

Lorenz Moser

**Weiterentwicklung eines
Simulators für die
Positionierung in
wireless ad hoc Netzwerken**

*Semesterarbeit SA-2002.20
Sommersemester 2002*

Tutor: Jan Beutel

*Supervisor:
Prof. Dr. Lothar Thiele*

Inhaltsverzeichnis

Zusammenfassung	4
1 Einführung	5
1.1 Wireless ad hoc Netzwerke	5
1.2 Warum Positionierung	5
1.3 Kurzbeschreibung Bluetooth	6
1.4 Positionierungsmöglichkeiten	6
1.4.1 Winkelmessung	6
1.4.2 Zeitmessung	7
1.4.3 Phasenmessung	7
1.4.4 Signalstärkemessung	8
1.5 Mathematik hinter der gewählten Positionierungsvariante	8
1.5.1 Ideale Situation	8
1.5.2 Reale Situation	9
1.5.3 QR-Zerlegung	9
2 Simulator Grundgerüst	10
2.1 Warum Simulation	10
2.2 Simulatorkonzept	10
2.3 Simulatorbefehle	11
2.3.1 Bestehende Befehle	11
2.3.2 Neue Befehle	12
3 Simulator Weiterentwicklung	13
3.1 Verbesserungen	13
3.1.1 Random Init	13
3.1.2 Szenario I/O	13
3.1.3 Log File	14
3.2 Terrain-Algorithmus / Prinzip der Zustandsmaschine	14
3.3 Zustandsmaschine, die Messfehler berücksichtigt	16
3.4 Implementation der Zustandsmaschine	18
3.4.1 Normale Knoten	18
3.4.2 Anker	19
3.4.3 Update	20
4 Bedienungsanleitung	22
4.1 Kreieren eines Szenarios	22
4.2 Einlesen eines Szenarios	23
4.3 Der Befehl ‚help‘	24
4.4 Der Befehl ‚terrain‘	24
4.5 Der Befehl ‚info‘	25

4.6 Die Log-Funktion	25
4.7 Der Befehl ‚save‘	26
4.8 Der Befehl ‚data‘	27

Abbildungen	28
--------------------------	-----------

Screenshots	28
--------------------------	-----------

Glossar	29
----------------------	-----------

Literaturverzeichnis	30
-----------------------------------	-----------

Zusammenfassung

Ein wireless ad hoc Netzwerk ist eine Form von Netzwerk, in dem die einzelnen Netzwerkknoten unabhängig von einer zentralen Steuerung miteinander kabellos kommunizieren. Es sind verschiedene Einsatzmöglichkeiten für solche Netzwerke denkbar. Möglich ist zum Beispiel der Aufbau eines Verbundes von Sensoren, der sich selbst organisiert und Daten liefert. Damit die Daten sinnvoll ausgewertet werden können, ist es bei vielen Messungen wichtig, den genauen Ort der Messung zu kennen. GPS ist ein Verfahren, seine eigene absolute, auf einer Landkarte nachvollziehbare Position zu bestimmen. GPS funktioniert aber innerhalb von Gebäuden nicht oder nur sehr ungenau.

Bluetooth ist eine Technik, die es ermöglicht, kleine Funkknoten mit niedrigem Energieverbrauch zu bauen und daher sehr gut geeignet, ein wireless ad hoc Netzwerk zu realisieren. Bluetooth regelt aber nur den Transport von Daten von einem Bluetooth Knoten zu einem andern. Welche Daten transportiert werden, hängt von den Anwendungen ab, die auf Bluetooth aufsetzen. Eine Anwendung könnte die Bestimmung der absoluten Positionen der Netzwerkknoten eines wireless ad hoc Netzwerks sein. In dieser Arbeit wird ein Simulator weiterentwickelt, mittels dem man die Funktionsweise von Algorithmen zur Positionsbestimmung von Netzwerkknoten in wireless ad hoc Netzwerken studieren kann.

Zuerst wird ein Überblick darüber gegeben, welche Möglichkeiten der Positionsbestimmung existieren. Es werden Vor- und Nachteile der einzelnen Varianten aufgezeigt und begründet, warum die Messung der Distanz zu andern Knoten via empfangene Signalstärke diejenige ist, die weiterverfolgt wird. Dann wird die Mathematik erklärt, wie ein Knoten mittels Distanzen zu andern Knoten seine eigene Position errechnen kann.

Der Simulator, wie er vor dieser Semesterarbeit bestand, wird vorgestellt, sowohl sein programmiertechnischer Aufbau, als auch die grundlegenden Funktionen. Für diese Simulation wichtige Begriffe können im Glossar nachgelesen werden. Dann werden die in dieser Arbeit vorgenommenen Änderungen und Verbesserungen des Simulators detailliert präsentiert. Es handelt sich dabei einerseits um Möglichkeiten, Netzwerkszenarien und die dazugehörigen Positionsberechnungen ein- und auszulesen, andererseits um die Implementation des Terrain-Algorithmus, eine Erweiterung der Positionsberechnung, ebenfalls basierend auf der Messung der Distanzen zu andern Knoten.

Am Schluss des Berichtes folgt eine Bedienungsanleitung für den Simulator, so wie er am Ende dieser Arbeit besteht. Es wird detailliert erklärt, wie die einzelnen Funktionen zu nutzen sind, wie Bildschirmausgaben und Files zu deuten sind, und welche Struktur ein Input-File haben muss, damit es vom Simulator gelesen werden kann.

Einführung

Es folgt eine kurze Einführung zum Thema wireless ad hoc Netzwerke, Bluetooth und zu existierenden Positionierungsverfahren.

1.1 *Wireless ad hoc Netzwerke*

Wireless ad hoc Netzwerke sind eine spezielle Form von Netzwerken. Ihr Hauptmerkmal ist die Selbstkonfiguration, d.h. sie haben keine zentrale Steuerung. Die Idee dahinter ist, dass man die Netzwerkknoten an ihren Standorten platziert und sie dann selbständig gegenseitig Kontakt miteinander aufnehmen und sich organisieren (ad hoc). Es ist offensichtlich, dass der Aufbau eines solchen Netzes am einfachsten kabellos via Funk geschehen kann (wireless). Die einzelnen Netzwerkknoten sind gekennzeichnet durch kleine und billige Bauweise, niedrigen Energieverbrauch und lokale Funkreichweite (3 – 10 Meter).

Denkbare Anwendungen für solche Netzwerke sind:

- kabellose Heimnetzwerke,
- smarte Hausnetzwerke (Kühlschrank, Briefkasten usw.),
- Überwachungsaufgaben,
- Lagerbewirtschaftung,
- Allgemeine Sensornetzwerke,

um nur einige aufzuzählen. [1]

1.2 *Warum Positionierung*

Um die in den Netzwerkknoten gewonnenen Daten auswerten zu können, ist es oftmals notwendig, den genauen Ort der Messung zu kennen. GPS-Empfänger sind technisch soweit entwickelt, dass sie die Grössenanforderungen an einen ad hoc Netzwerkknoten erfüllen würden. GPS funktioniert aber innerhalb von Gebäuden gar nicht oder nur sehr ungenau. Es muss also eine andere Form der Positionsbestimmung gefunden werden. Grundsätzlich stehen verschiedene Möglichkeiten zur Verfügung, in einem Verbund von Netzwerkknoten die eigene Position zu bestimmen. Es ist aber nicht jede gleichermassen geeignet, um in ad hoc Netzwerken realisiert zu werden. Ein Vergleich der Varianten folgt in Unterkapitel 1.4

1.3 *Kurzbeschreibung Bluetooth*

Eine heute verfügbare Technik zur Realisation von wireless ad hoc Netzwerken stellt das ‚Bluetooth Radio System‘ dar. Die Konzeption von Bluetooth erfüllt sehr gut die Anforderungen von wireless ad hoc Netzwerken. Im Gegensatz zum GSM (Global System for Mobilcommunication, unser Mobiltelefonsystem), das mit Zellen und dazugehörigen Basisstationen arbeitet, sind bei Bluetooth Basisstationen und Mobilstationen identisch, d.h. jeder Bluetooth Knoten ist beides und eine zentrale Koordination ist nicht nötig.

Bluetooth ist zuständig für die Datenübertragung von einem Knoten zu einem andern Knoten via Funk. Welche Daten übertragen werden, wird den Applikationen überlassen, die auf Bluetooth aufsetzen. Bluetooth kümmert sich also um die Suche von Kommunikationspartnern, Verbindungsaufbau, Medium access control, Energieverbrauch usw. Die benutzten Frequenzen liegen im Bereich 2.45 GHz, dem sogenannten ‚Industrial, Scientific, Medical (ISM) band‘. Diese Frequenzen können ohne Lizenzen benutzt werden.[2]

1.4 Positionierungsmöglichkeiten

Verfahren zur Bestimmung der Position von Funkknoten können auf vier verschiedenen physikalischen Messgrößen aufbauen([3],[4]):

- Winkelmessung
- Zeitmessung
- Phasenmessung
- Signalstärkemessung

1.4.1 Winkelmessung

Bei der Winkelmessung (AOA, angle of arrival) werden die geometrischen Winkel (\neq Phasenwinkel) der von verschiedenen anderen Netzwerkknoten ankommenden Signale gemessen. Diese anderen Netzwerkknoten (A – C) müssen ihre Position kennen und dem messenden Knoten (M) mitteilen. Dieser kann dann seine Koordinaten (X_0/Y_0) bestimmen.

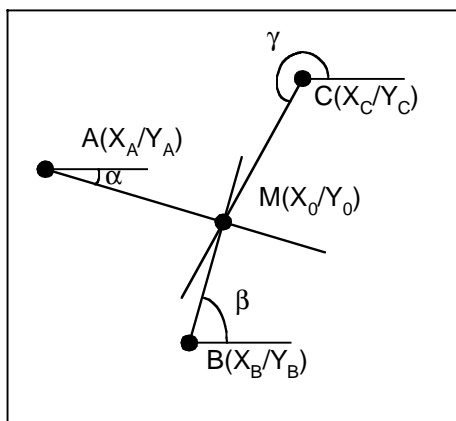


Abbildung 1:
 Kennt man die Winkel α , β & γ und die dazugehörigen Positionen von Knoten A, B & C, kann man mittels Trigonometrie die Position von Knoten M berechnen.

Um die Winkel messen zu können, reicht aber eine einzelne physikalische Antenne nicht aus. Vielmehr braucht man ein Array von Antennen, in dem jede Antenne in genau eine Richtung sendet und horcht. Dies ist ein Grund, weshalb diese Methode für wireless ad hoc Netzwerke zu aufwändig ist. Netzwerkknoten solcher Netzwerke zeichnen sich ja gerade durch einfache Bauweise aus. Ein weiterer Minuspunkt dieser Methode ist ihre potentielle Ungenauigkeit. In Gebäuden können Signale durch Wände reflektiert und abgeschirmt

werden, was zu Fehlern in der Winkelmessung von bis zu 180° (=max) führen kann. Die Methode der Winkelmessung ist also für unsere Zwecke doppelt ungeeignet.

1.4.2 Zeitmessung

Es gibt zwei Möglichkeiten, mittels Zeitmessung eine Position zu bestimmen. Man kann die Signallaufzeit vom Sender zum Empfänger messen, und daraus auf die Distanz zwischen diesen beiden Knoten schliessen (TOA, time of arrival). Diese Distanzen ergeben Kreise um die Knoten mit bekannten Positionen, auf denen der Knoten mit der unbekannt Position liegen muss. Hat man 3 Kreise, deren Mittelpunkte nicht auf einer Linie liegen, schneiden sie sich in einem Punkt (Triangulation). Dies ist der Punkt, wo der Knoten M (X_0/Y_0) liegt, dessen Position zu bestimmen ist.

