Jan Hase
Kilian Gärtner
Robert Schmidt

07.02.2010

Indoor Navigation

Die Suche nach dem optimalen Weg in Gebäuden

Inhalt

Einleitung:

Die Suche nach dem kürzesten oder auch dem schnellsten Weg beschäftigt die Menschheit schon seit dem Gedanken an die optimale Nutzung ihrer Zeit.
Ende des 20. Jahrhunderts erschienen für erschwingliche Preise die GPS gestützte Navigationssysteme, welche einerseits als elektronische Karte, andererseits aber auch zur Findung des optimalen Weges dienen.
Wir, Jan Hase, Robert Schmidt und Kilian Gärtner, fragten uns, ob diese Suche nach dem optimalen Weg auch für Innenräume, wie z.B. in Museen oder größeren Universitäten, möglich sein würde.
Unsere Vision ist die, dass ein jeder Mensch in einem größeren Gebäude über ein mobiles Gerät ohne viel Aufwand seine eigene Position im Gebäude bestimmen und den optimalen Weg zu seinem Ziel finden kann.
Mehrere Probleme mussten analysiert und gelöst werden. Damit jeder Mensch diese Technologie nutzen kann, war es wichtig, die Kosten niedrig zu halten.
Deswegen einigten wir uns auf folgendes:
Wir wollten kein neues Gerät voraussetzen, sondern, sofern möglich, auf ein Gerät zurückgreifen, was bereits weit verbreitet ist: Das Handy oder das Smartphone.
Mit der Frage nach dem Gerät ging auch die Frage nach der Programmiersprache einher. Wir entschieden uns aufgrund der Komptabilität mit vielen Handyarten für Java ME, einer objektorientierten Sprache, welche mittels einer Virtual Machine Programmcode ausführbar machte. Jedoch stießen wir im Laufe der Entwicklung auf eine entscheidende Grenze der Sprache, weswegen ein Wechsel auf eine andere Programmiersprache notwendig war.
Des Weiteren wollten wir einen eigenen Wegfindungsalgorithmus finden, der wenig Speicher und Rechenzeit verbraucht.
Ein großes Problem war jedoch die Findung einer Lösung für die Ortung des Handys im Raum, da GPS zu ungenau ist und auch innerhalb von Gebäuden nicht gewährleistet ist, dass es funktioniert. Hier entschieden wir uns für eine Lösung per lokale Sendestationen, die ein Signal senden, welches vom Programm genutzt wird, um die Position zu bestimmen.
Die Lösungsfindung der Probleme fassten wir in einem Projekt zusammen, was uns seit September 2009 beschäftigt und unzählige Arbeitsstunden benötigte.
Dieses Projekt nannten wir „Indoor Navigation“. In folgender Dokumentation wollen wir erläutern, wie wir arbeiteten und was wir dabei erforschten.

Vorgehensweise

Als wir mit dem Projekt begannen, war uns klar, dass dieses Projekt viel mehr Aufwand erfordert als kleinere Programme für die Schule zu schreiben.
Denn es musste:

1. ein Wegfindungsalgorithmus entwickelt werden,
2. ein Algorithmus zum Orten innerhalb von Gebäuden entwickelt werden und
3. beides kombiniert auf ein mobiles Gerät transportiert werden.

