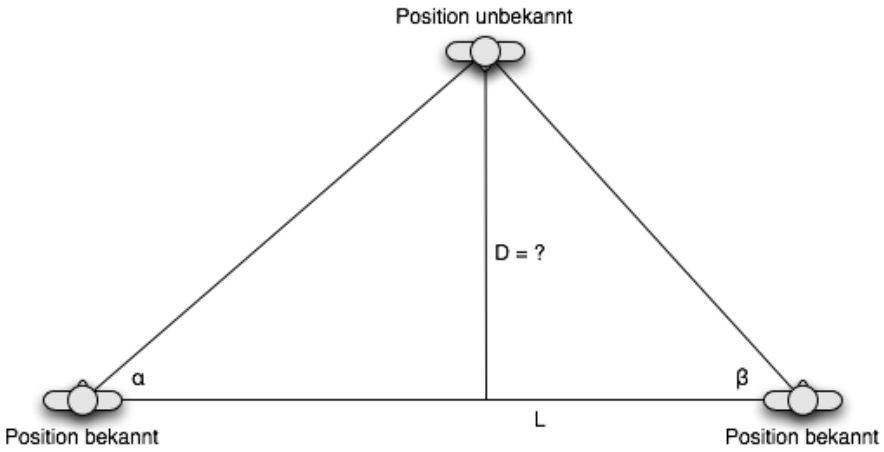


Abbildung 7: Angulation

Die Funktion zur Bestimmung der Distanz ist recht einfach. Man setze in die folgende Formel die bekannten Größen aus Abbildung 7 ein:

$$D = \frac{L}{\left(\frac{1}{\tan(a)} + \frac{1}{\tan(\beta)}\right)} \tag{9}$$

2.3.1.1 Beispiel

Zwei Personen möchten die Position einer dritten Person bestimmen. Beide wissen ihren jeweiligen Standort, sowie den Abstand (L) zueinander.

Folgende Situation aus Abbildung 8 sei gegeben:

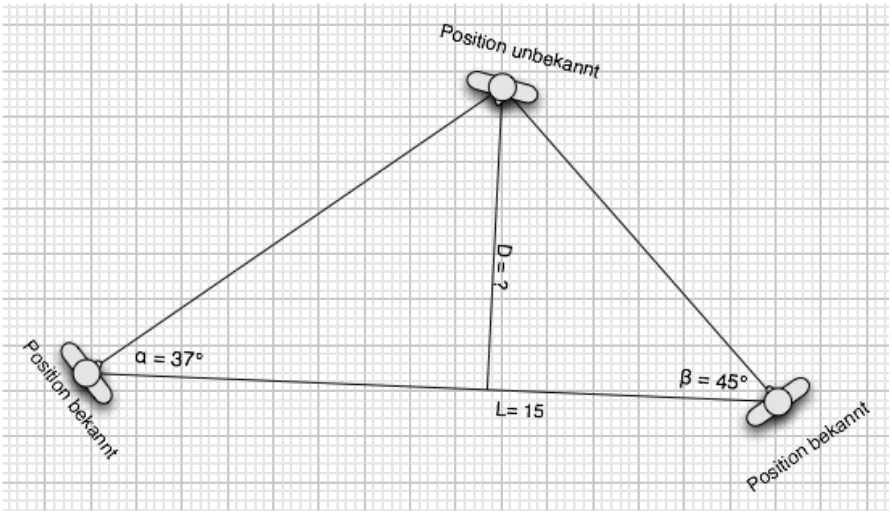


Abbildung 8: Beispiel einer Angulationsrechnung

Gegeben:

a = 37°

ß = 45°

L = 15 Längeneinheiten

Wendet man nun die Formel zur Bestimmung der Distanz (9) an, erhält man:

$$D = \frac{15}{\left(\frac{1}{\tan(37)} + \frac{1}{\tan(45)}\right)} = 6.4\ Längeneinheiten \tag{10}$$

Mit Hilfe der Trigonometrie ließen sich auch so alle weiteren Distanzen in diesem Dreieck bestimmen.

2.3.1.2 Bewertung

Das Verfahren der Angulation ist ein sehr effizienter Lokalisierungsalgorithmus, welcher mit geringer Rechendauer zu guten Ergebnissen führen kann. Die technischen Voraussetzungen der Winkelbestimmung führen jedoch im Bereich der Funkortung bei sehr vielen System zum Ausscheiden der Angulationsverfahren. Der Grund dafür liegt in den Effekten wie Shadowing und Fading, welche es als sehr schwer gestalten, den direkten Winkel eines eingehenden Signals zu erfassen. Die Gefahr der Messabweichungen durch diese Effekte gestaltet sich im Bereich des Hochfrequenzfunks als schwierig und oftmals zu aufwendig kompensierbar, weshalb in diesem Umfeld häufig auf den Lokalisierungsalgorithmus der Lateration zurückgegriffen wird.

2.3.2 Lateration

Die Lateration ist ein weiteres Verfahren um die Position eines Punktes zu bestimmen. Sie basiert anders als die Angulation jedoch nicht auf Winkelmessung, sondern auf direkter Distanzmessung zu Referenzpunkten, deren Positionen bekannt sind [14]. Um eine zuverlässige Aussage über die momentane Position geben zu können, benötigt der Algorithmus das Wissen über mindestens drei bekannte Referenzpunkte mit ihren Positionen. Aus diesem Grunde wird das Verfahren auch oftmals Trilateration genannt.

Je mehr Referenzpunkte dem Verfahren allerdings zur Verfügung stehen, desto höher wird auch seine Genauigkeit bei der Lokalisierung. Zu jedem der Referenzpunkte muss

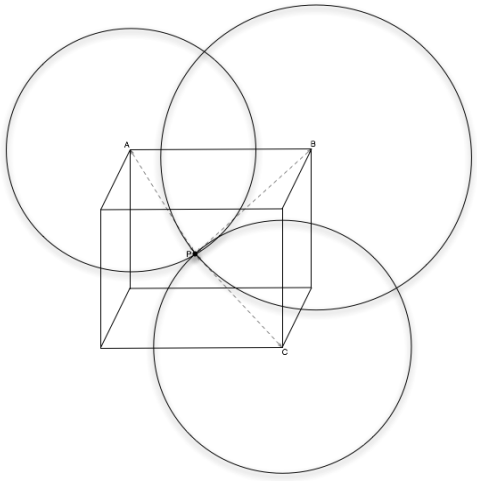


Abbildung 9: Schnittpunkt dreier Aufenthaltsräume

dem Verfahren die Information über die momentane Distanz vorliegen [15].

Jede dieser Distanzen stellt für das Verfahren einen Aufenthaltsraum in Form eines Kreises dar. Im Optimalfall kommt es zu einem gemeinsamen Schnittpunkt aller Kreise, welcher den momentanen Aufenthaltsort skizziert. (Siehe dazu Abbildung 9)

2.3.2.1 Rechenweg

Die Berechnung des Aufenthaltsortes lässt sich mit Hilfe der folgenden Gleichungen und dem Wissen über die Ausbreitungsgeschwindigkeit v_s der eingesetzten Übertragungstechnologie berechnen.

$$d = \sqrt{(x - x_0)^2 + (y - y_0)^2} \tag{11}$$

$$d = (v_s t_0)^2 \tag{12}$$

v_s = Ausbreitungsgeschwindigkeit

Mithilfe der Gleichungen (1) und (2) lässt sich nun mit Hilfe des Wissens über die Ausbreitungsgeschwindigkeit die Gleichung (3) lösen:

$$A\vec{x} = \vec{b} \tag{13}$$

, wobei die Matrix A sowie die beiden Vektoren $\vec{x}$ und $\vec{b}$ sich wie folgt berechnen.

$$A = \begin{pmatrix} 2(x_1 - x_0) & 2(y_1 - y_0) \\ 2(x_2 - x_0) & 2(y_2 - y_0) \\ 2(x_{(n-1)} - x_0) & 2(y_{(n-1)} - y_0) \end{pmatrix}, x = \begin{pmatrix} x \\ y \end{pmatrix} \tag{14}$$

$$b = \begin{pmatrix} x_1^2 - x_0^2 + y_1^2 - y_0^2 - v_s^2(t_1^2 - t_0^2) \\ x_2^2 - x_0^2 + y_2^2 - y_0^2 - v_s^2(t_2^2 - t_0^2) \\ \cdots \\ x_n^2 - x_0^2 + y_n^2 - y_0^2 - v_s^2(t_n^2 - t_0^2) \end{pmatrix}, n \geq 3 \tag{15}$$

Ist $n \geq 3$ lassen sich mithilfe der überbestimmten Matrix A die Werte x und y berechnen. Setzt man diese nun in die Gleichung (9) ein, erhält man die gewünschte Distanz d .

2.3.2.2 Bewertung

Leider sind die einzelnen Entfernungsmessungen zu den jeweiligen Referenzpunkten selten so genau, dass bei der Lateration ein eindeutiger Schnittpunkt ermittelt werden kann.

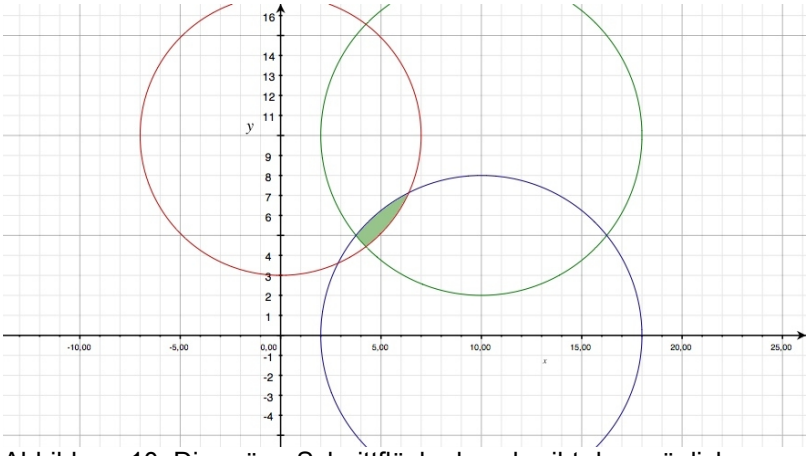


Abbildung 10: Die grüne Schnittfläche beschreibt den möglichen Aufenthaltsraum

Der Regelfall ist viel mehr, dass sich aus den Kreisen eine Schnittfläche bzw. ein Wahrscheinlichkeitsraum ergibt (s. Abbildung 10).

Für diesen Raum gibt es verschiedene Herangehensweisen, um aus ihm einen eindeutigen Schnittpunkt zu bestimmen. Eine Herangehensweise wäre z.B., das Zentralmoment dieser Schnittfläche zu bestimmen. Es wird deutlich, dass die Erhöhung der Anzahl an Referenzpunkten eine direkte Verbesserung im Messergebnis zur Folge hat. Eine größere Anzahl von Referenzpunkten schränkt den als wahrscheinlich geltenden Aufenthaltsraum direkt ein und ermöglicht so, eine präzisere Aussage über die Lage des gemeinsamen Schnittpunktes zu treffen. Die Vorbedingung hierfür ist allerdings in jedem Falle, dass die Radien einen recht niedrigen Fehler aufweisen, sodass auch alle Kreise eine gemeinsame Schnittfläche haben.

2.3.3 Fingerprinting

Anders als die beiden vorherigen Verfahren benötigt ein Fingerprinting-Algorithmus kein Wissen über die Position von Referenzpunkten [14]. In der „Anlernphase“ des Algorithmus werden an möglichst vielen Orten Messungen vorgenommen und die dort ermittelten Signalstärken abgespeichert. Möchte man nun in der „Produktivphase“ des Algorithmus seine Positionen ermitteln, werden die an diesem Ort ermittelten Signalstärken mit den zuvor gemessenen Signalstärken verglichen. Der Ort der am meisten Übereinstimmung innerhalb der Signalstärken hat, wird als aktuelle Position herangezogen.

2.3.3.1 Bewertung

Der Algorithmus bietet die Möglichkeit, sehr einfach und präzise ein zuverlässiges Ergebnis bei der Ortung zu erhalten. Der Vorteil gegenüber den anderen Algorithmen liegt in der etwas anderen Ausrichtung in den Anforderungen. Da der Algorithmus nicht auf ein Koordinatensystem sondern auf eine Menge an möglichen Aufenthaltsorten abbildet, ist seine Fehlerquote deutlich geringer. Er bietet zudem aufgrund seiner einfachen Installation und Wartung einen weiteren Vorteil im Sinne des Zeitaufwandes. Steigt jedoch die Anzahl an möglichen Aufenthaltsorten stark an, geht der Vorteil zu den zuvor vorgestellten Algorithmen verloren, da dann sehr viele Messungen durchgeführt werden müssen.

Auch bei diesem Algorithmus bedarf es einer effektiven Filterung und Vorverarbeitung der Messdaten um Messfehler auszugleichen, sodass auch ein angelerntes Muster wiedererkannt werden kann.

2.3.4 Künstliche, neuronale Netze

Wie in 2.2 beschrieben, ergibt sich ein großes Problem für alle bisher vorgestellten Verfahren aus den ständigen Schwankungen der empfangenen Signalpegel durch Störeinflüsse wie Fading und Shadowing.

Ohne Korrekturrechnung muss mit erheblichen Fehlern bei der Lokalisierung gerechnet werden, da sich die fehlerhaften Messwerte über den Lokalisierungsalgorithmus fortpflanzen [16].

Das Verfahren der Korrektur kann hierbei sehr komplex werden, da die individuellen Eigenschaften des Raumes und der Hardware berücksichtigt werden müssen. Eine Erfassung und Berücksichtigung aller wirksamen Störeinflüsse scheint nur unter Laborbedingungen realisierbar, sodass für die meisten Lokalisierungsverfahren in kleinen Räumen letztendlich mit einem großem Restfehler bei der Ortung gerechnet werden muss.

An dieser Stelle bietet sich der Ansatzpunkt für gänzlich andere Verfahren, nämlich die künstlichen, neuronalen Netze (KNN).

2.3.4.1 Hintergrund

KNN fanden erstmals öffentlich Erwähnung um 1943 durch die Untersuchungen von Warren McCulloch und Walter Pitts zu den Vorgängen im menschlichen Gehirn [17]. Ziel der Forschungen war die Aufstellung eines Rechenmodells, das die Fähigkeiten des Hirns nachvollziehbar machen sollte. Die von McCulloch und Pitts vorgestellte „Threshold-Logic“ ermöglichte erstmals auf Basis von Netzwerken von „McCulloch-Pitts-Einheiten“ das Abbilden aller denkbaren, logischen Funktionen, sowohl zustandsfrei (Feed Forward) als auch zustandsbehaftet (Recurrent).

Ab 1958 entwickelt eine Forschergruppe um Frank Rosenblatt das Perceptron, welches einen allgemeineren Ansatz als McCulloch-Pitts-Einheiten bot und direkt mit reellen Zahlen umgehen konnte. Im nachfolgenden Jahr entwickelte Rosenblatt einen überwachten Lernalgorithmus, der es ermöglichte, das Perceptron anhand von repräsentativen Beispieldaten zu konditionieren. Widdrow und Hoff erweiterten 1960 den Lernalgorithmus für ihren eigenen Ansatz, das „Adaptive Linear Neuron“. Die „Delta Regel“ erlaubte adaptives Lernen, bei dem das Neuron deutlich schneller zu

einem korrekten Verhalten konvergierte als beim klassischen „Perceptron Learning Algorithm“ und zudem die Verwendung der reellen Zahlen als Eingangsgewichte komfortabel ermöglichte [18].

Nach einer Durststrecke von mehreren Jahren wurden in den 1980er Jahren die Fähigkeiten des Ansatzes weiter erforscht und die Technologie erlebte eine Renaissance. Die Erforschung der KNN brachte in den folgenden zehn Jahren zahlreiche Interpretationen und neue Ansätze hervor, die den Einsatz in vielen Bereichen praktisch sinnvoll machten.

Neben vielen anderen Anwendungsbereichen in Forschung und Wirtschaft finden KNN sehr häufig Erwähnung im Bereich der Musterkennung. Viele moderne Verfahren zur Text- und Spracherkennung basieren zumindest zu Teilen auf KNN. Die Fähigkeit der KNN, Muster anhand von Erlerntem wieder zu erkennen, sollte sich gewinnbringend auf das Problem eines Lokalisierungsalgorithmus übertragen lassen.

2.3.4.2 Ansatz

Für die ersten Überlegungen soll an die Lokalisierungsfunktion L aus Kapitel 2.1.6 erinnert werden. Gängige, rein arithmetische Verfahren, wie die Lateration, beruhen auf einer Distanzwertermittlung, die dem eigentlichen Lokalisierungsalgorithmus vorangehen muss. Die Lokalisierungsfunktion kann somit als Hintereinanderausführung von Distanzwertermittlung d und Lokalisierungsalgorithmus l als $L = l \circ d$ verstanden werden. Die Schwierigkeit liegt unter anderem darin, die eingangs erwähnten Messfehler aus d in l so zu berücksichtigen, dass die Lokalisierungsfunktion L trotzdem noch korrekte Ergebnisse liefert.

Nach Rojas [17] kann ein Netz abstrakter, künstlicher Neuronen als „Mapping Machine“ betrachtet werden, welche Abbildungen $F : \mathbb{R}^n \rightarrow \mathbb{R}^m$ realisiert. Diese Abbildung wird im folgenden „Netzfunktion“ genannt. Mit welchen Methoden die Abbildung realisiert wird, ist hierbei vorerst unbekannt, es wird aber davon ausgegangen, dass ein solches System die Abbildung durch geeignete Verfahren *erlernen* kann.

Die grundlegende Idee ist nun, die „Mapping Machine“ auf die Problemstellung der Lokalisierung anzuwenden, also dass genau die Lokalisierungsfunktion L realisiert werden soll. Entscheidend ist hierbei der Aspekt, dass Distanzwertermittlung und

Lokalisierungsalgorithmus somit zusammenfallen. Die Abbildung folgt also direkt $R \to A$. Diese Annahme führt hinsichtlich der Kardinalität zu Einschränkungen für die Netzfunktion. n und m können somit angegeben werden, sodass gilt:

$$F' : \mathbb{R}^{|R|} \to \mathbb{R}^2 \tag{16}$$

Die Mächtigkeit der Eingabemenge entspricht also der Anzahl der Referenzpunkte, während die Ausgabe im zweidimensionalen Raum (nämlich A) liegen soll.

Am Beispiel der Distanzwertermittlung über RSSI könnte diese Abbildung so realisiert werden, dass die RSSI-Werte von drei verschiedenen Accesspoints ($RSSI_i,\ i = 0..2$) auf die zweidimensionalen Koordinaten $(x, y) \in A$ abgebildet werden.

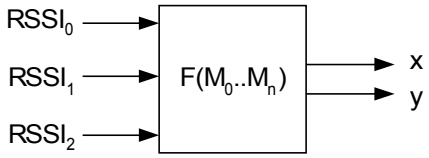


Abbildung 11: KNN als Blackbox

Es gibt eine Vielzahl von Klassen von KNN, welche diese Abbildung auf unterschiedlichste Art und Weisen realisieren. Die größte Gemeinsamkeit bei den Klassen, die Gegenstand dieser Arbeit sind, liegt hierbei in der Praxis des „Offline-Trainings“, bei dem ein KNN vor seinem Einsatz das erwünschte Verhalten erlernen muss. Die genauen Vorgänge des Trainings unterscheiden sich über die unterschiedlichen Netztypen deutlich voneinander. Ziel des Trainings ist, dass das die Netzfunktion so gut wie möglich der Lokalisierungsfunktion entspricht.

Unabhängig von der Wahl des Netztypen gestaltet sich die Implementation aus praktischer Sicht dabei in zwei Phasen:

1. Erfassen von Mustern an verschiedenen Orten im Raum und Training des Netzes, bis das gewünschte Verhalten erreicht ist (Offline-Phase)
2. Produktive Nutzung des Netzes für die Lokalisierung (Online-Phase)

Aus der Sicht des Anwenders ergeben sich hierbei vorerst keine Unterschiede zum Ablauf des Anlernens eines Fingerprinting-Algorithmus, welcher z.B. auf einem Entscheidungsbaum basieren kann. Der entscheidende Vorteil bei der Verwendung

von KNN liegt in deren Fähigkeit zu generalisieren [19]. Das bedeutet, dass ein korrekt eingestelltes KNN auch dann plausible Ergebnisse liefern sollte, wenn Messwerte anliegen, die während der Lernphase nicht als Trainingsmuster zur Verfügung standen. Wie praxistauglich das generalisierte Ergebnis für die Messung ist, muss dabei für jeden Netztyp noch untersucht werden.

Die Erwartung, die aus dieser These entsteht ist die, dass ein Lokalisierungsalgorithmus auf Basis von KNN auch dann in der Lage ist eine gültige Position zu ermitteln, wenn sich die Bedingungen für die Messwerte ändern oder geändert haben.

Für nähere Betrachtungen müssen die individuellen Eigenschaften der verschiedenen Klassen von KNN betrachtet werden. Im Rahmen dieser Arbeit wurden zwei verschiedene Ansätze untersucht, das Multi-Layer-Perceptron und Radiale Basisfunktionen, welche beide als universelle Funktionsapproximatoren verstanden werden können. Die Struktur und das Training unterschied sich jedoch in einigen Punkten erheblich voneinander.

2.3.4.3 Multi-Layer-Perceptron

Das Multi-Layer-Perceptron (MLP) geht auf die Arbeit von Frank Rosenblatt zurück, welcher 1958 das einschichtige Perceptron beschrieb und dessen Arbeit in den 60er-Jahren durch Papert und Minsky verfeinert und mathematisch kritisch eingeordnet wurde [18]. Es handelt sich hierbei um eine lineare Anordnung von elementaren Recheneinheiten (Neurone oder Units), die eine Eingabeschicht (Retina) und eine Ausgabeschicht (Reaktionsschicht) über gerichtete, gewichtete Pfade miteinander verbinden.

Wie auch das klassische Perceptron besteht das MLP aus einer Menge von kleinen, simplen Recheneinheiten (Neuronen), die nach einem bestimmten Schema verknüpft werden.

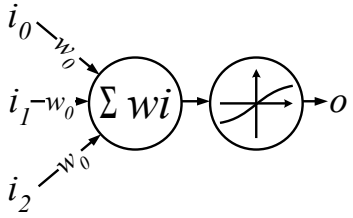


Abbildung 12: Neuron

Abbildung 12 zeigt ein solches Neuron. Es besteht aus drei maßgeblichen Komponenten:

1. Gewichteter Eingangsvektor (Input $\vec{I} = \{i_{0..2}\} \subset \mathbb{R}$ mit Verbindungsgewichten (Weight) $\vec{W} = \{w_{0..2}\} \subset \mathbb{R}$)
2. Propagierungsfunktion $\sum w \cdot i$, in welcher alle gewichteten Eingangswerte aufsummiert werden
3. Aktivierungsfunktion, in der die Ausgabe o aus dem Ergebnis der Propagierungsfunktion errechnet wird. Für die folgenden Betrachtungen wird hier die sigmoide Fermifunktion $f_{\log} = \frac{1}{1+e^{-x}}$ herangezogen.

Das Schema der Verknüpfung, das im Rahmen dieser Arbeit eine tragende Rolle spielen wird, ist die sogenannten „Feed-Forward-Verknüpfung ohne Shortcuts“. Diese Art der Verknüpfung besteht aus mindestens drei vertikal angeordneten Schichten von Neuronen, der Eingabeschicht, einer verdeckten Schicht und der Ausgabeschicht.

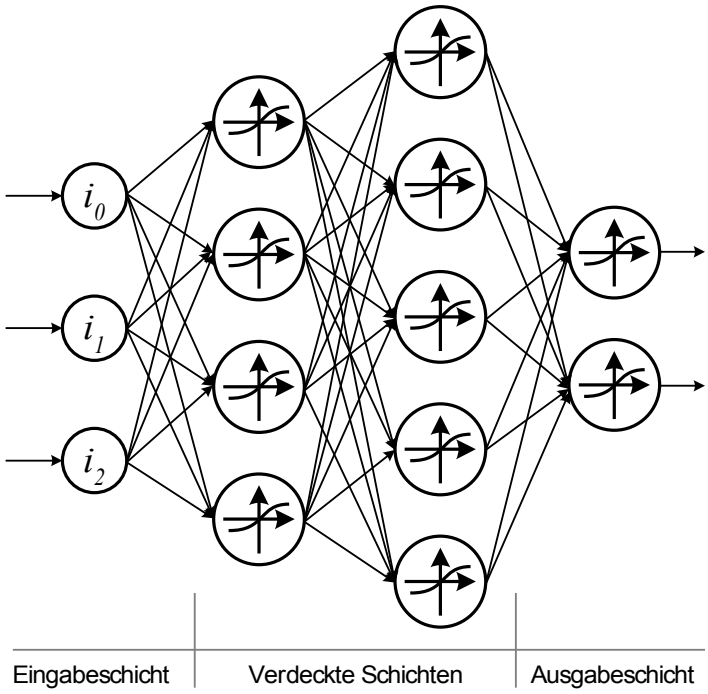


Abbildung 13: MLP mit zwei verdeckten Schichten

Abbildung 13 zeigt ein MLP mit zwei verdeckten Schichten. Die Eingabeschicht besteht

nicht aus Neuronen, wie sie hier definiert wurden, sondern reichen einfach den anliegenden Eingabewert durch an die erste verdeckte Schicht. Sie werden in der Literatur aber auch in der Regel als Neurone bezeichnet [18].

Die Schichten sind untereinander vollverknüpft, es geht also von jedem Ausgang der Neurone eine Verbindung an den Eingangsvektor eines jeden Neurons der nachfolgenden Schicht. Hieraus ergibt sich auch, dass der Eingangsvektor jedes Neurons so viele Elemente hat, wie die vorangegangene Schicht Neurone. Der Index des Eingangswertes auf $\vec{I}$ eines Neurons entspricht dabei dem Positionsindex des verknüpften Neurons in der Vorgängerschicht.

Es ist bewiesen, dass MLP mit zwei verdeckten Schichten einer bestimmen Größe (diese muss aber selbst gewählt werden) beliebige Probleme approximieren können (Kolmogrov-Theorem, [17]), wobei die Möglichkeit besteht, dass noch mehr verdeckte Schichten bessere Ergebnisse liefern können. Es wird daher im Folgenden grundsätzlich von einem MLP mit mindestens zwei verdeckten Schichten ausgegangen, sodass sichergestellt ist, dass, wenn es eine Lösung gibt, diese auch rein theoretisch gefunden werden kann, wenn die Anzahl der Neurone gut gewählt wurde.

Die optimale Anzahl an Schichten und der Neurone pro Schicht kann überhaupt nur unter der genauesten Kenntnis der Problemgröße angegeben werden und auch hier gehen die Meinungen schon weitläufig auseinander [20]. Da die Problemgröße der Anwendung im Falle dieser Arbeit nicht genau bekannt ist, bleibt die Dimensionierung des MLP grundsätzlich eine Frage, die durch Experimente beim Design des Netzes beantwortet werden muss.

Damit das Netz das gewünschte Verhalten für die Anweisung aufweist, muss es trainiert werden. Im Allgemeinen kann im Rahmen eines Trainings jeder Parameter des Netzes modifiziert werden, hierzu gehören:

- Aufbau oder Abbau neuer Verknüpfungen
- Veränderung der Verbindungsgewichte
- Veränderung der Aktivierungsfunktion
- Aufbau oder Abbau neuer Neurone (und der respektiven Verknüpfungen)

Welche dieser Aktionen möglich ist, hängt vom eingesetzten Lernalgorithmus ab. Der in dieser Arbeit vorgestellte Algorithmus „Backpropagation Momentum“ wirkt lediglich auf die Verbindungsgewichte, weswegen die anderen Aktionen nicht weiter behandelt werden.

Lernverfahren

Das MLP wird nach seiner Instantiierung grundsätzlich mit zufällig eingestellten Gewichten versehen. Es muss daher davon ausgegangen werden, dass ein MLP ohne Training generell unerwartete Ergebnisse liefert.

Eines der ersten Perceptron-Lernverfahren, das in der Industrie große Beachtung fand war „Backpropagation“, welches 1974 von Werbos vorgeschlagen wurde [17] und mit einer kleinen Modifikation (Momentum-Term [19]) als Lernverfahren für alle folgenden Untersuchungen am MLP gelten soll. Das Konzept soll an einem kleinen Beispiel verdeutlicht werden. Dafür wird vorerst nicht von einem MLP sondern einzig von einer Ein- und Ausgabeschicht (Single-Layer-Perceptron) mit einem einzelnen Neuron ausgegangen, das das binäre OR-Problem lösen soll. Das Verfahren lässt sich eins zu eins auf das MLP übertragen.

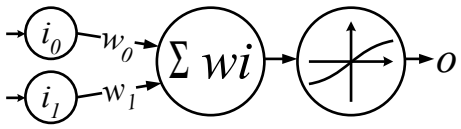


Abbildung 14: SLP der Größe 1

Als Menge T der Trainingsmuster stehen alle vier binären Eingabemöglichkeiten des OR-2 zur Verfügung. Ein Trainingsmuster ist dabei ein Tupel $t_j = (\vec{t}_j, s_j) \in T$, welches aus einem Eingabevektor $\vec{t}_i$ und dem erwarteten Ergebnis s_j besteht.

Das Verfahren legt in mehreren Iterationen und zufälliger Reihenfolge die verfügbaren Trainingsmuster an die Eingänge des Netzes an und ermittelt im Anschluss die Abweichung der Netzausgabe o vom erwarteten Ergebnis js_i des jeweiligen Musters.

Aus der Abweichung $e_j = (s_j - o)^2$ wird ein Anpassungswert für die Verbindungsgewichte $w_{j_{0..1}}$ errechnet, der den Gesamtfehler des Netzes in der nächsten Iteration verringern soll.

