**Julius-Maximilians-Universität Würzburg**
Fakultät für Mathematik und Informatik

**Informationstechnik für Luft- und Raumfahrt**
Lehrstuhl für Informatik 8 Prof. Dr. Sergio Montenegro


# Bachelorarbeit

## Implementierung und Evaluierung eines kartenbasierten Lokalisationsverfahrens für einen autonomen Quadrokopter

Vorgelegt von

Damian Rothkegel

Matr.-Nr.: 1747052

Prüfer: Prof. Dr. Sergio Montenegro

Betreuender wissenschaftliche Mitarbeiter: Dipl.-Ing. Nils Gageik

Würzburg, 16. 08. 2013

# Erklärung

Ich versichere, dass ich die vorliegende Arbeit einschließlich aller beigefügter Materialien selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe.
Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Werken entnommen sind, sind in jedem Einzelfall unter Angabe der Quelle deutlich als solche kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form noch nicht als Prüfungsarbeit eingereicht worden.
Mir ist bekannt, dass Zuwiderhandlungen gegen diese Erklärung und bewusste Täuschungen die Benotung der Arbeit mit der Note 5.0 zur Folge haben kann.

Würzburg, 16. 08. 2013

Damian Rothkegel

# Aufgabenstellung

Die Fortschritte im Bereich Sensorik und Mikrotechnik ermöglichen heutzutage den kostengünstigen Bau kleiner unbemannter Luftfahrzeuge (UAV, unmanned aerial vehicle, Drohne) wie Quadrokopter. Die Forschung und Entwicklung dieser Systeme wurde in den letzten Jahren aufgrund der vielfältigen Anwendungsmöglichkeiten stark vorangetrieben. Wenngleich im Bereich UAV viel geforscht wurde, ist das Thema Autonomes Flugobjekt längst noch nicht vollständig behandelt. Insbesondere der Indoor-Betrieb ist aufgrund fehlender absoluter Positionsstützung durch GPS problematisch. Der Aufbau eines eigenen autonomen Systems wird daher am Lehrstuhl Aerospace Information Technology der Uni Würzburg erforscht und erprobt. Im Rahmen dieses Forschungsvorhabens ist ein System zu entwickeln, dass in der Lage ist die Position des Quadrokopters an Hand von Wandabstandsinformationen (Abstandssensoren) und unter zu Hilfenahme einer Karte zu bestimmen.

Hauptaugenmerk dieser Arbeit ist die Entwicklung eines Algorithmus zur Lokalisation für die Verwendung mit Abstandssensoren. Der Algorithmus ist entsprechend der zu verwendenden Sensorik, namentlich Infrarot bzw. Ultraschall, auszulegen. Als weitere Stütze zur Lokalisation stehen ein optischer Flusssensor sowie eine Karte einer bereits implementierten Mapping-Software zur Verfügung.
Im Rahmen der Arbeit ist zunächst der Stand der Technik im Bereich autonome Lokalisation aufzuarbeiten und zu beschreiben. Die implementierte Lösung ist in das bestehende System zu integrieren und an diesem ausgiebig zu evaluieren. Die Arbeit ist umfangreich zu dokumentieren.

Aufgabenstellung (Stichpunktartig):

- Aufarbeitung Stand der Technik: Lokalisationsverfahren und SLAM
- Implementierung Datenfusion: OF + Abstandssensorik zur Lokalisation
- Einbettung Mapping und QT, Integration in Quadcopter
- Evaluierung am Quadcopter
- Dokumentation

# Zusammenfassung

Im Rahmen dieser Arbeit wurde ein Verfahren zur kartengestützten Lokalisation eines autonomen Quadrokopters entwickelt. Dieses benötigt außer einer Karte Abstands- sowie Bewegungsinformationen, welche der Quadrokopter ohne externe Systeme messen kann. Dadurch ist kein Verfahren wie GPS nötig, was dieses System für den Indoorbetrieb prädestiniert. Desweiteren wurde darauf geachtet, dass der Algorithmus auch mit eher unzuverlässigen bzw. ungenauen Sensoren (wie z.B. Ultraschall-Abstandsmessern) noch brauchbare Ergebnisse erzielt. Die Anfangsposition des Quadrokopters sollte bekannt sein, der Algorithmus ist aber auch in der Lage ohne diese Information eine Lokalisation durchzuführen.

Es konnte gezeigt werden, dass ein Lokalisationsalgorithmus für die Verwendung mit Abstandssensoren so implementiert werden kann, dass dieser bereits nach ca. 10-20 Iterationen eine Positionsschätzung liefert. Außerdem zeigte sich, dass das Verfahren auch mit eher ungenauer Sensorik noch funktionsfähig ist. Obwohl die Zuverlässigkeit des Lokalisationsverfahrens noch einer Steigerung bedarf, konnte mit dieser Arbeit eine Grundlage für einen Ausbau der Autonomie des Quadrokopters geschaffen werden.

# Inhaltsverzeichnis

**1. Einleitung** **1**

**2. Grundlagen & Stand der Technik** **3**

2.1. Mathematische Grundlagen . . . . . 3

2.1.1. Pose . . . . . 3

2.1.2. Zustandsschätzung . . . . . 4

2.1.3. Bayes Filter-Algorithmus . . . . . 5

2.1.4. Kalman Filter . . . . . 5

2.2. Kartendarstellung . . . . . 6

2.2.1. Feature-basiert . . . . . 6

2.2.2. Occupancy Grid Map . . . . . 7

2.3. Lokalisationsverfahren . . . . . 8

2.3.1. Koppelnavigation . . . . . 9

2.3.2. EKF-Lokalisation . . . . . 10

2.3.3. Lokalisation per Histogramm . . . . . 12

2.3.4. Monte Carlo Localization . . . . . 14

2.4. Das SLAM-Problem . . . . . 14

2.4.1. SLAM mittels Kalman Filter . . . . . 15

2.4.2. GraphSLAM . . . . . 16

2.4.3. FastSLAM . . . . . 16

2.5. Zusammenfassung . . . . . 17

**3. Konzept** **18**

3.1. Überblick . . . . . 18

3.1.1. Anforderungen . . . 18
3.1.2. Auswahl eines geeigneten Lokalisationsverfahrens . . . 20
3.2. Struktur . . . 21
3.2.1. Übersicht . . . 21
3.2.2. Partikel . . . 22
3.2.2.1. functionAge . . . 24
3.2.2.2. functionCell . . . 24
3.2.2.3. functionReading . . . 25
3.2.3. Weiterentwicklung der Partikel . . . 28
3.2.3.1. Bewegungsmodell . . . 30
3.2.3.2. Sensormodell . . . 31