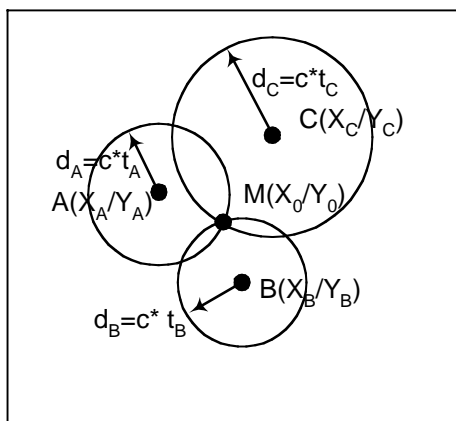


Abbildung 2:
 Kennt man die Signallaufzeiten t_A , t_B und t_C und die dazugehörigen Positionen von Knoten A, B & C, kann man über die Lichtgeschwindigkeit (c) Kreise errechnen.

Nachteil dieser Methode ist, dass die Uhren im Sender und Empfänger synchronisiert sein müssen. Dies widerspricht fundamental der Idee eines Netzwerkes ohne zentrale Koordination.

Die zweite Möglichkeit, mittels Zeitmessung die Position zu bestimmen, ist die Messung der Zeitdifferenz zwischen den Ankunftszeiten von Signalen, die gleichzeitig von verschiedenen Knoten ausgesendet wurden (TDOA, time difference of arrival). Anstatt Kreisen ordnen sich auf dies Art und Weise Hyperbeln um die sendenden Knoten an. Am Schnittpunkt dieser Hyperbeln liegt die gesuchte Position. Hier ist ebenfalls eine Synchronisation von Uhren notwendig, allerdings jetzt zwischen den Uhren der verschiedenen Sender. Diese Methode ist somit für wireless ad hoc Netzwerke ebenfalls ausser Betracht.

1.4.3 Phasenmessung

Bei der Phasenmessung wird der Phasenwinkel des ankommenden Signales gemessen. Dieser lässt sich mit hoher Präzision bestimmen. Ein Problem tritt auf, wenn die zu messenden Abstände zwischen den Knoten grösser sind als die Wellenlänge des Signales. Man kann dann nicht ermitteln, wie oft der Phasenwinkel schon einen Nulldurchgang hatte, d.h. man kann die Position nur auf Vielfache der Wellenlänge genau bestimmen. Unser Ansatz geht davon aus, dass man die Netzwerkknoten mittels Bluetooth realisiert. Diese Technik operiert in Frequenzen um 2.45 GHz.

$$\text{Wellenlänge } \lambda = \frac{c}{f} = \frac{2.998 \cdot 10^8 \frac{m}{s}}{2.45 \cdot 10^9 \text{ Hz}} = 12.2 \text{ cm}$$

Die sich ergebende Wellenlänge von 12.2 cm ist zu klein, um alleine für die Positionsbestimmung nutzbar zu sein. In Frage käme höchstens eine Feinabstimmung, wenn man mittels einer anderen Positionierungsmethode bereits bestimmt hat, wieviele Nulldurchgänge der Phasenwinkel des Signales bereits hatte.

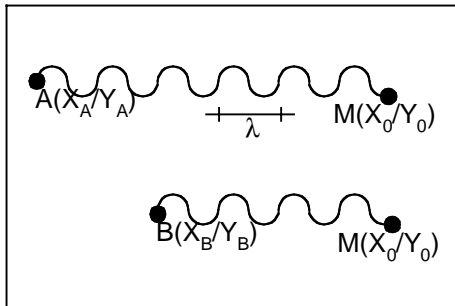


Abbildung 3:
Obwohl die Knoten A und B unterschiedliche Abstände zum Knoten M haben, misst dieser bei beiden Signalen Phasenwinkel 270° .

1.4.4 Signalstärkemessung

Die letzte Variante zur Positionsbestimmung ist die Messung der Signalstärke des ankommenden Signales. Die Dämpfung des Signales in der Luft ist bekannt. Ist die vom Sender abgestrahlte Leistung und die vom Empfänger aufgenommene Leistung bekannt, kann man aus dem Leistungsunterschied und der Dämpfung den Abstand zwischen Sender und Empfänger bestimmen. Sind erst einmal die Distanzen bekannt, kann genau gleich wie bei der Methode TOA (vgl. Abschnitt 1.4.2) mittels Triangulation die Position bestimmt werden. Natürlich führen Wände und Reflexionen zu Fehlern in der Messung (wie bei allen andern Verfahren ebenfalls), trotzdem erfüllt diese Methode die Anforderungen von wireless ad hoc Netzwerken wie einfache Bauweise der einzelnen Knoten und keine zentrale Koordination am besten.

1.5 Mathematik der gewählten Positionierungsvariante

In diesem Abschnitt wird genauer erklärt, wie man aus Distanzmessungen eine Position berechnen kann. Beim Vorstellen der verschiedenen Positionierungsvarianten wurde ein wichtiger Punkt ausgeblendet: die Ungenauigkeiten in den Messungen. Statt der effektiven Distanz d erhält man bei der Messung eine fehlerbehaftete Distanz $d - \delta$.

1.5.1 Ideale Situation

Im idealen Fall (Messung ohne Fehler) kann man für den Knoten $M(X_0/Y_0)$ bei vier umliegenden Knoten A, B, C & D, die ihre Position kennen, folgendes Gleichungssystem aufstellen (Pythagoras in der Ebene):

$$(1) (X_A - X_0)^2 + (Y_A - Y_0)^2 = d_A^2$$

$$(2) (X_B - X_0)^2 + (Y_B - Y_0)^2 = d_B^2$$

$$(3) (X_C - X_0)^2 + (Y_C - Y_0)^2 = d_C^2$$

$$(4) (X_D - X_0)^2 + (Y_D - Y_0)^2 = d_D^2$$

Der Knoten M hat die unbekannte Position (X_0/Y_0) , die durch Auflösen des Gleichungssystems berechnet wird.

Eine Anmerkung: Die Anzahl der Gleichungen kann im idealen Fall (absolut genaue Werte, kein Messfehler auf der Distanz) zwischen mindestens drei Gleichungen bis zu unendlich vielen Gleichungen betragen. Liegen mindestens drei Knoten mit bekannten Positionen nicht auf einer Gerade, bekommt man immer eine eindeutige Lösung.

1.5.2 Reale Situation

In der Realität kann aber nicht exakt gemessen werden. Der Messfehler δ muss berücksichtigt werden. Die gemessene Distanz D minus den Messfehler δ ergibt die wirkliche Distanz d . Man erhält folgendes Gleichungssystem:

(*) wirkliche Distanz $d = D - \delta$

$$\begin{aligned} (1) \quad & (X_A - X_0)^2 + (Y_A - Y_0)^2 = (D_A - \delta_A)^2 \\ (2) \quad & (X_B - X_0)^2 + (Y_B - Y_0)^2 = (D_B - \delta_B)^2 \\ (3) \quad & (X_C - X_0)^2 + (Y_C - Y_0)^2 = (D_C - \delta_C)^2 \\ (4) \quad & (X_D - X_0)^2 + (Y_D - Y_0)^2 = (D_D - \delta_D)^2 \end{aligned}$$

Multipliziert man die Klammern aus und zählt dann von den ersten drei Gleichungen jeweils die vierte ab, ergeben sich folgende drei Gleichungen (in Matrizenform):

$$\begin{bmatrix} 2(X_D - X_A) & 2(Y_D - Y_A) \\ 2(X_D - X_B) & 2(Y_D - Y_B) \\ 2(X_D - X_C) & 2(Y_D - Y_C) \end{bmatrix} \cdot \begin{pmatrix} X_0 \\ Y_0 \end{pmatrix} - \begin{pmatrix} X_D^2 - X_A^2 + Y_D^2 - Y_A^2 - (D_D^2 - D_A^2) \\ X_D^2 - X_B^2 + Y_D^2 - Y_B^2 - (D_D^2 - D_B^2) \\ X_D^2 - X_C^2 + Y_D^2 - Y_C^2 - (D_D^2 - D_C^2) \end{pmatrix} = \begin{pmatrix} \delta_A^2 - \delta_D^2 - 2(\delta_A D_A - \delta_D D_D) \\ \delta_B^2 - \delta_D^2 - 2(\delta_B D_B - \delta_D D_D) \\ \delta_C^2 - \delta_D^2 - 2(\delta_C D_C - \delta_D D_D) \end{pmatrix}$$

1.5.3 QR-Zerlegung

Das im vorigen Abschnitt erhaltene Gleichungssystem lässt sich in kompakter Form schreiben :

$$(5) \quad \mathbf{Ax} - \mathbf{c} = \mathbf{r}$$

Dieses Gleichungssystem ist im Normalfall gar nicht mehr exakt lösbar, weil sich die Messfehler so auf die Kreisradien auswirken, dass sich die Kreise nicht mehr in einem Punkt schneiden. Es gibt nun aber ein mathematisches Verfahren, mit dem eine Lösung angenähert werden kann. Die Methode der kleinsten Quadrate (QR-Zerlegung) sucht einen Wert für (X_0/Y_0) so, dass die Länge des Vektors \mathbf{r} minimal wird [5]. Dabei kann (5) auch mehr als drei Gleichungen enthalten. Enthält (5) zum Beispiel sieben Gleichungen, bedeutet das für unser Positionierungsproblem, dass die Distanzen zu acht andern Knoten, die ihre Positionen schon kennen, in die Rechnung einfließen.

Simulator Grundgerüst

Vor Beginn dieser Arbeit bestand bereits die Grundform des Simulators mit einigen Basisfunktionen. Diese Grundform wird in diesem Kapitel erläutert [6].

2.1 Warum Simulation

Laut der Vorlesung Simulation in Produktion & Logistik ist der Einsatz von Simulation sinnvoll, wenn[7]:

- Auf einem Fachgebiet Neuland beschritten wird
- Die Grenzen analytischer Methoden erreicht sind
- Komplexe Wirkungszusammenhänge die menschliche Vorstellungskraft überfordern
- Das Experimentieren am realen Modell nicht möglich bzw. zu kostenintensiv ist.

Im Falle der Positionierung in wireless ad hoc Netzwerken sind alle vier Voraussetzungen erfüllt. Die Entwicklung von Algorithmen zur Positionsbestimmung in wireless ad hoc Netzwerken steckt noch in den Kinderschuhen. Die momentan denkbaren Algorithmen arbeiten mit vielfachen rückgekoppelten Iterationen, wo zusätzlich noch zufällige Fehler eine Rolle spielen. Ein analytischer Beweis für das Funktionieren scheint kaum möglich. Die gesamten Rückkoppelungen der einzelnen Knoten untereinander sprengen das menschliche Vorstellungsvermögen. Und ein Implementieren der Positionsalgorithmen auf Bluetooth Knoten ist um einiges aufwändiger, als das Programmieren desselben Algorithmus in einer modernen Programmiersprache in einem Simulator.

2.2 Simulatorkonzept

Eine Simulation kann in drei Hauptkomponenten unterteilt werden:

- Modellteil: In diesem Teil müssen die Eigenschaften der Umwelt des technischen Systems simuliert werden, indem sich dieses in der Realität gezwungenermaßen bewegen würde. Ein Beispiel: Bei einer Simulation für die Positionierung von Netzwerkknoten sind die realen (für die Knoten unbekannt und zu bestimmenden) Positionen der Knoten in diesem Teil gespeichert.
- Technikteil: Hier befinden sich alle Funktionen, die das technische System auch in der Realität besitzen wird. Ein Beispiel ist der Positionierungsalgorithmus. Er wird bei Realisation auf jedem Knoten zu finden sein.
- Kontrollteil: Dies ist das Interface zum Bediener der Simulation. Es steuert den Ablauf der Simulation und enthält Komponenten, die mit dem Modellteil zusammenspielen, wie auch Komponenten, die mit dem Technikteil zusammenspielen.