Abbildung 15 zeigt hierbei anschaulich, wie sich die Einstellung der Gewichte $w_{j_{0..1}}$ (auf der x- und y-Achse aufgetragen) auf den Gesamtfehler E auswirkt. Zu Beginn des Lernens sind die Gewichte durch Zufall so eingestellt, dass das Perceptron Ergebnisse an einer sehr hoch gelegenen Stelle im Fehlergebirge liefert (ca. -2,-6), die Ausgabe also sehr weit entfernt vom gewünschten Verhalten liegt. Nach dem Verfahren des Gradientenabstiegs wird nun für jedes w_{j_i} ein Δw_{j_i} errechnet und addiert, sodass die neuen Gewichte den Punkt im Fehlergebirge markieren, der mit einer festgelegten Schrittweite η auf dem steilsten Abstieg (negativer Gradient) zu erreichen ist.

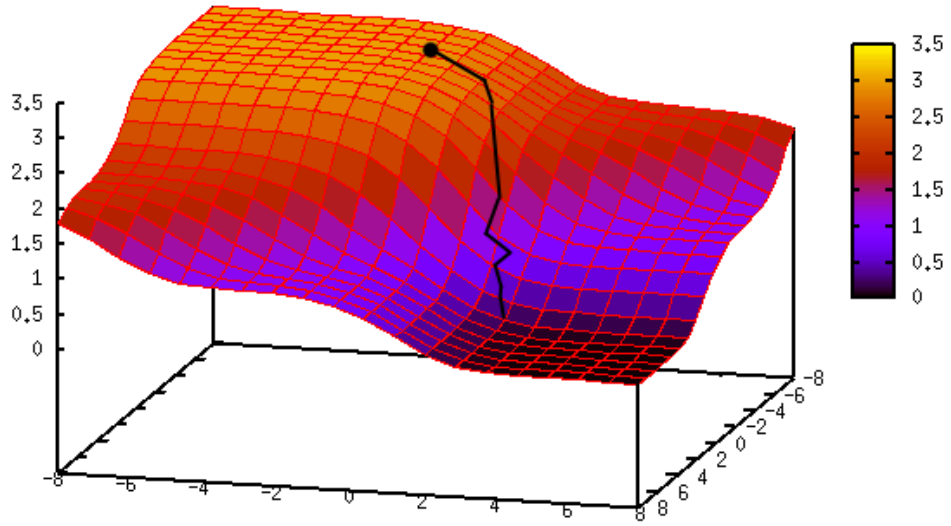


Abbildung 15: Gradiantenabstieg im OR-2-Fehlergebirge eines SLP

Die schwarze Linie markiert den steilsten Abstieg auf dem Fehlergebirge, der Fehler wird also minimiert. Je nach Zerklüftung des Fehlergebirges kann es allerdings auch dazu kommen, dass das Verfahren sich im Abstieg „verläuft“, falls eine falsche Schrittweite gewählt wird. Die Einstellung der korrekten Parameter für Backpropagation muss hierbei nach Erfahrung oder experimentell ermittelt werden.

Abbildung 15 zeigt weiterhin, dass das SLP das OR-2-Problem tatsächlich lösen kann, da es im Fehlergebirge ein Minimum (schwarzer Bereich) bei etwa 0 gibt, also einen Bereich in dem für alle Muster gilt, dass die Netzausgabe dem gewünschten Verhalten entspricht, da der Fehler minimal ist.

Bewertung

Ein MLP kann beliebige Funktionen approximieren. Mit dem MLP aus Abbildung 13 wurde mit Hilfe von Backpropagation z.B. eine gut passende Funktion (rote Kurve in Abbildung 16) für eine willkürliche und kleine Folge von Messwerten (blaue Punkte) gefunden .

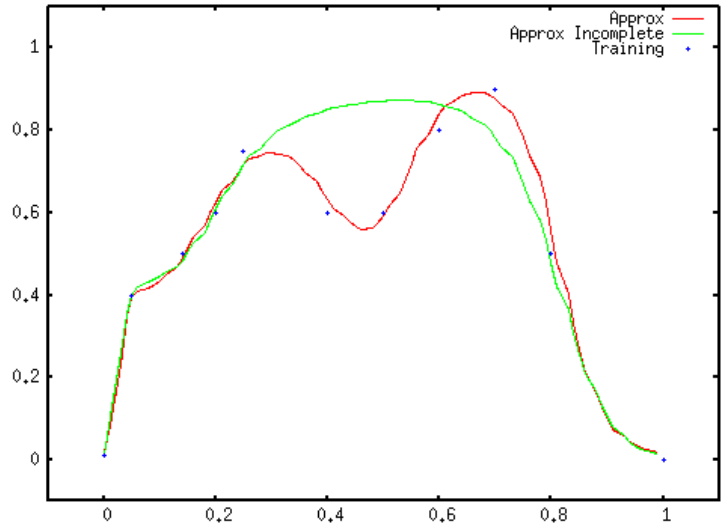


Abbildung 16: Durch MLP approximierte Funktion

Es scheint also auch aus dieser praktisch erlangten Erkenntnis heraus möglich, MLP für das Lokalisierungsproblem einzusetzen. Unwägbarkeiten ergeben sich aus der Frage der Dimensionierung des MLP und der korrekten Einstellung des Lernalgorithmus. Ferner ist die Qualität der Approximation auch von der Vollständigkeit der Trainingsmuster abhängig, wie die grüne Kurve in Abbildung 16 zeigt, bei der die beiden Messwerte auf $x = 0.4, 0.5$ fehlen. Hieraus lässt sich ableiten, dass in Umgebungen mit sehr starken Signalschwankungen auch relativ viele, eng gerasterte Messungen vorgenommen werden müssen, was einen hohen, praktischen Aufwand bedeutet.

Problematisch ist für das Verfahren, wenn neue Messungen in der Approximation nachträglich berücksichtigt werden sollen. Wird davon ausgegangen, dass für einen neuen Trainingsdatensatz mit einem gegebenen Eingabevektor bereits früher ein Trainingsdatensatz existierte, so kann nachträgliches Training mit den neuen Daten dazu führen, dass früher Gelerntes wieder verworfen wird (Stabilitäts-Plastizitäts-Dilemma [18])

Vorteile ergeben sich aus der vergleichsweise einfachen Implementierung und der

Tatsache, dass auch komplexe Muster von Signalpegeln berücksichtigt werden können, sofern diese angelernt werden.

Während der Trainingsaufwand sehr hoch ist und der Überwachung durch den Menschen bedarf, kann der Rechenaufwand für eine Approximation in der produktiven Phase als sehr gering betrachtet werden. Es werden pro Neuron lediglich $|I|$ Additionen benötigt, plus den Aufwand der Fermifunktion. Der Rechenaufwand steigt linear mit der Anzahl der Verbindungen.

2.3.4.4 Radiale Basisfunktionen

Eine weitere Variante von KNN als Funktionsapproximatoren stellt das Paradigma der Radialen Basisfunktionen (RBF) nach Poggio und Girosi dar, welches wesentlich später als das Perceptron entstand [18].

Zunächst gibt es zwischen den RBF und dem MLP eine Menge von Gemeinsamkeiten: Auch die RBF sind als mehrschichtiges Feed-Forward-Netz mit vollverknüpften Schichten organisiert. Ebenfalls reicht die Eingabeschicht die anliegenden Eingabewerte einfach durch.

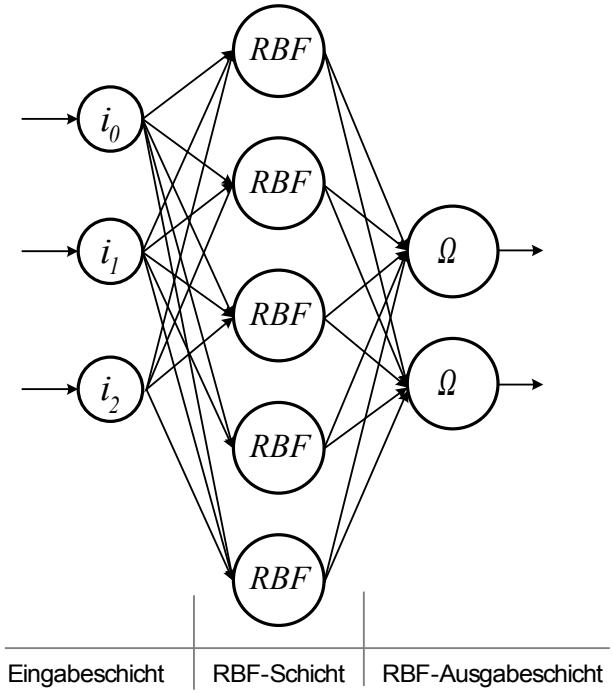


Abbildung 17: RBF-Netz

Im Gegensatz zum MLP ist die Anzahl der Schichten festgelegt. Neben der obligatorischen Ein- und Ausgabeschicht, deren Neuronenzahl sich nach der Größe

des Problems richtet, gibt es nur eine verdeckte Schicht, deren Neurone auch als RBF-Neurone bezeichnet werden und deren Menge im Folgenden als H deklariert sei. Die Verknüpfungen zwischen Eingabe- und RBF-Neuronen sind ungewichtet, die Eingaben werden also direkt in die RBF-Neurone geschickt. Die Verknüpfungen zwischen RBF- und Ausgabeneuronen sind wie beim MLP gewichtet.

Jedes RBF-Neuron repräsentiert hierbei in einem $|I|$ -dimensionalen Raum eine radialsymmetrische Funktion, z.B. die Gaussglocke.

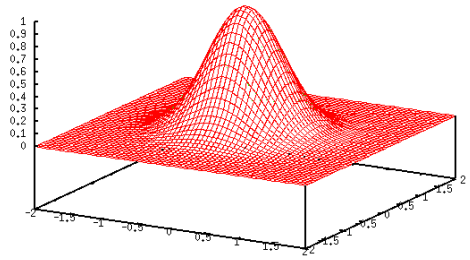


Abbildung 18: Gaussglocke

Die Aktivierung eines Neurons berechnet sich aus der euklidischen Distanz von Eingabevektor und Zentrum der Funktion, welche als Parameter für die Gaussfunktion eingesetzt wird [18]:

$$f_{h_{c,\sigma}}(\vec{x}) = e^{\frac{\|\vec{x},\vec{c}\|}{2\cdot\vec{\sigma}^2}} \tag{17}$$

Je weiter eine Eingabe also vom Zentrum eines RBF-Neurons entfernt ist, desto weniger ist dieses erregt.

Die Parameter c und und σ geben das Zentrum bzw. die Breite der Funktion an.

Wie beim MLP kann jedes Neuron auch als eine Hintereinanderausführung von Propagierungs- und Aktivierungsfunktion gesehen werden. Die Propagierungsfunktion entspricht hierbei der Errechnung der euklidischen Distanz des Neurons zur Eingabe, die Aktivierungsfunktion der radialsymmetrischen Funktion, welche auf die Distanz angewendet wird.

In der Ausgabeschicht werden die gewichteten Aktivierungen der RBF-Neurone schließlich aufsummiert. Es gilt somit für ein Ausgabeneuron o bei gegebenem Eingangsvektor $\vec{x}$ [18]:

$$\Omega_o(\vec{x}) = \sum_{h \in H} w_{o,h} \cdot f_h(\vec{x}) \tag{18}$$

Entscheidend ist hier die korrekte Einstellung der Gewichte, um für jedes Ausgabeneuron anhand der Gewichtung einen korrekten Output zu erzeugen. Die Einstellung der Gewichte wird während des Trainings vorgenommen.

Lernverfahren

In der einfachsten Form von RBF wird für jedes Trainingsmuster ein RBF-Neuron in der verdeckten Schicht angelegt.

Wie beim MLP kann davon ausgegangen werden, dass es eine Menge von Trainingsmustern T mit $|T|$ Eingabemustern gibt. Somit können $|T|$ Gleichungen nach Gleichung 18 angegeben werden, nämlich für jedes Trainingsmuster t genau eine. Gesucht sind die $|H|$ Gewichte eines jeden Ausgabeneurons.

Sei S die Menge der Trainingsausgaben, $\bar{G}$ die $|H| \times |O|$ Matrix der gesuchten Gewichte und $\bar{M}$ eine Matrix $|T| \times |H|$, die für alle Trainingseingaben aus T die Ausgaben der RBF-Neurone H nach Gleichung 18 enthält, so lässt sich ein Gleichungssystem aufstellen, welches die Lösungen für alle Trainingsausgaben erhält. Dieses System lässt sich nun nach G umstellen und liefert somit die gesuchten Gewichte [18]:

$$\begin{aligned} S &= M \cdot G \\ \Leftrightarrow \quad M^{-1} \cdot S &= G \end{aligned} \tag{19}$$

Da für jede Trainingseingabe genau ein RBF-Neuron maximal aktiviert sein wird, entspricht diese Berechnung einer exakten Interpolation der Trainingsdaten. Jedes Neuron entspricht einer Stützstelle in der Ausgabe.

Dieses Verfahren funktioniert nur unter der Bedingung, dass die Anzahl der RBF-Neurone der Anzahl der Trainingsmuster entspricht, anderenfalls lässt sich $\bar{M}$ nicht invertieren. Es gibt hier allerdings einige Ansätze, das Problem zu lösen. Der populärste Ansatz ist die Anwendung der Deltaregel, um nach dem Prinzip des Gradientenabstiegs günstige Gewichte für die Ausgangsgewichte zu berechnen. Somit könnte im Rahmen eines Trainings auch die Anzahl der RBF-Neurone variiert werden, um bestmögliche Ergebnisse zu erzielen. Zusätzlich könnten auch c und σ der

radialsymmetrischen Funktionen noch im Rahmen eines Trainings optimiert werden. Für die weiteren Betrachtungen des Verfahrens reicht die einfache Variante aber aus.

Bewertung

RBF-Netze ermöglichen wie MLP die Abbildung der geforderten Lokalisierungsfunktion durch Training mit repräsentativen Referenzpunktmessungen.

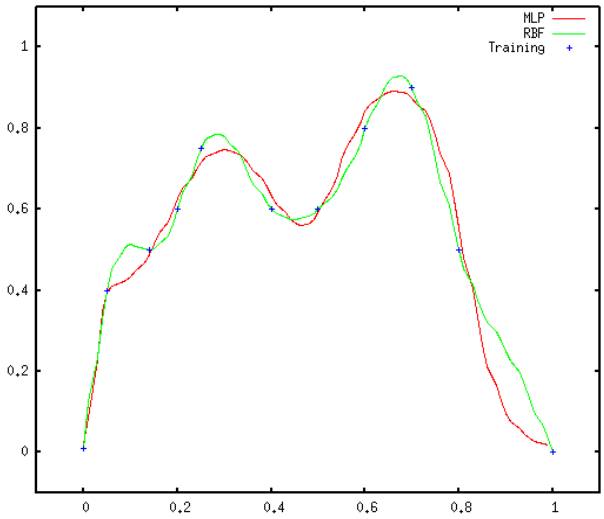


Abbildung 19: Lösung eines Problems durch MLP und RBF im Vergleich

Abbildung 19 zeigt das unterschiedliche Verhalten von RBF und MLP. Das interpolierende Verhalten des RBF-Netzes (grüne Kurve) lässt sich gut erkennen, die Trainingsmuster (blaue Punkte, Eingabe = x, Ausgabe = y) werden genau getroffen. Die Interpolation des RBF-Netzes liefert ein ähnliches gutes Gesamtergebnis wie das MLP, auch wenn z.B. bei x = 0.1 deutlich wird, wie sich das Verhalten der Gauss-Funktion vom Verhalten der Fermi-Funktion der Aktivierungsfunktion des MLP unterscheidet.

Die Generalisierungsfähigkeit des RBF-Netzes lässt sich allerdings schon durch Variation von σ sehr differenziert einstellen. Hauptkriterium ist hier der Abstand der Trainingseingaben voneinander. Zu große σ können zu oszillierendem Verhalten führen (grüne Kurve in Abbildung 20), zu kleine σ führen zu einer schlechten Generalisierungsfähigkeit (blaue Kurve).

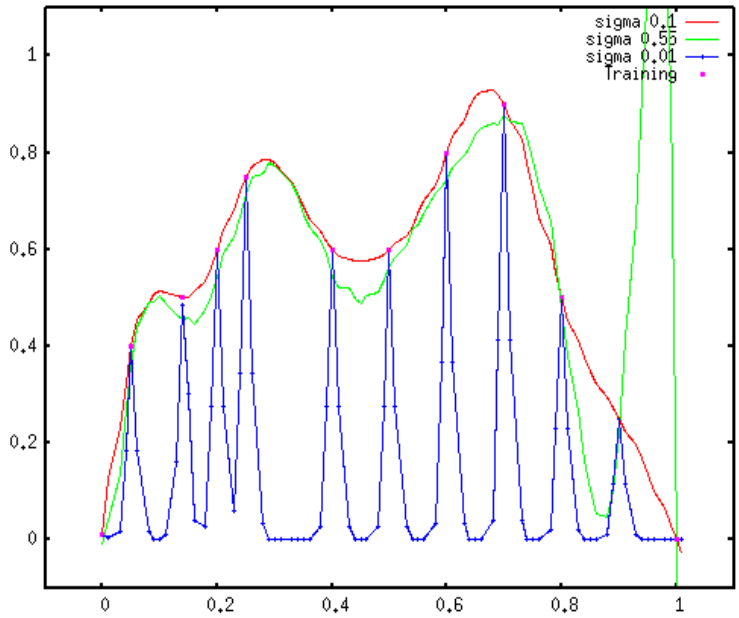


Abbildung 20: Auswirkung der Variation von Sigma bei RBF

Große Unterschiede bestehen im Extrapolationsverhalten von RBF im Vergleich zu MLP. Existieren keine Trainingseingaben für bestimmte Bereiche, tendieren RBF dazu, dort gar keine Aktivierung aufzuweisen, also umgangssprachlich formuliert, zu sagen: „Ich weiß es nicht“ [18].

Diese Eigenschaft kann gut oder schlecht bewertet werden. Zum einen wird ein MLP immer einen plausiblen Wert liefern, man weiß aber nicht, wie zutreffend dieser in Wirklichkeit ist; hier ist das RBF eher pessimistisch, liefert dafür aber auch keine möglicherweise komplett verkehrten Ergebnisse.

Das Verhalten lässt sich an einer Folge von zehn fiktiven Distanzwerten im Intervall [0;1] mit Schrittweite 0,1 skizzieren.

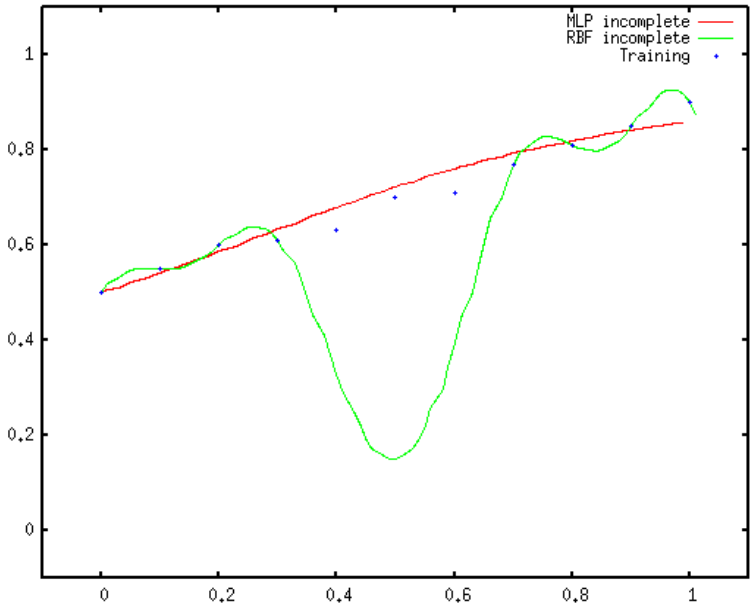


Abbildung 21: Extrapolationsverhalten von MLP und RBF

1	0.0	0.5
2	0.1	0.55
3	0.2	0.6
4	0.3	0.61
5	0.4	0.63
6	0.5	0.7
7	0.6	0.71
8	0.7	0.77
9	0.8	0.81
10	0.9	0.85
11	1.0	0.90

Abbildung 22: Fiktive Distanzwerte

Abbildung 21 zeigt die in Schritten (x-Achse) von 0,1 gemessenen Distanzwerte (y-Achse) als blaue Punkte. Diese Daten wurden als Trainingseingabe für das MLP auf Abbildung 13 und für ein RBF-Netz mit $\sigma = 0{,}1$ verwendet, wobei die Messungen 5, 6 und 7 (s. Abbildung 22) vor dem Training entfernt wurden, um das Verhalten der Netze auf die fehlenden Daten zu evaluieren.

Die rote Kurve zeigt das MLP, welches die fehlenden Punkte recht gutmütig überbrückt. Die Ausgabe des RBF-Netzes fällt hier anschaulich an der grünen Kurve stark ab. Ein größeres σ hätte zwar den Abfall etwas dämpfen können, hätte aber in anderen Bereichen des Netzes zum Überschwingen führen können. Dieser Fall lässt weiterhin erkennen, dass es äußerst sinnvoll sein kann, die Variation von σ mit in das Training einzubeziehen, wodurch sich aber wieder ein ähnlich großer Trainingsaufwand wie beim MLP ergibt.

Das Training eines RBF-Netzes ist in dieser schlichten Form sehr einfach, da lediglich Parameter für bekannte Funktionen eingestellt werden müssen. Es handelt sich praktisch gesehen nicht um Training, sondern um die Lösung einfacher Gleichungen, sofern Parameter wie σ nicht weiter angepasst werden sollen (s. Lernverfahren).

Bei einer großen Anzahl an Trainingsmustern wird der Berechnungsaufwand allerdings sehr schnell sehr groß, da in diesem Fall für die Erstellung der Matrix $\bar{M}$ jedes RBF-Neuron jede Eingabe einmal berechnen muss. Der Rechenaufwand steigt somit auch

mit der Größe der Eingabedimension, was für die Anwendung der Indoor-Lokalisierung aber noch unkritisch ist, da hier im der Praxis nicht mehr als vier bis fünf Referenzpunkte in einer Zelle gemessen werden. Die aufwändige Berechnung der Gewichte muss allerdings nur neu erfolgen, wenn neue Trainingsmuster hinzukommen. Der Rechenaufwand für die Online-Phase ist somit, wie auch beim MLP, gering.

Es muss davon ausgegangen werden, dass diese einfache Variante bei sehr vielen Messungen von Referenzpunkten nicht ohne eine Reduzierung von RBF-Neuronen und, damit einhergehend, anschließendes Training nach Delta-Regel praktizierbar sein wird.

3 Verwandte Arbeiten

3.1 Am Markt erhältliche Produkte

3.1.1 Chipcon CC2431 Location Engine

3.1.1.1 Hardwarebeschreibung

Texas Instruments bietet mit seinen SOCs aus der Chipcon-Familie eine sehr reichhaltige Palette an Hardwarelösung für drahtlose Sensornetzwerke im Mote-Format [21]. Der für die Indoor-Lokalisierung besonders interessante Part an diesem SOC ist die sogenannte Location-Engine, ein in Hardware gegossener Algorithmus zur Bestimmung der eigenen Position in einem hierarchischen Netzwerk von Motes [22].

Das Messverfahren setzt auf die Erfassung der Signalstärke (RSSI) aller in der Umgebung befindlichen Referenz-Motes.

3.1.1.2 Theoretische Leistungsfähigkeit

Die Hardware ist laut Hersteller in der Lage, eine Ortungsgenauigkeit von bis zu 25cm zu erreichen [22]. Hierfür werden mindestens drei Referenz-Motes benötigt. Die Verlässlichkeit der Ergebnisse selbst steigt mit einer größeren Anzahl an Referenznodes, bis zu 16, noch weiter an, während die Genauigkeit auf 25cm beschränkt ist.

Laut TI kann ein Feld von maximal 63,75 x 63,75 Metern abgedeckt werden.

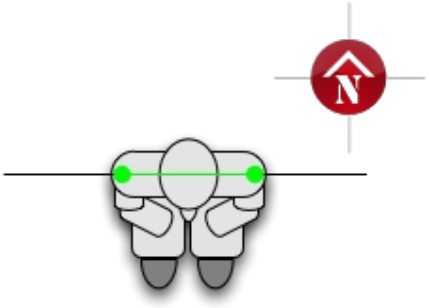


Abbildung 23: Erkennung der Orientierung mit zwei Motes

Mit einer Ortungsgenauigkeit von 25cm ließen sich schon die gängigsten

Anwendungsfälle in der Indoor-Lokalisation abbilden. Um eine Aussage über die Orientierung von Objekten zu treffen, müssten jeweils zwei Motes in einem Mindestabstand von mehr als 50cm an diesem angebracht sein. Die Abbildung 23 stellt den Vorgang der Orientierungserkennung mit Hilfe von zwei Motes (grün) dar. Die Lagebeziehung der beiden Motes gäbe Aufschluss über die Orientierung der markierten Person.

3.1.1.3 Allgemeiner Implementationsansatz

Das grundlegende Konzept sieht vor, dass über die Location-Engine der momentanen Aufenthaltsort geschätzt wird. Die ermittelten Datensätze werden dann an einen „Location-Service“ gesendet und dort abgespeichert. (Abbildung 24)

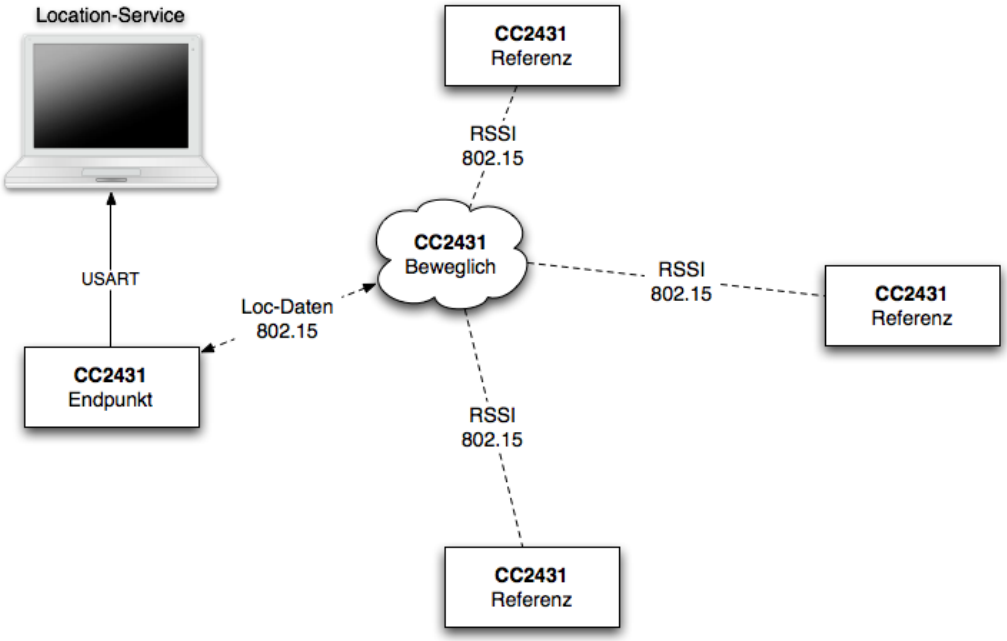


Abbildung 24: Funktionsskizze der CC2431 Location Engine

Die Testumgebung muss hierfür mit einer gewissen Anzahl Referenz-Motes ausgestattet werden, deren Standorte bekannt sind. Diese senden über die Funktionen der Location-Engine ein Beacon-Signal in einem festen Intervall. Dieses Beacon-Signal wird von der beweglichen Mote, dessen Position ermittelt werden soll, erfasst und für die Auswertung herangezogen. Der Auswertungsprozess ist bereits in der Location-Hardware des CC2431 implementiert. Der Location-Service lauscht im drahtlosen IEEE 802.15-Netz und wartet darauf, dass ein bewegliches Mote seine Position ermittelt hat und seine neuen Koordinaten bekannt macht.

Der Ansatz von TI versucht die Zuverlässigkeit des Gesamtsystems noch weiter zu verbessern, indem die Daten der Location-Engine im Location-Service nicht nur abgelegt werden (Tracking) sondern auch mithilfe statistischer Verfahren auf Basis alter Werte verglichen und ggf. korrigiert werden können. Die korrigierten Daten werden dann an das bewegliche Mote zur Übernahme zurückgesendet.

3.1.1.4 Erstellen von Software für CC2431

TI bietet zwar eine Referenzimplementation für die Infrastruktur um die Location Engine an, doch wird bei einer eingehenderen Auseinandersetzung mit der Technik schnell der Bedarf nach einer eigenen Implementierung erwachsen, die mehr auf den jeweiligen Use-Case zugeschnitten wäre.

Für die Hardware stehen nur eine kleine Menge an Entwicklungswerkzeugen zu Verfügung. Während es zwei größere, kommerzielle Lösungen gibt, stehen auch freie Lösungen zur Verfügung, die sich hinsichtlich Funktionsumfang und -struktur aber deutlich von den kommerziellen Komplettlösungen unterscheiden: Die kommerziellen Lösungen von IAR [23] und Keil [24] bieten eine komplette Entwicklungsumgebung inkl. Compiler, Bibliothek und integrierter Entwicklungsumgebung. Die freien Lösungen bestehen aus einer Ansammlung einzelner Tools, die vornehmlich der Erstellung der für den CC2431 benötigten Binaries dienen. Eine Softwarebibliothek oder Entwicklungsumgebung muss hingegen manuell hinzugefügt werden.