Wir begannen mit der Suche nach einem Wegfindungsalgorithmus. Wir setzten uns in vielen Sitzungen nach der Schule zusammen und überlegten, wie man einen Wegfindungsalgorithmus aufbaut. Die Datenstruktur zur Speicherung der Karte war entscheidend, denn Speicher war nur begrenzt vorhanden. Wir diskutierten, ob man die Karte als Graph ansehen kann, wie andere Algorithmen es sehen oder aber als 2D-Gebilde mit X- und Y-Koordinaten. Zum Anfang wollten wir mittels Brute Force alle Wege ausprobieren, jeden Weg in einem Teilbaum speichern und dann nach jedem Erreichen des Zieles mit dem bisher besten Weg vergleichen. So hätten wir sicher den optimalen Weg gefunden, doch der Baum wäre extrem schnell gewachsen und hätte damit den Speicherplatz des Gerätes überfordert
Wir einigten uns auf das 2D-Gebilde und speicherten den Weg in einer Liste von Koordinatenpaaren.
Jan Hase fiel bei der Betrachtung der Skizzen eines Tages eine Gesetzmäßigkeit auf, die als Basis für unseren ersten Algorithmus dienen sollte. Wir implementierten den Algorithmus vollständig in Java und optimierten ihn zugleich. Für die Implementierung haben wir die Entwicklungsumgebung NetBeans 6.8 genutzt, welche uns vertraut ist.
Des Weiteren benutzten wir für die Verteilung des Quelltextes ein SVN Repository im Internet, sodass jeder von zu Hause ohne Probleme am Projekt arbeiten konnte. Nach getaner Arbeit wurde seine Arbeit hochgeladen und jeder aus dem Team konnte unabhängig von der Zeit diese Änderung runterladen und weiterarbeiten. Diese Technologie nutzten wir bereits intensiv bei der Erstellung schulinterner Programme.
Zugleich teilte sich das Team in zwei Gruppen. Kilian Gärtner fing mit der Übersetzung von Java SE, das für Desktop Anwendungen verwendet wird, auf Java ME, was für mobile Geräte verwendet wird, an. Robert Schmidt und Jan Hase suchten nach einer Möglichkeit mit Java ME Zugriff auf das WLAN zu bekommen, sodass die WLAN Knoten der Umgebung ausgewertet werden können.
Kurz vor der vollständigen Übersetzung des Quelltextes war uns klar, dass mit Java ME das Ziel nicht verwirklicht werden konnte. In einer Eilsitzung entschieden wir nach kurzem Abwägen der Konsequenzen, auf eine andere Programmiersprache umzusteigen, die uns bei der Erfüllung des Projektes behilflich sein sollte. Dies war C#.
C# ist Teil von Microsofts .NET Umgebung. Da C# Programme auf Windows Mobile ausgeführt werden können und C# zu 90% Java ähnelt, entschieden wir uns dafür.
Die Entwicklungsumgebung, Visual Studio 2008 Professional, war im Team ebenfalls bekannt, sodass die Eingewöhnungszeit minimal gehalten wurde.
Die Konvertierung dauerte gerade einmal drei Tage und wurde ohne größere Probleme fertiggestellt.
Als die ersten brauchbaren Programme fertig waren, mussten wir jedoch unseren üblichen Arbeitsplatz von der Schule auf Jan Hases Wohnung verlegen, wo wir eine Testumgebung mit drei Access Points installierten. Zugleich schrieb Kilian Gärtner an der Dokumentation, während Jan Hase und Robert Schmidt sich intensiv mit der Problembehebung beim Ortungsalgorithmus beschäftigen.

Ortung

Einleitung

Eine Ortung außerhalb von Gebäuden, wie zum Beispiel bei der Straßennavigation, ist mit der heutigen Technologie fast schon eine Selbstverständlichkeit geworden. Innerhalb von Gebäuden ist diese Problematik jedoch noch nicht vollständig gelöst.
GPS gestützte Systeme ermöglichen eine Ortung von bis auf 10 Meter Genauigkeit im zivilen Gebrauch, und im militärischen Gebrauch ist die Abweichung noch geringer. In Relation zu der Fläche, auf der man sich bewegt, sind 10 Meter eine sehr geringe Abweichung.
In Gebäuden sind jedoch schon Abweichungen von 5 Metern problematisch, da die Fläche viel kleiner ist. Jede Abweichung ist gleichbedeutend mit einem Verlust des Nutzens des Navigationsgeräts.
Selbst wenn GPS eine Genauigkeit von einem Meter erreichen würde, wird das GPS-Signal von Gebäuden gestört und kann so innerhalb von Gebäuden nicht empfangen werden.
In Gebäuden jedoch halten verschiedene Materialen das Signal ab, sodass eine Mauer das Signal erheblich stören kann. Auch Menschenmassen stören das Signal mittels ihrer Körpers ungewollt und behindern den Vorgang der exakten Ortung des Nutzers.
Einige Projekte, wie zum Beispiel das Projekt **MagicMap**, benutzen WLAN Signale für die Positionsbestimmung.
Sogenannte Access Points senden das WLAN Signal und dienen der drahtlosen Vernetzung von Computern.
Da viele Gebäude bereits Access Points besitzen und diese auch billig und unkompliziert zu installieren sind, haben wir uns dafür entschieden, ebenfalls diese Technologie zur Ortung zu nutzen. So gewährleisten wir eine freie und vielfältige Verfügbarkeit der Technologie.
Auf Grund der schon angesprochenen Problematik der Menschenmassen, welche das WLAN Signal stören bzw. abschwächen, haben wir uns für ein Verfahren entschieden, welches „*Fingerprint*“ genannt wird.