Auf dieser Dreiteilung basiert auch der Netzwerksimulator. Es gibt eine Klasse Controller (im File controller.h & .cpp), wo alle möglichen Befehle, die der User aufrufen kann, programmiert sind (Kontrollteil). Die Klasse SimulatePositions (im File simulatePositions.h & .cpp) enthält die Positionen der einzelnen Knoten und allgemeine Daten zur Simulation wie Fehlerrate für Berechnungen oder die Anzahl der Knoten im Netzwerk (Modellteil). Die Klasse PositionNode (im File positionNode.h & .cpp) stellt die Modellierung der Knoten selbst dar. Hier speichert ein Knoten seine eigene errechnete Position (Technikteil). Daneben gibt es noch weitere Files und Klassen, die für das optimale Zusammenspiel dieser drei Hauptklassen verantwortlich sind.

2.3 Simulatorbefehle

Der Simulator kennt verschiedene Befehle. Ein Teil davon bestand bereits vor dieser Arbeit, der andere Teil kam im Laufe dieser Arbeit hinzu.

2.3.1 Bestehende Befehle

Im Simulator, wie er vor dieser Arbeit bestand, waren folgende Befehle implementiert:

- help
Gibt alle vorhandenen Befehle aus.
- inquiry
Dieser Befehl benötigt noch als Argument einen spezifischen Knoten. Er wird also folgendermassen aufgerufen: ‚inquiry Node5‘
Falls es einen Node5 gibt, werden alle andern Knoten aufgelistet, die in der Reichweite von Node5 sind. Die Reichweite variiert aber (wird mittels der Error Rate gesteuert).
- reset
Mt diesem Befehl werden alle gemachten Berechnungen wieder gelöscht, d.h., nur fixe Knoten kennen ihre Position noch, normale Knoten verlieren ihr errechnetes Wissen über ihre Position wieder. Somit kann dasselbe Szenario immer wieder neu durchgerechnet werden, um das Verhalten des Algorithmus zu vergleichen.
- restart
Dieser Befehl dient dazu, ein neues Szenario in den Simulator zu laden. Als Verbesserung in dieser Arbeit kann neu entweder von einem File eingelesen werden oder es können Eckwerte für ein neues Szenario vorgegeben werden.
- fix
Dieser Befehl benötigt als Argument noch einen spezifischen Knoten. Mit ‚fix Node2‘ wird Node2 zu einem Anker gemacht, falls es Node2 im Netzwerk gibt. Dieser Knoten kennt nun seine reale Position.
- once
Dies ist die Urform der Positionierung. Jeder Knoten macht hier genau einmal, was bereits in Abschnitt 2.5 beschrieben wurde. Jeder Knoten holt sich die errechneten oder fixierten Positionen seiner erreichbaren Nachbarn und den Abstand zu ihnen (fehlerbehaftet wegen der Error Rate). Findet er dies von mindestens drei Knoten, führt er mit den gewonnenen Daten eine Triangulation mittels QR-Zerlegung durch und errechnet eine Position für sich.

- **info**
Liefert die grundlegenden Informationen eines Knotens auf den Bildschirm: seine reale Position, seine errechnete (oder fixierte, falls Anker) Position, und die Abweichung zwischen errechneter und wirklicher Position.
Wird der Befehl ohne Argument (,info') gestartet, werden die Informationen von allen Knoten aufgelistet. Man kann aber auch diejenigen Knoten spezifizieren, von denen man die Informationen haben will (,info Node3 Node7', nützlich bei einer grossen Anzahl Knoten).
- **quit**
Beendet die Simulation.

Dies waren die Befehle, die der Simulator vor bereits kannte. Im folgenden Abschnitt werden die Befehle vorgestellt, die im Zuge des Ausbaus des Simulators in dieser Arbeit neu implementiert wurden.

2.3.2 Neue Befehle

Es wurden folgende Befehle hinzugefügt:

- **save**
Speichert das Szenario im ersten Teil eines Files in einlesbarer Form, im zweiten Teil des Files stehen sodann die Ergebnisse der Berechnungen für jeden Knoten (vgl. Abschnitt 3.1.2)
- **terrain**
Durchführen von Positionsberechnungen mittels des Terrain-Algorithmus. Gibt man ,terrain 50' ein, wird fünfzig Mal zufällig ein Knoten ausgesucht und dieser führt dann, abhängig vom Zustand, in dem er sich befindet, eine Aktion durch. (vgl. Abschnitt 3.4)
- **data**
Der Befehl ,data Node7' gibt alle Informationen auf den Bildschirm aus, die über Node7 vorhanden sind. Das sind z. B. reale Position, berechnete absolute Position, Positionen in den einzelnen ABC-Koordinatensystemen, sowie die ganze History der einzelnen Positionen. Die History enthält alle je berechneten Werte zu einer Position.
- **set**
Mit diesem Befehl wird die Wahrscheinlichkeit, mit der bei normalen Knoten in den Zuständen $n \geq 3$ Refinement gemacht wird, bestimmt. Der Befehl wird so aufgerufen: ,set 0.7'. Damit wird die Wahrscheinlichkeit für Refinement in den erwähnten Zuständen auf 70% gesetzt (vgl. Abschnitt 3.4.1).

All diese Befehle (bestehende wie neue) können in die Kommandozeile nach dem Start des Simulators eingegeben werden.

Simulator Weiterentwicklung

Um die Möglichkeiten des Simulators zu erweitern, wurden einige Funktionen hinzugefügt. Zusätzlich wurde der Terrain-Algorithmus als Zustandsmaschine implementiert.

3.1 Verbesserungen

3.1.1 Random Init

Verschiedene Funktionen im Simulator benötigen Zufallszahlen. Diese werden von der c++ library-Funktion rand() geliefert. Wird diese Funktion nicht initialisiert, liefert sie immer wieder dieselben Zahlenreihen. Um das zu vermeiden, wird beim Start des Simulators nach irgendeiner ganzen Zahl gefragt, mit der die Initialisierung durchgeführt wird. Selbstverständlich muss diese Zahl bei jeder Initialisierung eine andere sein, ansonsten auch dieselben Zahlenreihen resultieren.

3.1.2 Szenario I/O

Vor dieser Arbeit hat der Simulator die Szenarien selber kreiert. Er hat dabei auf einem X- & Y-Field von 20x20 eine Anzahl von zwanzig Knoten zufällig verteilt. Die Error Rate war 0.1, und die Ranges der Knoten 10. Anker musste man nach dem Kreieren des Szenarios selber bestimmen (Befehl ,fix').

Nun wurden zwei Varianten geschaffen, wie man selber ein Szenario einbringen kann:

- Einlesen aus einem File
- Vorgabe von Eckwerten

Beim *Einlesen aus einem File* kann das Szenario wunschgemäss konfiguriert werden. Die Ausdehnung des X- & Y-Fields sowie die Error Rate muss bestimmt werden. Danach wird jedem Knoten manuell Position und Range zugewiesen. Schliesslich muss der Knotentyp bestimmt werden (normaler Knoten oder Anker). All diese Informationen muss man in einer klar definierten Form in ein .txt File schreiben, damit der Simulator in der Lage ist, das Szenario einlesen zu können. Dieses File kann gespeichert und immer wieder von neuem eingelesen werden, um das Verhalten des Algorithmus über viele Durchläufe auswerten zu können.

Bei der *Vorgabe von Eckwerten* fragt der Simulator den Benutzer, wie viele Knoten er auf welcher Feldgrösse verteilen soll. Die Lage der Knoten in diesem Feld wird anschliessend zufällig bestimmt. Man gibt auch die Error Rate an, und eine Range, die für alle Knoten gilt. Nachdem die Knoten verteilt sind, können einige von Hand mittels dem Befehl 'fix' zu Anker gemacht werden.

Für die Ausgabe der Szenarien und Resultate steht der Befehl 'save' zur Verfügung. Er wird ohne Argument aufgerufen (,save') und schreibt Daten in ein File. Das File besteht aus zwei Teilen:

Im ersten Teil wird das aktuelle Szenario in exakt der Form gespeichert, in der es wieder eingelesen werden kann (vgl. *einlesen aus einem File*). Man kann also mittels *Vorgabe von Eckwerten* ein Szenario kreieren und mittels des Befehls ,save' in ein File schreiben. Löscht man in diesem File den zweiten Teil und gibt dem File den Namen des Input-Files, kann man das kreierte Szenario später wieder einlesen. Das Szenario ist damit nach dem Beenden des Simulators nicht verloren.

Im zweiten Teil stehen die Resultate der Positionsberechnungen für jeden Knoten: die reale Position, die errechnete Position, die Anzahl der Berechnungen, die jeder Knoten gemacht hat, sowie die Range und ob ein Knoten ,fix' ist oder nicht.

3.1.3 Log-File

Die Log-Funktion ist eine weitere Ausgabefunktion des Simulators. Sie läuft selbständig im Hintergrund und kann vom Bediener nicht explizit mittels einem Befehl aufgerufen werden. Stattdessen weiss die Simulation zu jedem Befehl, welche Werte sie aufzeichnen muss. Immer, wenn der Bediener einen Befehl ausführt, wird im Hintergrund automatisch das Ausführen dieses Befehles in einem File aufgezeichnet und die für diesen Befehl relevanten Daten dazugeschrieben. Dies ermöglicht das Auswerten des Files nach einer Testreihe. Das File muss aber nach Beenden des Simulators umbenannt oder verschoben werden, weil es bei einem Neustart vom neuen Log-File überschrieben wird.

3.2 Terrain Algorithmus / Prinzip der Zustandsmaschine

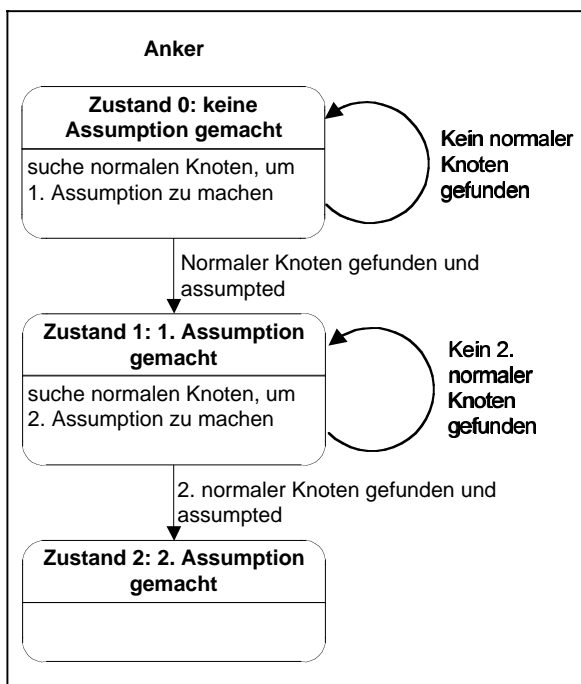


Abbildung 4:
Prinzip des Terrain-Algorithmus als Zustandsmaschine für Anker.
(ACS: ABC-Koordinatensystem)

Bisher konnte der Simulator nur mit dem Befehl ,once' Positionen bestimmen. Jeder Knoten hat dazu versucht, mindestens drei erreichbare Knoten in seiner Nachbarschaft zu finden, die entweder settled oder fixed sind. Es gibt hier ein Problem beim Start des Algorithmus. Zu Beginn ist noch kein Knoten settled. Man benötigt daher mindestens drei Anker. Sind diese aber unglücklich verteilt, besteht die Möglichkeit, dass für keinen normalen Knoten drei Anker erreichbar sind. Der Algorithmus startet folglich nie.