Lösung	**IAR Workbench**	**Keil µVision**	**SDCC**	**TinyOS**
Hersteller	IAR Systems	Keil	Open source	Open source
Lizenz	Kommerziell	Kommerziell	Frei	Frei
Compiler	Ja	Ja	Ja	SDCC oder Keil
HAL-Bibliothek	Optional	Optional	Nein	Ja
IDE	Ja	Ja	Eclipse	Eclipse
Status	Stabil	Stabil	Stabil	Alpha

Tabelle 1: Gegenüberstellung verschiedener Entwickler-Tools für TI CC2431

Für den Einsatz in einer öffentlichen Einrichtung wie der HAW wurde davon ausgegangen, dass dem Einsatz von freien Lösungen gegenüber rein kommerziellen Varianten der Vorzug gegeben werden sollte, da hier zum einen durch den

vorliegenden Quellcode eine Unabhängigkeit von einzelnen Herstellern erreicht wird und der didaktische Wert größer ist – die Funktionsweise kann bis ins letzte nachvollzogen werden. Bei Bedarf kann man somit auch an der Basis Veränderungen vornehmen.

Durch die großen Unterschiede zwischen den beiden freien Lösungen, SDCC und TinyOS, ergeben sich deutlich unterschiedliche Entwicklungsabläufe, die im Folgenden kurz skizziert werden sollen.

3.1.1.5 Plain-C mit SDCC

Die reinste Lösung, der Hardware am nächsten und der unmittelbare Ansatz ist die Realisierung aller notwendigen Programmteile in einfachem Ansi-C. Dieser Ansatz kann mit dem genannten Compiler SDCC [25] verfolgt werden.

Die Standarddistribution von SDCC liefert bereits einige, nützliche Header-Dateien, die bereits alle essentiellen defines enthalten, um die Register des CC2431 beim Namen anzusprechen. Durch ein Makro ist es auch möglich einzelne Bits der Register direkt anzusprechen, ohne große Schiebe- und Maskierungsoperationen.

Ebenfalls werden alle essentiellen Funktionen für Interrupt-Verarbeitung geliefert, sodass die Hardware vollständig kontrolliert werden kann [26].

Implementationsansatz

Als Basis für die Implementation wird eine leichtgewichtige C-Bibliothek erstellt, die alle essentiellen Funktionen implementiert, die für die Lokalisation benötigt werden. Hierzu zählen:

- Konfiguration nach Boot (Takt, Energie, I/O-Mapping)
- Einfaches I/O über USART
- Steuerung der Location-Engine
- I/O über Radio
- Ggf. Komfortfunktionen wie Temperatur-Sensor

Die Bibliothek wird mit SDCC vorkompiliert und kann dann für die jeweilige Anwendung

hinzugelinkt werden.

Die jeweilige Anwendung kann auf dieser Bibliothek fußen und je nach Anwendungsgebiet angepasst werden.

Grenzen des Ansatzes

Die allergrößte Hürde an diesem Ansatz ist die Komplexität der Anwendung. Da keine HALs verwendet werden, muss sehr viel Basiscode erzeugt werden. Die hardwarenahe Programmierung ist mühselig und benötigt sehr viel Zeit. Ansonsten sind keine Grenzen gesetzt, da man die volle Kontrolle über die Hardware hat.

3.1.1.6 TinyOS

TinyOS ist eine sehr vielfältige Entwicklungsplattform für autonome Sensornetzwerke auf Basis populärer Mote-Produkte [27]. Es liefert ein ereignisgesteuertes Laufzeitsystem, welches priorisiertes Multithreading, Messagequeues und einen sehr mächtigen Hardware Abstraction Layer bietet. Für die effektive Modellierung von Ereignissen wurde die Programmiersprache NesC eingeführt, eine Erweiterung zu Ansi-C, welche auf ereignisgesteuerte Systeme perfekt zugeschnitten ist.

TinyOS selbst liegt in Form von Quellcode in seinem Installationsverzeichnis vor und kann als mächtige Softwarebibliothek für die Entwicklung eben solcher Sensornetzwerke verstanden werden. Zusätzlich zu diesem Quellcode wird ein Buildsystem auf Basis von Make eingeführt, welches mithilfe des NesC-Compilers den NesC-Quellcode des Entwicklers in ein intermediate Ansi-C File überführt (mangling). Dieses intermediate File wird dann vom C-Compiler der Wahl übersetzt [28].

TinyOS unterstützt selbst den CC2431 nicht, es gibt allerdings eine Arbeitsgruppe in Stockholm, die tinyos8051wg [29], welche sich um die Portierung zum Vorgänger CC2430 kümmert. Der CC2431 ist binärkompatibel zum 2430 und sollte somit ebenfalls unterstützt werden.

Die Portierung ist allerdings noch nicht abgeschlossen, sodass manche Funktionen der Hardware noch nicht zur Verfügung stehen. So fehlen zum Beispiel sämtliche Funktionsaufrufe an die im CC2431 zur Verfügung stehende Location Engine. (Stand März 2009)

Implementationsansatz

Im Rahmen der Vorstudien zu dieser Arbeit sollte der Einsatz von TinyOS auf CC2431 untersucht werden.

Da der CC2430 nicht über die Location-Engine des CC2431 verfügt, gibt es auch noch keinen HAL aus der tinyos8051wg hierzu. Es müsste nach TinyOS-Standard ein Modul geschaffen werden, welches die Funktionen der Location-Engine über NesC zur Verfügung stellt. Dieser Arbeitsschritt wäre im Kern identisch mit dem Ansi-C-Ansatz, nur dass bei TinyOS der ganze Rest bereits zur Verfügung steht und nicht ganz von vorne angefangen werden müsste.

Grenzen des Ansatzes

Leider ist es im Laufe der Untersuchungen nicht gelungen, lauffähigen Code für den CC2431 mit Hilfe des tiny8051wg-Ports zu erstellen. Der Fehler wird im von NesC erstellten Code vermutet. Dieser scheint nicht unter allen Umständen fehlerfrei mit SDCC kompiliert werden zu können.

Vorläufige Untersuchungen trafen auf einen Fehler der internen Ereignissteuerung, welche über Interrupts gesteuert wird. Der Interrupt-Handling-Code, der von NesC erstellt wird, scheint nicht kompatibel mit SDCC zu sein. Weitere Untersuchungen wurden abgebrochen, um die anderen Ansätze im zeitlichen Rahmen weiterverfolgen zu können.

TinyOS ist jedoch eine sehr vielversprechende und für Entwickler angenehme Plattform und die Programmiersprache NesC bietet willkommene Ergänzungen zu Ansi-C.

3.1.2 UWB Location Detection (Ubisense)

Das Ortungsverfahren der Firma Ubisense [30] basiert auf einer eigens hergestellten Hardwarelösung auf Basis von UWB. Das Akronym UWB steht für die Ultra Wideband Technologie welche sich dadurch auszeichnet, sehr kostengünstig und energieeffizient zu arbeiten. Die Ortung findet in Echtzeit statt und gibt eine Aussage über die Lage der mobilen Endpunkte in 3 Dimensionen.

Das System besteht aus mehreren Referenzpunkten, welche an festen Plätzen installiert werden, sowie beliebig vieler sogenannten „Tags“ welche kontinuierlich eine

Abbildung 26: Ubisense-Compact Tag (Quelle: [31])

Abbildung 25: Ubisense-Referenzpunkt (Quelle: [32])

Art Beacon Signal aussenden. Anhand dieses Signales welches über die Referenzpunkte eingefangen wird, ist eine zentrale Location Engine in der Lage, die Position jedes Tags zu bestimmen. Die Position eines Tags wird im Ubisense-System mittels einer Kombination aus AOA- und TDOA-Algorithmen realisiert. Um die Zeitsynchronisation zwischen den Referenzpunkten sicherzustellen werden die Referenzpunkte mit einer Timing Leitung verbunden.

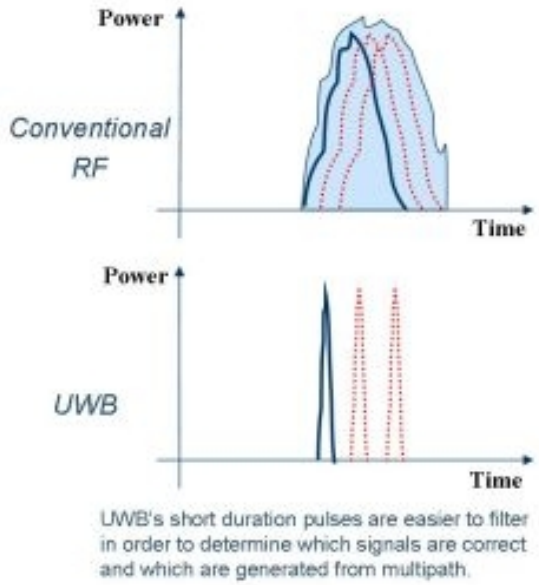


Abbildung 27: Pulserkennung bei UWB (Quelle: [33])

Neben der Möglichkeit über das Ultra Wideband die Position eines Tags zu ermitteln, ermöglicht Ubisense in ihrer Lösung die bidirektionale Kommunikation mit jedem Tag auf einem 2,4 GHZ Funkkanal.

Die Vorteile des Ortungsverfahrens von Ubisense liegen in der Genauigkeit. Die durchschnittliche Präzision der Lokalisierung mittels UWB beträgt nach Herstelleraussagen 15 cm [33]. Diese Genauigkeit ist auf die technische Möglichkeit der Verwendung von Laufzeitalgorithmen anstelle von Signalstärke-Algorithmen zurückzuführen. Die Abbildung 27 verdeutlicht das Vorgehen der Laufzeitmessung. UWB ist in der Lage aufgrund von zeitlich sehr kurzen Signalimpulsen eine Reflexion vom direkten Signal zu unterscheiden.

Ein weiterer Vorteil ist die vielseitige Integrationsmöglichkeit in bestehende Systeme. So können Referenzpunkte über Standard Netzwerke wie LAN oder WLAN verbunden werden und es stehen Schnittstellen in Programmiersprachen wie c++, C# und Java zur Verfügung. Zudem gilt das UWB-System als extrem skalierbar da es keine zentralen Instanzen im System gibt, welche einen Flaschenhals darstellen könnten.

Als nachteilig ist zu vermerken, dass der Einsatz jeder Art von UWB-Lösungen eine spezielle Hardware benötigt. Es wäre zu klären, inwiefern ein UWB-Tag in ein bestehendes, mobiles Endgerät integriert werden könnte. Zudem benötigt UWB eine eigenständige Installation und Wartung der Referenzpunkte.

Da momentan keine Möglichkeiten bekannt sind, UWB-Tags (Abbildung 26) in bestehende mobile Geräte zu integrieren, beschränkt sich der Anwendungsbereich momentan auf die Ortung von mobilen Punkten, welche beispielsweise mit hoher Genauigkeit in einem Arbeitsablauf zentral getrackt werden müssen.

3.1.3 WLAN Location Detection

3.1.3.1 EPE

Die Firma Ekahau Inc. [34] vertreibt ein WLAN-basierendes Ortungssytem mit dem Namen Ekahau Positioning Engine (EPE). Das System basiert auf der Messung von Signalstärken. Es ist in der Lage mit verschiedenen mobilen Endgeräten gleichzeitig zu arbeiten. Dies ist möglich, da die Signalstärken vor der eigentlichen Berechnung angepasst and die jeweilige Hardware normalisiert werden. Die Lokalisierung wird mit Hilfe eines Fingerprinting-Algorithmus gelöst, wodurch keine Abbildung auf ein Koordinatennetz möglich ist, sondern Verfahren zur symbolischen Lokalisierung verwendet werden. Fingerprinting benötigt sehr detailierte Informationen über

Grundrisse und Raumpläne und eine eingehende Vermessungsphase sowie ständige Kontrolle und Pflege. Ist ein solches System jedoch erst einmal richtig eingemessen, verspricht es einen hohen Grad an Genauigkeit und Zuverlässigkeit.

3.1.3.2 Magic Map

Magic Map ist eine von der freien Universität zu Berlin entwickelte Lokalisierungstechnologie. Sie wird beschrieben als ein so genanntes Hybrides Lokalisationsverfahren, welches mehrere Ortungstechnologien parallel verwendet um daraus die höchste Genauigkeit zu ermitteln. So ist das System in der Lage, mit Hilfe verschiedener Technologien wie ZigBee und WLAN sowie auch unterschiedlicher Distanzwertalgorithmen parallel die Position zu bestimmen. Magic Map benötigt zur Funktionalität anders als andere Verfahren nicht zwingend das Wissen über die Position von Referenzpunkten. Um die Genauigkeit allerdings zu erhöhen, ist es wünschenswert diese dem System bekannt zu machen. Zudem besitzt es die Möglichkeit über Peer to Peer Kommunikation der mobilen Endgeräte untereinander die Genauigkeit zu erhöhen [35]. In der Basiskonfiguration stellt Magic Map jedoch nur den Aufbau eines RSSI basierenden Lokalisierungssystems mittels Lateration dar.

Der strukturelle Aufbau des Systems ist sehr modular gehalten. So unterscheidet Magic Map zwischen folgenden Komponenten [36]:

- Stumbler

  Er zeichnet die RSSI Signalstärken von Referenzpunkten auf und vermittelt diese weiter an verarbeitende Dienste. Da Magic Map nicht allein auf WLAN Technologie festgelegt ist, werden auch Stumbler für alle verfügbaren Technologien benötigt.

- Server

  Der Server übernimmt die Rolle eines Koordinators, welcher die von anderen Knoten eingehenden Signale und Messwerte weiter verteilt.

- Positioning Engine

  Sie der eigentliche Kern von Magic Map. Die Positioning Engine selbst ist nicht nur als zentrale Instanz zu verstehen, sondern lässt sich ebenfalls als verteilter Algorithmus auf mehrere Geräte verteilen. Die Skalierbarkeit des Algorithmus

ist hierdurch stark im Vorteil.

- P2P-Kommunikation

  Sie dient der Kommunikation von mobilen Endgeräten untereinander. Die Kommunikation findet in diesem Falle nicht über den Server statt, sondern wird direkt unter den Clients hergestellt. Verwendung findet diese Technologie im Bereich der Nahfeldortung. Hier verhilft sie der Präzision einer Lokalisierung.

- Tracker

  Er fungiert als eine Art Event Listener auf verschiedene Ereignisse. Ein Tracker kann auf bestimmte Positionsdaten angesetzt werden, welche er im laufenden beobachtet und bei bestimmten Ereignismustern vor einstellbare Aktionen auslöst.

3.1.3.3 Skyhook Wireless (XPS)

Bei dem Lokalisierungssystem der Firma Skyhook Wireless inc. [37] handelt es sich weniger um eine Softwarelösung als um einen Internet Dienst. Er erlaubt es mit dem Internet verbundenen Endgeräten sehr schnell sowie in- und ausserhalb von Gebäuden die Position zu ermitteln. Die dafür von Skyhook Wireless erfundene Technologie heißt XPS, einer Kombination aus WLAN Trilateration , GPS und der GSM-Lateration (Siehe Abbildung 28).

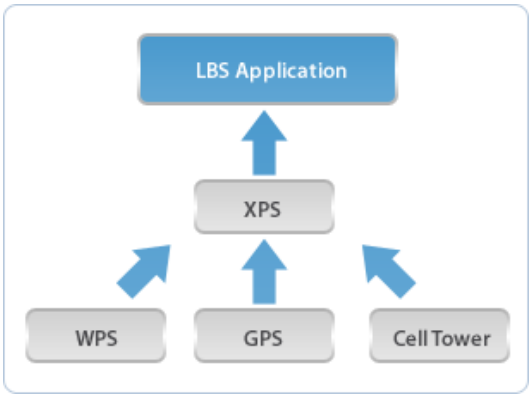


Abbildung 28: XPS Skyhooks Wireless Positioning System (Quelle: [37])

XPS [37] benötigt nicht alle aufgeführten Technologien um ein aussagekräftiges Ergebnis liefern zu können. Je mehr Technologien das System jedoch miteinander kombinieren kann, je geringer fällt der Fehler des Gesamtsystems aus. Zudem kann

XPS durch geschickte Kombination der Technologien die Rechenzeit hingegen einer einzelnen Technologie deutlich verringern. Als Beispiel gilt hier der Einsatz von GPS. Allein betrieben benötigt es zur ersten groben Lokalisierung sehr lange. Der Grund dafür ist, dass ein „orientierungsloses“ GPS Endgerät keinerlei Informationen über die Satellitenkonstellationen am Himmel vorliegen hat. Um diese zu ermitteln benötigt das Endgerät die für den Standort wichtigen „Almanach-Daten“. Diese zu empfangen kann jedoch mehrere Minuten dauern. Steht dem GPS allerdings schon Vorkenntnis über den ungefähren Aufenthaltsbereich durch z.B. GSM-Lateration zur Verfügung, fällt die Lokalisierung deutlich schneller aus.

Bei der Erfassung der Referenzstationen für weitere Laterationsalgorithmen geht Skyhook Wireless eigene Wege. So sind auf der Welt mehrere Autos betrieben von Skyhook Wireless unterwegs, um mithilfe mehrerer Empfänger die Signalstärken und Positionen von Accesspoints und Handymasten zu erfassen [38]. Zudem hat man selbst die Möglichkeit seinen Accesspoint mit genauer Position auf der Website des Herstellers zu registrieren.

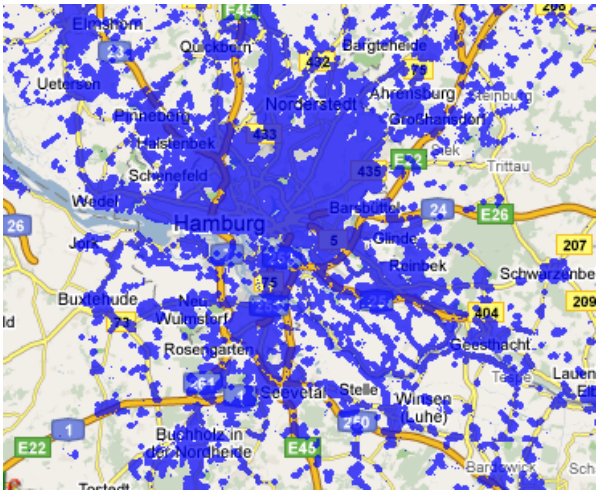


Abbildung 29: Netzabdeckung durch Skyhook Wireless in Hamburg (Quelle: [38], Stand Mai 09)

Die maximal zu erwartende Genauigkeit des XPS Systemes liegt bei 5 bis 10 Metern. Die Netzabdeckung beschränkt sich ohne GPS jedoch mehr auf Ballungsgebiete.

Für Entwickler stellt Skyhook Wireless ein Software Development Kit (SDK) zur Verfügung welches es einem erlaubt, diese Lokalisierungstechnik in seine eigenen Softwarelösungen zu integrieren.

3.2 Arbeiten an der HAW Hamburg

3.2.1 IMAPS

IMAPS ist ein verteiltes Indoor-Lokalisierungssystems auf Basis von Ultraschall und Zigbee, welches Sebastian Gregor 2006 im Rahmen seiner Bachelorarbeit [9] geschaffen hat und auch in folgenden Arbeiten an der HAW Anwendung fand [39][40]. IMAPS basiert auf der Grundlage des Cricket-Konzeptes von Nissanka B. et al. vom MIT, welches 2000 auf der sechsten ACM MOBICOM vorgestellt wurde [41].

3.2.1.1 Konzept

IMAPS basiert wie Cricket auf einer Infrastruktur von eigenständigen Sendeeinheiten, welche Beacons genannt werden. Diese werden im jeweiligen Raum an der Decke installiert und senden in zufälligen Zeitabständen Funk- (Zigbee) und Ultraschall-Signale aus. Diese Signale können von Listenern, Geräten, die an Ihrer Position im Raum interessiert sind, empfangen und ausgewertet werden. Ein Listener besteht hierbei aus einer kleinen Hardware, welche analog zum Beacon aufgebaut ist und dessen Signale empfangen kann. Die Distanz zu einer Beacon wird dabei aus der Differenz der Empfangszeitpunkte von Funk- und Schallsignal ermittelt. Die Distanzwerte mitsamt der Positionen der Beacons, welche über das jeweilige Funksingnal des Beacons übertragen wurden, werden im Anschluss vom Listener an einen über RS232 angeschlossenen Rechner übergeben, welcher einen Laterationsalgortihmus auf die ermittelten Distanzwerte ausführt. Der Listener führt somit eine Selbstortung durch. Die Auswertung der Position kann vom System flexibel gehandhabt werden und sowohl auf Koordinaten als auch symbolische Positionen abgebildet werden.

3.2.1.2 Kritik

Die Genauigkeit der Distanzmessung ist dank Ultraschall auf kurzen Distanzen bemerkenswert hoch. Typische Messfehler überschritten 50mm nicht. Der Einsatz eines Laterationsverfahrens bietet sich bei solch genauen Daten an. Die Lateration kann wegen ihrer einfachen Berechnung direkt am Endgerät durchgeführt werden, sodass keine weitere Datenkommunikation mit einem zentralen Dienst stattfinden muss. Ebenfalls kann bei einer guten Kollisionserkennung auf komplexe

Fehlerkorrekturen verzichtet werden, wodurch der Bedarf an Kalibirierungmessungen entfällt und die Infrastruktur somit schnell in Betrieb genommen werden kann. Multipath-Effekte können durch die langen Signallaufzeiten recht gut erkannt werden. Kurze Messintervalle erlauben auch die Möglichkeit, Mobilität festzustellen.

Der Einsatz von Ultraschall beschränkt den Einsatzbereich allerdings auf einen einzelnen Raum, da die Schallwellen Wände nicht durchdringen können. Größere Bereiche müssen pro Raum mit Beacons ausgerüstet werden. Ist die Anforderung an die Präzision der Ortung nicht zu hoch entstehen somit ggf. unnötige Kosten.

Da das System auf eigenständiger Hardware aufbaut, ist eine aufwändige, vorangehende Installation des Systems nötig. Ebenso kann aus diesem Grund nur spezielle Client-Hardware zum Einsatz kommen, welche in Gregors Prototypen aus einem sperrigen Laptop bestand. Thomas Pfaff thematisierte letztere Einschränkung in seiner Bachelorarbeit 2007 und skizzierte deren Umgehung durch den Einsatz von Bluetooth als Kommunikationsmedium zwischen Listener und z.B. PDA oder einem anderen, ultramobilen Endgerät. Der zusätzliche Hardwareaufwand für den Ultraschall- und Zigbee-Empfang bleibt in Form eines dedizierten Empfangsmoduls aber bestehen. Die Beacons müssen außerdem mit Strom versorgt werden, was bei einer hohen Beacon-Dichte pro Raum über die Möglichkeiten der vorhandenen Hausverkabelung hinausgehen wird, sodass mit Batterien gearbeitet oder eine zusätzliche Verkabelung stattfinden muss.

3.2.1.3 Einordnung

IMAPS stellt eine konkrete Alternative zu dem in dieser Arbeit untersuchten Verfahren dar. Abgesehen vom gleichen Ziel - den Ort innerhalb eines geschlossenen Gebäudes zu ermitteln, gibt es allerdings so gut wie keine Gemeinsamkeiten. Sowohl die Distanzwertermittlung als auch der Lokalisierungsalgorithmus basieren auch wegen der unterschiedlichen Qualität der Ausgangsdaten auf völlig unterschiedlichen Ansätzen. Ebenfalls ist die Anforderung der Mobilität mit beiden Verfahren sehr unterschiedlich zu bewerten. Während IMAPS grundsätzlich den Fokus auf Genauigkeit und Entkopplung legt, wird der Ansatz dieser Arbeit geleitet von Anforderungen an die Praktikabilität der Integration und Reduzierung der Implementationskosten

3.3 Andere Arbeiten

3.3.1 Paper von Julian K. Buhagiar, Carl J. Debono

Buhagiar und Debono [5] untersuchten verschiedene Paradigmen von KNN theoretisch auf die Anwendbarkeit in physikalischen Lokalisierungsverfahren.

3.3.1.1 Konzept

Ziel der Untersuchung war, bei einer gegebenen Umgebung, auf Basis einer WCDMA-Mobilfunkinfrastruktur (z.B. UMTS) die optimale KNN-Variante empirisch zu ermitteln. Die Untersuchung deckt dabei drei Paradigmen von KNN ab, radiale Basisfunktionen (RBF), Multi Layer Perceptron (MLP) mit Backpropagation und Self-Organizing Maps (SOM).

Unabhängig von der Wahl der KNN-Variante soll der Algorithmus in zwei Phasen unterteilt sein, die "Neural Network Training algorithm"-Phase, in der das jeweilige KNN anhand der erhobenen Messungen trainiert wird und die "Neural Network Detection Algorithm"-Phase, in der das KNN zur Lokalisierung verwendet wird. Der Lokalisierungsalgorithmus soll dabei auf einem zentralen Server beim Anbieter ablaufen, um auch noch andere Location Based Services anbieten zu können.

Das Paper benennt verschiedene Effekte des Hochfrequenzfunks als Quelle für Störungen und Messungenauigkeiten. Hervorgehoben werden verschiedene *Fading*- und *Shadowing*-Effekte, die im Rahmen einer künstlichem Umgebung simuliert wurden, um die Störanfälligkeit der verschiedenen KNN-Varianten zu untersuchen. Die Ergebnisse des Papers zeigen, dass alle drei KNN-Varianten generell für Lokalisierungsalgorithmen verwendet werden können, dass selbst-organisierende Netze aber für die vorgestellte Umgebung die größten Erfolgsaussichten versprechen. Weiterhin zeigten die Untersuchungen, dass die Güte der Lokalisierung proportional zur Dichte der Sendemasten steigen soll und das Szenario somit besonders für urbane Umgebungen mit einem hohen Bedarf an Sendemasten zur Sicherstellung der Übertragungskapazitäten sinnvoll erscheint.

3.3.1.2 Kritik

Buhagiar und Debono zeigen auf, dass KNN prinzipiell für die Aufgabe der Lokalisierung geeignet scheinen und dass die theoretisch erreichbare Leistungsfähigkeit eines solchen Ansatzes zu brauchbaren Ergebnissen führt. Die vorgestellte Umgebung weist jedoch zahlreiche Unterschiede zu einer typischen Indoor-Umgebung auf. Sowohl die Fläche des Raumes, als auch die Dichte der Sendemasten führen eine wesentlich größere Spreizung der empfangenen Signalpegel am Endgerät nach sich, da die Distanzen im Vergleich zu einem Indoor-Szenario um den Faktor 10 bis 100 überschritten werden.

Buhagiar und Debono stellen selbst fest, dass eine kontinuierliche Varianz der empfangenen Singalpegel für eine zuverlässige Ortung unerlässlich scheint. Gerade im Indoor-Bereich bieten die nahen Wände ein gegenüber Outdoor-Szenarien stark erhöhtes Potential für Fading-Effekte, welche die Varianz der Messwerte erheblich schmälern. Die Verlässlichkeit des Verfahrens muss im Indoor-Bereich somit neu untersucht werden.

Die Erkenntnisse der Untersuchungen gehen weiterhin auf eine simulierte Umgebung zurück, sodass nicht ausgeschlossen werden kann, dass die Ergebnisse in einer realen Umgebung abweichen können.

3.3.1.3 Einordnung

Das vorgestellte, zwei-phasige Vorgehensmodell zur Implementation von KNN in einem Lokalisierungsdienst liefert eine allgemeingültige Grundlage für derartige Verfahren. Dieser Ansatz stellt somit auch die Basis für das Verfahren in dieser Arbeit dar.

Während das Paper vorrangig den Algorithmus betrachtet und nur vage Aussagen zur Infrastruktur macht, soll in dieser Arbeit der Transfer des Ansatzes in ein reales Szenario stattfinden und auch Fragen der Anwendbarkeit und Kosteneffizienz berücksichtigen. Durch die angesprochenen Unterschiede zwischen dem im Paper vorgestellten Outdoor-Szenario und dem in dieser Arbeit zugrundeliegenden Indoor-Szenario muss das Verfahren allerdings zuvor neu bewertet werden.

4 Analyse

Nachdem in dieser Arbeit Verfahren zur Lokalisierung sowie auch auf dem Markt erhältliche Systeme vorgestellt wurden, beschäftigt sich das folgende Kapitel damit, eine Anforderungsanalyse vorzunehmen um sich für einen Implementationspfad festlegen zu können.

4.1 Anforderungen

Angelehnt an etablierte, mobile Lokalisierungssysteme ([30][34][36][37][5]) wurden Anforderungen an ein eigenes System aufgestellt. Im Folgenden werden die Anforderungen unterteilt in harte und weiche Anforderungen. Harte Anforderungen sollten von einem System erfüllbar sein, weiche sind eine Option, die den Nutzwert des Systems noch weiter steigern, aber nicht für alle Szenarien zwingend benötigt werden.

4.1.1 Harte Anforderungen

H1. Das System muss in der Lage sein anhand von eigenständig ermittelten Distanzwerten auf die Position des Endgerätes im Raum zu schließen.

H2. Das Endgerät soll mobil sein und eventuelle Datenkommunikation drahtlos erledigen.

H3. Die Position im Raum muss auf ein Koordinatensystem abzubilden sein.

H4. Das System muss auch ohne Sichtkontakt zwischen Referenzpunkten und Endgerät funktionieren, da Abschattungen durch Menschen, die sich im Raum bewegen, spontan entstehen können.

H5. Das System muss über mehrere Räume hinweg funktionieren.

H6. Das System muss temporäre Störungen soweit ausgleichen können, dass die Qualität der Lokalisierung für die Anwendung stets ausreichend ist.

4.1.2 Weiche Anforderungen

W1. Die Lokalisierung sollte eine Genauigkeit von < 1m erreichen.

W2. Das System sollte aufgrund der Kostenentwicklung auf Consumer-Hardware aufbauen und keinen zusätzlichen, infrastrukturellen Aufwand nach sich ziehen.

W3. Es sollte möglich sein, das System schrittweise anzulernen und zu erweitern, insbesondere über mehrere Räume. Hierbei sollten Offline-Phasen kurz gehalten oder sogar vermieden werden.

W4. Es sollte möglich sein, die Orientierung des Endgerätes im Raum ermitteln zu können.