**4. Implementierung 32**
4.1. Bestehendes System . . . 32
4.2. Strukturierung der Software . . . 33
4.2.1. Sensormodell . . . 35
4.3. Einbindung in bestehendes System . . . 36
4.4. Beschreibung der Benutzeroberfläche . . . 36

**5. Evaluierung 40**
5.1. Evaluierung mit simulierten Sensorwerten . . . 40
5.1.1. Lokalisation mit bekannter Startpose . . . 42
5.1.1.1. Geringer Bewegungsfehler . . . 42
5.1.1.2. Mittlerer Bewegungsfehler . . . 42
5.1.1.3. Großer Bewegungsfehler . . . 45
5.1.1.4. Fazit . . . 45
5.1.2. Lokalisation mit unbekannter Startpose . . . 45
5.1.2.1. Lokalisation mit 50 Partikeln . . . 47
5.1.2.2. Lokalisation mit 1000 Partikeln . . . 48
5.1.2.3. groupProbability = 0.6 . . . 49
5.1.2.4. groupProbability = 0.7 . . . 49
5.1.2.5. groupProbability = 0.8 . . . 49

5.1.2.6. Einschränkung der Orientierung . . . . . 50
5.1.2.7. Fazit . . . . . 51
5.2. Evaluierung mit realen Sensorwerten . . . . . 53
5.2.1. Erster Versuch . . . . . 55
5.2.2. Zweiter Versuch . . . . . 55

**6. Fazit** **57**
6.1. Ergebnisse . . . . . 57
6.2. Ausblick . . . . . 58

**7. Literaturverzeichnis** **59**

**A. Parameter** **61**

**B. Sichtlinien-Algorithmus** **63**

**C. Parametrisierung des Bewegungsmodells** **66**

# 1. Einleitung

Technische Systeme erfuhren in den letzten Jahrzehnten große Fortschritte. Roboter wurden immer schneller, stärker und effizienter und erschlossen damit immer neue Anwendungsgebiete. Gleichzeitig entwickelte sich auch die Computerindustrie rasanter als von manchem erwartet, was Ingenieuren und Wissenschaftlern immer mehr Freiheiten bei der Entwicklung von Drohnen und Robotern ließ. Damit wurde es möglich, diesen mehr und mehr „Intelligenz“ und somit auch Autonomie zu verleihen. Roboter können schon längst nicht mehr nur Autos zusammenbauen, heutzutage mähen sie den Rasen, helfen Ärzten bei komplizierten Eingriffen, werden vom Militär eingesetzt oder messen sich in Fußballmeisterschaften.

Bei Drohnen handelt es sich um sog. UAVs (**U**nmanned **A**erial **V**ehicles, dt. Unbemannte Luftfahrzeuge). Ein Spezialfall von Drohnen sind Quadrokopter, welche über vier nach unten wirkende Rotoren verfügen. Dadurch wird dem Quadrokopter sowohl ein „Stehen“ in der Luft wie bei Hubschraubern als auch ein Manövrieren im dreidimensionalen Raum ermöglicht.
An der Julius-Maximilians-Universität Würzburg wird im Rahmen des AQopterI8-Projekts ein Quadrokopter mit folgendem Ziel entwickelt: *„Das fertige System soll in die Lage versetzt werden, autonome Aufgaben aus den Bereichen Search & Rescue (z.B: Feuerwehreinsatz) sowie Überwachung und Kontrolle (Industrie- & Chemieanlage) durchzuführen.“* (Gageik [2013])
Dabei ist der Begriff der *Autonomie* besonders hervorzuheben. Je autonomer sich ein Quadrokopter (und auch ein Roboter) verhält, desto weniger ist Überwachung und Kontrolle durch eine Person notwendig.
Ein wichtiger Aspekt eines autonomen Quadrokopters ist die Möglichkeit zur Lokalisation. Diese sollte ohne externe Hilfe und in möglichst vielen Situationen funktionieren. In Gebäuden kann allerdings nicht davon ausgegangen werden kann, dass „klassische“ Verfahren wie GPS funktionieren, weshalb hier auf andere Methoden zurückgegriffen werden muss. Desweiteren ist zu be-

achten, dass das AQopterI8-Projekt auch mit der Absicht möglichst niedriger Kosten entwickelt wird, was zu einer Verwendung von Low-Cost-Sensorik führt. Die Verwendung der unruhigen Plattform Quadrokopter stellt außerdem hohe Anforderungen an die Fehlertoleranz des Lokalisationsalgorithmus. Das Lokalisationsverfahren muss also sowohl robust als auch schnell zu berechnen sein.

In Kapitel 2 werden zunächst gängige Lokalisationsverfahren vorgestellt und ein Überblick über sog. SLAM-Verfahren (Simultaneous Localization And Mapping, dt. simultane Lokalisation und Kartenerstellung, s. Kap. 2.4) gegeben, welche auch eine Lokalisation in vorher vollkommen unbekannten Gebieten ermöglichen. Nach einer Gegenüberstellung dieser Verfahren wird dann in Kapitel 3 das für dieses Projekt geeignetste Verfahren ausgewählt und entwickelt. Kapitel 4 behandelt die konkrete Umsetzung des zuvor erarbeiteten Konzepts. Diese wird dann in Kapitel 5 evaluiert. Abschließend werden in Kapitel 6 die erzielten Ergebnisse diskutiert und ein Ausblick auf mögliche weitere Entwicklungen gegeben.

# 2. Grundlagen & Stand der Technik

In diesem Kapitel werden, nach kurzer Erläuterung der wichtigsten mathematischen Grundlagen, einige Verfahren zur Lokalisation und zur Lösung des sog. SLAM-Problems (s. Kap. 2.4) vorgestellt.
Da sich die Thematik dieser Arbeit nicht nur auf bodengebundene Roboter, sondern auch auf Fluggeräte wie Quadrokopter erstreckt, werden diese Systeme im weiteren Verlauf zusammenfassend als *Agenten* bezeichnet.