Der Terrain-Algorithmus löst dieses Problem, in dem er nicht von Anfang an versucht, absolute Positionen zu berechnen, sondern zuerst nur relative Positionen der Knoten untereinander bestimmt. Er führt zuerst den sogenannten ABC-Algorithmus (Assumption Based Coordinates) durch [8] (vgl. auch Abbildungen 4 & 5).

Der ABC-Algorithmus funktioniert folgendermassen: Jeder Anker setzt sich selbst in den Ursprung (0/0) seines relativen Koordinatensystems (in Zukunft ABC-Koordinatensystem genannt). Dann sucht er einen normalen Knoten und berechnet die Distanz r_{01} zu diesem Knoten (Signalstärkemessung). Diesem Knoten werden die Koordinaten $(r_{01}/0)$ zugewiesen und damit die sogenannte erste Assumption, er wird also ins ABC-Koordinatensystem dieses Ankers assumed. Das ABC-Koordinatensystem wird damit so in der Ebene ausgerichtet, dass dieser Knoten auf der x-Achse liegt. Dann sucht der Anker einen weiteren normalen Knoten für die sogenannte zweite Assumption, dieser Knoten wird somit auch assumed. Durch Distanzmessungen können dessen Koordinaten im ABC-Koordinatensystem berechnet werden:

$$X_2 = \frac{-(r_{12}^2 - r_{01}^2 - r_{02}^2)}{2 \cdot r_{01}} \quad Y_2 = \sqrt{r_{02}^2 - X_2^2}$$

r_{01} : Distanz von Anker zu 1.Assumption
 r_{02} : Distanz von Anker zu 2.Assumption
 r_{12} : Distanz von 1.Assumption zu 2.Assumption

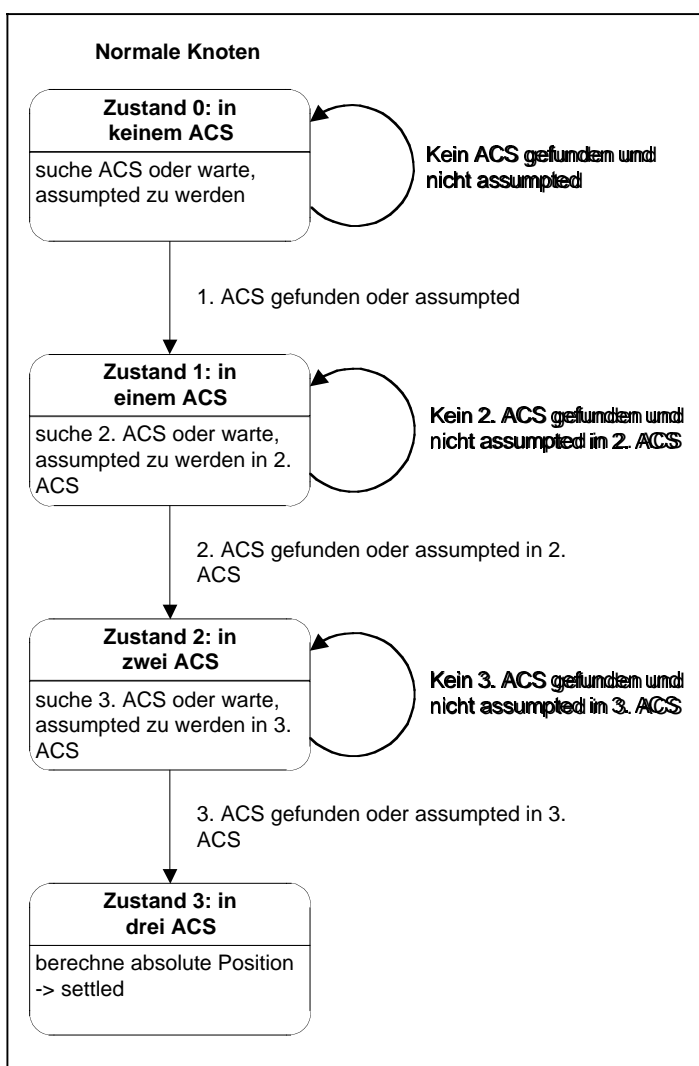


Abbildung 5: Prinzip des Terrain-Algorithmus als Zustandsmaschine für normale Knoten. (ACS: ABC-Koordinatensystem)

Jetzt kennen drei Knoten ihre Position im ABC-Koordinatensystem. Andere normale Knoten können von nun an ihre Position in diesem ABC-Koordinatensystem dank Triangulation mittels QR-Zerlegung berechnen. Dieses ABC-Koordinatensystem breitet sich durchs ganze Netzwerk aus. Weil der Anker die Koordinate (0/0) hat, kennt jeder Knoten, der in diesem ABC-Koordinatensystem eine Position hat, den Abstand zum zugehörigen Anker.

Von jedem Anker breiten sich auf diese Weise ABC-Koordinatensysteme durch das Netzwerk aus. Jeder normale Knoten versucht, in möglichst viele solcher ABC-Koordinatensysteme hineinzukommen (=seine Position im ABC-Koordinatensystem zu bestimmen). Hat ein normaler Knoten eine Position in drei verschiedenen ABC-

Koordinatensystemen errechnet, kennt er die Distanz zu drei Anker. Es ist nun ohne weiteres

möglich, bei der Ausbreitung eines ABC-Koordinatensystems immer die reale Position des zugehörigen Ankers mitzugeben. Damit kennt der Knoten nicht nur die Distanzen zu drei Anker, sondern auch die realen Positionen dieser Anker. Er kann also wieder durch Triangulation mittels QR-Zerlegung seine Position berechnen. Dies aber ist nun eine

absolute Position, der Knoten wird damit settled und hat theoretisch sein Positionierungsproblem gelöst (warum nur theoretisch, siehe Abschnitt 3.3).

All diese Berechnungen finden im realen Netzwerk nicht sequentiell statt, sondern parallel. Es werden mehrere Anker gleichzeitig versuchen, eine Assumption vorzunehmen. Einige normale Knoten können bereits an der Berechnung ihrer absoluten Position sein, während andere noch nicht einmal in einem einzigen ABC-Koordinatensystem ihre Position kennen. Um das ganze etwas klarer zu machen, kann man eine Zustandsmaschine ausarbeiten, die auf jedem Knoten läuft. Anker (Abbildung 4) haben eine andere Zustandsmaschine als normale Knoten (Abbildung 5). Man verlässt also die Netzwerksicht zugunsten der Knotensicht.

3.3 Zustandsmaschine, die Messfehler berücksichtigt

Die im vorigen Abschnitt vorgestellten Zustandsmaschinen zeigen nur das Prinzip des Terrain-Algorithmus. Ungenauigkeiten in der Berechnung der Positionen infolge Messfehler sind aber ausgeblendet worden. Es ist davon auszugehen, dass die errechneten Positionen sowohl bei den Positionen in ABC-Koordinatensystemen wie bei den absoluten Positionen mehr oder minder ungenau sind.

Die berechneten Positionen müssen folglich immer wieder upgedatet werden. Dadurch erhofft man sich, dass sich zufällige Messfehler im Mittel aufheben und man sich somit der realen Position immer mehr annähert [8]. Die beiden Zustandsmaschinen müssen daher um diese Updates erweitert werden. Es resultieren folgende, wesentlich umfangreichere Zustandsmaschinen:

Bei den Ankern müssen die Assumptions upgedatet werden. Im Zustand 1 wird entweder die 1. Assumption upgedatet oder versucht, die 2. Assumption vorzunehmen, im Zustand 2 wird eine der beiden vorgenommenen Assumptions upgedatet (vgl. Abbildung 6).

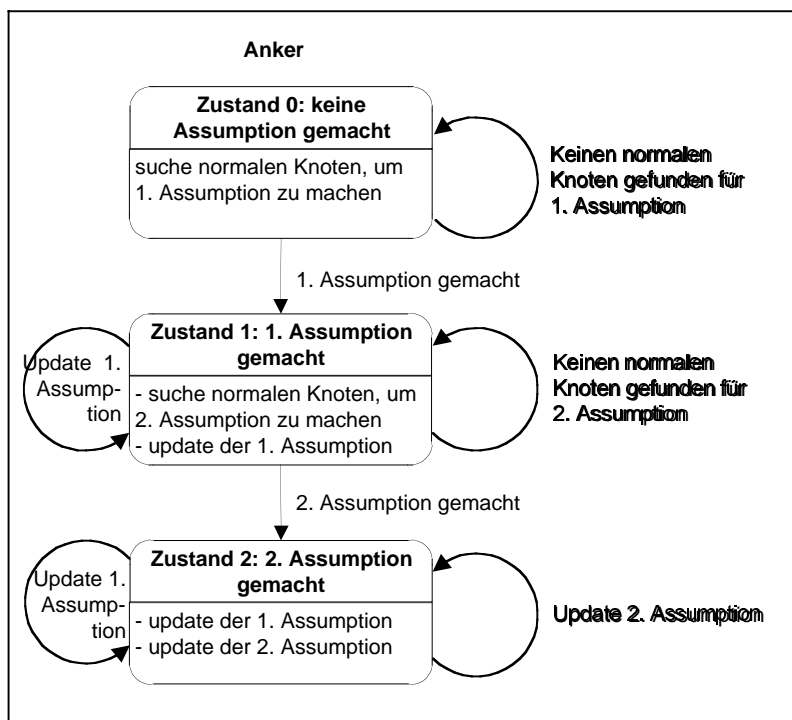
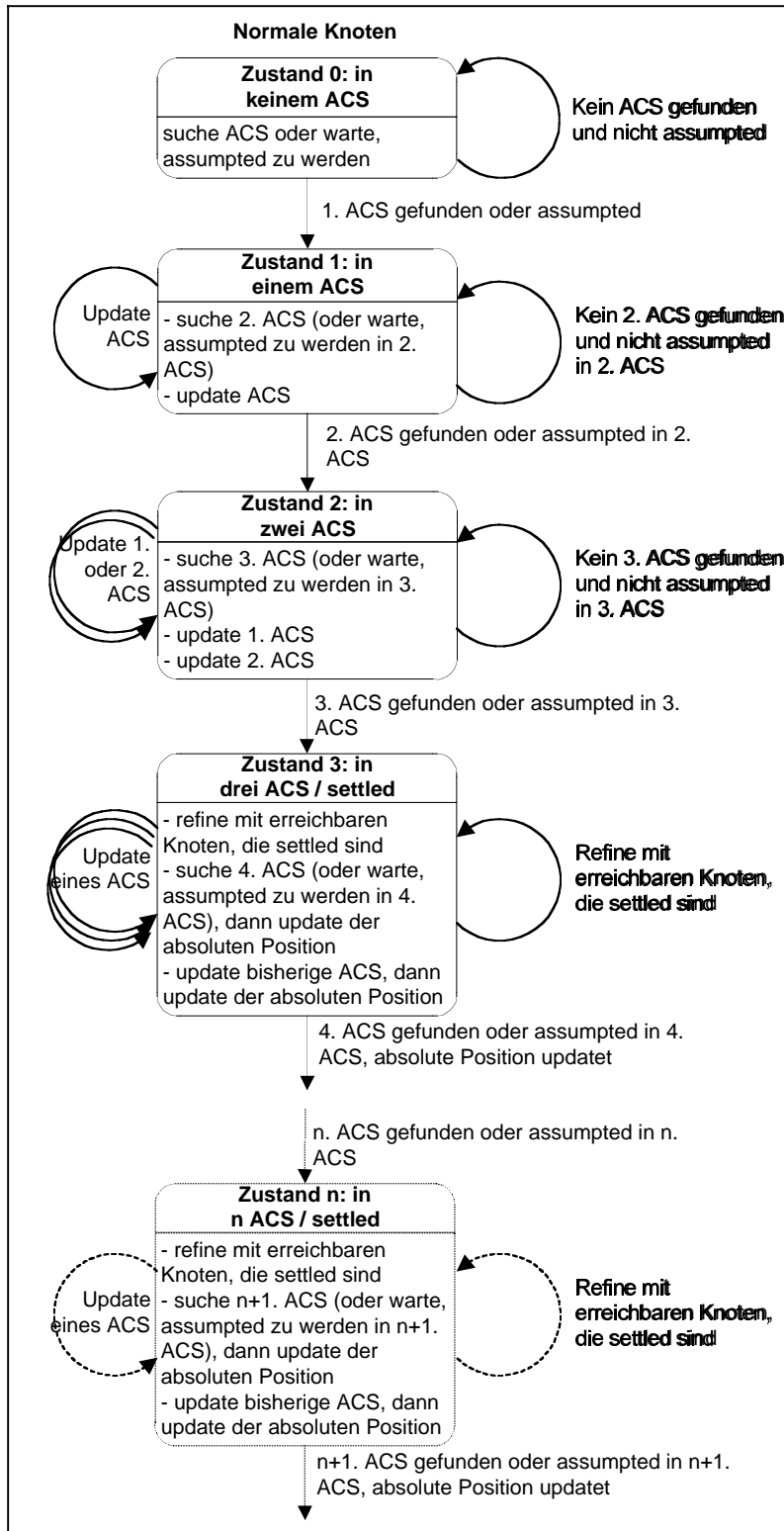