4.2 Vorauswahl

Die Anforderung W2 soll von Anfang an erfüllt sein, sodass das ausgewählte System tatsächlich nur mit Hardware, die für den Consumer-Markt bestimmt ist, funktionieren soll. Für die folgenden Ausführungen soll von einer IEEE 802.11g-WLAN-Infrastruktur ausgegangen werden. Diese signifikante Einschränkungen wurde beschlossen, um auch der Anforderung H2 gerecht zu werden und eine praktische Evaluation mit engem finanziellen Rahmen durchführen zu können. H4 ist somit im Rahmen eines Wohnraumes ebenfalls erfüllt, ebenso wie W4, da WLAN-Accesspoints günstig nachträglich installiert werden können.

4.3 Auswahl eines Verfahrens

Um die zuvor erläuterten Probleme einer WLAN-Installation in der Praxis zu ergründen, wurde sich eingehend damit beschäftigt, eine Testumgebung aufzubauen in welcher die Charakteristik von WLAN in Indoor Umgebungen erläutert werden kann.

Die Abbildung 30 zeigt die zur Verfügung stehende Testumgebung. Sie ist 5,7 x 11 Meter groß und beinhaltet ein kleines Badezimmer. Der Rest des Raumes ist offen. Da sich die Testumgebung in einem Dachgeschoss befindet, besitzt sie zu den beiden längeren Seiten große Dachschrägen. In der Testumgebung wurden an drei Stellen WLAN Accesspoints installiert, welche als Referenzstationen dienen sollen. Sie sind in der Abbildung 30 als $R_{o..n}$ eingezeichnet. Sie wurden alle in einem einheitlichen Abstand von 15cm über dem Boden installiert.

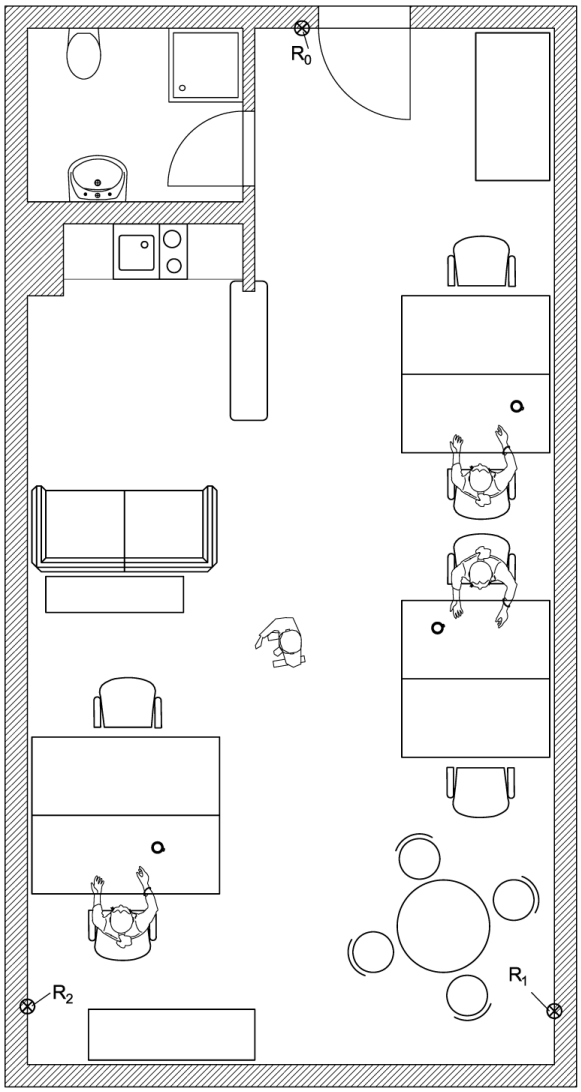


Abbildung 30: Indoor-Lokalisierungsszenario

Die Testumgebung beinhaltet mehrere Tischgruppen mit Metallfüßen. Zudem herrscht kontinuierlich Bewegung durch andere Mitarbeiter im Raum. Diese Änderungen sollten das System in der Praxis möglichst wenig beeinflussen.

Um einen Gesamtüberblick über den Raum zu erhalten, wurde sich dafür entschieden, den Raum gleichmäßig mit einem Raster zu unterteilen und an jedem so entstehenden Rasterpunkt Referenzmessungen vorzunehmen, um einen Einblick in die Richtcharakteristik der Accesspoints zu gewinnen.

Wie in Abbildung 31 zu sehen ist, wurde sich für ein Messraster mit einem Meter Abstand entschieden, welches im Raum mittels Klebestreifen auf den Boden

aufgetragen wurde.

Abbildung 31: Testumgebung mit aufgetragenem Messraster (hervorgehoben)

Für den Raum von 5,7m x 11m wurden so nun 55 Rasterpunkte erstellt, welche in einem nachfolgenden Schritt dazu verwendet werden konnten, um an jedem der Rasterpunkte Referenzmessungen vorzunehmen. Die Abbildung 33 zeigt die Aufteilung des gewählten Rasters im Raum. Bei den Referenzmessungen geht es darum, die Signalpegel der Referenzstationen $R_{o\text{-}n}$ zu ermitteln. Sie werden dazu verwendet um Aussagen über die Ausleuchtung der Funksignale im Raum zu treffen.

Pro Messpunkt wurden über einen längeren Zeitraum mehrere Messungen vorgenommen und bei nachfolgenden Auswertungen die arithmetischen Mittelwerte aller Messungen verwendet. Dieses Verfahren verhalf dazu, Ausreißer welche durch unvorhersehbare Störungen auftraten, zu eliminieren.

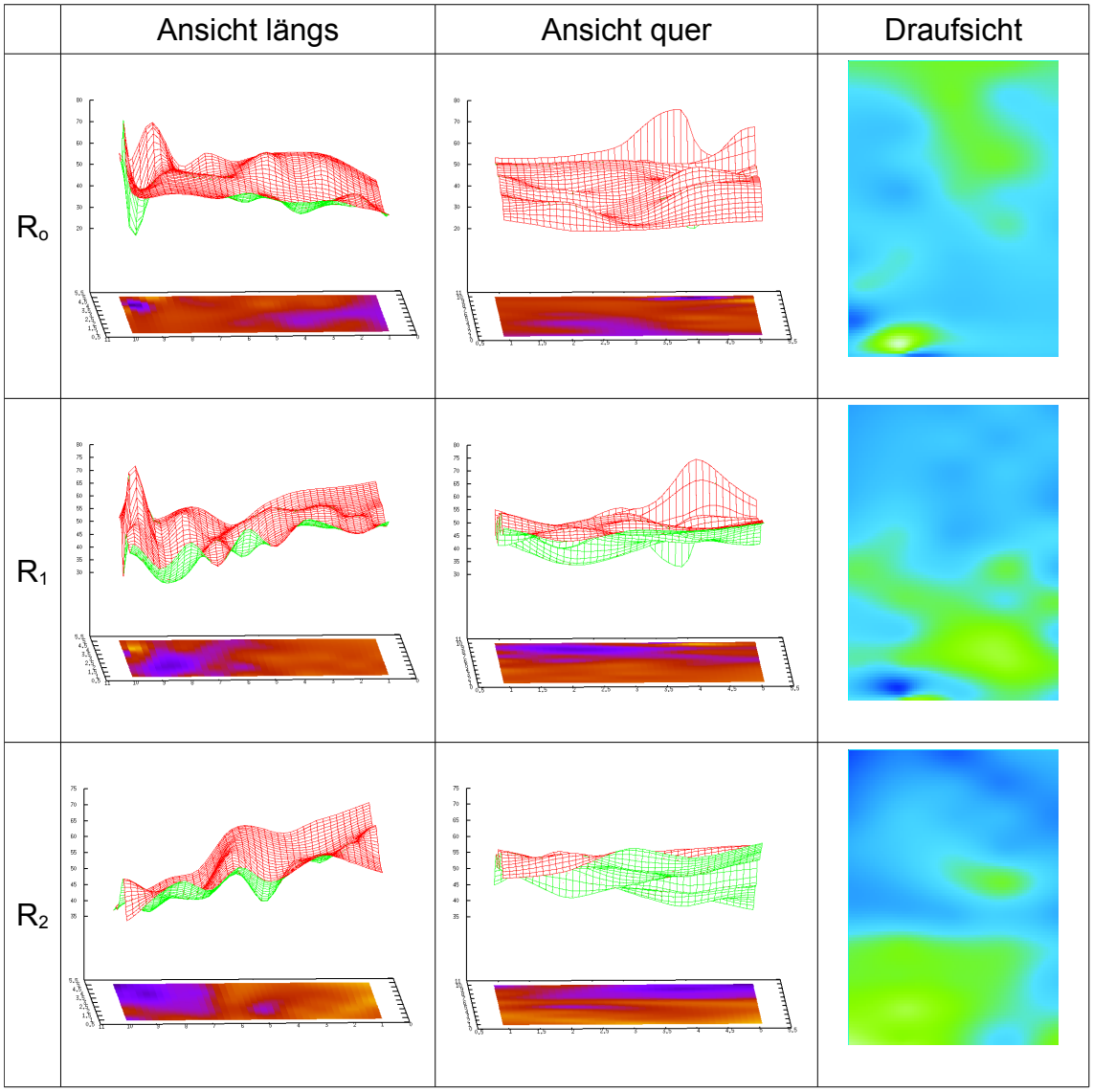


Abbildung 32: Darstellung der Signalpegelverläufe dreier Referenzpunkte im Raum

Anhand der gesammelten Signalpegelmessungen ließ sich nun für jede Referenzstation eine individuelle Karte über die Ausleuchtung der Messdaten ermitteln. So ergab sich für jeden der drei verfügbaren Referenzstationen ein Set an Messergebnissen (Abbildung 32). Die Achsen X und Y beschreiben die Raumabmessungen, Z den zugehörigen RSSI-Wert.

Wie in den Draufsichten der Abbildung 32 zu erkennen ist, ist das Abstrahlverhalten der Signalpegel im Raum keineswegs gleichförmig in alle Richtungen verteilt. Es wird viel mehr durch die antennenspezifische Charakteristik sowie durch *Fading-* und *Shadowing* Effekte im Zusammenhang mit der Umgebung beeinflusst. Das Resultat ist ein nur sehr schwer vorhersagbares Abstrahlverhalten, welches sich nur sehr schwer

modellieren lässt.

Um einen Eindruck des Abstrahlverhaltens aller Referenzpunkte in der Testumgebung zu erhalten, wurde ein Summenbild aller Signalpegelmessungen der Referenzstationen erstellt. Dieses Summenbild wurde maßstabsgetreu über den Grundriss der Testumgebung gelegt (Abbildung 33).

Wie anhand der Messungen in der Testumgebung deutlich wird, bedarf es bei herkömmlichen Verfahren wie der RSSI-Lateration einer eingehenden Vorverarbeitung der Messdaten, um zufriedenstellende Ergebnisse erlangen zu können. Besteht das Bestreben die Position möglichst genau zu ermitteln, so ist dies mit dem RSSI Verfahren allein nur möglich, in dem die Umgebung in der das System verwendet werden soll, extrem wenigen Veränderungen ausgesetzt und die verfügbare Rechenkapazität zur Positionsbestimmung deutlich höher, als die eines mobilen Endgerätes ist. Ist dies nicht der Fall, gibt es die Möglichkeit, mehrere Algorithmen zur Distanzwertermittlung miteinander zu verknüpfen. Diese Vorgehensweise wird unter anderem von Systemen wie Magic Map (Kapitel 3.1.3.2) und auch Ubisense UWB (Kapitel 3.1.2) verfolgt.

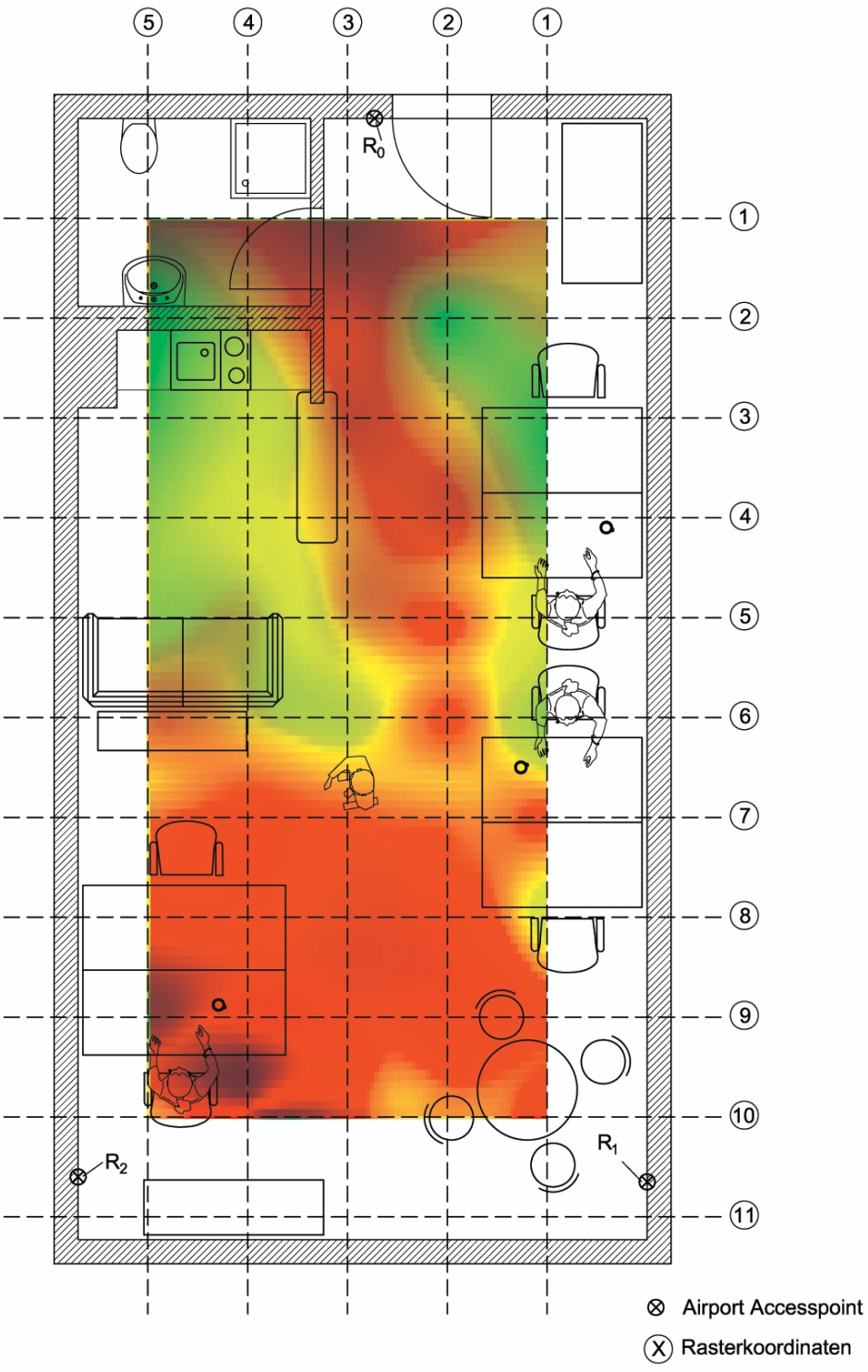


Abbildung 33: Summenbild aller Signalpegelmessungen der drei Referenzpunkte

Stellt man nun die verbleibenden, verfügbaren Systeme gegenüber und vergleicht sie anhand der gegebenen Anforderungen ergibt sich einem folgendes Bild:

Anforderung	Skyhook	RSSI WLAN Lateration (z.B. Magic Map)	EPE
Genauigkeit	-	?	-
Consumer Hardware	x	x	Teilweise
Ausfallsicherheit	x	?	x
Erweiterbarkeit	-	x	-
Störanfälligkeit	-	x	-
Kosten	x	x	-
Positive Eigenschaften	4	3 bis 5	2 bis 3
x= ja -=nein ?=Keine Information			

Tabelle 2: Gegenüberstellung verschiedener auf dem Markt erhältlicher Lokalisierungssysteme in Bezug auf die gegebenen Anforderungen.

Betrachtet man die Anzahl der als positiv zu wertenden Eigenschaften, so stehen die Systeme von Skyhook Wireless sowie ein auf RSSI basierendes System wie Magic Map in direkter Konkurrenz. Da bei dem System von Skyhook jedoch keine Möglichkeit besteht, sich eine private Indoor-Umgebung nur für eigene Zwecke zu schaffen und die Erfassung der Referenzstationen nicht selbst gesteuert werden kann, stellt sich dieses System trotz der hohen Punktzahl als ungeeignet dar. Der Sieger der Gegenüberstellung ist somit eindeutig ein lokal bereitgestelltes, auf RSSI basierendes Lokalisierungssystem wie Magic Map.

Warum jedoch die Punktzahl an positiven Eigenschaften bei Magic Map "3 bis 5" beträgt, bedarf einer genaueren Erklärung. Die Genauigkeit des Magic Map-Systems ist beeinträchtigt, wenn die Raumtopologie veränderbar ist. Zudem liegt die Stärke des Magic Map Systems darin, sich mehrere Technologien zu Hilfe zu nehmen. So besitzt Magic Map die Möglichkeit der Erweiterung durch andere Übertragungstechnologien [35] sowie auch verschiedener Distanzwertalgorithmen. Zudem versucht das System Schwankungen durch Peer-to-Peer Topologien zu eliminieren, was im Umkehrschluss bedeutet, dass die Anzahl mobiler Endgeräte einen direkten Einfluss auf das Lokalisierungsergebnis hat.

Diese Eigenschaften beeinträchtigen die Punktzahl der Gegenüberstellung in den Kategorien "Genauigkeit" und "Ausfallsicherheit".

Ruft man sich nun die gegebenen Anforderungen noch einmal ins Gedächtnis, fällt es schwer, ein auf RSSI basierendes System als das geeignetste für den Bereich Indoor-Lokalisierung zu nennen. Es bedarf eines fehlertoleranten Verfahrens zur Lokalisierung, welcher sich zur Nutzung mit herkömmlicher Consumer-Hardware eignet und der in der Lage ist, zu erlernen mit den vorhandenen Einschränkungen umzugehen.

Künstliche, neuronale Netzen sollten den Einschätzung aus Kapitel 2.3.4.2 zufolge hierzu in der Lage sein und wurden deshalb für den weiteren Verlauf dieser Arbeit als Verfahren ausgewählt.

4.4 Anforderungen an das Verfahren

Mit der Entscheidung für den Einsatz künstlicher, Neuronaler Netze auf Basis von RSSI-Messungen musste im Folgenden untersucht werden, wie diese Technik sinnvoll evaluiert werden kann. Von zentralem Interesse war insbesondere die Frage nach der Wahl der KNN-Variante und der Parametrisierung des Lernalgorithmus, deren Beantwortung aber die Erstellung eines geeigneten Implementationspfades voran gehen musste.

Als erster, grober Entwurf des Implementationspfades wurden vier ineinandergreifende Arbeitsschritte definiert:

I1. Definition der Referenzpunkte

I2. Aufzeichnung der Trainingsmuster

I3. Design und Training des KNN

I4. Verifikation des Systems

4.4.1 Definition der Referenzpunkte

Die Definition wurde für den vorliegenden Raum bereits in Kapitel 4.3 vorgenommen, es kommen die Referenzpunkte R_0 bis R_2 zum Einsatz. Inwiefern die Wahl der Referenzpunkte maßgeblich für die Leistungsfähigkeit der Anwendung ist, konnte an dieser Stelle noch nicht abgesehen werden und wird auch bei der Implementation in anderen Räumen stets ein ungewisser Faktor sein. Es bietet sich an, die Referenzpunkte so zu wählen, dass möglichst große Abstände zwischen ihnen entstehen, sodass der sehr schmalbandige Wertebereich des RSSI bestmöglich ausgeschöpft wird.

4.4.2 Aufzeichnung der Trainingsmuster

Der Aufzeichnung der RSSI-Werte im Raum musste bereits eine größere Aufmerksamkeit zuteil werden, da hier gleich mehrere Probleme zu lösen waren. Zunächst musste ein Applikation gefunden werden, die in der Lage sein sollte, die RSSI-Werte zu allen Referenzpunkten kontinuierlich aufzuzeichnen, sodass die

Messdaten später leicht weiterverarbeitet werden können. Hierbei ist insbesondere darauf zu achten, dass die Reihenfolge der gemessenen Referenzpunkte stets die gleiche sein muss, da jeder Eingang des KNN einen bestimmten Referenzpunkt repräsentiert.

Im Anschluss sollten die RSSI-Werte normiert werden. Dieser Vorgang ist der Tribut an die herstellerspezifischen Wertebereiche des RSSI.

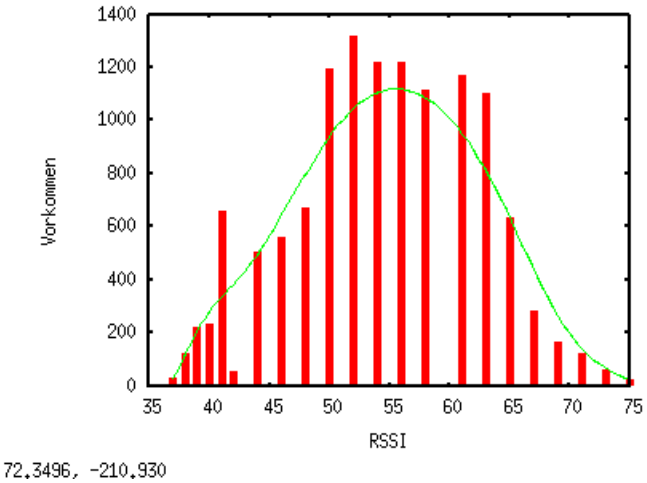


Abbildung 34: RSSI-Verteilung im Raum

Untersuchungen des Raumes zeigten eine relativ enge Konzentration von RSSI-Werten zwischen 40 und 65. Wie in Abbildung 34 dargestellt, ist unterhalb von 30 ($RSSI_{min}$) und oberhalb von 80 ($RSSI_{max}$) nicht mehr mit signifikanten Messwerten zu rechnen. Unter der Annahme, dass der RSSI zwischen 0 und 100 liegt, wurde somit festgelegt:

$$RSSI_{norm} = \frac{(|RSSI| - RSSI_{min})}{RSSI_{max} - RSSI_{min}} \quad (20)$$

Sollte der RSSI-Wert für andere Hardware nicht zwischen ±100 und 0 liegen, wobei 0 die größtmögliche Signalstärke repräsentiert, müsste eine weitere Normierung vorgeschaltet werden.

Das Ergebnis der Normierung liegt somit immer zwischen 0 und 1 für $RSSImin$ bzw. $RSSImax$, was unter anderem für MLP mit sigmoider Aktivierungsfunktion in der Ausgabeschicht eine schnelleres Lernen begünstigt.

Für die Erhebung der Messwerte wurde im Rahmen der Vorstudien eine eigene

Applikation entwickelt, die in der Lage sein sollte, die Erhebung und Normierung der Messwerte zu erledigen. Der „Datenlogger“ ist ein einfaches Kommandozeilentool, welches in Java geschrieben wurde und die Tatsache ausnutzt, dass Apple mit seinem Betriebssystem Mac OS 10.5 ein sehr komfortables WLAN-Kommandozeilentool ausliefert, welches eine Tabelle aller Accesspoints und deren RSSI-Werte erzeugt (s. Abbildung 35)[42]. Der Datenlogger kann also nur auf diesem Betriebssystem ausgeführt werden.

```
$ /System/Library/PrivateFrameworks/Apple80211.framework/Versions/A/Resources/airport -s
                            SSID BSSID             RSSI CHANNEL SECURITY
                  WLAN-6E2A95 00:1d:19:6e:2a:44 -77  4       WPA(PSK/TKIP/TKIP)
                     kiwigear 00:09:5b:eb:17:aa -41  11      WPA(PSK/TKIP/TKIP)
                   JUBUROENK2 00:11:50:8c:e0:7a -84  1       WEP
                     MnM oben 00:21:91:0b:4f:95 -63  6       WPA2(PSK/AES/AES)
                    JUBUROENK 00:15:e9:0d:ac:22 -76  6       WEP
                  referenzNico 00:1d:4f:aa:0b:e5 -50  3       NONE
                 referenzZahni 00:19:e3:e5:6b:7f -52  5       WPA2(PSK/AES/AES)
            referenzLautsprecher 00:1f:f3:04:32:3b -49  8       NONE
```

Abbildung 35: Ausgabe des WLAN-Kommandzeilentools

Die Anwendung des Programms besteht aus der Eingabe der aktuellen Position $\vec{l}_{real} \in A$ auf der Kommandozeile und einer anschließenden Messung, welche sich der Ausgabe, wie in Abbildung 35 gezeigt, bedient und daraus einen Trainingsdatensatz (R, A) erzeugt. Die Reihenfolge der benötigten Referenzpunkte wird dabei vorher in einer Konfigurationsdatei festgelegt.

Weiterhin muss auch der Ortsvektor $\vec{l}_{real}$ normiert werden, sodass die Werte zwischen 0 und 1 liegen. Dieser Schritt ist sehr wichtig, da viele Netzparadigmen mit logistischem Verhalten der Ausgangsneurone sonst nur sehr langsam oder gar nicht trainierbar werden [18]. Die Normierung kann hier ganz einfach durch Division durch Breite bzw. Höhe erreicht werden:

$$\vec{l}_{real\,norm} = \vec{l}_{real} \div \begin{pmatrix} \vec{w} \\ h \end{pmatrix} \tag{21}$$

Die Ausgabe des Datenloggers entspricht dem SNSS-Pattern-Format, sodass für Schritt 3) sofort nutzbare Daten zur Verfügung stehen.

4.4.3 Design und Training des KNN

Für die ersten Untersuchungen bot sich die Verwendung von JavaNNS [43] an. Dieses Tool, welches an der Universität Tübingen entwickelt wurde, bietet zahlreiche Möglichkeiten, KNN zu modellieren und zu trainieren.

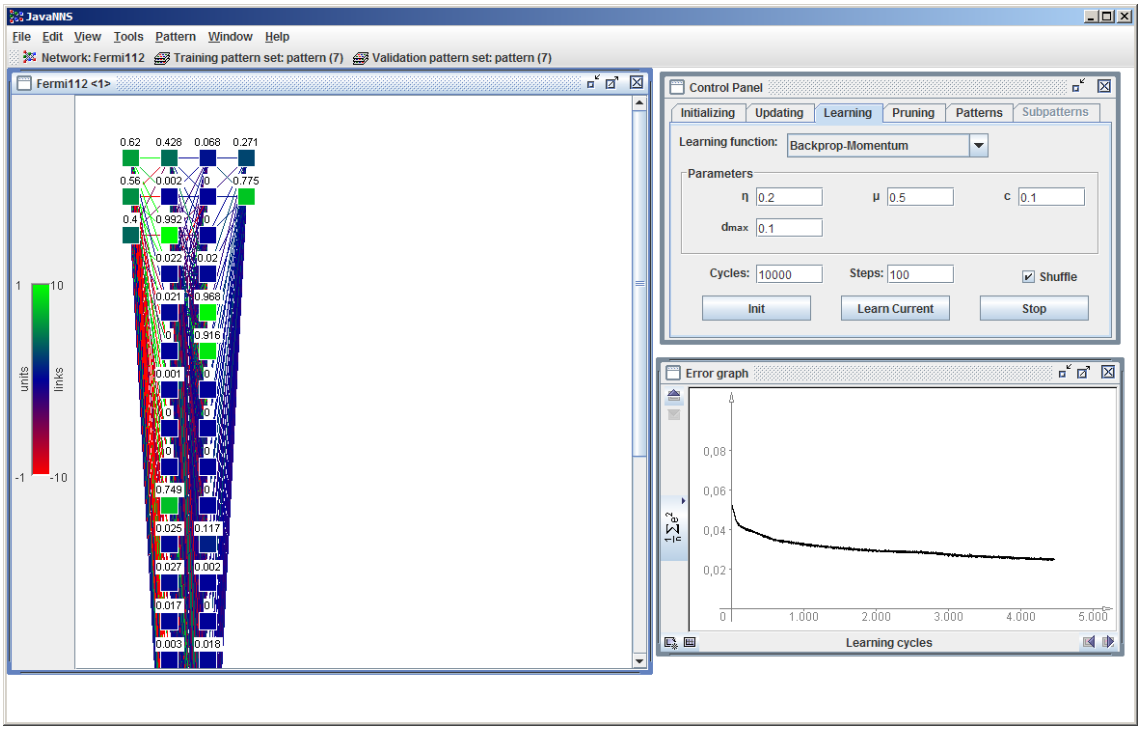


Abbildung 36: JavaNNS 1.1 im Einsatz

Der Arbeitsablauf in JavaNNS sieht vor, dass zunächst ein KNN angelegt wird. Das Anlegen des Netzes erfolgt hierbei schichtweise. Für jede Schicht kann angegeben werden, um welchen Typ von Neuron es sich handeln und welche Aktivierungsfunktion verwendet werden soll. Im Anschluss werden die Verbindungen zwischen den einzelnen Neuonen nach Feed-Forward-Schema hergestellt. Diese Arbeitsschritte werden grafisch anschaulich dargestellt, wie Abbildung 36 zeigt.

Nach dem Anlegen des Netzes lädt man eine oder mehrere Pattern-Files (*.pat), welche die Trainingsdaten enthalten. Der Inhalt der Pattern-Files muss je nach eingesetztem Netztypen angepasst sein, für MLP und RBF ist er allerdings identisch. Nach einem obligatorischen Header folgen $|T|$ Musterdefinitionen, die stets aus zwei Zeilen bestehen. Die erste Zeile enthält die normalisierten RSSI-Werte als Eingabe, die zweite Zeile die normalisierte Ortsangabe $\vec{l}_{realnorm}$ als

Trainingsausgabe.