## 2.1. Mathematische Grundlagen

Zunächst werden in möglichst kompakter und verständlicher Form einige mathematische Zusammenhänge erläutert, die für das Verständnis dieser Arbeit von Bedeutung sind. Auf weiterführende Literatur wird an den entsprechenden Stellen verwiesen.

### 2.1.1. Pose

Im Folgenden wird der Begriff der *Pose* häufig Verwendung finden. Die Pose beschreibt die X-, Y- und ggf. Z-Koordinaten der Position eines Agenten (je nach dem, ob man sich in einer Ebene oder im Raum bewegt) sowie dessen Orientierung $\theta$ ggü. einem globalen kartesischen Referenzsystem. In dieser Arbeit findet nur die zweidimensionale Pose $(x, y, \theta)$ Anwendung.
Abb. 2.1 verdeutlicht eine Pose. Desweiteren ist zu sagen, dass Agenten häufig vereinfacht durch einen Kreis dargestellt werden, wobei eine Linie vom Mittelpunkt zum Kreisrand die Orientierung darstellt.

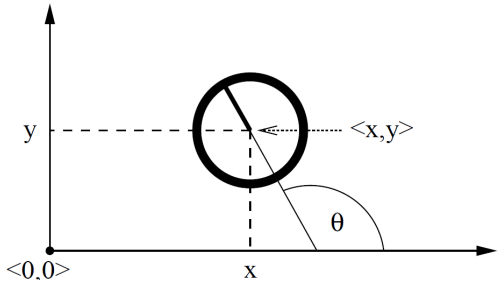



**Abbildung 2.1.:** Pose eines Roboters. Quelle: Thrun et al. [2005]

### 2.1.2. Zustandsschätzung

Das Lokalisierungsproblem lässt sich in etwas abstrakter Form als das Problem, den aktuellen Systemzustand zu schätzen, beschreiben. Dieser Zustand umfasst dabei sowohl alle für das autonome System relevanten Eigenschaften der Umgebung (also die Position von Wänden und Türen, der Aufenthaltsort von Personen, etc.), als auch den Zustand des Agenten selbst (Pose, Geschwindigkeit, Zustand der Sensoren, etc.). Da viele dieser Eigenschaften kaum direkt gemessen werden können muss sich der Agent auf seine Sensorik sowie bisher erfasste Messungen verlassen, um aus den Messdaten auf die benötigten Eigenschaften schließen zu können. Genau das ist mit Zustandsschätzung gemeint: Es soll aus fehlerbehafteten Messungen, die nur einen kleinen Teil der Umgebung erfassen können, der aktuelle Zustand des Systems berechnet werden. Der Agent *glaubt* also, den Systemzustand $x$ zum Zeitpunkt $t$ berechnet zu haben, was sich wie folgt ausdrücken lässt:

$$bel(x_t) \;=\; p(x_t \mid z_{1:t}, u_{1:t}) \tag{2.1}$$

$z_{1:t}$ sind alle Messungen, $u_{1:t}$ alle Steuerbefehle, die bis zum Zeitpunkt $t$ durchgeführt wurden. Damit beschreibt $p(x_t \mid z_{1:t}, u_{1:t})$ - respektive $bel(x_t)$ - die Wahrscheinlichkeit, dass $x_t$ der aktuelle Systemzustand ist, wenn zuvor die Messwerte $z_{1:t}$ erfasst und die Steuerbefehle $u_{1:t}$ ausgeführt wurden.

Desweiteren lässt sich ausdrücken, welcher Systemzustand erwartet wird, bevor Messungen zum Zeitpunkt $t$ durchgeführt werden:

$$\overline{bel}(x_t) \;=\; p(x_t \mid z_{1:t-1}, u_{1:t}) \tag{2.2}$$

Der vor der Durchführung der Messungen erwartete Systemzustand muss nach Durchführung der Messungen *korrigiert* werden, um die aktuelle Systemzustandsvermutung zu erhalten. Diese

Aufteilung in *Vorhersage- und Berichtigungsschritt* ist charakteristisch für viele Lösungen des Lokalisierungs- und SLAM-Problems (vgl. Thrun et al. [2005], Kap. 2.3).

### 2.1.3. Bayes Filter-Algorithmus

Eine allgemeine Lösung zur rekursiven Berechnung der aktuellen Systemzustandsvermutung ist der Bayes Filter-Algorithmus. Er basiert auf dem Bayestheorem, welches die Umkehrung einer bedingten Wahrscheinlichkeit erlaubt:

$$p(x \mid y) \;=\; \frac{p(y \mid x)p(x)}{p(y)} \tag{2.3}$$

Das bedeutet: Sind die Wahrscheinlichkeiten für das Eintreten von $x$ und $y$ bekannt, und ist die bedingte Wahrscheinlichkeit, dass $y$ eintritt, wenn $x$ eingetreten ist, bekannt, lässt sich die Wahrscheinlichkeit berechnen, dass $x$ eintritt, wenn $y$ eingetreten ist.
Wie bereits in Kap. 2.1.2 erwähnt wird die Berechnung in einen Vorhersage- und einen Korrekturschritt aufgeteilt. Hierzu werden genaue mathematische Modelle der Sensorik und der Bewegung des Agenten benötigt, welche in der Praxis meist nur schwierig erstellt werden können.
Da es sich beim Bayes Filter-Algorithmus um eine sehr allgemeine Lösung handelt, die viel zu komplex zu berechnen wäre, wurden verschiedene Annäherungen und Vereinfachungen entwickelt, die eine konkrete Berechnung durch einen Computer erlauben. Eine Anwendung probabilistischer Verfahren wie des Bayes Filter-Algorithmus auf das Lokalisierungsproblem bei mobilen Robotern wird auch als *Markov-Lokalisation* bezeichnet (vgl. Fox [1998]). Einige dieser Umsetzungen werden in Kap. 2.3 besprochen.