Fingerprint Verfahren

Dieses Verfahren beruht darauf, dass vor der eigentlichen Ortung Referenzmessungen durchgeführt werden. Der Verantwortliche für das Gebäude installiert mehrere Access Points im Gebäude und führt nun eine Referenzmessung durch. Bei dieser Messung wird systematisch der Raum begangen und an der jeweiligen Position alle verfügbaren Access Points mit ihrer zu dem Zeitpunkt vorhandenen Signalstärke gespeichert. Nachteil dieses Verfahren ist das Messen selber. Der Messende muss systematisch den Raum durchqueren und messen, was bei größeren Räumen wie z.B. ein Flughafen eine Schwierigkeit darstellt.
Der Vorteil ist jedoch, dass man anhand dieser Referenzpunkte ein relativ genaues Ergebnis beim späteren Orten erhält, da man nur die Punkte in der Karte absuchen muss, die ähnliche Signalstärken wie zur Zeitpunkt der Ortung besitzen.
Mithilfe von statistischen Erhebungen können die Referenzdaten verbessert werden, indem immer wieder gemessen wird und mit den vorherigen Messungsergebnissen ein Mittelwert für den Referenzpunkt bestimmt wird.

Die Grafik veranschaulicht eine fiktive Messung eines Raumes

y\x	0	1	2	3	4	5
0	A	A	A C	C A	C	C
1	A	A	A C	C A	C	C
2	A B	A B	A B C	C A D	C D	C D
3	B A	B A	B A D	D B C	D C	D C
4	B	B	B D	D B	D	D
5	B	B	B D	D B	D	D

Fiktive Messung eines Raumes

Grüne Farbe = Sendebereich von Access Point **A**
Gelbe Farbe = Sendebereich von Access Point **B**
Blaue Farbe = Sendebereich von Access Point **C**
Rote Farbe = Sendebereich von Access Point **D**

Unser Algorithmus

Unser Algorithmus für die Ortung funktioniert auf die oben beschriebene Weise.
Die Bestimmung der Position beginnt mit der Suche nach aktuell verfügbaren Access Points. Dies wird mithilfe einer externen Bibliothek Namens OpenNETCF.Net, welche Klassen zur Ermittlung der Signalstärke und der Namen von Access Points bereitstellt, ermittelt.
Danach werden sie nach ihren Namen geordnet.
Für die Veranschaulichung bedienen wir uns der oben zu sehenden Grafik.
Nehmen wir an, wir haben die Access Points **A** und **B** gemessen, wobei **A** eine Signalstärke von -39dbm und **B** eine bessere Signalstärke von -32dbm besitzt.
Nun werden alle Referenzpunkte der Ebene betrachtet und abgeglichen, ob die Referenzpunkte dieselbe Anzahl an Access Points besitzen wie die aktuell gemessen und ob die Namen gleich sind.
Damit würden nur noch die Referenzpunkte R1(0/2), R2 (1/2), R3(0/3) und R4(1/3) als mögliche Standorte übrig bleiben.

y\x	0	1	2	3	4	5
0	A	A	A C	C A	C	C
1	A	A	A C	C A	C	C
2	A B	A B	A B C	C A D	C D	C D
3	B A	B A	B A D	D B C	D C	D C
4	B	B	B D	D B	D	D
5	B	B	B D	D B	D	D

Mögliche Standorte

Die Referenzpunkte mit den Koordinaten (2/2) und (2/3) kommen nicht in Frage, da dort 3 Access Points gemessen wurden, zurzeit der Ortung aber nur zwei Access Points gemessen wurden.
Nun überprüfen wir, welcher der übrig gebliebenen Punkten am ehesten dem Standort entspricht. Dies geschieht über die Differenzenbildung der Signalstärken.
R1 und R2 liegen näher an A als an B, sodass die Signalstärke bei der Referenzmessung bei **A** an R1 und R2 größer gewesen sein muss als die Signalstärke von **A** bei R3 und R4.
Nehmen wir an, dass:

1. R1 mit A mit -30dbm und B mit -50 dbm
2. R2 mit A mit -35dbm und B mit -55 dbm
3. R3 mit A mit -40dbm und B mit -30 dbm
4. R4 mit A mit -60dbm und B mit -40 dbm
5. Standort S mit A -39dbm und B mit -32dbm