Kommentare können mit einem vorangestellten # angegeben werden. Ein Beispiel eines einzelnen Trainingsmusters kann Abbildung 37 entnommen werden.

```
# Input pattern 0:
0.42 0.5 0.47
# Output pattern 0: 1 1
0.05 0.1
```

Abbildung 37: Trainingsmuster nach SNNS-Format

Nach dem Laden der Musterdateien kann das Training beginnen. Je nach Netztyp und Lernalgorithmus können verschiedene Parameter eingestellt werden, beim MLP mit Backpropagation-Momentum z.B. Schrittweite, Momentum und Anzahl der Trainingszyklen.

Der Fortschritt des Trainings kann im "Error Graph" begutachtet werden. Da hier das summierte Fehlerquadrat auf der Y-Achse aufgetragen ist, muss die Aussagekraft der Zahlen in den Hintergrund rücken, da sich aus dieser Zahl nicht direkt die Qualität des Trainings ableiten lässt. Die Praxis hat gezeigt, dass das Training dann beendet werden kann, wenn der Fehlergraph keine signifikanten Abstiege pro Zyklus mehr anzeigt.

4.4.3.1 Umsetzung in Programmcode

Die KNN, die mit JavaNNS generiert werden, können nicht direkt in Programmcode eingebettet werden, da die Network-Files (*.net), welche JavaNNS erzeugt, wenn man ein Netz speichert, nur deklarativen Charakter haben und keine Programmlogik enthalten. Es gibt hierfür allerdings einen kleinen Compiler namens „snns2c“, welcher 1995 von Bernward Kett geschrieben wurde. Der Compiler kann Network-Files in C-Quellen übersetzen, welche dann in beliebigen Programmcode eingebettet werden können [44].

```
$ ./snns2c wlannns4.net net.c measure
============================================================
   snns2c by Bernward Kett (1995)
============================================================
   converts wlannns4.net to net.c
   Function-Name measure
============================================================
loading net...
dividing net into layers ...
sorting layers ...
writing net ...
```

Abbildung 38: Übersetzungsaufruf für snns2c

Die Parameter, die dem Compiler übergeben werden sind:

1. Pfad zum Network-File
2. Pfad der zu erstellenden C-Datei (H-Datei wird mit gleichem Namen und anderer Dateiendung automatisch generiert)
3. Name der Methode, die die Netzfunktion realisieren soll

Der Compiler generiert dann gemäß dem Aufruf in Abbildung 38 eine Header- und eine Code-Datei. Vermutlich aufgrund des Alters des Compilers und der zu der Zeit verfügbaren Rechenleistung sind noch alle Gleitkommazahlen als `float` definiert. Vor der Weiterverarbeitung sollten sie daher als `double` umdeklariert werden, um die Fehlerfortpflanzung zu minimieren [16].

```
extern int measure(double *in, double *out, int init);
```

Code 1: Signatur der von snns2c erstellen Netzfunktion

Die Verwendung der Funktion ist dann denkbar einfach. Man definiere zwei `Arrays of double`, deren Längen $|I|$ bzw. $|O|$ entsprechen und führe die Methode mit diesen Arrays als Parameter aus. Das `in-Array` muss hierbei die normalisierten RSSI-Werte enthalten. `init` kann 0 sein. Nach erfolgter Berechnung enthält des `out-Array` die Netzausgabe, welche dann wieder denormalisiert werden kann.

4.4.3.2 Abschließende Bemerkungen zu JavaNNS

JavaNNS bietet durch seine Vielfalt von Netzparadigmen, Aktivierungsfunktion und Lernverfahren eine hervorragende Plattform für grundlegende Experimente und Erhebungen, da alle Parameter des Netzes schnell und unkompliziert manipuliert werden können. Besonders interessant ist JavaNNS im Zusammenhang mit der Beurteilung von Lernalgorithmen, da am Fehlergraphen einfach abgelesen werden kann, zu welchen Fehlerraten Netze bei bestimmten Parametern konvergieren.

Die Umsetzung der Netze in Programmcode ist leider mühsam und zudem auch eingeschränkt. snns2c ist wesentlich älter als die letzte Überarbeitung des SNNS-Kernels (1995 (Abbildung 38) vgl. 1998 [43]), sodass z.B. bei der Umsetzung von RBF-Netzen gelegentlich Fehler auftreten. Somit können RBF nicht ohne weiteren Aufwand mit JavaNNS in eine produktive Anwendung implementiert werden. Weiterhin gilt, dass jedes neue Training eine Kette von Aktionen nach sich zieht:

1. Netz in JavaNNS überarbeiten und speichern
2. Netz mit snns2c neu übersetzen
3. Quellcode der Ausgabe von snns2c anpassen
4. Produktivsystem neu übersetzen und bereitstellen

Zudem muss das Produktivsystem in der Lage sein, C-Quellen einzubinden, sodass die Auswahl der Implementierungssprache eingeschränkt ist oder zumindest ein weiterer Aufwand bei der Integration entsteht.

4.4.4 Verfikation des Systems

Die Verifikation kann jeweils in zwei Phasen durchgeführt werden. Zum einen kann während des Trainings des KNN, in der Offline-Phase, eine Verifikation stattfinden. Hierfür wird neben den Trainingsmustern noch ein weiterer Satz Muster zur „Offline-Verifikation“ zur Verfügung gestellt [19][45], die nicht in das Training einfließen. Während des Trainings wird der momentane Gesamtfehler dann ausschließlich anhand der Verifikationsmuster ermittelt. So kann sichergestellt werden, dass das Netz die Trainingsmuster nicht nur stur auswendig lernt, sondern generalisierendes Verhalten aufweist. Die Anzahl der Verifikationsmuster sollte hierbei nach praktischen Erfahrungen 20% bis 30% der Anzahl der Trainingsmuster betragen. Der Vorteil der Verifikation in dieser Phase liegt darin, dass Fehler noch korrigiert werden können, da das Training noch nicht abgeschlossen ist.

Besonders interessant für den Anwender ist die Verifikation der Lokalisierung im produktiven Betrieb. In dieser Phase soll von „Online-Verifikation“ gesprochen werden. Gerade für die Anwendung der Indoor-Lokalisierung ist eine Online-Verifikation von Interesse, da nicht bekannt ist, ob alle Situationen, die sich auf die Signalpegel der Referenzpunkte auswirken auch mit den Trainingsmustern repräsentativ widergespiegelt werden. Ein gezieltes, nachträgliches Training des Netzes anhand der Ergebnisse der Online-Verifikation ist mit den vorgestellten Netzparadigmen allerdings leider problematisch, insbesondere bei MLP, da ein nachträgliches Training bereits gelerntes Verhalten zerstören kann (Stabilitäts-Plastizitäts-Dilemma). Die Live-Verifikation beschränkt sich also zunächst darauf, dem Anwender ein Maß für die Leistungsfähigkeit der Anwendung zu liefern. Maßnahmen zur Verbesserung müssen

in einer weiteren Offline-Phase nach Ermessen des Anwenders getroffen werden. Es wäre jedoch denkbar, ein hybrides System zu entwickeln, dass während des Online-Betriebes eine Kopie eines KNN trainieren kann, welches dann später in einer atomaren Operation das alte KNN im laufenden Betrieb ersetzen könnte.

4.4.5 Bewertung

Die Umsetzung von KNN für Indoor-Lokalisierung wird über einen vierstufigen Implementationspfad abgebildet. Nur, wenn alle Stufen schnell und flexibel abzubilden sind, kann eine Evaluation der Technologie in der Praxis sinnvoll durchgeführt werden.

Der in den Vorstudien evaluierte Implementationspfad mit JavaNNS bedarf mehrerer, manueller Benutzereingriffe und Umwege über das Dateisystem. Zusätzlich kommt es zu sporadischen Problemen bei der Übersetzung des KNN in Quellcode. Somit scheidet dieser Pfad für die Implementation der Anwendung aus Zeitgründen aus.

Es wurde beschlossen KNN in dem zu schaffenden System im sogenannten „Neuronalen Kernel“ selbst zu implementieren. Der vierstufige Implementationspfad soll in eine neue, kompakte Anwendung eingebettet werden, die die Anforderungen H1 bis H6 und W2 bis W3 ad-hoc erfüllen soll. W1 und W4 sollen durch die Anwendung evaluierbar sein.

5 Design

Die Anforderung an die Lokalisierung mit Hilfe von KNN bringen Designentscheidungen mit sich, die im Folgenden erläutert und spezifiziert werden.

Da sich nach der Evaluationsphase die Zusammenarbeit mit KNN aus dem Programm JavaNNS als nicht praktikabel erwiesen haben (s. Kapitel 4.4.5), entstand der Bedarf nach einer möglichst universell einsetzbaren Bibliothek zur Verwendung von KNN. Diese sollte selbst implementiert werden. Sie sollte die Funktionen der MLP sowie der RBF Netze mit sich bringen, um direkte Vergleiche in der Praxis abbilden zu können.

Um die zu entstehende KNN Bibliothek soll eine Client/Server Infrastruktur geschaffen werden. Diese soll dazu dienen, Trainingsdaten zu sammeln und zu speichern und den Algorithmus der Lokalisierung auf eine zentrale Serverinstanz zu verschieben. Die clientseitige Berechnung der KNN erwies sich aufgrund der gesteigerten Anforderungen an die Prozessorleistung für Erfüllung der Anforderung W3 als Kandidat zur Auslagerung und sollte deshalb vom Server übernommen werden.

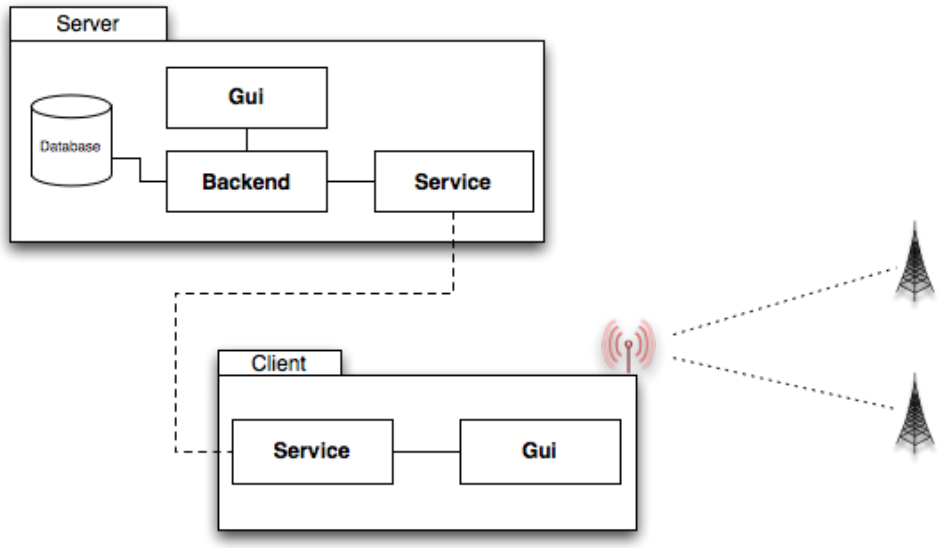


Abbildung 39: Packagediagramm des Designentwurfs

Zudem soll das System einen Service zur Verfügung stellen, der die Kommunikation zwischen mehreren Clients und dem Server abwickelt. Der Service sollte sehr universell einsetzbar sein um ihn in jede Art von Client implementieren zu können.

5.1 Neuronaler Kernel

Als Kernkomponente für die Realisation der Netzfunktion soll eine Library entstehen, welche die in dieser Arbeit vorgestellten Netzparadigmen MLP und RBF mitsamt den vorgestellten Lernalgorithmen implementiert.

5.1.1 Schnittstellenbeschreibung

Der neuronale Kernel soll kompakt gehalten werden und lediglich der Abbildung der Netzfunktion dienen. Die Schnittstelle `INeuralNetwork` stellt im Wesentlichen drei Methoden zur Verfügung, die der Eingabe (`setInput`), der Berechnung (`calculate`) und der Ausgabe (`getOutput`) dienen. Ein Lernalgorithmus `ILearnAlgorithm` muss eine Reihe von Trainingsmustern in das angegebene Netz integrieren können. Hierfür stehen eine inkrementelle Methode (`train`) und eine vollständige Methode (`trainAll`) zur Verfügung.

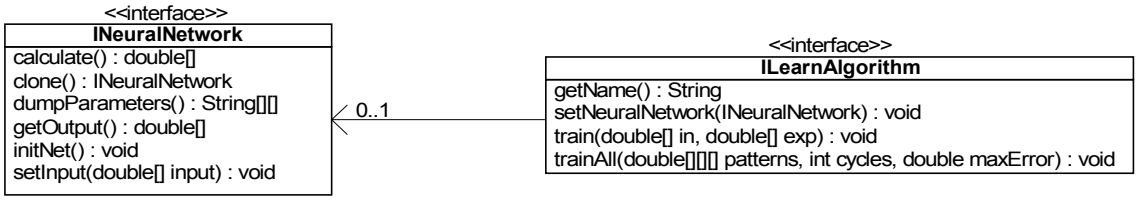


Abbildung 40: Schnittstellen des neuronalen Kernels

Die Datenstruktur der Trainingsmuster besteht dabei je nach Trainingsverfahren (inkrementell, vollständig) aus zwei Arrays, die je eine Trainingseingabe und die passende, erwartete Ausgabe beinhalten oder einem Array von Arrays von Trainingseingabe und Trainingsausgabe.

Die Anzahl der Lernzyklen, `cycles`, soll beim vollständigen Training die maximal gewünschte Zahl von Trainingsschritten festlegen, `maxError` den gewünschten, maximalen Gesamtfehler der Netzfunktion. Der Trainingsdurchlauf gilt als beendet, wenn die Anzahl der Zyklen oder der gewünschte Gesamtfehler erreicht wird, je nach dem was zuerst eintritt.

Das Erreichen der maximalen Anzahl von Zyklen impliziert, dass dementsprechend noch nicht der gewünschte Gesamtfehler erreicht wurde. Es sollte dann ein weiterer Trainingsdurchlauf gestartet werden.

5.1.2 Trainingssequenz

Ein Training gestaltet sich anhand der gegeben Schnittstellen als recht einfache und kurze Folge von Aufrufen. Das beispielhafte Sequenzdiagramm zeigt auch, wie die Implementation eines Trainings innerhalb eines Aufrufes von `trainAll` aussehen könnte. Die Methode `adjustParameters` ist klar erkennbar kein Mitglied der Schnittstellenbeschreibung, wird hier aber als Stellvertreter für die Anpassung des Netzes an die Trainingsparameter verwendet.

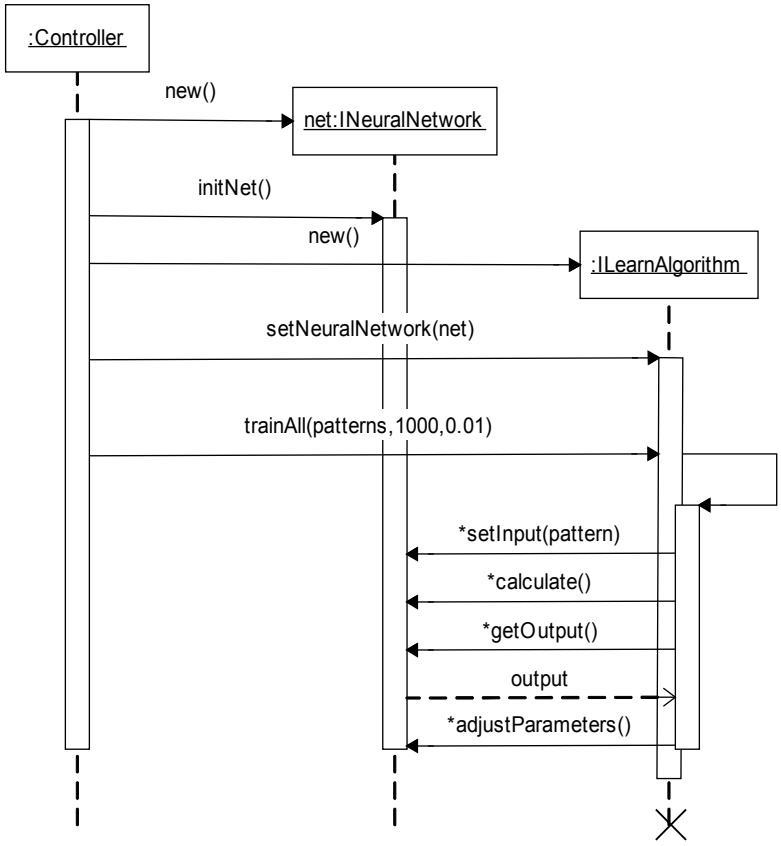


Abbildung 41: Einfache, vollständige Trainingssequenz

Abbildung 41 zeigt einen beliebigen Controller, welcher den Kernel verwendet. Nach der Erstellung und Initialisierung eines Netzes, wird dieses an den Trainingsalgorithmus übergeben, welcher durch den Aufruf von `trainAll` mit einer Reihe von Trainingsmustern hier `1000` Lernzyklen und mit einem maximalen Fehler von `0,01` trainieren soll.

5.1.3 Komponenten

Um die geforderten Netzparadigmen abbilden zu können, müssen die Schnittstellen so implementiert werden, dass die folgenden Funktionen zur Verfügung stehen:

5.1.3.1 Multi Layer Perceptron (MLP)

MLP sollen mit beliebig vielen verdeckten Schichten schnell und einfach erzeugt werden können. Da die Anzahl der Referenzpunkte variieren kann, muss auch die Größe der Eingabeschicht variabel gehalten werden. Der Schritt zu einer variablen Größe der Ausgabeschicht ist dann auch nicht mehr weit. Deshalb soll das Netz auch hier flexibel sein, auch wenn für die Anwendung eine feste Größe der Ausgabeschicht mit zwei Neuronen schon ausreichte.

5.1.3.2 Backpropagation Momentum Learning Algorithm

Als Lernverfahren für MLP soll Backpropagation Momentum implementiert werden. Dieses Verfahren basiert auf dem klassischen Backpropagation-Verfahren, erweitert dieses jedoch durch einen sogenannten Momentum-Term, der während des Trainings hilft, über ausgedehntere Plateaus im Fehlergebirge hinwegzukommen, sodass das Training sicherer gegen einen niedrigen Gesamtfehler konvergiert.

Das Backpropagation-Verfahren soll einzelne Trainingsdatensätze nach und nach verarbeiten können (inkrementell).

5.1.3.3 Radial Basis Functions (RBF)

Zusätzlich zu MLP sollen noch RBF zum Einsatz kommen oder zumindest evaluiert werden. Hierfür sollen RBF nach der Vorgabe $Anzahl\ Trainingsmuster\ |T| = Anzahl\ verdeckte\ Neuronen\ |H|$ implementiert werden.

5.1.3.4 RBF Learning Algorithm

Entsprechend der Vorgabe $|T|=|H|$ muss der Lernalgorithmus nur die einfache Aufstellung der Stützstellen für die Interpolation beherrschen.

5.2 Lokalisierungsdienst

Der Lokalisierungsdienst soll mobilen Endgeräten die Möglichkeit der Erfassung von Messdaten sowie der Lokalisierung zur Verfügung stellen. Um die Kompatibilität über mehrere Endgeräte hinweg zu gewährleisten, soll das Übertragungsprotokoll des Lokalisierungsdienstes möglichst universell und für jede Art von Endgerät verständlich sein. Hier wäre zu prüfen ob sich eine Implementation als Webservice oder als Socket Protokoll anbietet..

Der Dienst soll den Endgeräten folgende Funktionalitäten bieten:

LS1. Abrufen aller verfügbaren Räume (Areas)

LS2. Abrufen aller benötigten Referenzpunkte eines Raums

LS3. Abrufen einer Liste an bekannten Orten eines Raums

LS4. Übermitteln einer Messung bestehend aus Liste von RSSI-Werten der Referenzpunkte und dem momentanen Aufenthaltsort.

LS5. Ermitteln des aktuellen Ortes im Raum

5.3 Backend

Das Backend ist als zentrale Instanz des Systems zu sehen. Es soll den Zugriff auf ein Objektmodell gemäß des Klassendiagramms aus Abbildung 47 zur Verfügung stellen. Das Backend sollte sowohl für die Administrationsoberfläche sowie auch für den Lokalisierungsdienst erreichbar und verwendbar sein. Der Zugriff auf die Objekte sollte gemäß eines Zugangs Controllers (DAO) zur Verfügung stehen.

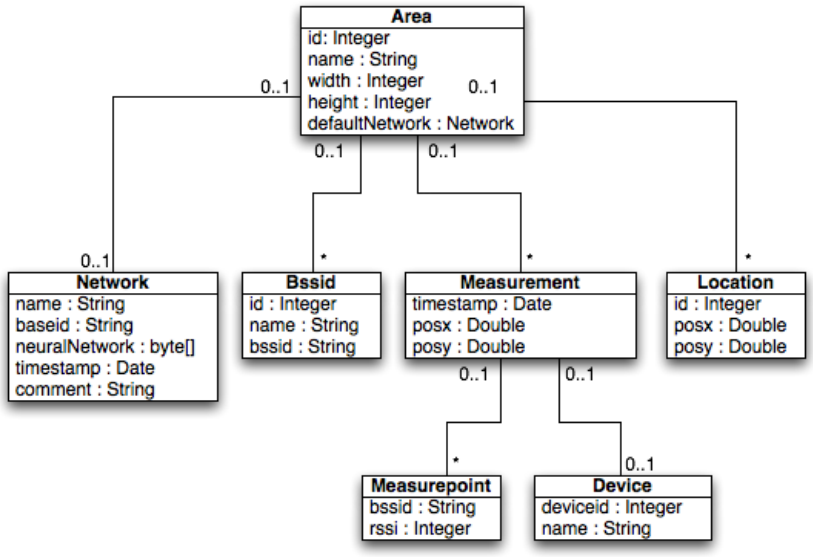


Abbildung 42: Klassendiagramm des Backends

Die folgende Abbildung zeigt das Klassendiagramm des Backends. Mit Hilfe dieser Klassenstruktur werden Referenzmessungen (`Measurement`), antrainierte KNN und Referenzpunkte persistiert. Oberste Entität ist der Raum (`Area`). Ihm sind alle anderen Entitäten untergeordnet.

5.3.1 Beschreibung der Entitäten

Anhand des Klassendiagramms bedingt es einer detaillierteren Beschreibung einiger Komponenten.

Klasse `Area`:

Ihr sind alle anderen Entitäten untergeordnet. Sie beschreibt einen Raum A mit seinen Abmessungen 'width' und 'height'.

Klasse `Network:`

Die Klasse `Network` hält jeweils immer ein Neuronales Netz vor. Dieses liegt in serialisierter Form vor und kann bei Bedarf deserialisiert und verwendet werden. Zudem hält die Klasse eine Versionsnummer sowie das Erstellungsdatum und eventuelle Kommentare vor.

Klasse `Bssid:`

Bei Objekten vom Typ `Bssid` handelt es sich um Referenzstationen. Eine Referenzstation ist immer einem Raum untergeordnet und besitzt immer einen Schlüsselwert namens 'bssid' und einen Beschreibungstext names 'name'. Referenzstationen dienen als Inputs des Neuronalen Netzes.

Klasse `Measurement:`

Ein `Measurement` beschreibt eine Messung von RSSI-Werten. Ein `Measurement` beinhaltet die Information der Position und des genauen Zeitpunktes an dem die Messung vorgenommen wurde. Zudem gibt sie Aufschluss über das Gerät welches die Messung getätigt hat. Ähnlich der `Bssid` ist ein `Measurement` auch immer direkt einem Raum untergeordnet.

5.3.2 Das Areakonzept

Nach Kapitel 2.1.4 soll die Lokalisierungsfunktion L eines Lokalisierungssystems auf einen Raum A abbilden. Um Anforderung W3 zu erfüllen, muss ein solches System in der Lage sein, mehrere Räume zu kennen und den richtigen Raum während der Lokalisierung auszuwählen. Somit soll das System beliebig viele A parallel verwalten können und dafür ausgelegt sein, dass sich Positionen innerhalb des A in der Realität überschneiden. Zudem soll die Möglichkeit gegeben sein, dass A vollständig ineinander geschachtelt werden können, so dass es möglich ist, innerhalb einer erfassten Fläche bestimmte Unterbereiche detaillierter zu erfassen. Dies bietet die Möglichkeit, an diesen ausgewählten Bereichen eine höhere Genauigkeit bei der Lokalisierung zu erlangen. Das Konzept bietet zudem die Möglichkeit, bestimmte A zu warten ohne das Gesamtsystem in seiner Funktionalität einzuschränken und bietet die Möglichkeit, mit Hilfe neu erzeugter A das Gesamtsystem schrittweise zu erweitern. Mit Hilfe dieser Herangehensweise ist es möglich, die Anforderungen aus W3 vollständig zu erfüllen.

Die Client-Implementierung soll durch das Areakonzept keinerlei Veränderungen erfahren. Die Wahl des geeignetsten A in dem sich der Client befindet soll auf Seiten des Backends anhand der empfangenen Referenzstationen und zugehöriger Signalpegel entschieden werden.

5.4 Client

Der Client nimmt bei der Erfüllung der Anforderungen eine besondere Rolle ein, da diese auf einem Endgerät installiert werden soll und somit ausschlaggebend für das Erlebnis des Benutzers ist.

Der Client besteht aus zwei Komponenten, zum einen aus einem mobilen Gerät (Anforderungen H2, H4, H5), welches über IEEE 802.11g verfügt (Anforderung H1, H2) und im Einzelhandel zu erwerben sein muss (Anforderung W2) und zum anderen aus einer Anwendung, welche in der Lage ist, über die drahtlose Netzwerkverbindung mit dem Lokalisierungsdienst zu kommunizieren (Anforderung H2). Mögliche Geräte wären PDAs oder Smartphones mit deren jeweiligen Software Development Kits.

5.4.1 Programmmodi

Die Software des Clients muss über zwei Betriebsmodi verfügen, die der Offline- und der Online-Phase der KNN-Implementierung gerecht werden:

5.4.1.1 Capture Mode

In diesem Modus führt der Client Messungen der Referenzpunkte durch und sendet diese samt einer vom Benutzer ausgewählten Position im Raum an den Lokalisierungsdienst. Jede Messung entspricht somit einem Trainingsmuster.

Als Vorbedingungen für den Betrieb im Capture Mode gelten:

1. Der Raum A muss bekannt sein

   Die Clientanwendung erhält durch eine Anfrage an den Lokalisierungsdienst die Liste der verfügbaren Räume (Aufruf LS1), sodass der Anwender schließlich in einer Auswahl den zutreffenden Raum selektieren kann.

2. Der Aufenthaltsort $\vec{l}_{real}$ im Raum muss bekannt sein

   Der Anwender muss angeben, wo im Raum er sich befindet. Als Orientierungshilfe kann sich der Anwender eine Liste von wohl-bekannten Orten (Koordinaten) abrufen (Aufruf LS3).

3. Die für den Raum benötigten Referenzpunkte müssen messbar sein

   Damit ein Trainingsmuster gültig ist, muss für alle Elemente des Eingabevektors ein Wert bekannt sein. Die Anwendung muss sicherstellen, dass während einer Messung alle Referenzpunkte erfasst werden. Welche Referenzpunkte dafür benötigt werden, erfährt die Anwendung vom Lokalisierungsdienst (Aufruf LS2).

Nach der Erfassung der Signalpegel aller benötigten Referenzpunkte kann das Ergebnis an den Lokalisierungsdienst übermittelt werden (Aufruf LS4). Die gesammelten Datensätze können dann in einer Offline-Phase vom Lokalisierungsdienst zum Training eines KNN herangezogen werden.

5.4.1.2 Location Mode

Der Location Mode dient der Abfrage des Lokalisierungsdienstes in einer Online-Phase. Der Client misst alle Referenzpunkte, die gerade in Empfangsreichweite liegen. Um die Zuverlässigkeit der Messung zu erhöhen sollen mehrere Messungen erfolgen und die Ergebnisse kumuliert werden. Liegen für einen Referenzpunkt mehrere Messungen vor, was wahrscheinlich ist, soll der Messwert über den Median gemittelt werden. Hiervon darf man sich erhoffen, dass gröbste Schwankungen der Messwerte ausgefiltert werden.

Der Client sendet eine Lokalisierungsanfrage (Aufruf LS5) an den Dienst und wartet auf die Antwort. In der Antwort sind der erkannte Raum A und der vermutete Aufenthaltsort $\vec{l} \in A$ enthalten.

6 Realisierung

6.1 Umsetzung des Designs

Das in dieser Bachelorarbeit entwickelte System soll in der Lage sein, mit Hilfe neuronaler Netze die Positionen eines mobilen Endgerätes zu ermitteln. Es wird auf der Verwendung von RSSI Signalpegelwerten basieren, mit denen ein Neuronales Netz angelernt wird um die Richtcharakteristik der Indoor Umgebung zu erlernen. Im Optimalfall soll das System in der Lage sein, Strukturveränderungen sowie Störungen im Raum zu erkennen und auszugleichen.

Ein solches System lässt sich in vier Komponenten einteilen.

1. Eine eigene Klassenbiblitothek für neuronale Netze, der Neuronale Kernel.
2. Einen Lokalisierungsdienst, welcher über mehrere Schnittstellen Positionsanfragen entgegen nimmt und mit Hilfe eines neuronalen Netzes beantwortet. Des weiteren ermöglicht der Dienst auch, Trainingsdaten zum Trainieren des neuronalen Netzes zu sammeln. Er wurde sowohl als Webservice als auch als Socket Protokoll implementiert.
3. Mobile Clients (Siehe Abbildung 50) welche die Aufgabe übernehmen, Messdaten von verschiedenen Positionen zu ermitteln, sowie später auch Positionsanfragen an den Lokalisierungsdienst zu stellen.
4. Eine Administrationsoberfläche, in welcher man Neuronale Netze mit Hilfe der gesammelten Messdaten erstellen, trainieren und versionieren kann.