### 2.1.4. Kalman Filter

1960 entwickelte Rudolf E. Kálmán den nach ihm benannten Kalman-Filter. Basierend auf Bayes' Filter erlaubt er, sowohl vergangene als auch in der Zukunft liegende Zustände eines linearen, zeitdiskreten Systems zu schätzen, wobei die Messfehler als normalverteilt angesehen werden (vgl. Welch und Bishop [2001]). Dieser Filter kann sehr recheneffizient implementiert werden und liefert für Systeme, die den Anforderungen (linear, normalverteiltes Rauschen) entsprechen, sehr gute Ergebnisse. Durch Modifizierungen ist es auch möglich, ihn bis zu einem gewissen Grad auf nichtlineare Systeme anzuwenden. Dazu kann z.B. mittels einer Taylorentwicklung das System

lokal linearisiert werden, was *Extended Kalman Filter* (EKF) genannt wird. Diese Linearisierung kann auch durch statistische Techniken wie die *Unscented Transformation*, welche im Unscented Kalman Filter (UKF) zum Einsatz kommt, berechnet werden, wodurch teilweise bessere Ergebnisse erzielt werden können. Das geht allerdings auf Kosten der Recheneffizienz (s. Thrun et al. [2005]).

## 2.2. Kartendarstellung

Eine wichtige Rolle spielt die Karte der Umgebung, da sie die Grundlage der Lokalisierung darstellt. Um ein möglichst gutes Verhältnis zwischen Detailgrad und Speicherbedarf zu erhalten ist es notwendig, gewisse Ungenauigkeiten in Kauf zu nehmen und für den Agenten unwichtige Details zu vernachlässigen. Hier haben sich einige Ansätze durchgesetzt, von denen zwei der verbreitetsten hier kurz vorgestellt werden.

### 2.2.1. Feature-basiert

Die Idee dieser Karte ist es, komplett aus sog. Features (manchmal auch „Landmarks“, dt. Landmarken, genannt) aufgebaut zu sein. Ein Feature ist ein vom Agenten leicht zu identifizierendes Merkmal der Umgebung, z.B. Bäume, Türen oder künstliche Markierungen wie bunt angestrichene Pfähle. Der große Vorteil dieser Repräsentation der Umgebung ist, dass ein Feature in der Regel nur mit einem 2- oder 3-Tupel, welches die Koordinaten in der Ebene oder im Raum darstellt, gespeichert werden kann. Dies macht diese Darstellung besonders effizient hinsichtlich des Speicherbedarfs. Allerdings birgt dieses Verfahren auch Risiken, wobei insb. zwei zu nennen sind: Zum einen müssen die Features aus den Messungen extrahiert werden. Dies kann, je nach Art der Features, komplexe Prozesse wie die Analyse von Kamerabildern erforderlich machen. Zum andern muss gewährleistet werden, dass die erkannten Features intern auch den richtigen Features der Karte zugeordnet werden (häufig als „Data Association Problem“, dt. Datenassoziationsproblem, bezeichnet).

### 2.2.2. Occupancy Grid Map

Ein Gegenstück zu feature-basierten Karten sind Occupancy Grid Maps. Statt sich auf besondere Eigenschaften der Umgebung zu konzentrieren, wird über den gesamten relevanten Bereich ein (meist zweidimensionales) Gitter gelegt. Den dadurch entstandenen Zellen wird nun jeweils ein Wert zugeordnet, der die Wahrscheinlichkeit, dass die Zelle belegt ist, repräsentiert (s. Hähnel [2005]). Dies kostet in der Regel viel Speicher, da das Gitter nicht zu grob sein darf, um die Umgebung genau genug abzubilden. Dafür bietet diese Kartendarstellung aber den Vorteil, dass zur Erstellung einfache Entfernungsmessungen reichen, die auch relativ einfach in die Karte integriert werden können (s. Schmitt [2012]). Es ist also im Gegensatz zu feature-basierten Karten nicht nötig, aus den Messwerten zunächst die Features zu extrahieren.
Ein praktischer Nebeneffekt ist außerdem, dass die Daten auch von Menschen sehr gut als Karte interpretiert werden können (s. Abb. 2.2).

**Abbildung 2.2.:** Occupancy Grid Map einer großen Ausstellungsfläche. Je dunkler ein Pixel, desto wahrscheinlicher ist die zugehörige Zelle belegt. Quelle: Thrun et al. [2005]

## 2.3. Lokalisationsverfahren

Ein Agent muss, um autonom agieren zu können, stets Kenntnis über seine eigene Position im Raum haben. Während dies im Freien heutzutage dank GPS recht einfach zu bewerkstelligen ist, muss in Gebäuden auf andere Verfahren zurückgegriffen werden, da das schwache GPS-Signal dort nur noch schwierig zu empfangen ist (was zusätzlich durch Effekte wie Reflektionen in Häuserschluchten erschwert wird). Die gängigsten dieser Verfahren werden in den folgenden Teilkapiteln vorgestellt.

Grundsätzlich lassen sich Lokalisationsverfahren anhand mehrerer Aspekte unterscheiden (vgl. Thrun et al. [2005], Kap. 7.1). Eine der einfachsten Aufgaben besteht in der *lokalen Lokalisation*, da hier die Anfangsposition im Raum bekannt ist. Der Agent besitzt somit einen Referenzpunkt, auf den er seine Berechnungen aufbauen kann. Ist die Anfangsposition jedoch unbekannt spricht man von *globaler Lokalisation*. Hier muss der Agent, bevor er seine Position verfolgen kann, diese zunächst bestimmen. Dies wird z.B. durch symmetrische und repetitive Umgebungen weiter erschwert, da meist mehrere Vermutungen existieren, an welcher Position man sich befinden könnte. Allerdings ist ein globales Lokalisationsverfahren, sobald es implementiert wurde, deutlich robuster als ein lokales Verfahren, da es in der Lage ist, den kompletten Verlust der Positionsvermutung auszugleichen. Sollte jedoch ein lokales Verfahren seine Position verlieren (z.B. durch zu große Messfehler oder eine falsche Anfangsposition) ist der Agent in sehr vielen Situationen unfähig, sich jemals wieder zu lokalisieren.
Ein weiterer Aspekt ist das sog. *Kidnapped Robot Problem*. Es beschreibt die Situation, in der der Agent plötzlich an eine andere Position „teleportiert“ wird, ohne dass er dies bemerkt. Ein verlässlicher Algorithmus sollte in der Lage sein, nach kurzer Zeit zu merken, dass seine Positionsvermutung auf einmal vollkommen falsch ist und deshalb die aktuelle Vermutung nicht weiter verfolgen. Eine „Teleportation“ ist selbstverständlich unwahrscheinlich, allerdings besteht immer die Möglichkeit, dass sich ein Algorithmus irrt und deshalb eine falsche Position verfolgt. In diesem Fall ist es wichtig, dass sich der Algorithmus von diesem Fehler erholt und seine Positionsvermutung berichtigt.