Bildet man nun die Summe der Beträge der Differenzen der Signalstärken für jeden Referenzpunkt, so ermittelt man den Referenzpunkt der am wahrscheinlichsten der Standort ist.
Die Rechnung am obigen Beispiel:

$$R1: |-30-(-39)|+|-50-(-32)| = 9+18 = 27$$
$$R2: |-35-(-39)|+|-55-(-32)| = 4+23 = 27$$
$$R3: |-40-(-39)|+|-0-(-32)| = 1+2 = 3$$
$$R4: |-60-(-39)|+|-40-(-32)| = 21+8 = 29$$

Damit hat R3 die geringste Differenz und ist somit am wahrscheinlichsten der Standort.

y\x	0	1	2	3	4	5
0	A	A	A C	C A	C	C
1	A	A	A C	C A	C	C
2	A B	A B	A B C	C A D	C D	C D
3	B A	B A	B A D	D B C	D C	D C
4	B	B	B D	D B	D	D
5	B	B	B D	D B	D	D

Wahrscheinlichster Standort

Wegfindung

Die Suche nach dem optimalen Weg ist ein Problem in der Informatik, welches einfach zu lösen ist, jedoch viel Speicher und viel Rechenzeit verbraucht. Normale Haushaltscomputer besitzen Arbeitsspeicher im Gigabyte Bereich, Prozessoren mit mehreren Kernen und Taktraten im Gigahertz Bereich und Festplattenspeicher im Terabyte Bereich. Hier ist der einzige Faktor bei der Suche nach dem optimalen Weg die Zeit, denn je mehr mögliche Wege es gibt, desto mehr Wege müssen berechnet und miteinander verglichen werden.
Unser Anwendungsbereich ist jedoch der Bereich der mobilen Geräte wie Handys, PDAs und Smartphones, wo die Leistung sehr begrenzt ist.
Vor allem der Speicher der mobilen Geräte ist klein und auch die Prozessoren sind im Hinblick auf geringem Stromverbrauch hin entwickelt worden, so dass deren Taktraten im mittleren Megahertzbereich liegen.
Deswegen ist es unabdingbar, dass der Wegfindungsalgorithmus nicht alle möglichen Wege berechnet, sondern nur jene, die wahrscheinlich sind, um den Speicherverbrauch zu verringern als auch die Rechenzeit zu verringern.
Hierbei ist zu beachten, dass bei der Schätzung auch Wege als unwahrscheinlich geschätzt werden können, die jedoch in Wirklichkeit gute Wege sind.

VDPA

Einleitung

Unter diesem Aspekt haben wir einen Algorithmus entworfen, der sich an der theoretischen Entfernung zum Ziel orientiert als auch an der Art und Weise, wie man zum Ziel gelangt. Wir tauften den Algorithmus **VDPA**, **V**alue **D**ependent **P**athfinding **A**lgorithm.
Die Karte, auf welcher der Weg gefunden werden soll, ist rechteckig aufgebaut, so dass eine Rasterung der Karte entsteht.

Verfahren

Bevor der Suchalgorithmus beginnt, werden die Wertigkeiten der Punkte der Karte berechnet.
Die Wertigkeit wird wie folgt berechnet:

Wenn man die Punkte P_W und P_Z hat, der aktuell betrachtete Punkt ist P_W und P_Z der Zielpunkt, dann bildet man den Betrag der Differenz der einzelnen Koordinaten:

$$|X_w - X_Z|$$
$$|Y_w - Y_Z|$$

Die größere der beiden Komponenten beschreibt die Anzahl der benötigten Schritte zum Ziel und die kleinere der beiden Komponenten beschreibt die Anzahl der Diagonalen auf dem Weg zum Ziel.

Die Anzahl der benötigten Schritte wiegt mehr als die Anzahl der Diagonalen, weswegen sich folgende Formel ergibt:

$$Anzahl\ der\ Schritte * 10 + Anzahl\ der\ Diagonalen = Wertigkeit\ des\ Punktes$$

Je geringer die Wertigkeit ist, desto geeigneter ist der Wegpunkt.