Abbildung 6: Terrain-Algorithmus als Zustandsmaschine für Anker, die Messungenauigkeiten mittels Updates begegnet.

Hat das Netzwerk total k Knoten, und sind davon a Anker, hat die Zustandsmaschine für die normalen Knoten $a+1$ Zustände (der Zustand gibt an, in wievielen ABC-Koordinatensystemen ein normaler Knoten ist). Sobald ein normaler Knoten im Zustand 3 oder darüber ist, folgt jedem Update eines ABC-Koordinatensystems oder dem Hinzukommen eines ABC-Koordinatensystems eine Neuberechnung der absoluten Position und damit auch ein Update.



Ein settled Knoten kann seine absolute Position noch auf andere Art und Weise updaten. Sobald mehrere Knoten eine absolute Position haben, muss ein Knoten nicht mehr auf seine ABC-Koordinatensysteme zurückgreifen, um sich upzudaten. Stattdessen kann er die Distanzen zu den andern Knoten, die bereits eine absolute Position haben, berechnen und mit den absoluten Positionen dieser Knoten seine eigene absolute Position neu bestimmen dank Triangulation mittels QR-Zerlegung (vgl. Abschnitt 1.5.3).

Abbildung 7: Terrain-Algorithmus als Zustandsmaschine für normale Knoten, die Messungenauigkeiten mittels Updates begegnet. (ACS: ABC-Koordinatensystem)

3.4 Implementation der Zustandsmaschine

Obwohl vom Prinzip des Terrain Algorithmus in Abschnitt 3.2 zur realen Form in Abschnitt 3.3 ein grosser Schritt in Richtung Funktionsfähigkeit unter realen Bedingungen (keine genauen Messungen möglich) gemacht wurde, blieb etwas immer noch ungeklärt: das Scheduling der einzelnen möglichen Aktionen eines Zustandes. Mit andern Worten: welche Aktion wird gemacht, wenn ein Knoten eine Aktion machen will (oder soll im Simulator). Je nach Zustand hat ein Knoten mehrere mögliche Aktionen, aus denen er auswählen kann.

Beispiel: mögliche Aktionen im Zustand 3 bei normalen Knoten

- Refine mit erreichbaren Knoten, die ebenfalls settled sind
- Update eines ABC-Koordinatensystems, dann Update der absoluten Position
- Suche eines zusätzlichen ABC-Koordinatensystems, dann Update der absoluten Position

Welche dieser drei Aktionen wird durchgeführt?

Eine Möglichkeit wäre, für jeden Zustand eine Scheduling-Regel zu entwerfen. Es zeigt sich aber, dass dies nicht für alle Zustände nötig ist.

3.4.1 Normale Knoten

Betrachten wir das Suchen eines zusätzlichen ABC-Koordinatensystems (in irgendeinem Zustand eines normalen Knoten). Damit ein Knoten seine Position im ABC-Koordinatensystem von Anker A berechnen kann, muss er mindestens drei andere Knoten in seiner Reichweite finden, die bereits eine Position in diesem ABC-Koordinatensystem haben. Da er nicht weiss, in welchen Knoten er welches ABC-Koordinatensystem finden wird, muss er alle erreichbaren Knoten fragen, in welchen ABC-Koordinatensystemen sie drin sind. Eventuell wird er dann von drei oder mehr Knoten (man braucht mindestens drei,

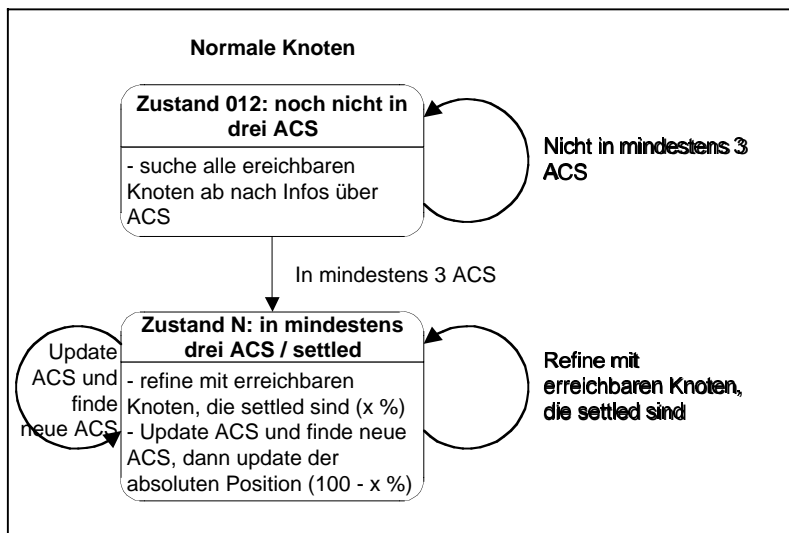


Abbildung 8:
Implementierte Form des Terrain-Algorithmus als Zustandsmaschine für normale Knoten. x ist einstellbar mit dem Befehl 'set'.
(ACS: ABC-Koordinatensystem)

um Triangulieren zu können) die gewünschten Infos über das ABC-Koordinatensystem von Anker A erhalten, es kann aber auch das Gegenteil der Fall sein. Es ist auch denkbar, dass er nur Infos über das ABC-Koordinatensystem von Anker B erhält. Oder Infos über die ABC-Koordinatensysteme von Anker A und Anker C. Vor seiner Anfrage weiss der fragende

Knoten nicht, was er für Informationen bekommen wird. Es wäre nun schade, alle Informationen, die nicht das ABC-Koordinatensystem von Anker A betreffen, nicht weiterzuverwenden. Viel besser ist es, die gesamten Informationen zu verwerten. Das kann dann heissen, dass ein Knoten beim Abfragen aller erreichbaren Knoten sowohl neue ABC-

Koordinatensysteme findet als auch solche, in denen er schon eine Position hat. Diejenigen, die er neu findet, werden seinen eigenen Daten hinzugefügt, diejenigen, in denen er schon eine Position hat, werden upgedatet.

Dies bedeutet, dass in einer Aktion sowohl mehrere neue ABC-Koordinatensysteme gefunden werden können als auch mehrere bereits bekannte upgedatet. Die Zustände 0, 1 & 2 der normalen Knoten können damit in einem Zustand zusammengefasst werden (dieser Zustand heisst in der Simulation `normal_states012`). Der Knoten sucht in diesem neuen Zustand in allen erreichbaren Knoten nach Informationen über ABC-Koordinatensysteme, und verwertet sie abhängig von seinen bereits vorhandenen Daten: neue Koordinatensysteme werden hinzugefügt, bereits bekannte upgedatet. Das Absuchen der erreichbaren Knoten und das Ordnen der gewonnenen Daten übernimmt in der Simulation die Funktion `searchCalculateUpdateABC`. Solange das Total aller bekannten ABC-Koordinatensysteme in einem Knoten kleiner als drei ist, bleibt der Knoten in diesem Zustand (Zustand `normal_states012`) und sucht weiterhin nach zusätzlichen ABC-Koordinatensystemen und findet dabei auch bereits bekannte, die er gleich updatet.

Hat er drei oder mehr gefunden, berechnet der Knoten seine absolute Position und wechselt dann in den Zustand 3 (respektive $n > 3$, wenn er im Total n Koordinatensysteme gefunden hat). Dieser Zustand heisst in der Simulation `normal_statesN`.

In den Zuständen $n \geq 3$ präsentiert sich die Situation dann anders. Der Knoten hat jetzt eine absolute Position berechnet. Hier kann einerseits weiterhin nach neuen und bereits bekannten ABC-Koordinatensystemen gesucht werden (wird auch in diesen Zuständen von der Funktion `searchCalculateUpdate` übernommen). Andererseits kann ein Knoten seine absolute Position auch mittels Refining updaten. Welche dieser zwei Möglichkeiten ausgeführt wird, muss in einer Scheduling-Regel festgelegt werden.

Die programmierte Scheduling-Regel beruht auf Wahrscheinlichkeiten. Diese können eingestellt werden. Beim Einlesen eines Szenario ist die Wahrscheinlichkeit für Refinement bei 80%, für Update via ABC-Koordinatensysteme dementsprechend bei 20%. Den zuständige Parameter kann man mit dem Befehl `set` ändern. Tippt man `,set 0.6'`, setzt man die Wahrscheinlichkeit von Refinement auf 60%, entsprechend diejenige von Update via ABC-Koordinatensysteme automatisch auf 40%.

3.4.2 Anker

Bei Anker gibt es im Zustand 0 kein Scheduling. Der Zustand hat nur eine mögliche Aktion, das Suchen eines normalen Knotens für die 1. Assumption.

Im Zustand 1 gibt es zwei mögliche Aktionen. Entweder wird ein normaler Knoten für die 2. Assumption gesucht, oder die 1. Assumption wird upgedatet. Da das Starten des Algorithmus davon abhängt, dass die zweite Assumption gemacht wurde, wird auf das Update der 1. Assumption verzichtet im Zustand 1. Zuerst sollen beide Assumptions gemacht werden, damit die ABC-Koordinatensysteme sich ausbreiten können, bevor upgedatet wird. So braucht es im Zustand 1 auch keine Scheduling-Regel, weil nur noch eine Aktion vorhanden ist.

Im Zustand 2 wird nur noch upgedatet. Entweder wird die 1. Assumption oder die 2. Assumption upgedatet. Welche gewählt wird, ist zufällig, beide haben eine Wahrscheinlichkeit von 50%.