Ein weiterer Aspekt sind *dynamische* Umgebungen, also Umgebungen, in denen sich immer

wieder etwas ändert. Dies kann bspw. durch umherlaufende Personen oder sich öffnende und schließende Türen verursacht sein. Unveränderliche Umgebungen werden als *statisch* bezeichnet.

Von großer Bedeutung ist die Art der Verteilung, die die Positionsvermutung(en) repräsentiert. Eine unimodale Verteilung kann nur eine Vermutung gleichzeitig modellieren, was zwar meist recheneffizient, dafür aber nicht sehr robust ist. Sie setzt voraus, dass der Lokalisierungsalgorithmus so genau arbeitet, dass nie die Möglichkeit mehrerer gleichberechtigter Vermutungen besteht. Noch schwieriger ist es, mit einer unimodalen Verteilung eine globale Lokalisierung durchzuführen. Dies ist in Umgebungen mit mehreren sehr ähnlichen Räumen fast immer zum Scheitern verurteilt, da sich der Algorithmus von Anfang an für einen dieser Räume entscheiden muss.
Genau hier können multimodale Verteilungen Abhilfe schaffen (wie u.a. von Cox und Leonard [1994] vorgeschlagen), da sie die Möglichkeit mehrerer Positionsvermutungen erlauben. Allerdings sind diese Verteilungen tendenziell schwieriger zu implementieren, da hier das gegenteilige Problem der Fall ist: Es muss dafür gesorgt werden, *dass* sich der Algorithmus möglichst schnell für eine der Vermutungen entscheidet - eine Information wie „Der Agent könnte hier oder hier oder aber hier sein“ ist z.B. für Navigationsaufgaben zu ungenau.

### 2.3.1. Koppelnavigation

Eines der einfachsten und intuitivsten Lokalisationsverfahren ist die Koppelnavigation (im Englischen Dead Reckoning). Hierbei wird zunächst die aktuelle Position als Ursprung definiert oder dem Agenten mitgeteilt, an welcher (globalen) Position er beginnt. Sobald sich der Agent bewegt, misst er seine Bewegung mittels dafür geeigneter Sensorik. Bei bodengebundenen Robotern mit Rädern werden häufig Inkrementalgeber genutzt, welche die Radumdrehungen mit mehreren Rechtecksignalen wiedergeben (sog. *Gray Codes*), woraus sich Drehrichtung und -geschwindigkeit berechnen lassen. Besteht hingegen kein Bodenkontakt können bspw. Beschleunigungs- und Drehratensensoren eingesetzt werden.
Nach einer gewissen Zeit (einige Millisekunden bis mehrere Sekunden) berechnet der Agent aus den Messdaten seine zuletzt ausgeführte Bewegung und addiert diese zur zuletzt bestimmten Position, wodurch er seine aktuelle Position erhält. Daraufhin wird wieder die Bewegung gemessen, diese auf die zuletzt bestimmte Position addiert usw. usf.
Der Vorteil dieses Verfahrens liegt eindeutig in seiner Einfachheit, da es nicht nur verständlich,

sondern auch effizient zu berechnen ist. Allerdings sind alle Messungen mit Fehlern behaftet, was sich insb. bei Low-Cost-Sensorik bemerkbar macht. Da bei der fortlaufenden Addition der Bewegung zur letzten Position nicht nur die Bewegung selbst, sondern auch der Messfehler aufsummiert wird, ist die berechnete Position mit einem umso größeren Fehler behaftet, je länger das Verfahren läuft. Dieser Fehler kann nur von außen, durch eine absolute Positionsreferenz, behoben werden, bspw. durch das Anfahren einer Dockingstation mit exakt bekannter Position.

### 2.3.2. EKF-Lokalisation

Eine deutliche Verbesserung der einfachen Koppelnavigation kann erreicht werden, wenn Messungen der Umgebung laufenden Einfluss auf die Positionsbestimmung haben, also eine Korrektur der Odometrie erfolgt. Die EKF-Lokalisation benötigt dazu eine feature-basierte Karte und einen entsprechenden Sensor, mit dessen Hilfe Features in der Umgebung des Agenten eindeutig und sicher identifiziert werden können. Desweiteren müssen die Anfangsposition des Agenten sowie ein Modell der Bewegung und der Messungen bekannt sein.
Der Algorithmus baut, wie der Name schon sagt, auf den Extended Kalman Filter auf. Er besteht im Wesentlichen aus drei Schritten:

**Vorhersage** Berechnung der nächsten Position und der erwarteten Messungen, basierend auf der vorherigen Position und dem Bewegungsmodell des Agenten. Die vermutete Position ist mit einer größeren Unsicherheit behaftet, da in diesem Schritt auch Messungenauigkeiten berücksichtigt werden.

**Fehler der erwarteten Messung** Nach Durchführung der Messungen werden diese damit verglichen, welche Messungen erwartet wurden. Aus der Abweichung lässt sich (aufgrund eindeutiger Zuordnung der Landmarken) berechnen, wo der Agent gestanden haben müsste, um die gemessenen Werte zu erhalten.

**Korrektur der Positionsschätzung** Der berechnete Messfehler wird auf die Positionsschätzung angewendet, wodurch der Agent seine Position nun besser und mit geringerer Unsicherheit kennt.