Beispiel mit Zielpunkt bei (1/1):

y\x	0	1	2	3	4
0	11	10	11	21	31
1	10	0	10	20	30
2	11	10	11	21	31
3	21	20	21	22	32
4	31	30	31	32	33

Wertigkeiten der Wegpunkte

Das Ziel hat die Koordinaten (1/1). Am Beispiel der Punkte P_1(1/0) und P_2(3/2) die Berechnung der Wertigkeit :

$$|P_1 - Ziel|$$
$$\Delta X = |1 - 1| = 0$$
$$\Delta Y = |0 - 1| = 1$$
$$Da\ \Delta Y > \Delta X\ gilt, gibt$$
$$\Delta Y\ die\ Anzahl\ der\ Schritte\ und$$
$$\Delta X\ die\ Anzahl\ der\ Diagonalen\ an$$
$$Wertigkeit\ für\ P_1 = \Delta Y * 10 + \Delta X$$
$$Wertigkeit\ für\ P_1 = 10 + 0$$
$$Wertigkeit\ für\ P_1 = 10$$

$$|P_2 - Ziel|$$
$$\Delta X = |3 - 1| = 0$$
$$\Delta Y = |2 - 1| = 1$$
$$Da\ \Delta X > \Delta Y\ gilt, gibt$$
$$\Delta X\ die\ Anzahl\ der\ Schritte\ und$$
$$\Delta Y\ die\ Anzahl\ der\ Diagonalen\ an$$
$$Wertigkeit\ für\ P_2 = \Delta X * 10 + \Delta Y$$
$$Wertigkeit\ für\ P_2 = 20 + 1$$
$$Wertigkeit\ für\ P_2 = 21$$

Sind alle Wertigkeiten der Ebene berechnet, beginnt der Pathfinder(dt. Pfadfinder) den geeignetsten Weg zu suchen, indem er sich alle umliegenden Punkte anschaut und den Wegpunkt mit der geringsten Wertigkeit betritt. Dies macht er solange, bis er den Zielpunkt erreicht hat.

Beispiel mit Startpunkt bei (3/4) und Zielpunkt bei (1/1):

y\x	0	1	2	3	4
0	11	10	11	21	31
1	10	0	10	20	30
2	11	10	11	21	31
3	21	20	21	22	32
4	31	30	31	32	33

Karte mit Start und Ziel

1. Schritt: Betrachtung der umliegenden Punkte(rot markiert)

y\x	0	1	2	3	4
0	11	10	11	21	31
1	10	0	10	20	30
2	11	10	11	21	31
3	21	20	21	22	32
4	31	30	31	32	33

Umliegende Punkte von Start

2. Schritt: Den Wegpunkt mit kleinster Wertigkeit nehmen (außer er ist -1 [nicht begehbar, z.B. Mauer]) und als Wegpunkt zum Pfad hinzufügen. Des Weiteren die Betrachtung der nächsten Punkte vom hinzugefügten Wegpunkt fortsetzen. Solange dies fortsetzen, bis das Ziel erreicht ist.

y\x	0	1	2	3	4
0	11	10	11	21	31
1	10	0	10	20	30
2	11	10	11	21	31
3	21	20	21	22	32
4	31	30	31	32	33

y\x	0	1	2	3	4
0	11	10	11	21	31
1	10	0	10	20	30
2	11	10	11	21	31
3	21	20	21	22	32
4	31	30	31	32	33

Nächste Wegpunkte

Nachdem alle Pfade beschritten wurden, wird eine Nachoptimierung durchgeführt. In dieser wird jeder Punkt des Pfades hinsichtlich seiner benachbarten Punkte überprüft. Wenn einer der benachbarten Punkte auch Element des Pfades ist und nicht der Nachfolger, so ist dies der neue Nachfolger. So werden die Fälle des im Kreis Laufens und unnötig entstehende Längen entfernt.

Problembehandlung

Sollte er jedoch merken, dass er bereits an diesem Punkt war, geht er alle Schritte bis zum bereits besuchten Punkt zurück und wählt den Punkt mit der zweit besten Wertigkeit als nächsten und versucht nun erneut, den Weg zu finden.