Als Referenzpunkte dienen dem System handelsübliche Accesspoints. Die Accesspoints des Systems müssen weder räumlich vermessen noch in irgendeiner Weise miteinander gekoppelt werden. Bei der Implementierung wurde die Programmiersprache Java verwendet.

6.1.1 Neuronaler Kernel

Der neuronale Kernel wurde als Java SE-Klassenbibliothek realisiert. Im Vordergrund stehen die Implementationen von MLP und RBF und den respektiven Lernalgorithmen. Zusätzlich zu den gegebenen Anforderungen wurde im Sinne der Vereinfachung der Analyse noch ein Fehlergraph realisiert, der Lernfortschritt visualisieren kann.

6.1.1.1 Multi Layer Perceptron

Das Multi Layer Perceptron ist als Sammlung von Schichten von Neuronen realisiert. Jedes Neuron (Klasse `Perceptron`) besteht aus einem Array von Eingabewerten, einem Propagierungswert (`netpj`) und einer Aktivierungsfunktion, die auf den Propagierungswert angewendet wird. Als Aktivierungsfunktion stehen Linear, Fermi (`Sigmoid`) und Tangens Hyperbolicus zur Auswahl.

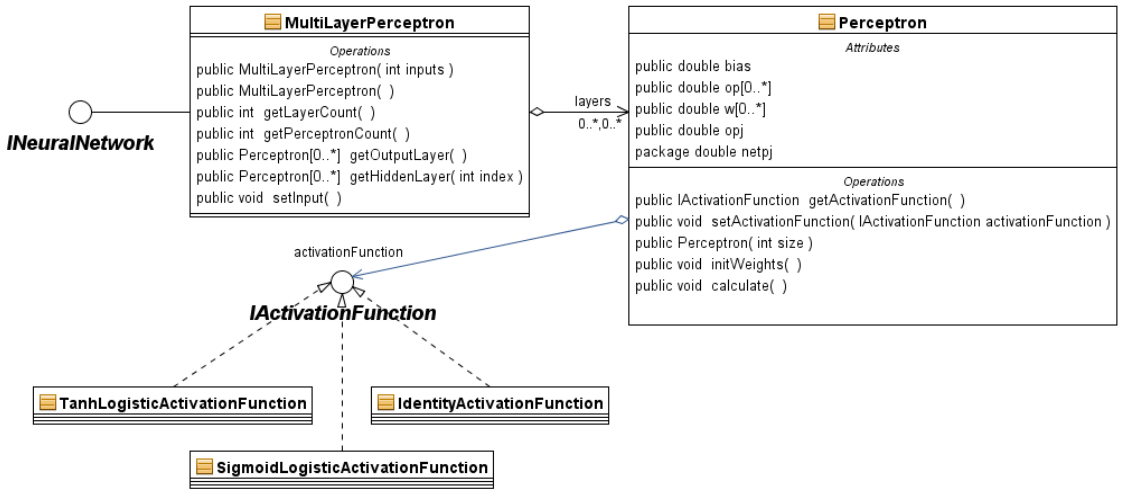


Abbildung 43: Klassendiagramm der MLP-Implementation

Die Instanziierung eines MLP erfolgt mit Hilfe einer `LayerDefinition`-Hilfsklasse. Eine `LayerDefinition` sagt aus, wie viele Neuronen in einer Schicht existieren sollen und welche Aktivierungsfunktion diese aufweisen sollen. Als Konstruktor-Argumente werden die Größe des Eingabevektors, gefolgt von beliebig vielen `LayerDefinition`-Instanzen erwartet. Die letzte `LayerDefinition`-Instanz beschreibt dabei die Ausgabeschicht. Daher muss auch mindestens eine `LayerDefinition` übergeben werden.

```
MultiLayerPerceptron network = new MultiLayerPerceptron(
  3,
  new LayerDefinition(11, new SigmoidLogisticActivationFunction()),
  new LayerDefinition(11, new SigmoidLogisticActivationFunction()),
  new LayerDefinition(2, new SigmoidLogisticActivationFunction())
);
```

Code 2: Instanziierungsaufruf eines 3-11-11-2-MLP mit Fermi-Aktivierungsfunktion

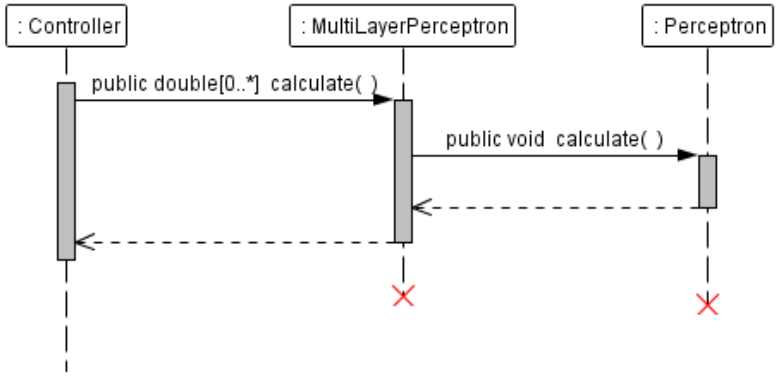


Abbildung 44: Kaskadierter Aufruf von `calculate`

Der Aufruf `calculate()`, welcher die Berechnung der Netzfunktion für den aktuell anliegenden Eingabevektor veranlasst, ruft eine schichtweise Verarbeitung der Eingabewerte in jedem jeweiligen Neuron hervor.

6.1.1.2 Backpropagation-Momentum-Trainingsalgorithmus

Als Lernalgorithmus für das MLP wurde Backpropagation-Momentum realisiert.

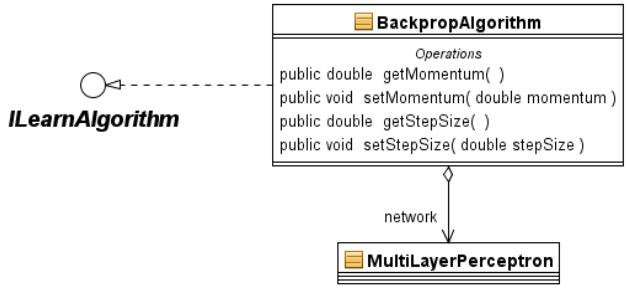


Abbildung 45: Klassendiagramm der Backpropagation-Implementation

Die Implementation kann sowohl inkrementell als auch vollständig trainieren. Die Schrittweite der Backpropagation kann über den Parameter `stepSize` eingestellt werden, der Momentum-Faktor über den Parameter `momentum`.

Die Parametrisierung hat einen erheblichen Einfluss auf die spätere Funktion des

trainierten Netzes. Während die Instanz des Trainingsalgorithmus längst wieder zerstört wurde, spiegelt sich der Erfolg des Trainings in den Gewichten der Neurone des MLP wieder.

Das MLP ist somit auch ein recht universell einsetzbares Netz, dessen Verhalten maßgeblich durch die Implementation des Trainingsalgorithmus beeinflusst wird.

6.1.1.3 Radiale Basisfunktionen

Radiale Basisfunktionen wurden ebenfalls nach Anforderung implementiert. RBF wurden self-contained implementiert, es gibt keine weiteren Entitäten, wie das `Perceptron` beim MLP, die für die Implementierung eine tragende Rolle spielen. Für die Berechnungen wird das JavaMatrix-Package verwendet [46]. Das System ist daher recht übersichtlich:

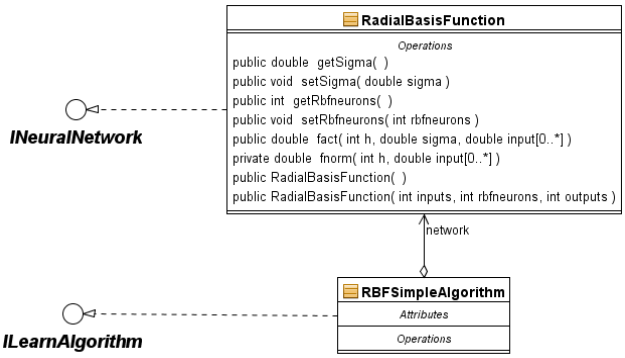


Abbildung 46: Klassendiagramm der RBF-Implementation

Im Gegensatz zum MLP, wo der Lernalgorithmus für das spätere Verhalten des Netzes ausschlaggebend ist, stecken die Parameter beim RBF im Netz selbst. Der einzig einstellbare Parameter zur Zeit ist `sigma`. Wie in Kapitel 2.3.4.4 erläutert, kann über diesen Parameter die Breite der Interpolationsfunktionen festgelegt werden. Da dieser Wert bei jeder Ausführung der Netzfunktion und nicht nur während des Trainings benötigt wird, enthält das Netz selbst den Parameter. Der Lernalgorithmus holt sich sigma dann vom jeweiligen Netz, mit dem er initialisiert wird.

Die Initialisierung eines RBF-Netzes gestaltet sich noch einfacher als beim MLP. Da RBF stets nur eine verdeckte Schicht haben, reichen als Argumente schlichtweg die Kardinalitäten von Eingabe-, verdeckter- und Ausgabeschicht.

```
RadialBasisFunction net = new RadialBasisFunction(3,40,2);
```
Code 3: Instanziierungsaufruf eines 3-40-2-RBF

6.1.1.4 RBF-Lernalgorithmus

Im Gegensatz zu Backpropagation unterstützt das Training der RBF kein inkrementelles Training, das heißt, der Algorithmus muss mit allen Trainingsmustern auf einmal parametrisiert werden, damit die entsprechenden Matrizen berechnet werden können.

Dafür ist die vorliegende Implementation über die Anforderung hinaus dazu in der Lage, auch mit mehr Trainingsdaten als RBF-Neuronen zu trainieren. Dies wurde erreicht, indem M^{-1} über die Moore-Penrose-Pseudoinverse bestimmt wird, sobald $|H| \neq |T|$ gilt. Die Deltaregel wurde nicht implementiert.

6.1.2 Lokalisierungsdienst

Die Funktionen des Lokalisierungsdienstes wurden mit zwei verschiedenen Technologien realisiert, um den Dienst für weitergehende Versuche mit verschiedenen Endgeräten nutzbar zu machen.

6.1.2.1 Webservice

Da ein Webservice lediglich die Funktionalität bietet, primitive Datentypen zu übertragen, findet bei der Abfrage von Informationen ein zweistufiger Aufruf statt, bei dem im ersten Schritt die id der gewünschten Objekte angefragt wird und durch einen weiteren Aufruf die zusätzlichen Informationen anhand der bekannten id abgerufen werden. Der Webservice wird über SOAP auf Basis von Apache JAX-WS bereitgestellt.

Protokolldefinition

```
public interface ILocationService {

    /** LS4).2
     * Adds a seen Bssid with it's measured RSSI value to a measurement.
     * A valid measurement handle (measurementId) has to be provided.
     */
    void addMeasurePoint(int measurementId, String bssid,int rssi);

    /** LS4).1
     * Requests a new measurement handle for adding location measurements.
     */
    int addMeasurement(String deviceName, int areaId,double posx,
```

```
        double posy);

    /** LS2)
     * Gets all Bssids that are associated to this area. The order of
     * the returend list is also the order of Bsids with their
     * respective RSSI values that are expected in the locate and
     * measure operations.
     */
    String[] getAreaBssids(int areaId);

    /** LS1).3
     * Returns an array of integers of the length of 2.
     * The width of the area is stored at index 0, the height at index 1
     * respectively.
     */
    int[] getAreaDimensions(int areaId);

    /** LS1).1
     * Gets the ids of all available areas.
     */
    int[] getAreaIds();

    /** LS1).2
     * Gets the name of a certain area.
     */
    String getAreaName(int areaId);

    /** LS5)
     * Gets a location estimate based on the provided list of visible
     * Bssids ans their corresponding RSSI values plus the
     * id of the corresponding area (cast to int please)
     */
    double[] getLocation(String[] bssids, int[] rssis);

    /** LS3).2
     * Returns an array of integers of the length of two that
     * contains the
     * X and the Y component of a well known location.
     */
    double[] getLocationCoordinates(int locationId);

    /** LS3).1
     * Gets a list of all ids of well known locations of a certain area.
     */
    int[] getLocationIds(int areaId);

}
```

Code 4: Interface-Definition des Webservices

Neben dem universell einsetzbaren Webservice wurde ein Socket-Protokoll für langsame Übertragungswege und Endgeräte mit geringerer Leistungsfähigkeit entwickelt, welches weniger Overhead als das mächtige SOAP-Protokoll erzeugen sollte, was sich in der Praxis auch bestätigte.

6.1.2.2 Socket-Protokoll

Das Socket-Protokoll stellt den selben Funktionsumfang wie den des Webservices zur Verfügung. Die Daten werden lediglich direkt über einen TCP-Socket in Form eines kompakten, für diese Anwendung ausgelegten Protokolls ausgetauscht. Das Protokoll sieht hierbei vor, dass das mobile Endgerät parametrisierte Anfragen stellen kann, welche eine direkte Antwort des Servers zur Folge haben.

Für jede Anfrage wird ein neuer Kanal geöffnet und nach der Übermittlung gleich wieder geschlossen. Jede Anfrage ist mit einem Line-break zu terminieren.

Protokolldefinition

Aktion	**Anfrage**	**Antwort**
LS1) Alle Räume holen	AREAS;\n	AREAS; (**'areaid'**,**'areaname'**,**'areawidth'**,**'areaheight'**;)*\n
LS2) Referenzpunkte eines Raums holen	BSSIDOFAREA;**'areaid'**;\n	BSSIDOFAREA; (**'bssid'**,**'name'**;)*\n
LS4) Messung hinzufügen	CAPTURE;**'areaid'**,**'posx'**,**'posy'**;**'devicename'**;(**'bssid'**,**'rssi'**;)*\n	CAPTURE;\n
LS5) Position ermitteln	LOCATE;(**'bssid'**,'rssi**';)***n	LOCATE;**'areaid'**,**'areaname'**,**'areawidth'**,**'areaheight'**;**'posx'**,**'posy'**;\n
LS3) Wohl bekannte Orte eines Raums holen	LOCATIONSOFAREA;**'areaid'**;\n	LOCATIONSOFAREA; (**'x'**,**'y'**;)*\n
Fehlerbehandlung		ERROR;**'errormessage'**;\n

Tabelle 3: Protokolldefinition des Socket-Protokolls

Datentypen

Variable	**Format**	**Wertebereich**
areaid	Integer	0 bis max
areaname	String	-
areawidth	Integer	1 bis max
areaheight	Integer	1 bis max
bssid	String	-
name	String	-
posx	Double	0 bis **areawidth**

* Beliebig häufige Wiederholung des Bereiches x

Variable	Format	Wertebereich
posy	Double	0 bis **areaheight**
devicename	String	-
rssi	Integer	-100 bis 100
x	Double	0 bis **areawidth**
y	Double	0 bis **areawidth**
errormessage	String	-

Tabelle 4: Datentypen des Socket-Protokolls

6.1.3 Backend

Das Backend stellt die in Abbildung 42 dargestellten Entitäten zur Verfügung. Es hält einzige Instanz eine Datenbankverbindung vor, in welcher Referenzmessungen und trainierte KNN abgelegt werden können.

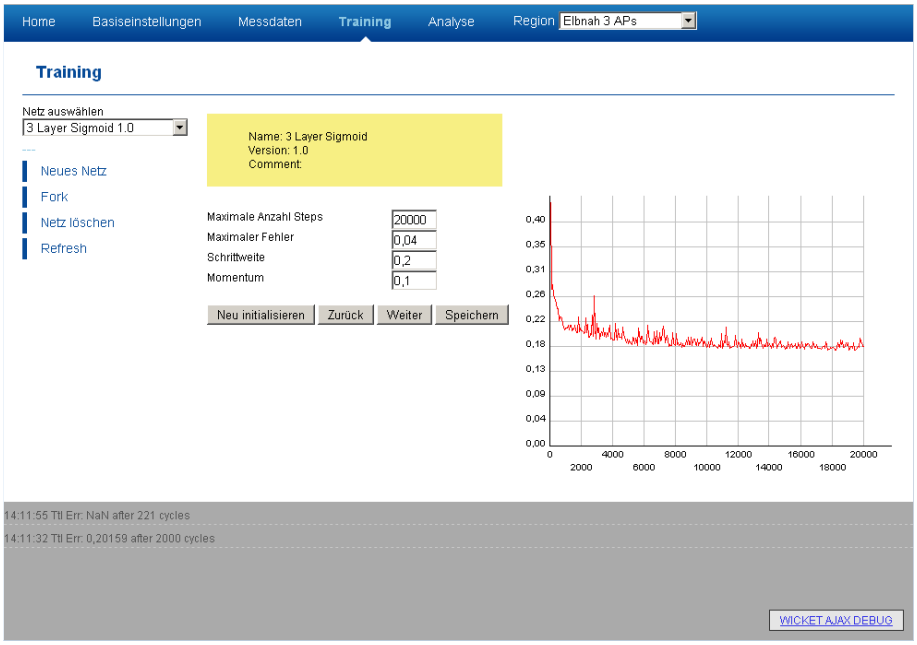


Abbildung 47: Trainingsoberfläche als Website

Das Backend stellt zudem eine Administrationsoberfläche über eine Webseite zur Verfügung. Sie bietet die Möglichkeit, sich einen Überblick über gesammelte Messdaten zu verschaffen, Räume zu verwalten und ein neuronales Netz bestmöglich mit Hilfe eines Fehlergraphen zu trainieren. Sie hat die Aufgabe, den Prozess des Trainings des KNN möglichst flexibel zu halten, um dem Anwender die Möglichkeit zu geben, verschiedene Szenarien und Netztopologien auszuprobieren. So bietet sie die

Möglichkeit, die Trainingsdaten einzuschränken und einzelne Trainingsschritte auch wieder rückgängig zu machen.

6.1.3.1 Verwendete Technologien

Da sich dazu entschieden wurde, das Backend über eine Internetseite bedienbar zu machen, wurde die gesamte Serverseite als eine Webapplikation entwickelt, welche mit Hilfe eines Java Servlet Containers verwendet werden kann. Hier wurde sich für den Apache Tomcat Application Server [47] entschieden. Als Datenbank wurde sich für die Open Source Softwarelösung MySQL [48] entschieden. Der Zugriff auf die Datenbank auf Code Ebene wurde mit Hilfe des Java Persistence Frameworks hibernate [49] realisiert, welches einen objektorientierten Zugriff auf die Daten aus der Datenbank gewährt. Um Abhängigkeiten zwischen dem Persistenz Layer und der da drüber liegenden Schichten zu vermeiden, wurden zugehörige Data Access Objects angelegt über welche der Zugriff auf die Daten gesteuert wird.

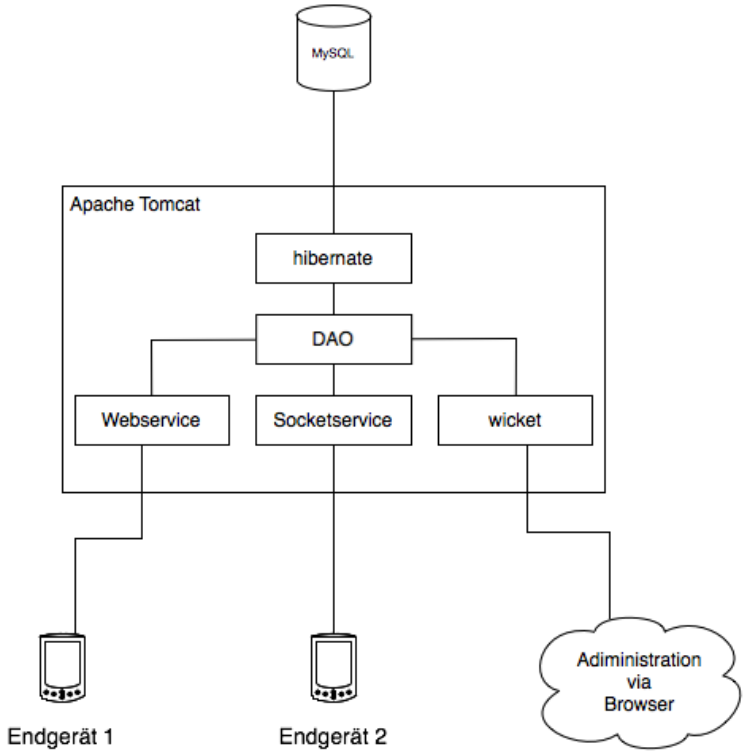


Abbildung 48: Die Aufteilung der einzelnen Komponenten und wie sie ineinander greifen

Seitens der Darstellungsebene für die Administrationsoberfläche im Web wurde das Java Web Application Framework Apache Wicket [50] verwendet. Es ermöglicht eine

weitere Vermeidung von Abhängigkeiten zwischen der Controller- und View-Ebene. In Abbildung 48 wird die Hierarchie der gerade erklärten Technologien noch einmal grafisch verdeutlicht.

6.1.4 Client

Der Lokalisierungsclient wurde auf den von der HAW zur Verfügung gestellten Handhelds vom Typ HP IPAQ 5500 unter Windows Mobile 2003 entwickelt. Dabei wurde das .NET Compact Framework 2.0 von Microsoft und das Smart Device Framework 2.3 von OpenNETCF Consulting [51] zum Ansprechen der WLAN-Hardware eingestzt.

Der Client bietet die Möglichkeit, über das Socket-Protokoll mit dem Backend zu kommunizieren und so Messdaten zu erfassen sowie auch später Lokalisierungsanfragen an das Backend zu stellen.

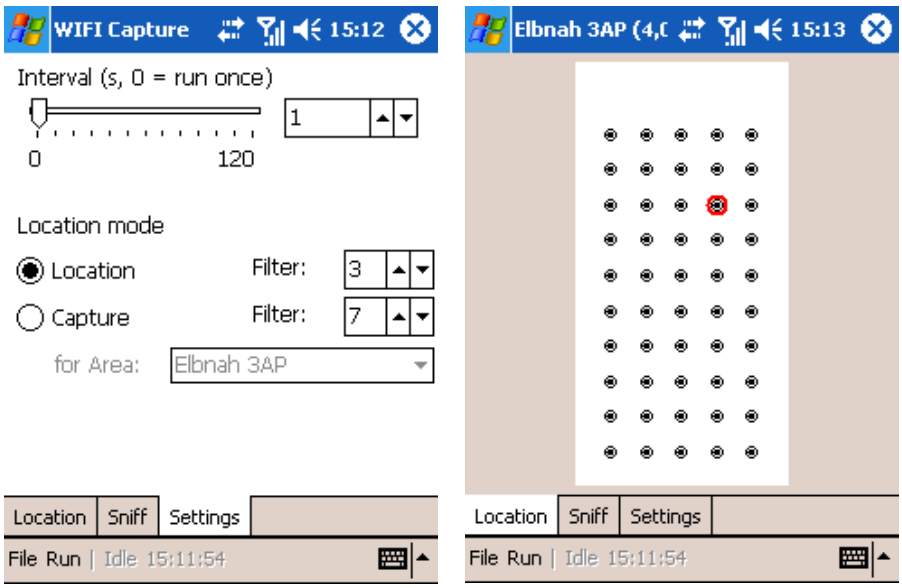


Abbildung 49: Benutzeroberfläche des mobilen Clients

Über einen Intervall-Regler (s. Abbildung 49) steht dem Anwender die Möglichkeit zur Verfügung, ein geeignetes Messintervall zu wählen, in dem die jeweilige Aktion ausgeführt werden soll. Zudem kann über die Option „Filter“ gewählt werden, über wie viele Messungen ein Medianfilter angewandt werden soll, um die Schwankungen der RSSI-Messwerte etwas zu glätten. Über die Optionen des „Location Mode“ kann eingestellt werden, ob sich der Client im Capture-Mode oder im Location-Mode befinden soll (s. Kapitel 5.4.1). Für den Capture-Mode ist entsprechend den Design-

Richtlinien die Angabe des Raumes (Area) vorgeschrieben, welcher aus einem Drop-Down ausgewähöt werden kann.

Während der Nutzungshase stellt der Client dem Anwender eine 2D-Draufsicht des Raumes zur Verfügung, in welcher der Anwender je nach Modus entweder seine momentane Position eingezeichnet bekommt, oder mit einem Stylus seinen momentanen Aufenthaltsort für die Aufzeichnung eines Trainingsmusters auswählen kann (s. zweiter Screenshot auf Abbildung 49). Zur besseren Orientierung während des Capture-Mode werden dem Anwender auch die wohlbekannten Orte im Raum angezeigt (schwarze Kreise).

Im Capture-Mode wird die gewählte Position des Anwenders (roter Kreis) über Touch-Eingabe so dem System bekannt gemacht und samt der gemessenen RSSI-Signalwerte über den Lokalisierungsdienst an das Backend übertragen. Der Client achtet hierbei darauf, dass auch für jeden benötigten Referenzpunkt des Raums RSSI-Werte erfasst werden. Ist das nicht der Fall, wird das Trainingsmuster verworfen und ein akustisches Fehlersignal ausgegeben. Im Location-Mode gilt diese Abfrage natürlich nicht, da der Anwender ja nicht wissen muss, in welchem Raum er sich befindet (Mutli-Area-Konzept).

Nach einer ersten Testphase wurde leider deutlich, dass sich die on-board WLAN-Karte des Handhelds als für die Anwendung ungeeignet erwies. Die Treiber der Karte stellten den RSSI-Signalpegel sehr grob in nur drei Schritten (33, 66, 99) über das gesamte Spektrum dar. Die Deaktivierung der internen WLAN Karte und Verwendung einer externen SD/IO-WLAN Karte „Go Wi-Fi! P300“ vom Hersteller Socket Mobile, inc. [52] (Abbildung 50) konnte hier durch eine wesentliche feinere Granularität bei den

Abbildung 50: SD/IO WLAN-Karte von Socket

RSSI-Werten Abhilfe schaffen.

6.2 Verwendung

6.2.1 Offline-Phase (Training)

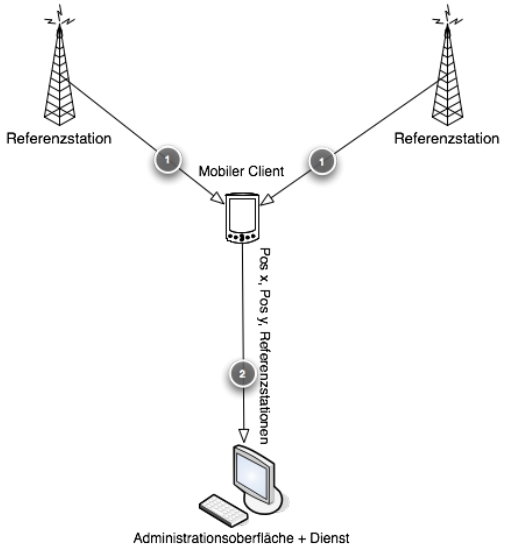


Abbildung 51: Anlernphase: So werden Referenzmessungen gesammelt

Der erste Schritt der Anlernphase ist das Sammeln von Referenzmessungen für den neu zu lernenden Raum. Hierfür wird der mobile Client eingesetzt.

Er übermittelt die vom Benutzer gewählte Messposition mitsamt der an diesem Punkt erfassten Referenzstationen über den Dienst an die Administrationsoberfläche.

Während der Anlernphase gibt es die Möglichkeit die Verteilung der bereits getätigten Referenzmessungen live an der Administrationsoberfläche zu verfolgen. Diese Funktion wird LiveView genannt (Siehe Abbildung 52).

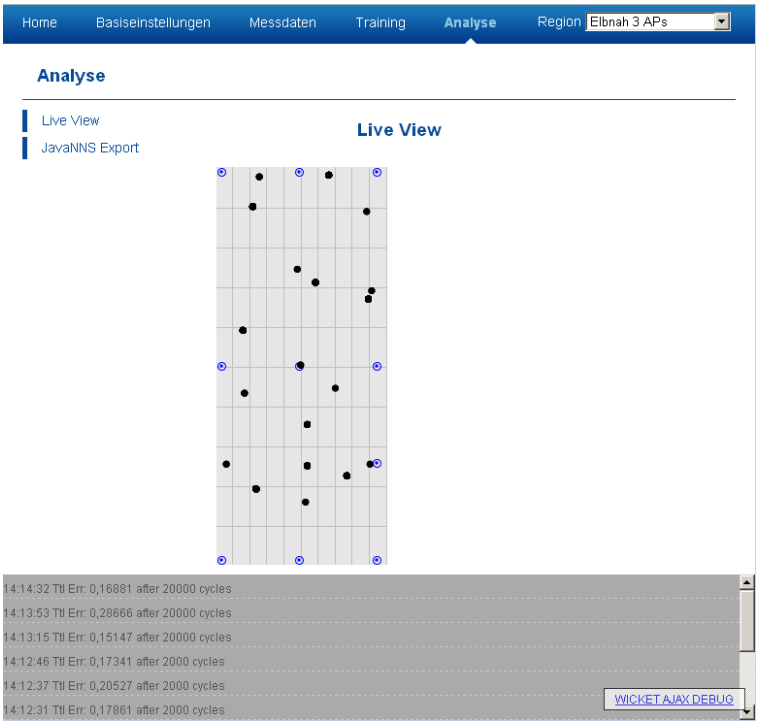


Abbildung 52: Administrationsoberfläche: Die LiveView Oberfläche gibt Auskunft über die Verteilung der Referenzmessungen im Raum

Sind für den Raum genügend Referenzmessungen gesammelt worden, wird in einem weiteren Schritt das KNN mithilfe der gesammelten Daten trainiert (Abbildung 47). Gilt ein Netz als fertig trainiert, besteht die Möglichkeit, den Zustand in die Datenbank zu speichern und dem jeweiligen Raum als Standard („Default-Netzwerk“) zuzuordnen (Abbildung 53).