Abb. 2.3 verdeutlicht diesen Algorithmus. Kreise mit einer Zahl sind Landmarken, durchgezogene Linien zeigen den tatsächlichen Weg des Agenten. Im oberen Bild repräsentiert die gestri-

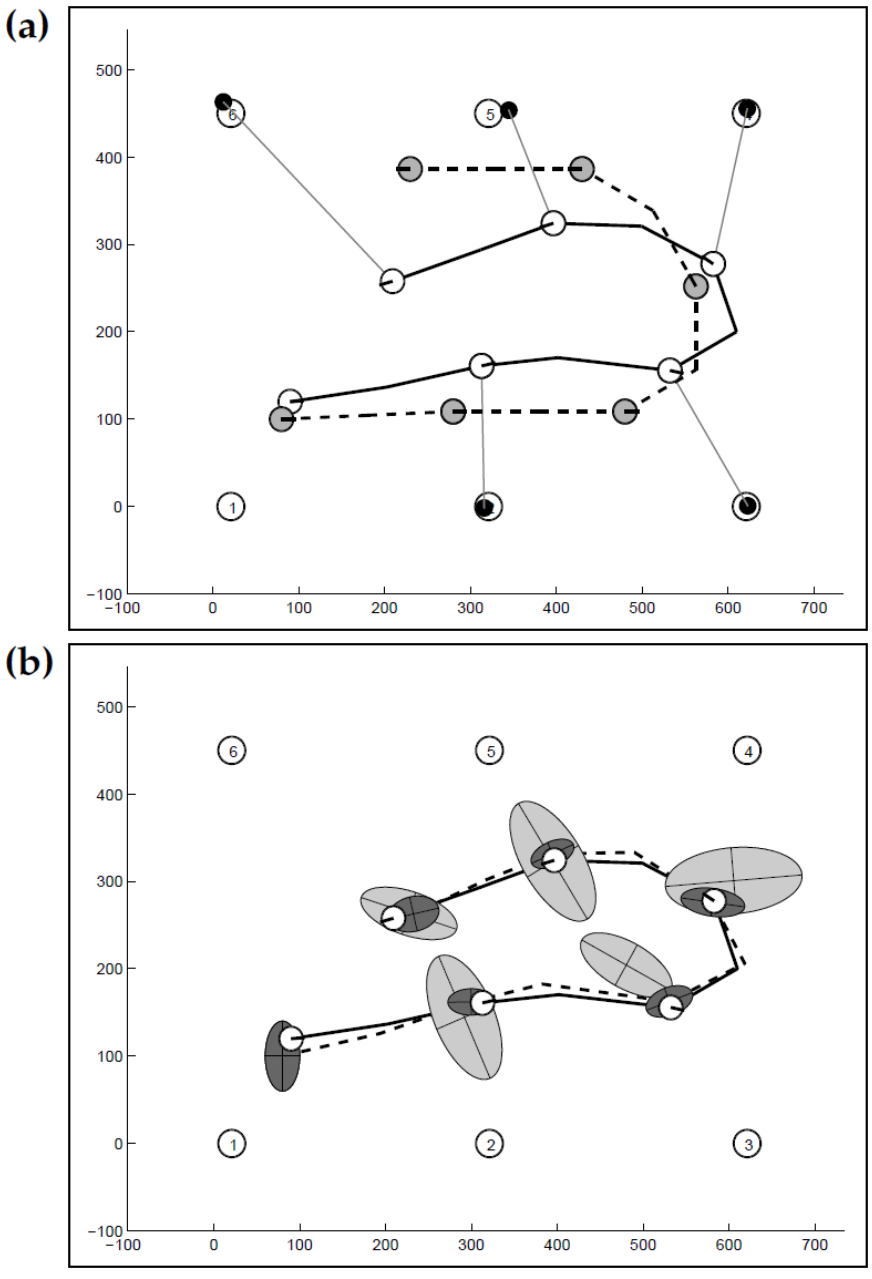



**Abbildung 2.3.:** Schematische Darstellung der EKF-Lokalisation. Quelle: Thrun et al. [2005]

chelte Linie den von der Odometrie des Agenten gemessenen Weg. Dünne Linien zeigen, welche Landmarken an der jeweiligen Position observiert wurden. Im unteren Bild ist die Positionsunsicherheit vor (hellgrau) und nach (dunkelgrau) der Korrektur durch den EKF-Algorithmus eingezeichnet. Die gestrichelte Linie repräsentiert den gemessenen und korrigierten Weg.
Die Verwendung der EKF-Lokalisation erlaubt das Verfolgen einer Positionsvermutung, wobei diese durch eine Normalverteilung repräsentiert wird. Es handelt sich um ein lokales Lokalisationsverfahren, da die Anfangsposition bekannt sein muss. Um eine zuverlässige Funktion zu gewährleisten, muss die Sensorik ausreichend genau arbeiten, was je nach Einsatzgebiet ein Problem sein kann. Es existieren zahlreiche Erweiterungen dieses Algorithmus, die sich verschiedener nachteiliger Aspekte annehmen und sich bspw. um das Problem der Datenassoziation kümmern oder das Verfolgen mehrerer Positionsvermutungen erlauben. Es ist zudem über Umwege möglich, die EKF-Lokalisation zu einem globalen Lokalisationsverfahren auszubauen (s. Thrun et al. [2005]).

### 2.3.3. Lokalisation per Histogramm

Ein gänzlich anderer Ansatz zur Lokalisation ist die Verwendung eines Histogramms (wie u.a. in Thrun et al. [2005] vorgestellt). Jeder Zelle der Karte wird eine Wahrscheinlichkeit zugeordnet, je nach dem, wie wahrscheinlich es ist, dass sich der Agent dort befindet. Bei jeder Bewegung werden diese Wahrscheinlichkeiten mitbewegt und den entsprechenden Zellen neu zugeordnet. Eine erneute Messung kann die Wahrscheinlichkeiten der Zellen wieder aktualisieren. Einfach gesagt können im Verlauf des Algorithmus immer mehr Zellen ausgeschlossen werden, wodurch sich im Endeffekt die aktuelle Position des Roboters herauskristallisiert. Abb. 2.4 verdeutlicht diesen Prozess nochmal an einem eindimensionalen Beispiel. Der Roboter ist in der Lage, Türen zu erkennen. $bel(x)$ stellt das Histogramm dar, an dem die aktuelle Positionsvermutung abgelesen werden kann, $p(z \mid x)$ gibt an, wie wahrscheinlich die aktuelle Messung an welchen Stellen der Karte ist.