Kritischer Fehler und das Ende von VDPA

Uns fiel jedoch bei den Tests ein gravierender Fehler im Algorithmus auf.
Im folgenden Beispiel ist dies zu erkennen:

55	54	53	52	51	50	51	52	53	54	55
54	44	43	42	41	40	41	42	43	44	54
53	43	33	32	31	30	31	32	33	43	53
52	42	32	22	21	20	21	22	32	42	52
51	41	31	21	11	10	11	21	31	41	51
50	40	30	20	10	0	10	20	30	40	50
51	41	31	21	11	10	11	21	31	41	51

Fehler im Algorithmus

Da der rot markierte Punkt besitzt eine geringere Wertigkeit als der dunkel grün markierte Punkt hat, darum fügt der Algorithmus den rot markierten Punkt hinzu, welcher jedoch, wie zu sehen ist, nicht der bessere Punkt ist. Da der Fehler bei der Formel liegt, war er nicht zu berichtigen, ohne sich einen neuen Algorithmus auszudenken, was wir dann auch taten.

CAPF

Einleitung

Als uns bewusst wurde, dass VDPA kein funktionierender Algorithmus ist, nutzten wir eine unser ersten Ideen zur Wegfindung in einem zwei-dimensionalen Raum. Die Idee bestand darin, die Koordinaten des Startpunktes an die Koordinaten des Zielpunktes anzugleichen. Sollten beide Koordinaten ungleich sein, dann sollte der Algorithmus diagonal gehen, wenn eine Koordinate gleich ist, sollte sie die andere anpassen. Dies funktioniert ohne Probleme in einem Raum ohne jedes Hindernis.
Jedoch ist dieser Fall als trivial zu werten, weswegen wir überlegten, wie wir mit Hindernissen umgehen sollen. Uns fiel dabei eine Gesetzmäßigkeit bezüglich der Koordinaten und des Abstandes auf, weswegen wir den VDPA entwickelten.
Diese Idee des Angleichens der Koordinaten bauten wir nun weiter aus und entwickelten den Algorithmus CAPF, **C**oordinate **A**djusting **P**ath**f**inder.

Verfahren

Um herauszufinden, in welche Richtung der Zielpunkt liegt, wird die Differenz der Koordinaten des Start- und des Zielpunktes berechnet. Die Differenz gibt der Wegsucheinheit Auskunft, in welche Richtung er gehen muss.

Beispiel mit Startpunkt bei (1/1) und Zielpunkt bei (7/7):

y\x	0	1	2	3	4	5	6	7
0								
1		S						
2								
3								
4								
5								
6								
7								Z

Karte mit Start und Ziel

Die X Richtung ist = 7 – 1 > 0 -> 1 in X-Richtung bewegen
Die Y Richtung ist = 7 – 1 > 0 -> 1 in Y-Richtung bewegen

Dies macht er solange, bis er entweder das Ziel erreicht hat oder aber, wie in dem Beispiel, in eine Ecke kommt, wo er keine der beiden Koordinaten angleichen kann.

y\x	0	1	2	3	4	5	6	7
0								
1		S						
2								
3								
4								
5								
6								
7								Z

Weg bis zur Ecke

Hier werden in die freien Richtungen so genannte **Scouts** ausgesendet, die nach einem Ausweg schauen. Diese laufen nur in eine Richtung und schauen nur in einer Richtung nach einem Ausweg. In diesem Fall wird ein Scout nach links geschickt, der nach unten nach einer Öffnung schaut und einer nach oben, der nach rechts schaut.

y\x	0	1	2	3	4	5	6	7
0								
1		S						
2								
3								
4								
5								
6								
7								Z

Scouts in zwei Richtungen

Wird ein Ausweg gefunden, so geht der Scout in diesen scheinbaren Ausweg und überprüft, ob er, wie im Beispiel um die Mauer herum kann. Falls ja, so setzt er ein zweites Signal und gibt beide an die Hauptsucheinheit zurück, die nun den Weg zum ersten Signal und den Weg zum zweiten Signal sucht.
Kommt der Scout nicht um die Mauer herum, so geht er an dieser entlang und sucht einen zweiten Ausweg und setzt dort erst das zweite Signal. Erst wenn ein Scout 2 Signale gesetzt hat, wird der andere nicht mehr weiter verfolgt.

y\x	0	1	2	3	4	5	6	7
0								
1		S						
2								
3								
4								
5	1. S							
6		2. S						
7								Z

Signale vom Scout

y\x	0	1	2	3	4	5	6	7
0								
1		S						
2								
3								
4								
5								
6								
7								Z

Weg von Start zum Signal

Die Hauptsucheinheit setzt von dort an nun die Suche nach dem ursprünglichen Ziel fort.