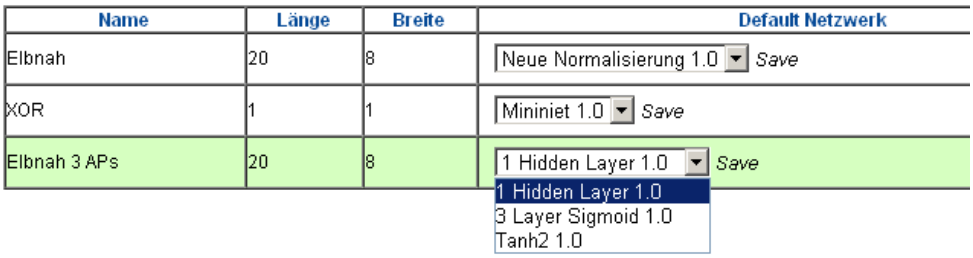


Abbildung 53: Zuordnung eines Default-Netzwerks für einen Raum

Der Anwender hat somit sogar die Möglichkeit, im laufenden Betrieb das Default-Netzwerk für den jeweilige Raum dynamisch zu wechseln, da es vom Lokalisierungsdienst bei jeder Client-Anfrage neu von der Persistence-Unit angefordert wird.

6.2.2 Online-Phase (Nutzung)

Die Benutzungsphase sieht folgenden Ablauf vor: Möchte ein mobiler Client Informationen zu seiner momentanen Position erhalten, kann er diese durch eine Anfrage über den Lokalisierungsdienst erhalten. Die Informationen die der Client dafür zu übermitteln hat, sind die am momentanen Standort verfügbaren Referenzstationen mit den jeweils zugehörigen RSSI-Werten. Seitens des Lokalisierungsdienstes wird danach das für den momentanen Raum eingestellte Default-Netzwerk zur Positionsermittlung herangezogen.

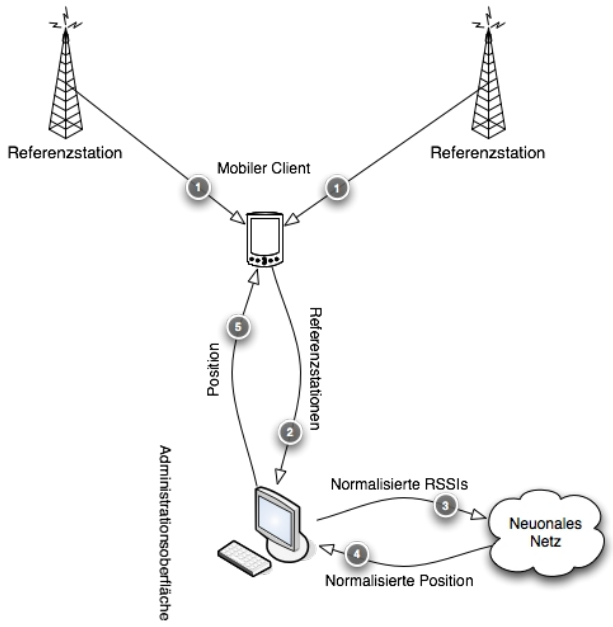


Abbildung 54: Benutzungsphase: Das Neuronale Netz ermittelt auf Anfrage die Position eines Clients

Erklärung zu Abbildung 54:

1. Mobiler Client ermittelt verfügbare Referenzstationen mit RSSIs.
2. Mobiler Client übermittelt Messungen aus 1. an Lokalisierungsdienst.
3. RSSIs der Referenzstationen werden normalisiert und an das Neuronale Netz übergeben.
4. Das Neuronale Netz liefert nach erfolgreicher Berechnung die ermittelte Position in normalisierter Form zurück.
5. Der Lokalisierungsdienst de-normalisiert die vom Neuronalen Netz ermittelte Position und übermittelt sie an den mobilen Client.

7 Evaluation

Das selbst entwickelte Lokalisierungssystem stellt in Hinsicht der Konfigurationsmöglichkeiten des Aufbaus eine Vielzahl an Funktionen zur Verfügung. So bietet es die Funktionalität, die Anzahl und Auswahl der Referenzstationen für jeden Raum individuell zu bestimmen sowie auch die Möglichkeit, neuronale Netze verschiedener Ausprägungen für jeden Raum anzulegen und zu erweitern. Das System benötigt keine Kenntnis über die jeweilige genaue Position einer Referenzstation.

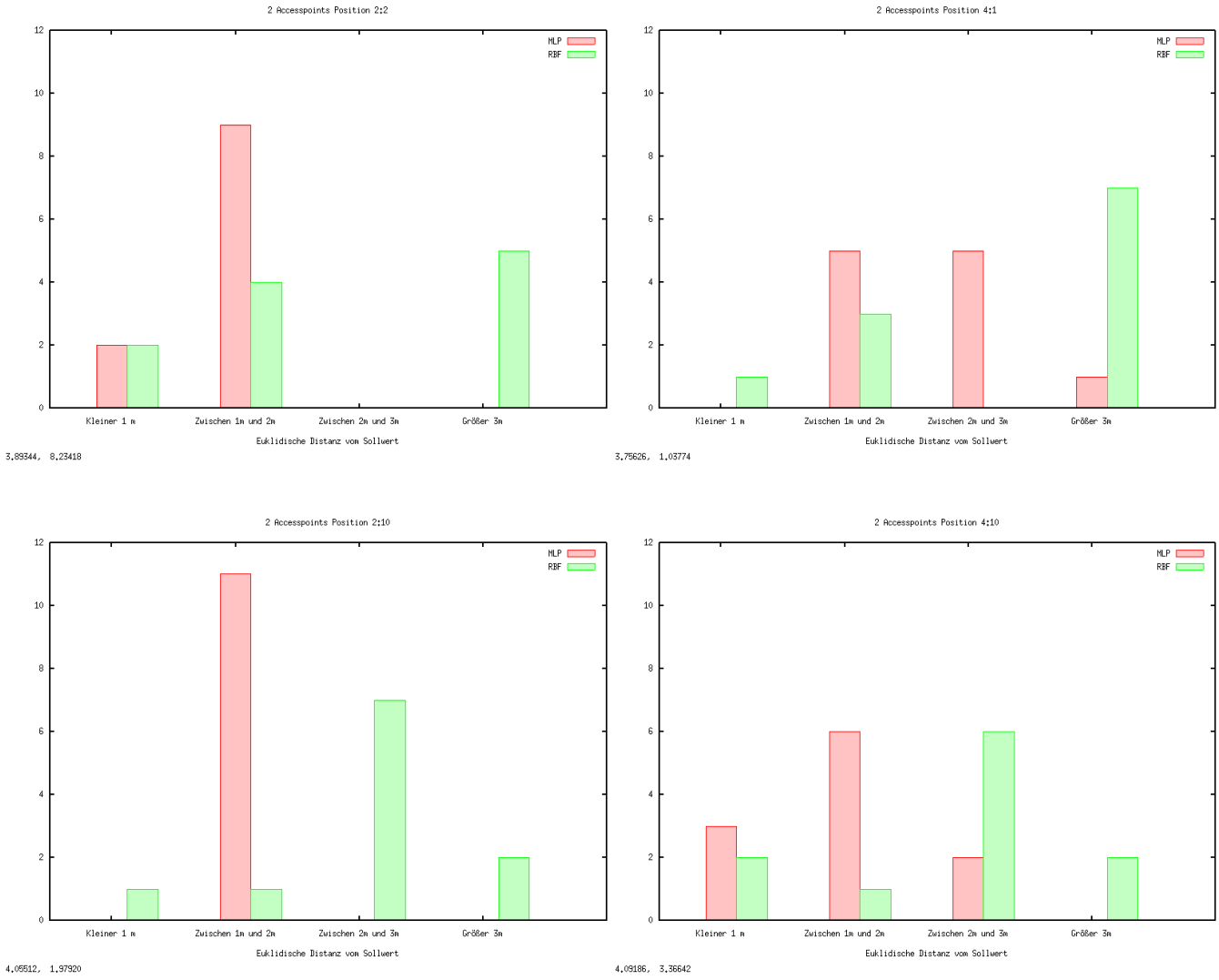


Abbildung 55: Leistungsvergleich von MLP und RBF als Histogramme

Um das System anhand seiner Genauigkeit und Zuverlässigkeit bewerten zu können, wurde entschieden, mehrere Testmessungen an verschiedenen Punkten in der Testumgebung zu tätigen, um anhand der gesammelten Daten zu einer

abschließenden Bewertung des Systems zu gelangen. Hierzu wurde die an den Testpunkten ermittelte Position mit der tatsächlichen Position in Relation gesetzt. Als Genauigkeitskriterium wurde die euklidische Distanz von der ermittelten zur tatsächlichen Position herangezogen.

Die erste und wichtigste Frage war die Wahl des richtigen KNN. Hierfür musste ein Testaufbau geschaffen werden, in dem es möglich war die Ergebnisse eines MLP direkt mit denen eines RBF KNN zu vergleichen. Die Ergebnisse dieser Messung sind in Abbildung 55 zu finden. Wie der Histogrammvergleich verdeutlicht, ist es zwar auch möglich mit einem RBF sehr hohe Genauigkeiten zu ermitteln. Die Stetigkeit der Ortungsergebnisse mit RBF sind jedoch deutlich schlechter als die mit MLP. Während sich die Ergebnisse einer Ortung mittels MLP meist in einem Bereich bündeln, bilden die Ergebnisse der RBF Ortung eine Verteilung über das gesamte Histogrammspektrum. Diese Ergebnisse stimmen mit den theoretischen Überlegungen aus Kapitel 2.3.4 überein. Um die Schwankungen für folgende Feldversuche zu minimieren, wurden die folgenden Untersuchungen lediglich nur noch mit Hilfe von MLP realisiert.

7.1 Bewertung der Leistungsfähigkeit

Ziel dieser Untersuchung war es, in einer praktischen Herangehensweise eine Vorschrift aufzustellen, nach der anhand der abzubildenden Funktion eine minimale Netzgröße bestimmt werden kann. Wählt man das Netz etwa zu groß, benötigt es eines teils deutlich höheren Rechenaufwandes während der Trainings- sowie auch der Produktivphase. Es wurden insgesamt sechs verschiedene Netzgrößen miteinander verglichen, welche sich in der Anzahl an Hidden-Layern sowie in der Anzahl an Neuronen pro Layer jeweils voneinander unterschieden. Während der Untersuchung wurde sich ausschließlich auf die Verwendung von MLP beschränkt. Das Ergebnis dieser Gegenüberstellung an verschieden großen KNN ist in Abbildung 56 zu sehen. Sie stellt die Lerngraphen mit dem Gesamtfehler auf der y-Achse und der Anzahl an Trainingsschritten auf der x Achse dar. Es wird deutlich das alle Netzkonfigurationen nahezu den selben Fehler von ca. 16% erreichen. Vergleicht man allerdings die Anzahl an Trainingsschritten zu dem Zeitpunkt wo der Netzfehler jeweils ca 16% beträgt, wird ersichtlich, dass ein größeres KNN erheblich mehr Lernschritte benötigt um die selbe Funktion im gleichen Maße zu approximieren.

Bei weiterer Analyse der Lerngraphen aus Abbildung 56 fällt auf, dass die größeren KNN erheblich stärker anfangen in den Ergebniswerten zu oszillieren. Wie in [53] beschrieben, ist der Grund für dieses Verhalten auf eine ungünstige Wahl der Netzparameter zurückzuführen, welche dem MLP den Gradientenabstieg im Fehlergebirge erschweren.

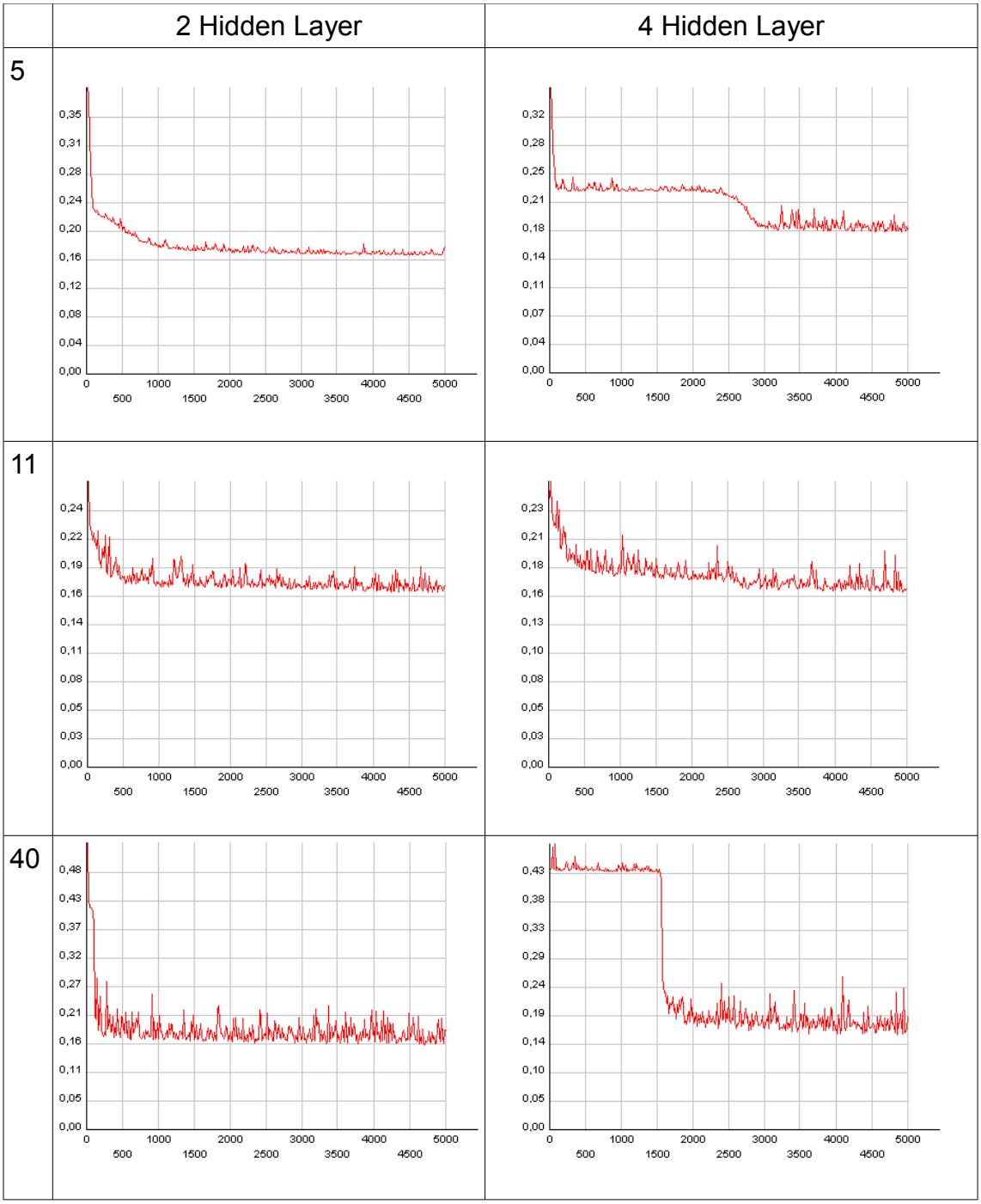
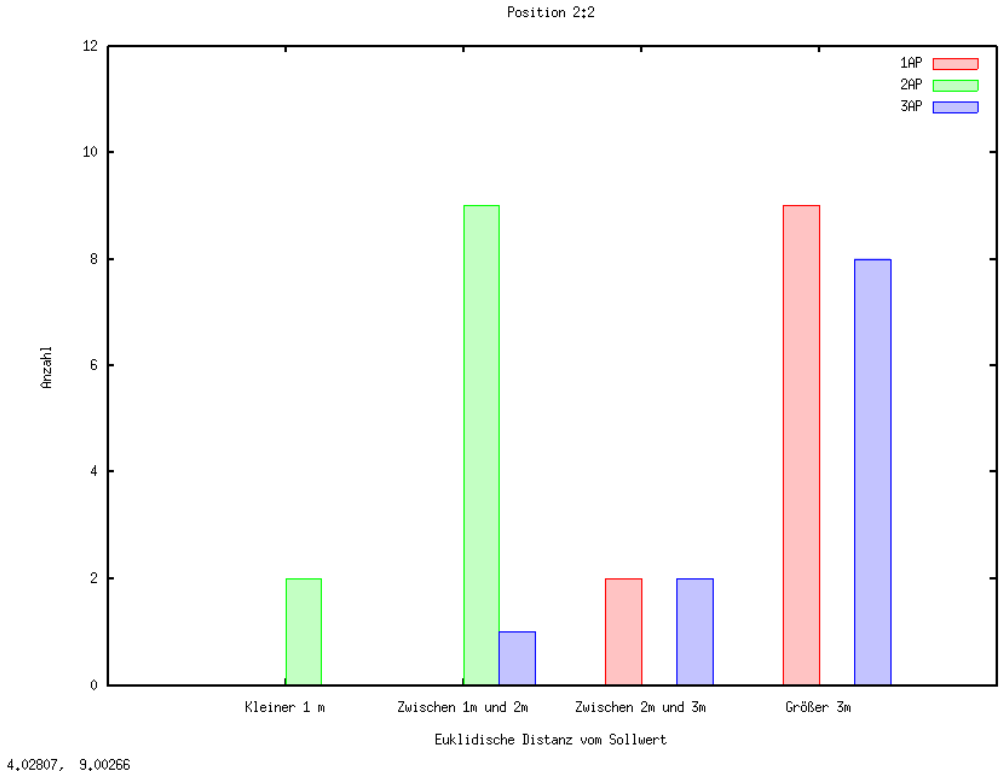


Abbildung 56: Lernverhalten verschieden dimensionierter MLP

Hierbei spielt vornehmlich die vermutlich zu groß gewählte Schrittweite eine entscheidende Rolle wodurch das KNN über die Minima im Fehlergebirge hinweg schreitet. Dieses Verhalten kann beim direkten Vergleich dieses Aufbaus leider nicht verhindert werden, da eine Änderung der Netzparameter die Vergleichbarkeit der verschieden großen MLPs beeinträchtigen würde. Es wird allerdings deutlich, dass die Netzparameter für ein kleines MLP keineswegs die selben wie die eines größer gewählten MLP sind.

In einem weiteren Versuch wurde sich damit beschäftigt, die Frage nach der benötigten Anzahl an Referenzstationen zu klären mit der die höchste Präzision zu erwarten ist. Hierbei sollte insbesondere geklärt werden, wie sich die Anzahl an Referenzstationen auf die Genauigkeit des Systems auswirkt. Für den Versuch wurden mehrere MLP mit unterschiedlicher Anzahl an Referenzstationen angelegt und nach dem selben Schema mit identischen Parametern angelernt. Für die so erstellten KNN wurden nun erneut Testmessungen an identischen Positionen vorgenommen und die Ergebnisse wieder mittels der euklidischen Distanz zum Sollwert als Fehler-Histogramm visualisiert. Die Abbildungen 57 und 58 stellen diese Histogramme für die verschiedenen Positionen in der Testumgebung dar.



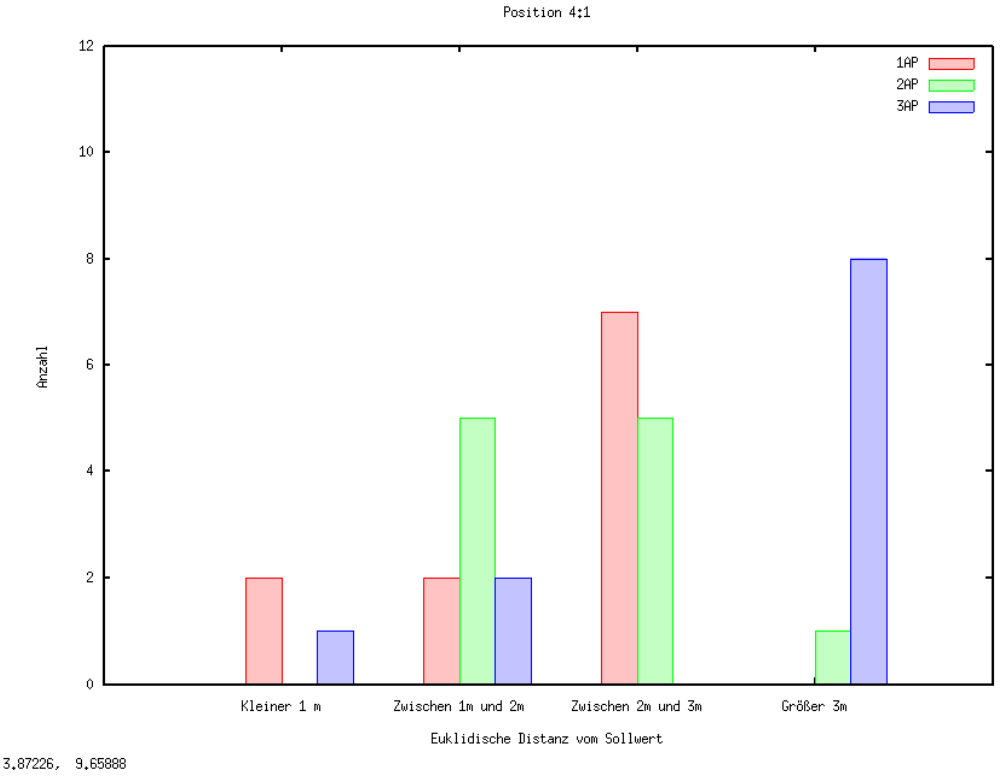
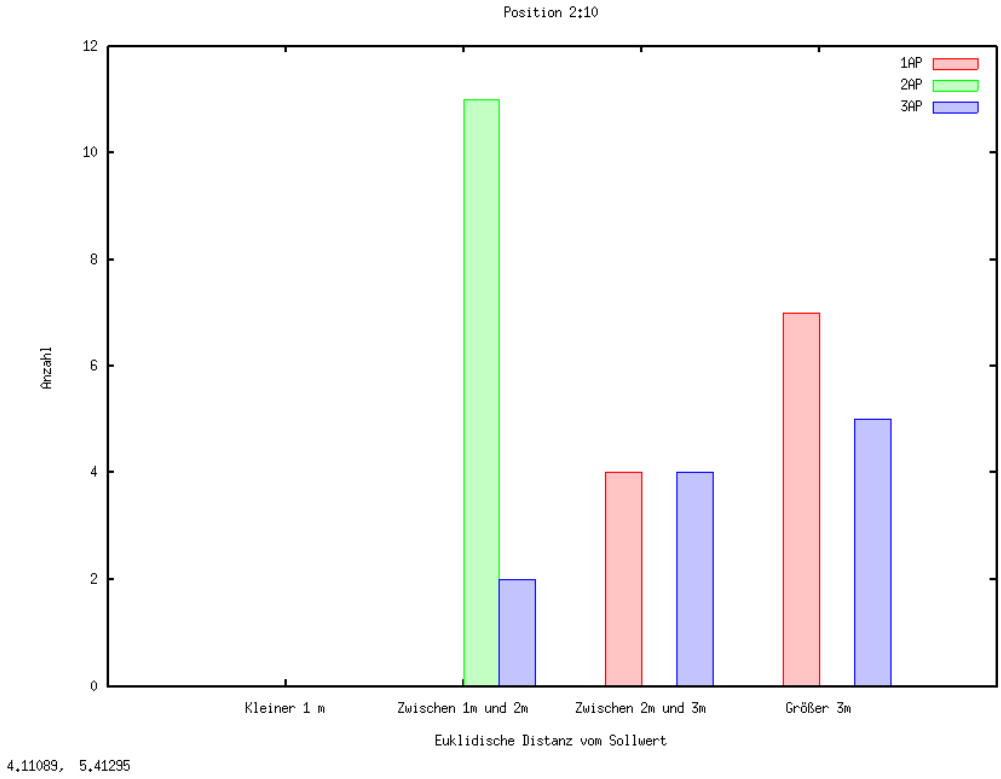


Abbildung 57: Leistungsvergleich von verschiedenen Umgebungen als Histogramme 1



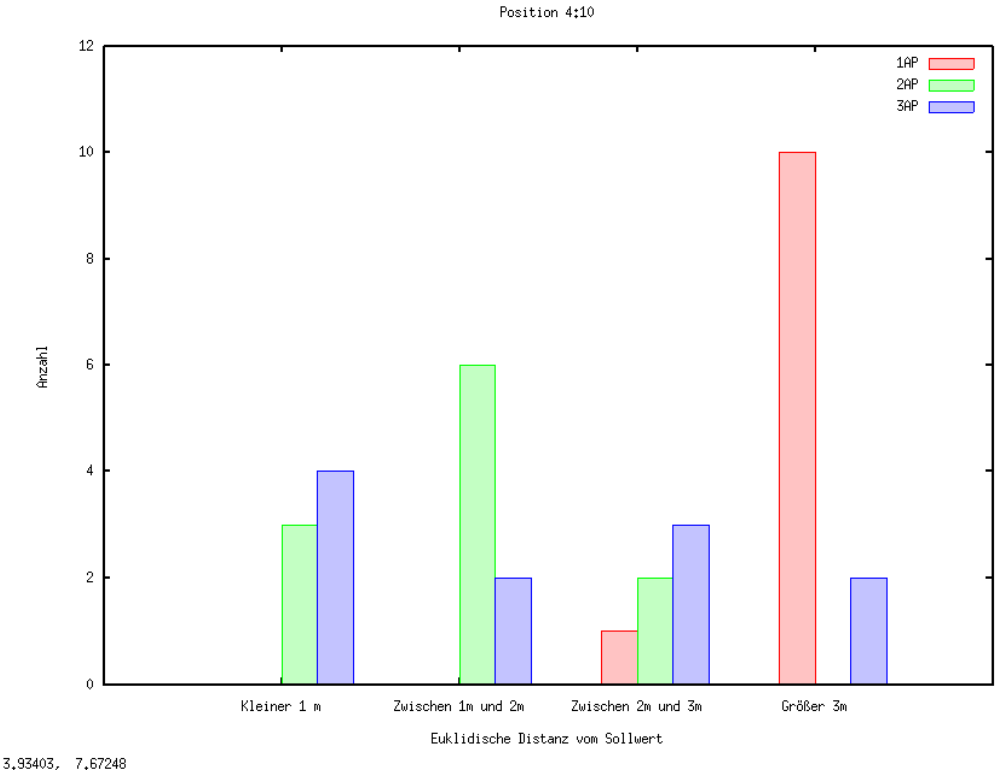


Abbildung 58: Leistungsvergleich von verschiedenen Umgebungen als Histogramme 2

Wie die Histogramme zeigen, hängt die Anzahl der verwendeten Referenzstationen keineswegs direkt mit der Genauigkeit des Gesamtsystems zusammen. Betrachtet man die Histogramme aller vier Referenzstationen, zeichnet sich sehr deutlich ab, dass die Ergebnisse mit zwei Referenzstationen (grün) sehr viel häufiger zu einem präziseren Ergebnis führen als die Ergebnisse einer Ortung über drei Referenzstationen (lila). Sogar die Ergebnisse einer Ortung mit nur einer Referenzstation stehen zur Ortung mit drei Referenzstationen in direkter Konkurrenz.

Die Wahl der richtigen Anzahl an Referenzstationen scheint also nicht abhängig von der geforderten Präzision zu sein, sondern viel mehr von den Eigenschaften der jeweiligen Umgebung. Die richtige Dimensionierung der Anzahl an Referenzstationen ist folglich ein essentieller Bestandteil der Anlernphase eines solchen Systems. Die für die Testumgebung am geeignetste Anzahl war die mittels zwei Referenzstationen.

Bei 44 Testmessungen ergaben sich hier acht Messungen als ungenauer als zwei Meter, sechs an Position 4:1 (Abbildung 57), zwei an Position 4:10 (Abbildung 58). Die Fehlerrate beträgt nach Formel 8 für $d = 2\mathrm{m}$ somit ca. 18,2%. Die jeweilige Leistungsfähigkeit der in Kapitel 3.1 vorgestellten Systeme kann abgesehen von den werbewirksamen Versprechungen auf einem ähnlichen Niveau gesehen werden, mit der Ausnahme von Ubisense, welches genauer sein kann.

System	**Genauigkeit**	**Fehlerrate**
Ubisense	0,15 m [32]	?
Magic Map einfach	5,00 m [35]	?
Magic Map mit intensiven Referenzmessungen	1,00 m [35]	?
Eigene Lösung	2,00 m	18,20%

Tabelle 5: Vergleich der Fehlerrate einzelner Systeme bei gegebener Genauigkeit

Die Übersicht aus Tabelle 5 dient nur einer Orientierung und muss als Aufstellung grundlegender Richtwerte verstanden werden. Die Leistungsdaten der anderen, verglichenen System basieren auf Marketingaussagen, wobei die Genauigkeit nur circa angegeben wird, was annehmen lässt, dass die Fehlerrate größer als Null ist. Zahlen über die Fehlerrate konnten jedoch leider nicht gefunden werden. Unter diesem Hintergrund scheint das vorgestellte Lokalisierungssystem trotz der frühen Entwicklungsstufe jedoch brauchbare Ergebnisse zu liefern.

Des weiteren scheint das vorgestellte System eine kostengünstige Lösung

darzustellen, mit einem vergleichbar geringen Aufwand für die Inbetriebnahme. Betrachtet man die Kosten des Prototypen so ergeben sich beispielsweise für die Konstellation mit zwei Referenzpunkten folgende Materialkosten:

Kategorie	**Produkt**	**Menge**	**Gesamtkosten**
Infrastruktur	Apple Airport Express 1st Gen.[1]	2	180,00 €
Server	Handelsüblicher PC[2]	1	470,00 €
Endgerät	HP iPaq-Handheld[3]	1	370,00 €
Gesamt			**1.020,00 €**

Tabelle 6: Kostenaufstellung des Prototypen (Material)

Geht man davon aus, dass eine WLAN-Infrastruktur und ein geeigneter Server bereits zur Verfügung stehen beschränken sich die Kosten sogar nur auf das Endgerät. Bei zusätzlich gegebenen Endgeräten beschränken sich die Kosten lediglich noch auf Installation, Inbetriebnahme und Wartung, sodass insbesondere Anforderung W2 als erfüllt betrachtet werden darf.