Dieses Verfahren bietet einige wichtige Vorteile gegenüber auf Bayes' Filter basierenden Algortihmen. Da man nicht an eine Normalverteilung gebunden ist, können auch komplexe Verteilungen repräsentiert werden - ohne deren mathematische Beschreibung kennen zu müssen. Dies macht auch die praktische Implementierung relativ einfach. Außerdem entfällt, wie bei Occupancy Grid Maps üblich, die Notwendigkeit, Features aus der Umgebung extrahieren zu müssen, es

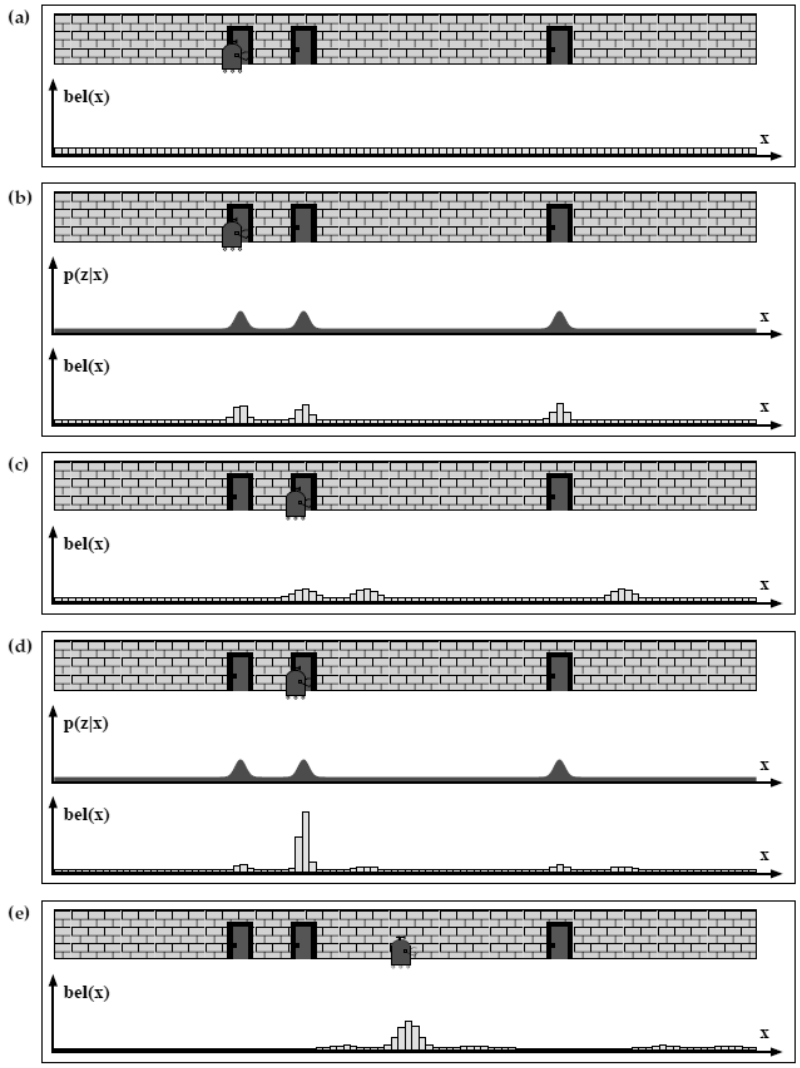



**Abbildung 2.4.:** Eindimensionale Lokalisation per Histogramm. Quelle: Thrun et al. [2005]

kann also direkt mit den rohen Messwerten gearbeitet werden. Desweiteren handelt es sich um ein globales Lokalisationsverfahren, wodurch der Agent in der Lage ist, sich auch ohne bekannte Anfangsposition zu lokalisieren.
Ein Nachteil ist die große zu verwaltende Datenmenge. Für eine genauere Lokalisierung sind kleinere Zellen nötig, welche mehr Speicherplatz und Rechenzeit benötigen. Dies lässt die Hardwareanforderungen, insb. für dreidimensionale Karten, schnell in exorbitante Höhen schießen. Abhilfe kann hier eine topologische, statt einer metrischen, Aufteilung der Karte schaffen. Detailarme Regionen können so effektiver beschrieben werden, während komplexen Umgebungen viel Speicher zugestanden wird. Hier können statt einer Occupancy Grid Map auch Landmarken Verwendung finden. All das erhöht jedoch wieder den Anspruch der Implementierung.

### 2.3.4. Monte Carlo Localization

Dieses Verfahren ist heutzutage aufgrund seiner Einfachheit und seines Potentials einer der populärsten Algorithmen zur Roboterlokalisation. Er wurde 1999 von Fox et al. [1999] vorgestellt. Auf der Karte werden sog. *Partikel* verteilt, wobei jedes Partikel eine Posevermutung darstellt. Bewegungen des Roboters werden auf alle Partikel übertragen. Nach jeder Messung werden sie nach Wichtigkeit gewichtet, wobei zur Messung passende Partikel höher gewichtet werden. In einem Resampling-Prozess werden von Zeit zu Zeit alle Partikel mit zu niedriger Wichtigkeit aussortiert, gleichzeitig werden in der Umgebung wichtiger Partikel neue Partikel erzeugt.
Monte Carlo Localization erlaubt eine globale Lokalisation und kann auf das Kidnapped Robot-Problem erweitert werden, indem immer einige Partikel zufällig hinzugefügt werden. Desweiteren ist es möglich, die Partikelzahl dynamisch zu verändern, was eine Anpassung an die momentan zur Verfügung stehenden Rechenkapazitäten ermöglicht. Wie bei der Lokalisation per Histogramm ist man nicht auf Normalverteilungen eingeschränkt und kann mehrere Vermutungen gleichzeitig verfolgen. Außerdem ist dieses Verfahren sowohl für Feature-basierte Karten wie auch für Occupancy Grid Maps geeignet.