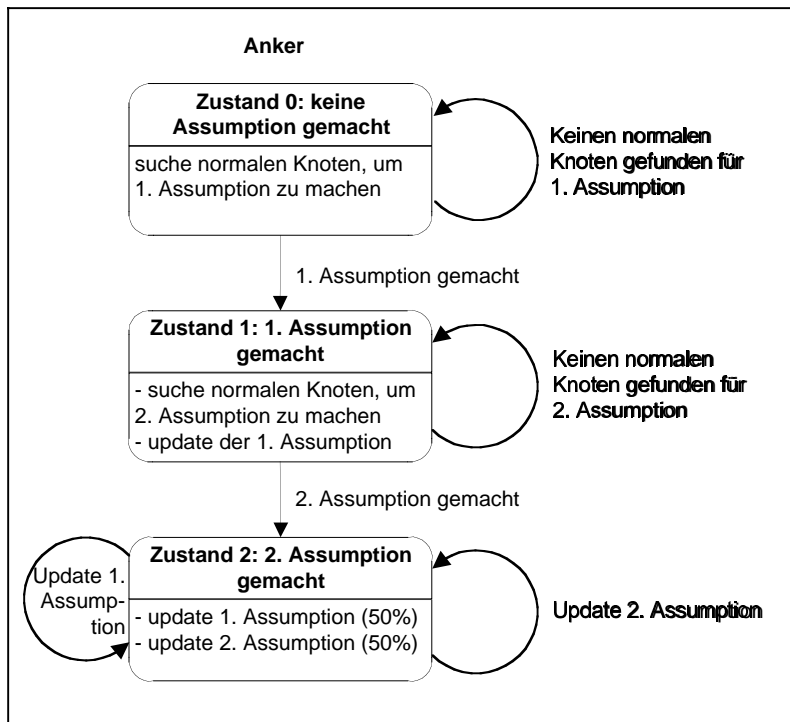


Abbildung 9:
Implementierte Form des Terrain-Algorithmus als Zustandsmaschine für Anker.

3.4.3 Update

In Abschnitt 3.3 wurde immer wieder von Update gesprochen. Was versteht man darunter? Update heisst, dass eine bereits berechnete Position neu trianguliert und berechnet wird. Das Ziel ist, dass sich im Durchschnitt die Fehler aufheben oder zumindest verringern. In [8] wird gezeigt, dass man damit Verbesserungen erzielt gegenüber den einzelnen Berechnungen.

Ein einfaches Beispiel: Knoten A hat in der realen Position einen X-Wert von 5. In der ersten Berechnung kommt man auf einen X-Wert von 4.5. Beim Update bekommt man einen X-Wert von 5.3. Der Durchschnitt dieser beiden Berechnungen ergibt einen X-Wert von 4.9, was besser ist als die Werte der beiden einzelnen Berechnungen.

Der Simulator zeichnet für jede berechnete Position alle jemals triangulierten Werte auf (sowohl für absolute wie für relative Positionen). Beim Update einer Position ergibt sich die upgedatete Position wie folgt: die upgedatete Position ist der Mittelwert der neuen Triangulation und aller bisherigen Triangulationen für diese Position. Diese Mittelwertbildung wird in der Funktion calculateFromHistory vorgenommen. Will man zu einem späteren Zeitpunkt eine andere Gewichtung der einzelnen Werte als nur den Durchschnitt aller Werte, braucht man nur diese Funktion zu ändern.

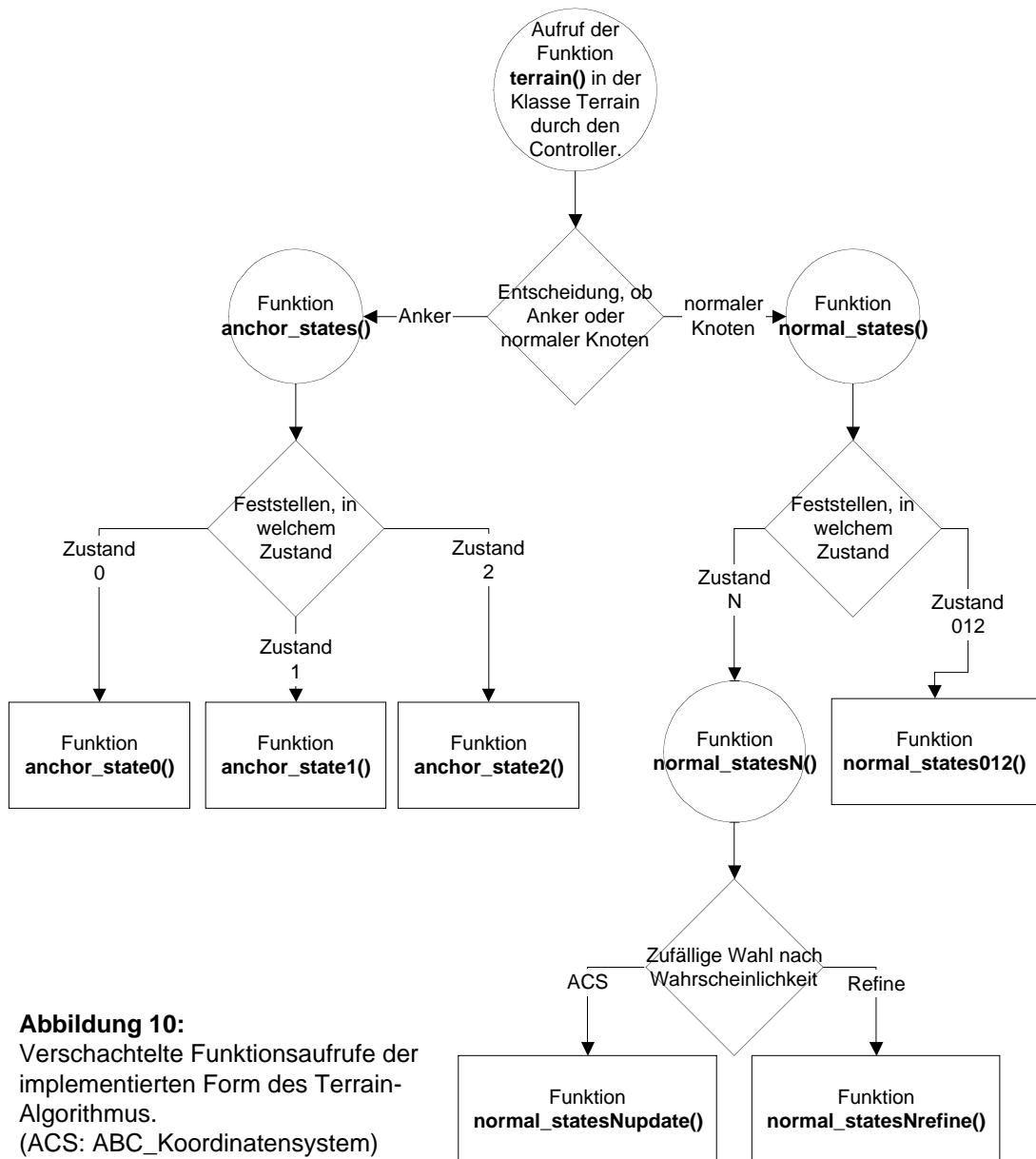


Abbildung 10: Verschachtelte Funktionsaufrufe der implementierten Form des Terrain-Algorithmus. (ACS: ABC_Koordinatensystem)

Bedienungsanleitung

Um ein schnelles Einarbeiten in den Simulator zu ermöglichen, werden einige Screenshots erklärt. Die Reihenfolge entspricht mehr oder weniger einem möglichen Simulationsdurchlauf.

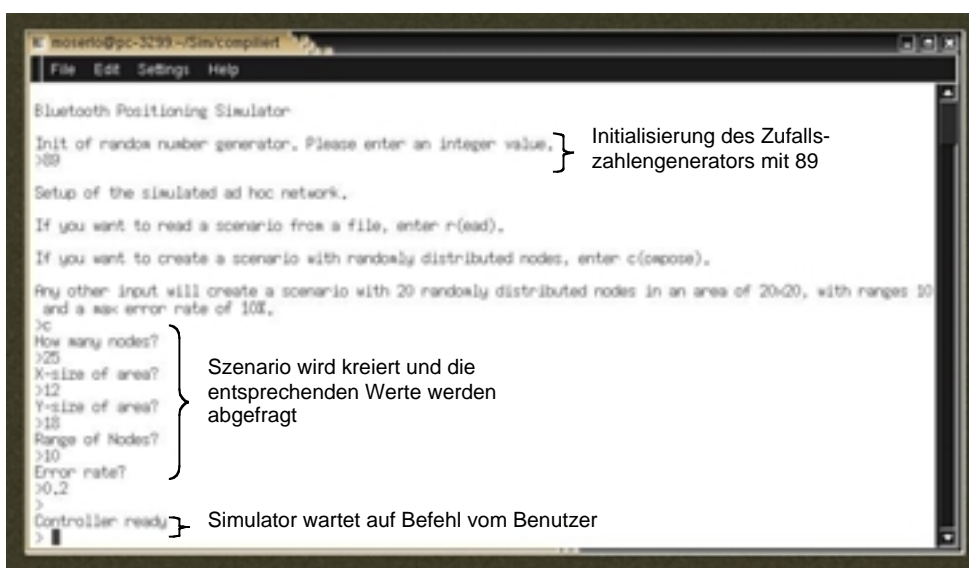
Der Simulator muss zuerst kompiliert werden. In der Shell kann man einfach den Befehl ‚make‘ im Verzeichnis, wo sich die Source-Files befinden, ausführen. Die Verlinkung der einzelnen Source-Files geschieht automatisch.

Ist die Compilation ohne Fehler abgeschlossen, kann mit dem Befehl ‚./simulator‘ (ebenfalls im Verzeichnis der Source-Files) der Simulator gestartet werden.

4.1 Kreieren eines Szenarios

Der Simulator fragt zuerst nach der Initialisierung des Zufallszahlengenerators. Wir geben in unserem Beispiel 89 ein (vgl. Screenshot 1).

Dann wird gefragt, ob man ein Szenario aus einem File lesen möchte oder ob man ein Szenario kreieren will. Wir entscheiden uns für das Kreieren eines Szenarios (Eingabe ‚c‘). Schrittweise fragt nun die Simulation nach der gewünschten Anzahl Knoten, nach der Grösse des X- & Y-Field, nach den Ranges der Knoten und nach der Error Rate (vgl. Screenshot 1). Ist das alles eingegeben, verteilt die Simulation die gewünschte Anzahl Knoten zufällig im durch X- & Y-Field aufgespannten Bereich (nicht sichtbar). Jetzt ist die Simulation bereit und wartet auf einen Befehl.



```

moserle@pc-3293 ~/Sim/compiled
File Edit Settings Help

Bluetooth Positioning Simulator
Init of random number generator. Please enter an integer value.
>89
Setup of the simulated ad hoc network.
If you want to read a scenario from a file, enter r(read).
If you want to create a scenario with randomly distributed nodes, enter c(create).
Any other input will create a scenario with 20 randomly distributed nodes in an area of 20x20, with ranges 10
and a max error rate of 10%.
>c
How many nodes?
>25
X-size of area?
>12
Y-size of area?
>15
Range of Nodes?
>10
Error rate?
>0.2
>
Controller ready
>
  
```

Initialisierung des Zufallszahlengenerators mit 89

Szenario wird kreiert und die entsprechenden Werte werden abgefragt

Simulator wartet auf Befehl vom Benutzer

Screenshot 1: Simulation wird bereit gemacht mittels kreieren eines Szenarios

4.2 Einlesen eines Szenarios

Anstatt ein Szenario zu kreieren, kann ein Szenario auch aus einem File eingelesen werden. Man gibt nach der Initialisierung des Zufallszahlengenerators einfach ‚r‘ ein anstatt ‚c‘. Der Simulator liest dann das File ‚/I_O/in.txt‘ ein. Wie der Pfad schon sagt, muss sich das File ‚in.txt‘ im Verzeichnis ‚I_O‘ befinden. Dieses Verzeichnis sollte im selben Verzeichnis sein wie die Source-Files. Findet der Simulator dieses File nicht, bricht er ab. Hat der Simulator das File eingelesen, ist er bereit und wartet auf einen Befehl.

```

Bluetooth Positioning Simulator
Init of random number generator. Please enter an integer value.
>978
Setup of the simulated ad hoc network.
If you want to read a scenario from a file, enter r(read).
If you want to create a scenario with randomly distributed nodes, enter c(ompose).
Any other input will create a scenario with 20 randomly distributed nodes in an area of 20x20, with ranges 10
and a max error rate of 10%.
>r
>
Controller ready
>
  