Sackgassenproblem:

Sollte jedoch der Algorithmus merken, dass er in einer Sackgasse gelandet ist, geht er bis zum letzten Signal zurück und setzt an dem Eingang der Sackgasse eine Mauer. Diese ist jedoch nur für diese eine Routenberechnung vorhanden und existiert, wenn eine neue Route berechnet wird, nicht mehr.

y\x	0	1	2	3	4	5	6	7	8
0									
1									
2									
3					S				
4									
5									
6									
7									Z

y\x	0	1	2	3	4	5	6	7	8
0									
1									
2									
3					S				
4									
5									
6									
7									Z

Sackgassenproblem

Dieses Prinzip basiert auf dem Backtracking Verfahren, was unter anderem für die Suche nach einem Ausweg aus einem Labyrinth genutzt wird.

Ergebnisse

Wegfindung:

Es ist eine Herausforderung, einen guten Weg zu finden, wenn man nur begrenzte Ressourcen wie auf dem Handy zur Verfügung hat. Man muss viele Fälle betrachten, um zu prüfen, ob der Algorithmus wirklich funktioniert.
Unser Ergebnis für CAPF ist das, dass der Algorithmus von der Theorie gut funktioniert, jedoch zum Ende hin es uns nicht möglich war, ihn so wie er in der Dokumentation steht, zu implementieren. Es gab bisher noch Fehler, welche uns unklar sind. Jedoch fand der Algorithmus immer einen guten bis den optimalen Weg unter Bedingungen, wo keine Sackgasse entstand.
Des Weiteren ist der Algorithmus mit seinen Scouts in der Lage, auf mehr als nur einem Thread zu laufen. Wenn mobile Geräte Mehrkernprozessoren besitzen wie heutige Desktop PCs, dann ist es kein Problem, aus CAPF ein Algorithmus zu machen, der mehrere Threads benutzt, was seine Rechenzeit erheblich verkürzen würde.

Ortung:

Für einen Test in einem größeren Gebäude wie einem Museum oder der kompletten Schule fehlte uns leider die Zeit. In Jan Hases Wohnung erreichten wir mit der Methode nur befriedigende Ergebnisse. Der Algorithmus ist sehr stark von dem Gerät abhängig, was das WLAN Signal sendet als auch vom dem Gerät, was es empfängt.
Das Fingerprint Verfahren ist jedoch nur ein Verfahren von vielen und kann mit anderen Verfahren kombiniert werden, um eine genaue Ortung zu ermöglichen.

Projekt:

Da der Wegfindungsalgorithmus in bestimmten Fällen nicht funktioniert und der Ortungsalgorithmus noch zu ungenau ist, ist das Projekt zwar nicht gescheitert, aber im jetzigen Zustand nicht massentauglich. Der Ortungsalgorithmus ist hier der entscheidende Faktor, ob das Projekt für den Privatanwender genutzt werden kann.
Jedoch können wir sagen, dass die Arbeit an dem Projekt anstrengend war und viel Zeit in Anspruch genommen hat. Trotzdem hat es uns dreien viel Spaß gemacht, in unserer Freizeit so ein Projekt anzufangen und zu bearbeiten und können stolz auf das sein, was wir in der Zeit seit September 2009 geleistet haben.

Quellen

Literaturverzeichnis:

http://www.ikaros-projekt.de/index.php?name=CmodsDownload&file=index&req=getit&lid=64 : 30.01.2010

Professor Dr. Uwe Großmann: WLAN Ortung innerhalb von Gebäuden mittels Signalstärkelinien

Stephan Müller: Positionierung im WLAN

http://www.magicmap.de/index.htm : 15.11.2009

Peter Ibach : Projekt mit ähnlicher Orientierung

Benutzte Programme:

NetBeans IDE 6.8 für Java Programmierung
http://www.netbeans.com/

Tortoise SVN zur Verwaltung des SVN Repository
http://tortoisesvn.net/

Visual Studio 2008 Professional für C# Programmierung
http://www.microsoft.com/germany/visualstudio/products/visual-studio/visual-studio-professional.aspx

Microsoft Office 2007 zur Erstellung der Dokumentation und zur Erstellung der Diagramme
http://office.microsoft.com/de-de/default.aspx

Bibliothek zur Analyse der Access Points
http://www.opennetcf.com/CompactFramework/Products/SmartDeviceFramework/tabid/65/Default.aspx