Da das System eine aktive Erweiterung im laufenden Betrieb erlaubt (s. Kapitel 6.2.1), ist es nicht nötig bei eventuellen Änderungen den Betrieb des Systems zu unterbrechen, es kann also im laufenden Betrieb optimiert und erweitert werden. Dank des Einsatzes von KNN ist das Verfahren in der Lage, geringe Schwankungen auszugleichen und Störungen im Raum entsprechend zu berücksichtigen. Betrachtet man den Rechenaufwand der Berechnung der aktuellen Position, stellt sich das System als verhältnismäßig ressourcensparend dar. Zwar bedingt es während der Trainingsphase eines KNN eines deutlich erhöhten Rechenaufwandes, jedoch ist der Aufwand während der späteren Verwendung sehr gering, sodass die Verzögerung durch die Berechnung der Lokalisierungsfunktion nur im ms-Bereich liegt.

Praktische Erhebungen auf einem handelsüblichen PC zeigen, wie in Tabelle 7 dargestellt, dass der Rechenaufwand für eine Lokalisierung so gering ist, dass die Netzgröße nur eine untergeordnete Rolle spielt.

1 http://www.preis.de/produkte/Apple-AirPort-Express-Basisstation/20031.html, (12.07.2009)
2 http://www.alternate.de/html/product/PC-Systeme_Komplettsysteme_Desktop_mit_Betriebssystem/Lenovo/ThinkCentre_A58_Tower_SMK79GE/340214/?tn=BUILDERS&l1=PC-Systeme&l2=Komplettsysteme (12.07.2009)
3 http://www.alternate.de/html/solrSearch/toArticle.html?articleId=259364&query=ipaq&referer=detail&link=solrSearch/listing.productDetails (12.07.2009)

Netztopologie	Typ. Lokalisierung[4]	Typ. Trainingsszyklus[5]
MLP 3-10-10-2	17,50 ms	8.968,00 ms
MLP-3-40-40-2	17,70 ms	60.797,00 ms

Tabelle 7: Verschiedene Latenzen für zwei KNN-Varianten

Die Tatsache, dass das Training für mittlere Netze aber schon mehrere Minuten in Anspruch nehmen kann, belegt, dass die zentrale Ausführung des neuronalen Algorithmus für ein System, in welchen sich die Netzparameter schnell verändern können, richtig war. In der vorliegenden Tabelle wird von 20.000 Trainingsschritten ausgegangen. Es zeigte sich, dass für das erreichen eines akzeptablen Netzfehlers durchaus mehrere hunderttausend Trainingsschritte durchgeführt werden müssen, abhängig von Trainingsparametern und Netztopologie, sodass ein schneller Server für das Training sinnvoll ist. Die Ausführung auf einem mobile Client scheint in diesem Fall nicht sinnvoll.

Die kurze Rechenzeit bei der Lokalisierung eröffnet die Überlegung, die Berechnung clientseitig auszuführen und lediglich die Netzparameter vom Server zu übertragen. Im Sinne der Privatsphäre ist dieser Ansatz sinnvoll, da der Client somit niemandem außer sich selbst seine Position mitteilen müsste. Da das entwickelte System in der Lage ist, KNN im laufenden Betrieb auszutauschen (s. Kapitel 6.2.1), müsste jedoch eine ständige Prüfung auf neue Revisionen stattfinden, was wieder einen größeren Zeitaufwand mit sich brächte. Zusätzlich müsste der Client wegen des Area-Konzeptes nicht nur einen Parametersatz, sondern ggf. eine sehr große Menge von Parametersätzen abfragen. Das Area-Konzept sieht keine künstliche Begrenzung der Anzahl von Räumen vor, sodass im schlimmsten Fall hunderte von KNN-Parametersätzen abgefragt werden müssten, was wieder zu einer erheblich erhöhten Latenz führte, sodass der zentrale Ansatz auch aus Sicht des Lokalisierungsaufrufs sinnvoll erscheint.

4 Dauer eines Lokalisierungsaufrufs abzüglich Kommunikationsaufwand
5 Backpropagation-Momentum 20.000 Schritte mit Schrittweite=0,2, Momentum=0,1

8 Schluss

Dieses abschließende Kapitel fasst die Ergebnisse der Arbeit zusammen und stellt Möglichkeiten für Weiterentwicklung und Ausbau vor.

8.1 Zusammenfassung

Das Ziel dieser Arbeit bestand in der Untersuchung von zwei verschiedenen Varianten von KNN auf die Einsatztauglichkeit in mobilen Indoor-Lokalisierungsszenarien. Nachdem in Kapitel 2.1 die Grundlagen für das Szenario definiert wurden, konnte in 2.3.4 gezeigt werden, dass beide vorgestellten Varianten, MLP und RBF grundsätzlich zur erfolgreichen Lösung des Problems eingesetzt werden könnten. Anhand konkreterer Anforderungen in Kapitel 4 (Analyse) wurde ein Design für eine verteilte Applikation auf Basis von IEEE 802.11g vorgestellt und implementiert (Kapitel 5 bzw. 6), welche eine praktisch orientierte Evaluation des Szenarios mit Standardhardware zulassen sollte, um die Leistungsfähigkeit des Ansatzes mit marktüblichen Verfahren zur Indoor-Lokalisierung vergleichen zu können.

8.2 Funktionsumfang

Das vorgestellte Design der Applikation wurde, wie in Kapitel 5 beschrieben, als leichtgewichtige Client-Server-Architektur umgesetzt. Clientseitig wurde eine Anwendung mit .NET 2.0 realisiert, welche auf einem mit Windows Mobile 2003-basierten PDA läuft und über eine WLAN-Karte die Referenzpunkte (Accesspoints) in der Umgebung erfassen kann. Serverseitig entstand ein Lokalisierungsdienst auf Basis von Java Standard Edition 6, welcher in einem Apache Tomcat 6-Container läuft. Die Anwendung des Lokalisierungsdienstes wurde in einem kompakten, persistierbaren Klassenmodell entworfen und bietet über ein schlankes Netzwerkprotokoll auf Grundlage von TCP Konnektivität zum Client. Die Funktionen des KNN wurden in einer separaten Klassenbibliothek implementiert, welche dann in den Lokalisierungdienst als "Neuronaler Kernel" importiert und integriert wurde.

Die entstandene Anwendung erfüllt alle Anforderungen, um einen Raum anzulegen, diesen über ein mobiles Endgerät zu vermessen und auf Basis der Messdaten, serverseitig ein oder mehrere KNN anzulegen und zu trainieren, um diese als

Lokalisierungsalgorithmus für den Dienst einzusetzen. Das grundlegende Ziel, einen kompakten aber auch vollständigen Lokalisierungsdienst zu schaffen, wie in Kapitel 1.1 (Zielsetzung) definiert, kann somit zunächst als erfüllt betrachtet werden. Einzig die Anforderung W3 wird nicht in Gänze erfüllt. Das Datenmodell und der Capture-Mode des Endgerätes berücksichtigen zwar das Area-Konzept (Kapitel 5.3.2) vollständig, jedoch ist die Erkennung des Raumes während der Lokalisierung noch nicht implementiert. Momentan befindet sich hier ein hart eincodierter Aufruf, der durch eine adäquate Erkennungsmethode ersetzt werden müsste, z.B. auf Basis von RBF oder nach dem Nearest-Neighbour-Verfahren. Erst dann kann der entsprechende Raum auch während der Lokalisierung automatisch erkannt werden kann.

8.3 Fazit

Mithilfe der entwickelten Anwendung konnte die Praxistauglichkeit des Verfahrens auf Basis von KNN in einem Büro-Szenario prinzipiell nachgewiesen werden. Die Ergebnisse der Evaluation in Kapitel 7 zeigten, dass die Leistungsfähigkeit des Ansatzes im Vergleich zu anderen am Markt vorhandenen, freien Systemen, welche nicht auf KNN basieren, auf einem akzeptablen Niveau liegt. Angesichts der recht einfachen Implementierung und einer Fehlerrate von 18,2% bei $g = 2\mathrm{m}$ besteht allerdings noch erhebliches Optimierungspotential.

8.4 Ausblick

8.4.1 Optimierung

Die Implementation der Anwendung stellt mit Backpropagation Momentum für das MLP zwar einen vollständig beschriebenen und bewährten Lernalgorithmus [18], es gibt aber eine Vielzahl von Lernalgorithmen, die eine schnellere Konvergenz gegen einen niedrigeren Gesamtfehler ermöglichen [17]. Bekannte Verfahren sind z.B. Resilient Propagation oder Quickprop [19]. Die Implementation eines solchen Verfahrens nach dem ILearnAlgorithm-Interface könnte die benötigte Zahl an Lernzyklen und somit den Zeitaufwand des Trainings minimieren. Außerdem wurde im vorgestellten Lernalgorithmus nur an den Eingangsgewichten der Neurone manipuliert. Die Größe oder Topologie des Netzes wurde nicht angepasst, die korrekte Parametrisierung obliegt somit dem Administratoren des Netzes. Lernalgorithmen, die in der Lage sind,

die Netztopologie während des Trainings ebenfalls zu verändern, könnten zu wesentlich akurateren oder auch schnelleren Ergebnissen führen, da eine bessere Anpassung des Netzes an die Problemgröße erfolgen kann [18]. Gerade im Lokalisierungs-Szenario, wo die genaue Problemgröße durch zahlreiche Störfaktoren nicht genau benannt werden kann, wäre ein Verfahren wünschenswert, dass seine "Leistungsfähigkeit" der Problemgröße von selbst anpassen kann.

Zusätzlich zum MLP kommen auch eine Fülle von anderen Netzparadigmen zur Lösung des Problems in Frage, wie z.B. das ebenfalls vorgestellte RBF-Netz. Buhagiar und Debono [5] verweisen auf die Leistungsfähigkeit von selbstorganisierenden Verfahren, wie z.B. Self Organizing Maps (SOM), welche sich besser an die stetig wechselnden Bedingungen der Signalpegel durch Fading etc. anpassen können sollen. Am Idaho National Laboratory haben Kurt Derr und Milos Manic LENSR entwickelt, ein junges Lokalisierungssystem, welches 2008 auf der 3rd IEEE Conference on Industrial Electronics and Applications vorgestellt wurde [54]. Das Verfahren basiert und Counter-Propagation, welches auf der Idee von SOM basiert [17]. Das Verfahren konnte aus zeitlichen Gründen leider nicht mehr in dieser Arbeit untersucht werden. Die Ergebnisse von LENSR scheinen die These zu bestätigen, dass selbstorganisierende Verfahren auch im Indoor-Bereich zu ausgezeichneten Ergebnissen führen, mit ca. 15% Fehlerrate bei $g = 1{,}5\,m$ [54].

Eine Information, die in den vorgestellten Verfahren nicht verarbeitet wird, ist die Wegstrecke, die ein Endgerät im Raum über mehrere Messungen hinweg zurücklegt. Diese kann in zustandsfreien Systemen wie MLP, RBF oder auch SOM nicht verarbeitet werden, da diese als zustandsfreie Systeme gelten, also von der Zeit abhängige Funktionen nicht abbilden können. Rückgekoppelte Netzparadigmen, deren Zustände auch immer abhängig von den vorherigen Zuständen sind [17], könnten an dieser Stelle als weiteres Forschungsobjekt gelten.

Neben den zahlreichen Optimierungsmöglichkeiten am neuronalen Kernel besteht auch Optimierungspotential bei der Infrastruktur. Besonders großes Potential kann beim IEEE 802.11g-Transceiver des Endgeräts gesehen werden. Auch wenn die Anforderung (W2) vorgibt, dass Consumer-Hardware zur Verwendung kommen soll, so ergeben sich selbst in diesem Marktsegment bereits zwischen einzelnen Produktserien erhebliche Unterschiede in der Konstanz und Granularität der gemessenen RSSI-Werte. Dieser Zusammenhang lässt sich bereits an den Unterschieden zwischen der

on-board-WLAN-Karte des vorliegenden HP iPAQ-PDA und der zugekauften WLAN-Karte von socket sehen. Bei der on-board-Karte schwankte der RSSI-Wert zwischen drei willkürlichen Werten, sodass eine Lokalisierung fast unmöglich war. Die socket-Karte lieferte bereits wesentlich akuratere Werte (Kapitel 6.1.4). Somit lässt sich festhalten, dass zumindest für die Sammlung der Trainingsmuster auf möglichst hochwertige WLAN-Karten zurückgegriffen werden sollte.

8.4.2 Symbolische Lokalisierung

Ein weiterer Ansatz, der im Rahmen dieser Arbeit leider nicht mehr untersucht werden konnte betrifft das Konzept der symbolischen Lokalisierung. Die Anforderung der physikalischen Lokalisierung bringt hohe Anforderungen an die Genauigkeit mit sich, da das Bestreben ist, $\vec{l}$ und $\vec{l}_{real}$ so nahe wie möglich aneinander zu bringen. Aus der Erwartungshaltung des Anwenders heraus wird davon ausgegangen, dass ein kleines g gewählt wird und die Fehlerrate somit schon bei kleinen Abweichungen unerfreulich hohe Werte erreicht.

Abhilfe könnte eine andere Art von Lokalisierung schaffen, die nicht in einen zweidimensionalen Vektorraum abbildet, sondern auf eine Menge von Wahrscheinlichkeiten für wenige, wohlbekannte Orte. Das Konzept der wohlbekannten Orte wurde bereits in die vorliegende Anwendung zur besseren Orientierung während der Trainingsphase implementiert (s. Kapitel 5.3), die Abbildung auf diese in einem Lokalisierungsalgorithmus aber noch nicht.

Der grundlegende Ansatz beschreibt ein KNN, mit n Ausgängen, wobei n der Anzahl der wohlbekannten Orte entspreche. Jeder Ausgang erhalte einen Wertebereich von 0 bis 1, der die Wahrscheinlichkeit für den Aufenthalt des Endgerätes an dem jeweiligen Ort angebe.

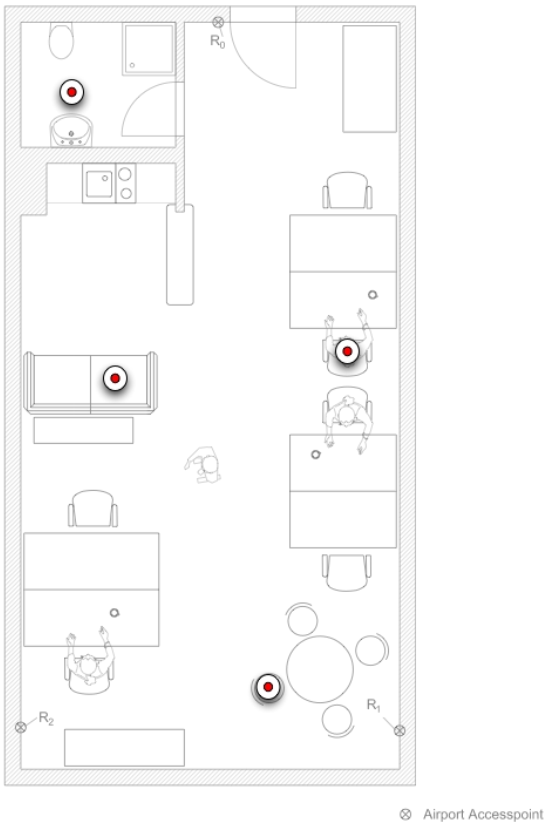


Abbildung 59: Beispiel für einige wohlbekannte Orte im gegebenen Lokalisierungsszenario

Abbildung 59 zeigt den in dieser Arbeit herangezogenen Raum und eine kleine Menge von möglichen, wohlbekannten Orten (rote Punkte). An diesen Punkten müssten in möglichst vielen Situationen Messungen der Referenzpunkte erhoben werden. Ein Trainingsdatensatz bestünde aus der in dieser Arbeit vorgestellten Menge von RSSI-Werten als Eingabevektor und einem One-Hot-codierten Ausgabevektor, der den Ort angebe, an dem gerade gemessen wurde. Bei einer späteren Lokalisierung wäre ein Vektor von Erregungsgraden für jeden Ort das Ergebnis, wobei das Maximum auf den Ort zeigte, für den der Aufenthalt am wahrscheinlichsten wäre. Gleichzeitig erhielte der Anwender durch diese maximale Erregung auch ein zuverlässiges Gütemaß für den jeweiligen Lokalisierungsvorgang. RBF-Netze erscheinen bei diesem Ansatz als sehr vielversprechende Variante. Da die Eingabedimension relativ kompakt wäre, wäre auch

das RBF-Netz nicht allzu groß (Kapitel 2.3.4.4). Das interpolierende Verhalten und die pessimistische Generalisierung bei entsprechend klein gewählten σ führten zu einem Verhalten, dass bei ähnlichen Eingaben eine sehr gute Einschätzung über die Aufenthaltswahrscheinlichkeit an einem bestimmten Ort gäbe. Wäre die Netzeingabe dagegen zu weit entfernt von den erlernten Mustern, wäre kein RBF-Neuron hinreichend stark aktiviert und die maximale Erregung läge nur bei wenigen Prozentpunkten oder sogar Null. Die Aussage des Netzes wäre dann korrekterweise "Ich weiß nicht, wo Sie sich befinden". Mit diesem Ansatz ließe sich eine außerordentlich niedrige Fehlerquote erreichen, da das Netz nur dann ein Schätzung des Ortes abgäbe, wenn die Daten halbwegs verlässlich wären. Anwendungen, die nur der Aufenthalt an bestimmten Orten interessiert, hätten durch die niedrige Fehlerquote und das Qualitätsmaß der jeweiligen Lokalisation sehr zuverlässige Informationen. Dieser Ansatz scheint für intelligente Lebensräume sehr vielversprechend und auf jeden Fall wert, weiterverfolgt zu werden.

9 Anhang

9.1 Literaturverzeichnis

[1]: BITKOM, 2007 : Umsatz mit mobilen Navigationsgeräten klettert auf 1 Milliarde Euro, CD-ROM://BITKOM1.pdf

[2]: BITKOM, 2008 : Neuer Absatzrekord bei mobilen Navigationsgeräten, CD-ROM://BITKOM2.pdf

[3]: GSA, 2009 : LOCATION-BASED SERVICES MARKET READY FOR TAKEOFF, CD-ROM://GNSS.pdf

[4]: Florian Hinz, 2008 Freie Universität Berlin: Mobile Nodes and Tags in Practical Applications, CD-ROM://Hinz1.pdf

[5]: J.K. Buhagiar, C.J. Debono, 2007 EUROCON 2007: An Evaluation of Neural Network Architecture Performance in Wireless Geo-Location, CD-ROM://Buhagiar1.pdf

[6]: Bernhard Walke, 2002 John Wiley and Sons: Mobile radio networks, 9780471499022

[7]: Sudhir Dixit, Tao Wu, 2004 John Wiley and Sons: Content networking in the mobile Internet, 9780471466185

[8]: Engin Yilmaz, 2007 Humboldt-Universität zu Berlin: Localization Using RSSI Readings, CD-ROM://Yilmaz1.pdf

[9]: Sebastian Gregor, 2006 HAW-Hamburg: Entwicklung einer Hardwareplattform für die Ermittlung von Positionsdaten innerhalb von Gebäuden, CD-ROM://Gregor1.pdf

[10]: Jan Blumenthal, Frank Reichenbach, Dirk Timmermann, 2006 Universität Rostock: Minimal Transmission Power vs. Signal Strength as Distance Estimation for Localization in Wireless Sensor Networks, CD-ROM://Timmermann2.pdf

[11]: Wild Packets, 2002 : Converting Signal Strength Percentage to dBm Values, CD-ROM://WildPackets1.pdf

[12]: Bill Moss, 2005 Clemson Linux Initiative: DBM to RSSI to Percent, CD-ROM://Clemson1.pdf

[13]: Thorsten Frunzke, 2006 Friedrich-Alexander-Universität Erlangen-Nürnberg: Adaptive Lokalisierungstechniken in drahtlosen Sensornetzen, CD-ROM://Frunzke1.pdf

[14]: Krzysztof W. Kolodziej, Johan Hjelm, 2006 CRC Press: Local Positioning Systems, 9780849333491

[15]: Dirk Timmermann, 2007 Universität Rostock: Ortung in Räumen, CD-ROM://Timmermann1.pdf

[16]: Peter Hartmann, 2004 vieweg: Mathematik für Informatiker, 9783528231811

[17]: Raúl Rojas, 1996 Springer Verlag: Neural Networks - A Systematic Introduction, CD-ROM://Rojas1-pdf

[18]: David Kriesel, 2005 dkriesel.com: Ein kleiner Überblick über Neuronale Netze, CD-ROM://Kriesel1.pdf

[19]: Prof. Dr.-Ing. Andreas Meisel, 2008 HAW-Hamburg: Musterklassifikation mit neuronalen Netzen, CD-ROM://Meisel1.pdf

[20]: Stefan Otto, 2002 Institut für Informatik, Technische Universität Clausthal: Neuronale Netze, CD-ROM://Otto1.pdf

[21]: Texas Instruments Norway AS, 2007 : CC2430 Data Sheet, CD-ROM://TI1.pdf

[22]: Texas Instruments Norway AS, 2007 : CC2431 Data Sheet, CD-ROM://TI2.pdf

[23]: IAR Systems AB, 2009 : IAR Embedded Workbench, http://iar.com/website1/1.0.1.0/50/1/ Letzter Abruf: 10.07.2009

[24]: Keil - An ARM Company, 2009 : µVision IDE Overview, http://www.keil.com/uvision/ Letzter Abruf: 10.07.2009

[25]: SDCC Community, 2009 : Small Device C Compiler, http://sdcc.sourceforge.net/ Letzter Abruf: 05.01.2009

[26]: SDCC Community, 2008 : SDCC Compiler User Guide, CD-ROM://SDCC1.pdf

[27]: TinyOS Community, 2009 : TinyOS Community Forum, http://www.tinyos.net/ Letzter Abruf: 05.01.2009

[28]: TinyOS Community, 2009 : Getting Started with TinyOS, http://docs.tinyos.net/index.php/Getting_Started_with_TinyOS Letzter Abruf: 05.01.2009

[29]: TinyOS8051WG Community, 2009 : TinyOS 8051 working group, http://www.tinyos8051wg.net/ Letzter Abruf: 05.01.2009

[30]: Ubisense AG , 2009 : Coverage, http://www.ubisense.de Letzter Abruf: 08.03.2009

[31]: Ubisense AG, 2007 : Ubisense Compact Tag, CD-ROM://Ubisense2.pdf

[32]: Ubisense AG, 2007 : Serie 7000 Sensor, CD-ROM://Ubisense1.pdf

[33]: Ubisense AG , 2009 : Ultrawideband, http://www.ubisense.de/content/14.html Letzter Abruf: 08.03.2009

[34]: Ekahau, Inc., 2009 : Wi-Fi RTLS and Site Survey Solutions, http://www.ekahau.com/ Letzter Abruf: 08.03.2009

[35]: Humboldt-Universität zu Berlin, 2009 Humboldt-Universität zu Berlin: MagicMap im Vergleich zu anderen Systemen, CD-ROM://MagicMap1.pdf

[36]: Peter Ibach, Tobias Hübner, Martin Schweigert, 2004 Humboldt-Universität zu Berlin: MagicMap – Kooperative Positionsbestimmung über WLAN, CD-ROM://Ibach1.pdf

[37]: Skyhook Wireless, Inc., 2009 : How It Works, http://www.skyhookwireless.com/howitworks/ Letzter Abruf: 08.03.2009

[38]: Skyhook Wireless, Inc., 2009 : Coverage, http://www.skyhookwireless.com/howitworks/coverage.php Letzter Abruf: 08.03.2009

[39]: Thomas Pfaff, 2007 HAW-Hamburg: Entwicklung eines PDA-basierten Indoor-Navigationssystems, CD-ROM://Pfaff1.pdf

[40]: Andreas Herglotz, 2006 HAW-Hamburg: Lokalisierung und Orientierung in Gebäuden - IMAPS und Headmounted Display im Einsatz als Museumsführer, CD-

ROM://Herglotz1.pdf

[41]: Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan, 2000 6th ACM MOBICOM: The Cricket Location-Support System, CD-ROM://Cricket1.pdf

[42]: Rob Griffiths, 2008 Macworld.com: Use a command-line AirPort tool, CD-ROM://Airport1.pdf

[43]: Igor Fischer, Fabian Hennecke, Christian Bannes, Andreas Zell, 2002 Eberhard Karls Universität Tübingen: JavaNNS : Java Neural Network Simulator, http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/manual/JavaNNS-manual.html Letzter Abruf: 01.02.2009

[44]: Bernward Klett, 1995 : SNNS2C Readme, CD-ROM://snns2c1.txt

[45]: Igor Fischer, Fabian Hennecke, Christian Bannes, Andreas Zell, 2002 Eberhard Karls Universität Tübingen: JavaNNS : User Manual, http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/ Letzter Abruf: 01.02.2009

[46]: Joe Hicklin, Cleve Moler, Peter Webb, Ronald F. Boisvert, Bruce Miller, Roldan Pozo, Karin Remington, 2005 : JAMA : A Java Matrix Package, http://math.nist.gov/javanumerics/jama/ Letzter Abruf: 12.06.2009

[47]: Apache Software Foundation, 2009 : Apache Tomcat, http://tomcat.apache.org/ Letzter Abruf: 20.06.2009

[48]: Sun Microsystems, Inc., 2009 : MySQL Documentation, http://dev.mysql.com/doc/ Letzter Abruf: 20.06.2009

[49]: Red Hat Middleware, LLC., 2009 : Relational Persistence for Java and .NET, https://www.hibernate.org/ Letzter Abruf: 20.06.2009

[50]: Martijn Dashorst, Eelco Hillenius, 2009 Manning Publications Co.: Wicket in Action, 9781932394986

[51]: OpenNETCF Consulting, 2009 : Smart Device Framework, http://www.opennetcf.com/Products/SmartDeviceFramework/tabid/65/Default.aspx Letzter Abruf: 10.07.2009

[52]: Socket Mobile, inc., 2009 : Go Wi-Fi!® SD P320/P320X (802.11a/b/g Cards), http://www.socketmobile.com/products/embedded-wireless-communications/go-wi-fi-abg/ Letzter Abruf: 10.07.2009

[53]: , 2008 Wikimedia Foundation, Inc.: Artificial Neural Networks, http://en.wikibooks.org/wiki/Artificial_Neural_Networks/Error-Correction_Learning#Error-Correction_Learning Letzter Abruf: 02.04.2009

[54]: Kurt Derr, Milos Manic, 2008 3rd IEEE Conference on Industrial Electronics and Applications: Wireless Indoor Location Estimation Based on Neural Network RSS Signature Recognition (LENSR), CD-ROM://LENSR1.pdf

[55]: Sun Microsystems, Inc., 2009 : MySQL, http://www.mysql.de/ Letzter Abruf: 20.06.2009

9.2 Glossar

AOA	Angle of Arrival, Distanzwertermittlungsverfahren
Application Server	Synonym für die Serverkomponente einer Webanwendung
DAO	Data Access Object, Object welches zu Zutritt zum Objektmodell in Objektorientierten Programmiersprachen gewährleistet
Embedded	Eingebettet, hier eingebettete Systeme in Low-Cost-Hardware
GPS	Global Positioning System, satellitengestütztes Ortungsverfahren
HAW	Hochschule für Angewandte Wissenschaften Hamburg
I/O	Input / Output, Datenverkehr in zwei Richtungen
IAR	IAR Systems, Hersteller einer IDE zur Entwicklung von Embedded Software
IDE	Integrated Development Environment, Entwicklerwerkzeug
JavaNNS	Software-Tool zur Erstellung von Neuronalen Netzen
KNN	Künstliche Neuronale Netze
MLP	Multi-Layer-Perceptron, Typ eines KNN
MOTE	Teilnehmer eines ZigBee Netzwerks
Multipath Fading	Störeffekt bei Signalübertragungen über Funk durch Reflexionen
MySQL	Relationales Datenbanksystem [55]
NTP	Netzwerkprotokoll zur Uhrensynchronisation
One-Hot	Bit Kodierung, bei der immer nur ein Bit zur Zeit aktiviert ist

PDA	Personal Digital Assistant, taschencomputer und Kalender
Persistence Framework	Objektrelationales Mapping auf Datenbanktabellen
RBF	Radiale Basisfunktionen
RSSI	Signalpegeleinheit für Funkübertragung
Shadowing	Störeffekt bei Signalübertragungen über Funk durch Hindernisse
SLP	Single Layer Perceptron, Ein Perceptron eines MLP
snns2c	Compiler um aus JavaNNS Dateien Maschinencode zu erzeugen
SOC	System-on-a-Chip, ein Ein-Chip-System
TDOA	Time Difference of Arrival, Distanzwertermittlungsverfahren
TOA	Time of Arrival, Distanzwertermittlungsverfahren
UWB	Ultra Wideband, Kommunikationstechnologie
Web Application Framework	Grundgerüst zur Entwicklung dynamischer Webseiten oder Webanwendungen
Wicket	Ein Java Web Application Framework der Apache Group
WIFI	Wireless Fidelity, siehe WLAN
WLAN	WLAN, Kommunikationstechnologie im Consumer Markt
XPS	Lokalisierungssystem der Firma Skyhook Wireless

9.3 Abbildungsverzeichnis

9.4 Quellcodeverzeichnis

9.5 Tabellenverzeichnis

9.6 Abgrenzung

Im folgenden ist definiert, welcher Autor für welches Kapitel maßgeblich verantwortlich war. O steht für Pascal Oblonczek, Z für Sebastian Zahn.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg,	
Ort, Datum	Unterschrift
Hamburg,	
Ort, Datum	Unterschrift