```

Screenshot 2: Simulator wird bereit gemacht mittels einlesen eines Szenarios

Es ist wichtig, die Daten im File ‚in.txt‘ korrekt anzuordnen, damit der Simulator das File lesen kann: In der ersten Zeile steht X-Field, Y-Field, Error Rate, jeweils getrennt durch einen Leerschlag. In den folgenden Zeilen steht in jeder Zeile genau ein Knoten: zuerst sein Name, dann seine X-Position, seine Y-Position, seine Range, und zuletzt, ob ein Knoten ein Anker ist oder ein normaler Knoten. Getrennt werden müssen die einzelnen Werte mit mindestens einem Leerschlag, es können aber auch mehrere sein.

```

X-Field = 15
Y-Field = 10
Error Rate = 0.2
Node1 12.19 9.22 8 1
Node2 8.67 3.57 8 0
Node3 9.46 5.62 8 0
Node4 2.06 7.24 8 0
Node5 14.62 1.42 8 1
Node6 5.58 1.51 8 0
Node7 13.78 4.17 8 1
Node8 11.83 9.09 8 0
Node9 0.98 4.13 8 0
Node10 8.76 5.56 8 0
  
```

Screenshot 3: Beispiel eines ‚in.txt‘-Files. Hat ein Knoten in der letzten Spalte eine 1, ist er ein Anker, hat er dort eine 0, ist er ein normaler Knoten.

4.3 Der Befehl ‚help‘

Nachdem der Simulator bereit gemacht wurde, wartet er auf eine Befehlsausgabe. Mittels ‚help‘ kann man sich alle zur Verfügung stehenden Befehle anzeigen lassen. Zuerst wird der Befehl angezeigt, der in die Kommandozeile der Shell eingegeben werden muss. Dann folgt, falls nötig, der Parameter zu diesem Befehl. In der zweiten Zeile wird erklärt, wozu der Befehl dient.



```
Node67 post 5.35, 5.39 estimated 17.57, -2.57 dist 15.26
> help
--- help function ---
- quit
  Quit the program
- help
  Prints a help text
- inquiry nodeName
  List all nodes reachable by nodeName
- once
  Every node is estimating his position once
- reset
  Reset all nodes
- restart
  Restart the simulation with new data
- info [nodeName1 nodeName2 ...]
  Prints the information about the specified nodes. No arguments: all nodes
- fix nodeName
  Fix the node to it's real position
- save
  save scenario into file
- terrain numberOfCalcs(int)
  number of randomly chosen nodes trying to make a calculation
- data nodeName
  prints all data of nodeName
- set resolution
  sets ratio of update by refining
```

Screenshot 4: Mit dem Befehl ‚help‘ sieht man alle zur Verfügung stehenden Befehle.

4.4 Der Befehl ‚terrain‘

Mit dem Befehl terrain wird der Terrain-Algorithmus gestartet. Er benötigt als Argument die Anzahl Wiederholungen des Terrain-Algorithmus. ‚terrain 10‘ sucht zehn Mal zufällig irgendeinen Knoten aus allen Knoten aus und führt jeweils eine Aktion durch, die im aktuellen Zustand dieses ausgewählten Knotens zulässig ist. Der ausgewählte Knoten und die durchgeführte Aktion werden auf dem Bildschirm ausgegeben.

Wie bereits in Abschnitt 4.4.1 ausgeführt wurde, kann in einigen Zuständen eine Aktion aus mehreren Teilaktionen bestehen. Ab der zweiten Teilaktion werden diese Teilaktionen nach rechts verschoben ausgegeben. So sieht man auf einen Blick, welche Knoten eine Aktion mit mehreren Teilaktionen ausführen.



```
> terrain 10
Nodes executing terrain algorithm:
Node13(normal node): found abc coord system which was assumed to it from Node1, doing nothing.
Node13(normal node): updated own position in abc coord system of anchor Node11.
Node13(normal node): found abc coord system which was assumed to it from Node15, doing nothing.
Node13(normal node): updated global position with abc info: 12.37, 5.21
Node5(normal node): global position refined with neighbour settled nodes: 11.38, 7.98
Node7(normal node): updated own position in abc coord system of anchor Node1.
Node7(normal node): updated own position in abc coord system of anchor Node11.
Node7(normal node): calculated own position in abc coord system of anchor Node15 and stored.
Node7(normal node): calculated global position: 6.33, 5.55
Node7(normal node): updated own position in abc coord system of anchor Node1.
Node7(normal node): updated own position in abc coord system of anchor Node11.
Node7(normal node): updated own position in abc coord system of anchor Node15.
Node7(normal node): updated global position with abc info: 4.58, 5.09
Node15(normal node): global position refined with neighbour settled nodes: 11.97, 5.02
Node14(normal node): global position refined with neighbour settled nodes: 12.62, 4.71
Node2(normal node): global position refined with neighbour settled nodes: 11.35, 6.94
Node16(normal node): global position refined with neighbour settled nodes: 8.48, 5.42
Node15(normal node): global position refined with neighbour settled nodes: 6.98, -7.38
Node12(normal node): updated own position in abc coord system of anchor Node1.
Node12(normal node): updated own position in abc coord system of anchor Node11.
Node12(normal node): updated own position in abc coord system of anchor Node15.
Node12(normal node): updated global position with abc info: -8.33, 0.32
```

Screenshot 5: Mit dem Befehl ‚terrain 10‘ wird der Terrain-Algorithmus zehn Mal ausgeführt.

4.5 Der Befehl ‚info‘

Hat man mittels dem Terrain-Algorithmus Berechnungen durchgeführt, kann man die Resultate mit dem Befehl ‚info‘ auf den Bildschirm ausgeben. Man sieht zeilenweise zuerst die reale Position eines Knotens. Dann wird angegeben, ob der Knoten ein Anker ist (‚external‘ im Screenshot 6) oder ein normaler Knoten. Normale Knoten können noch keine Berechnung gemacht haben (‚no estimated position‘ im Screenshot 6) oder dann wird ihre berechnete Position ausgegeben (‚estimated‘ im Screenshot 6). Am Schluss jeder Zeile steht die Distanz von der realen Position zu der errechneten Position und liefert eine Aussage über die Genauigkeit (wäre im Idealfall 0).

```
Node15(normal node): global position refined with neighbour settled nodes: 11,6
> info
Node1 pos: 6.28, 5.62 external: 6.28, 5.62 dist: 0.00
Node2 pos: 2.10, 3.28 estimated: 7.59, 12.32 dist: 10.63
Node3 pos: 3.01, 6.59 estimated: -5.38, 3.66 dist: 8.98
Node4 pos: 8.48, 1.60 no estimated position
Node5 pos: 0.26, 3.42 no estimated position
Node6 pos: 8.72, 2.53 estimated: 18.04, 8.17 dist: 10.89
Node7 pos: 0.74, 0.41 no estimated position
Node8 pos: 5.63, 6.92 no estimated position
Node9 pos: 4.12, 3.47 estimated: 3.59, 6.88 dist: 3.45
Node10 pos: 3.22, 2.21 estimated: -9.92, -2.35 dist: 13.91
Node11 pos: 6.06, 9.54 external: 6.06, 9.54 dist: 0.00
Node12 pos: 2.75, 0.21 estimated: -2.42, 2.28 dist: 5.57
Node13 pos: 7.56, 3.43 estimated: 12.54, 5.68 dist: 5.47
Node14 pos: 9.81, 9.04 estimated: 19.66, 6.44 dist: 10.19
Node15 pos: 4.88, 8.62 external: 4.88, 8.62 dist: 0.00
Node16 pos: 9.43, 1.16 estimated: 36.99, 9.16 dist: 28.70
Node17 pos: 4.23, 1.54 estimated: 4.73, 8.07 dist: 6.55
Node18 pos: 4.44, 7.25 estimated: 5.44, 5.56 dist: 1.97
Node19 pos: 8.12, 2.92 estimated: 20.41, 7.99 dist: 13.14
Node20 pos: 8.85, 8.39 estimated: 17.57, -2.37 dist: 13.86
```

Knoten 1 ist ein Anker

Knoten 5 hat noch keine Position errechnet

Knoten 13 hat eine Position errechnet mit einer Abweichung von 5.47 zu der realen Position

Screenshot 6: Ausgabe des Befehls ‚info‘

4.6 Die Log-Funktion

Die Log-Funktion läuft im Hintergrund und zeichnet jeden ausgeführten Befehl und dazugehörige Daten im File ‚/I_O/log.txt‘ auf. Man könnte sie auch Black-Box des Simulators nennen. Das File ‚log.txt‘ findet man im Verzeichnis ‚I_O‘, und dieses befindet sich im Verzeichnis, wo die Source-Files des Simulators sind.

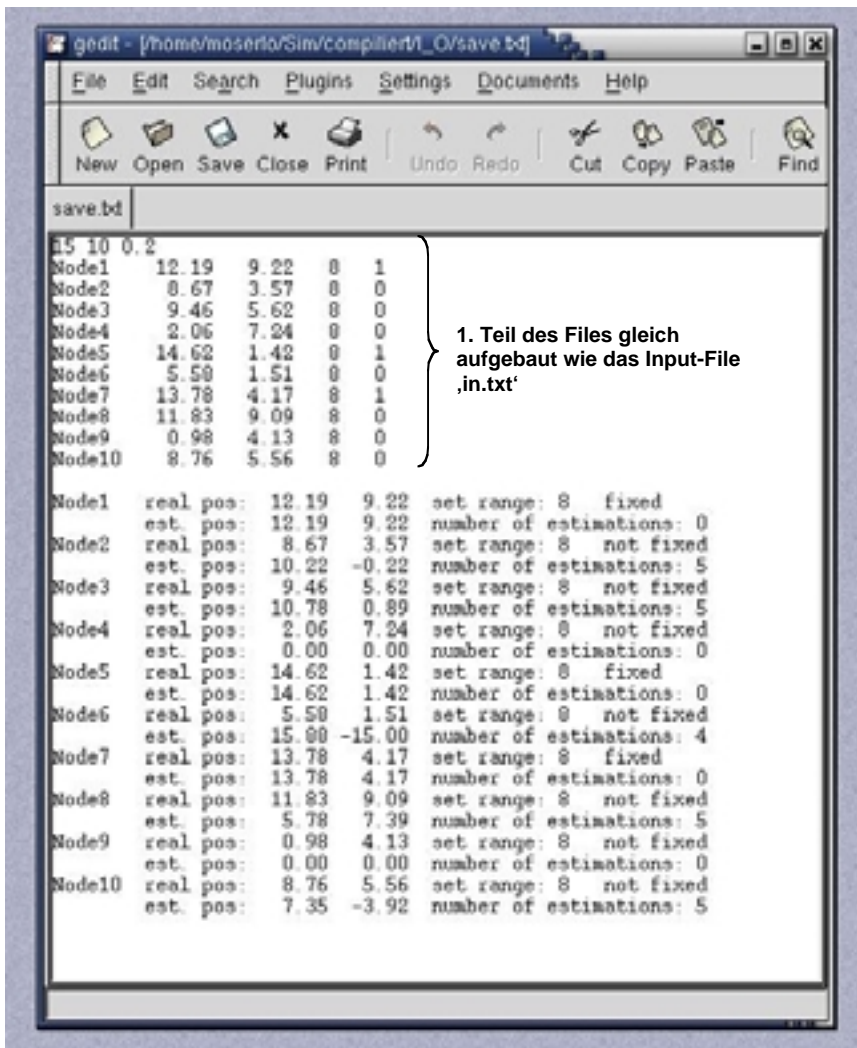
Solange der Simulator nicht beendet und neugestartet wird, werden zu loggende Daten ans Ende des Files angehängt. Sobald aber der Simulator neu gestartet wird (der Befehl ‚restart‘ startet den Simulator nicht neu, sondern nur das Szenario), wird das alte Log-File überschrieben durch ein neues. Die Einträge im Log-File sind selbsterklärend.

4.7 Der Befehl ‚save‘

Der Befehl ‚save‘ speichert Daten ins File ‚./I_O/save.txt‘. Dieses File ‚save.txt‘ wird also in das Verzeichnis ‚I_O‘ geschrieben. Wird der Befehl ‚save‘ wiederholt aufgerufen, wird das bestehende File ‚save.txt‘ durch das neue überschrieben.

Das File selbst ist zweiteilig. Im ersten Teil steht das eingelesene Szenario in genau derjenigen Form, wie es der Simulator einlesen kann. Löscht man den zweiten Teil des Files und ändert den Namen auf ‚in.txt‘, kann man das Szenario später wieder einlesen.

Im zweiten Teil des Files stehen dieselben Sachen wie der Befehl ‚info‘ liefert, ergänzt um die Range eines Knotens und die Anzahl der vorgenommenen Berechnungen.



```
gedit - [~/home/moser/er/Sim/compiled/I_O/save.txt]
File Edit Search Plugins Settings Documents Help
New Open Save Close Print Undo Redo Cut Copy Paste Find
save.txt
E5 10 0.2
Node1 12.19 9.22 8 1
Node2 8.67 3.57 8 0
Node3 9.46 5.62 8 0
Node4 2.06 7.24 8 0
Node5 14.62 1.42 8 1
Node6 5.58 1.51 8 0
Node7 13.78 4.17 8 1
Node8 11.83 9.09 8 0
Node9 0.98 4.13 8 0
Node10 8.76 5.56 8 0

Node1 real pos: 12.19 9.22 set range: 8 fixed
est. pos: 12.19 9.22 number of estimations: 0
Node2 real pos: 8.67 3.57 set range: 8 not fixed
est. pos: 10.22 -0.22 number of estimations: 5
Node3 real pos: 9.46 5.62 set range: 8 not fixed
est. pos: 10.78 0.89 number of estimations: 5
Node4 real pos: 2.06 7.24 set range: 8 not fixed
est. pos: 0.00 0.00 number of estimations: 0
Node5 real pos: 14.62 1.42 set range: 8 fixed
est. pos: 14.62 1.42 number of estimations: 0
Node6 real pos: 5.58 1.51 set range: 8 not fixed
est. pos: 15.00 -15.00 number of estimations: 4
Node7 real pos: 13.78 4.17 set range: 8 fixed
est. pos: 13.78 4.17 number of estimations: 0
Node8 real pos: 11.83 9.09 set range: 8 not fixed
est. pos: 5.78 7.39 number of estimations: 5
Node9 real pos: 0.98 4.13 set range: 8 not fixed
est. pos: 0.00 0.00 number of estimations: 0
Node10 real pos: 8.76 5.56 set range: 8 not fixed
est. pos: 7.35 -3.92 number of estimations: 5
```

1. Teil des Files gleich aufgebaut wie das Input-File ‚in.txt‘

Screenshot 7: Der Befehl ‚save‘ erzeugt das File ‚save.txt‘

4.8 Der Befehl ‚data‘

Um Ergebnisse der Berechnungen auf den Bildschirm auszugeben, kann man den Befehl ‚info‘ benutzen. Dieser Befehl liefert aber nur die Endergebnisse und keine Informationen darüber, wie es zu diesen Ergebnissen gekommen ist. Die Lücke wird geschlossen mit dem Befehl ‚data‘. Er benötigt als Argument noch den Knoten. ‚data Node4‘ liefert alle gespeicherten Daten zu Node4 inkl. der History für alle berechneten Positionen. Am besten erklärt man die Ausgaben direkt am Screenshot .

```

> data Node1
Data of Node1
Real Position: 6,28 5,62
MPosition: 6,28 5,62 1
ExternalPosition: 6,28 5,62 1
string assumed_1: Node10 1
string assumed_2: Node13 1
ratio for update by refining: 0,3
Vector MPos_history:
Vector EstimateError:
Vector EstimateError:
Vector abc_coord_systems:
-anchor_name Node1
  is assumed: 0
  abcPos 0 0 1
  Values in vector abcPos_history:
    abcPos_history empty
This was data of Node1.

> data Node7
Data of Node7
Real Position: 0,74 0,41
MPosition: 3,48655 7,64061 1
ExternalPosition: 0 0 0
NumberOfEstimations: 2
string assumed_1: 0
string assumed_2: 0
ratio for update by refining: 0,3
Vector MPos_history:
  MPos_history: 6,33268 5,50245
  MPos_history: 0,660416 9,72878 1
Vector EstimateError:
  ErrorDistance: 7,58755
  ErrorDistance: 9,31912
Vector abc_coord_systems:
-anchor_name Node1
  is assumed: 0
  abcPos 9,2448 1,61368 1
  Values in vector abcPos_history:
    abcPos_history 14,835 2,87969 1
    abcPos_history 3,6409 3,50411 1
    abcPos_history 8,20058 -0,719441 1
    abcPos_history 10,2027 0,688349 1
-anchor_name Node11
  is assumed: 0
  abcPos -0,701281 10,1758 1
  Values in vector abcPos_history:
    abcPos_history -0,0043979 0,0054 1
    abcPos_history -1,12316 1
-anchor_name Node15
  is assumed: 0
  abcPos 0,96658 9,9322 1
  Values in vector abcPos_history:
    abcPos_history 0,96658 9,9322 1
This was data of Node7.
  
```

Knoten 1 ist ein Anker

Die 1. Assumption ging an Knoten 10, die 2. Assumption ging an Knoten 13

Ein Anker ist nur in seinem eigenen Koordinatensystem drin und hat dort Position (0/0)

Knoten 7 ist ein normaler Knoten

Knoten 7 hat zweimal seine absolute Position berechnet

Knoten 7 ist in den ABC-Koordinatensystemen von den Ankern Knoten 1 , Knoten 11, Knoten 15

Screenshot 8: Die Funktion ‚data‘ liefert auch die History zu allen Berechnungen.

Abbildungen

1	Winkelmessung	6
2	Zeitmessung	7
3	Phasenmessung	8
4	Terrain-Algorithmus als Zustandsmaschine für Anker	14
5	Terrain-Algorithmus als Zustandsmaschine für normale Knoten	15
6	Terrain-Algorithmus als Zustandsmaschine für Anker, die updatet	16
7	Terrain-Algorithmus als Zustandsmaschine für normale Knoten, die updatet	17
8	Implementierte Form des Terrain-Algorithmus für normale Knoten	18
9	Implementierte Form des Terrain-Algorithmus für Anker	20
10	Verschachtelte Funktionsaufrufe für implementierte Form Terrain-Algorithmus	21

Screenshots

1	Kreieren eines Szenarios	22
2	Einlesen eines Szenarios	23
3	Beispiel eines ‚in.txt‘ Files zum Einlesen	23
4	Ausgabe des Befehls ‚help‘	24
5	Ausgabe des Befehls ‚terrain‘	24
6	Ausgabe des Befehls ‚info‘	25
7	Beispiel eines ‚save.txt‘ Files	26
8	Ausgabe des Befehls ‚data‘	27

Glossar

Absolute Position: Position im Koordinatensystem, welches durch X- und Y-Field aufgespannt wird (auch globale Position genannt).

Anker: Ein Knoten, der seine reale Position kennt. Ein normaler Knoten wird durch den Befehl ‚fix‘ zu einem Anker.

Erreichbarer Knoten: Ein Knoten E ist für den Knoten A dann ein erreichbarer Knoten, wenn sowohl die (fehlerbehaftete) Range von Knoten A als auch diejenige von Knoten E grösser oder gleich ist wie die Distanz zwischen diesen beiden Knoten.

Error Rate: Legt fest, mit welchem Fehler gemessene Distanzen belegt werden. Ist die Error Rate 0.2, kann die abgefragte Distanz d zwischen zwei Knoten zwischen $0.8d$ und $1.2d$ liegen. Damit wird die Ungenauigkeit in der Distanzbestimmung bei Messung der Signalstärke simuliert.

Fixed: Ein Knoten ist fixed, wenn er ein Anker ist.

Normaler Knoten: Netzwerkknoten, der mittels Funk mit andern Netzwerkknoten kommunizieren kann und versucht, seine reale Position zu errechnen.

Range: Funkreichweite eines Knotens. Will ein Knoten mit einem andern Knoten kommunizieren, müssen die Ranges von beiden Knoten jeweils grösser sein als die gemessene (fehlerbehaftete) Distanz zwischen diesen beiden Knoten. Die Range wird ebenfalls mit der Error Rate belegt, um damit Hindernisse oder Reflexionen zu simulieren. Ist die Error Rate 0.3, kann die Range R zwischen $0.7R$ und $1.3R$ sein.

Reale Position: Position, die ein Knoten im Koordinatensystem hat, welches von X- & Y-Field aufgespannt wird, aber selber nicht kennt und versucht, möglichst genau zu errechnen.

Relative Position: Position, die ein Knoten in einem ABC-Koordinatensystem hat (vgl. Abschnitt 3.2).

Refine: Update der eigenen absoluten Position mittels Triangulation zu andern Knoten, die ebenfalls bereits eine absolute Position haben.

Settled: Ein Knoten ist settled, wenn er eine absolute Position berechnen konnte.

Szenario: Eine Menge von Daten, die die Beschaffenheit des Netzwerkes beschreiben. Dazu gehören die Anzahl Knoten im Netzwerk und ihre Namen, die Error Rate, X- & Y-Field, die realen Positionen dieser Knoten, die Ranges jedes Knotens, und der Vermerk, ob ein Knoten ein normaler Knoten ist oder ein Anker.

Settled: Ein Knoten ist settled, wenn er eine absolute Position errechnen konnte. Diese Position kann von der realen Position abweichen.

X-Field, Y-Field: Diese zwei Werte geben die maximale Ausdehnung des simulierten Netzwerkes an und spannen das Koordinatensystem für die absoluten Werte auf.

Literaturverzeichnis

- [1] Jan Beutel, Geolocation in a PicoRadio Environment, Master thesis, ETH Zürich, Electronics Lab, 1999
- [2] Jaap C. Hartsen, The Bluetooth Radio System, IEEE Personal Communications, Februar 2000
- [3] James J. Caffery und Gordon L. Stüber, Overview of Radiolocation in CDMA Cellular Systems, IEEE Communications Magazine, April 1998
- [4] Christopher Drane, Malcolm Macnaughtan und Craig Scott, Positioning GSM Telephones, IEEE Communications Magazine, April 1998
- [5] Kaspar Nipp und Daniel Stoffer, Lineare Algebra – Eine Einführung für Ingenieure unter besonderer Berücksichtigung numerischer Aspekte, Verlag vdf, Zürich, 1992
- [6] Reto Strahl, Build your own World, Semesterarbeit. ETH Zürich, TIK, Juli 2001
- [7] Vorlesungsskript ‚Simulation in Produktion und Logistik‘, 1. Vorlesung, ETH Zürich, Sommersemester 2000
- [8] Chris Savarese, Jan M. Rabaey und Jan Beutel, Locationing in distributed ad-hoc sensor networks, Proceedings of ICASSP, 2001