## 2.4. Das SLAM-Problem

Die Abkürzung SLAM (**S**imultaneous **L**ocalization **A**nd **M**apping) beschreibt die Problematik, gleichzeitig eine Lokalisation durchzuführen und eine Karte der Umgebung zu erstellen. Falls

bereits eine Karte vorhanden ist (wie im vorherigen Kapitel beschrieben) ist eine Lokalisation vergleichsweise einfach durchzuführen. Ebenso ist das Aufzeichnen einer Karte bei bekannter Position relativ einfach. Beginnt der Agent jedoch ohne eine Karte und kann seine Position nur unzureichend genau verfolgen steht man vor dem SLAM-Problem. Dieses lässt sich auch als Henne-Ei-Problem betrachten: Ohne Karte keine Lokalisation, ohne Lokalisation keine Karte.
Die meisten SLAM-Verfahren lassen sich anhand folgender Aspekte einteilen:

**Full vs. Online SLAM** Während beim Online SLAM neben der Karte die aktuelle Pose des Roboters von Interesse ist, wird bei Full SLAM-Algorithmen der gesamte bisher zurückgelegte Weg gespeichert und im weiteren Verlauf berücksichtigt. Dies erhöht zwar die Genauigkeit des Algorithmus, wird aber schnell rechen- und speicherintensiv.

**Volumetric vs. Feature-based SLAM** Dies bezieht sich auf die Art der genutzten Kartendarstellung. Während bei Volumetric SLAM eine metrische Karte wie z.B. eine Occupancy Grid Map zum Einsatz kommt, wird die Karte bei Feature-based SLAM aus Landmarken aufgebaut. Vgl. auch Kap. 2.2.

**Active vs. passive SLAM** Bisher wurde immer angenommen, dass der Agent seine Steuerbefehle von außerhalb, z.B. durch einen Menschen erhält. Im Fall von SLAM spricht man dann von passive SLAM. Bei active SLAM steuert sich der Agent hingegen selbst, um aktiv zur Bildung der Karte und zur eigenen Lokalisierung beizutragen.

Die folgenden Kapitel geben eine kurzen Überblick über einige der gängigen Lösungen des SLAM-Problems.

### 2.4.1. SLAM mittels Kalman Filter

Hier sei besonders EKF SLAM erwähnt, welches mit zu den frühesten SLAM-Implementierungen gehört (vgl. Thrun et al. [2005], Kap. 10.2.1). Voraussetzung sind eindeutig identifizierbare Landmarken und ein dementsprechend guter Erkennungsalgorithmus. Vereinfacht gesagt handelt es sich um eine EKF-Lokalisation, bei der nicht nur die Roboterpose, sondern auch die Position aller erkannten Landmarken geschätzt wird. Der Algorithmus eignet sich nicht für große Karten, da die Berechnungen sonst zu rechenintensiv werden.
Es existieren diverse weitere Implementierungen mit Kalman Filter (vgl. Thrun et al. [2005]),

u.a. UKF SLAM, EIF (Extended Information Filter, überführt den EKF in die kanonische Form) SLAM und SEIF (Sparse EIF, führt eine zusätzliche *sparsification* der Daten durch, wodurch nur noch benachbarte Features korreliert werden; deutlich effizienter als EKF SLAM, allerdings etwas ungenauer) SLAM. Allen ist gemeinsam, dass sie nur für kleinere bis mittelgroße Karten geeignet sind.

### 2.4.2. GraphSLAM

Der GraphSLAM-Algorithmus ermöglicht eine Lösung des Full SLAM-Problems. GraphSLAM korreliert Features und Poses und baut daraus einen Graphen auf. Dieser Graph wird immer wieder überarbeitet, indem versucht wird, Features mit in der Realität identischen Koordinaten zu finden, was also ein Feature ist, das fälschlicherweise mehrmals in den Graphen eingetragen wurde. Dadurch wird der Graph immer wieder korrigiert, wodurch mit diesem Algorithmus gute Ergebnisse zu erzielen sind.
Die Grundlagen für diesen Algorithmus wurden 1997 durch eine Veröffentlichung von Lu und Milios [1997] gelegt.

### 2.4.3. FastSLAM

FastSLAM wurde 2002 von Michael Montemerlo entwickelt (Montemerlo et al. [2002]) und zählt heutzutage mit zu den effektivsten SLAM-Verfahren überhaupt. Aufbauend auf einem Partikelfilter löst dieser Algorithmus sowohl das Online als auch das Full SLAM-Problem und ist für Feature Based wie für Occupancy Grid Maps geeignet. Jedes Partikel enthält eine eigene Pfadschätzung sowie Schätzungen für alle Features der Karte, welche jeweils durch EKFs repräsentiert werden. FastSLAM existiert auch als Version 2.0, welche nochmals verbessert wurde (Montemerlo et al. [2003]).
Aufbauend auf FastSLAM 1.0 wurde der sog. *Distributed Particle (DP) SLAM* entwickelt, welcher den effizienten Umgang mit mehreren Kartenversionen gleichzeitig erlaubt (Eliazar und Parr [2003]). Dieser wurde ebenfalls zu Version 2.0 ausgebaut, mit nochmals gesteigerter Effizienz (Eliazar und Parr [2004]).

## 2.5. Zusammenfassung

Die bisher vorgestellten Techniken ermöglichen heutzutage nicht nur eine zuverlässige Lokalisation, sondern auch eine Lösung des SLAM-Problems. Dennoch ist die dazu benötigte Hardware (insb. mit einem hochwertigen Quadrokopter als Plattform) nach wie vor zu teuer, um Systeme zur möglicherweise einmaligen Verwendung zu entwerfen.
Um diesen Problem zu begegnen wird im Rahmen des *AQopterI8*-Projekts ein Quadrokopter von Grund auf neu entworfen, sodass dieser sowohl hinsichtlich Funktionalität als auch niedriger Kosten optimiert werden kann. Wie bereits in Kap. 1 erwähnt ist es von großer Bedeutung, diesen Quadrokopter mit einem hohen Grad an Autonomie auszustatten, damit er die Aufgaben erfüllen kann, die für Menschen evtl. zu gefährlich sind. Um einen wichtigen Beitrag zur Autonomie des Quadrokopters zu leisten wird im weiteren Verlauf dieser Arbeit ein Lokalisationsverfahren entwickelt, dass sowohl robust arbeitet als auch schnell zu berechnen ist.

# 3. Konzept

## 3.1. Überblick

In diesem Kapitel wird das dieser Arbeit zugrunde liegende Konzept vorgestellt. Dazu werden zunächst die an das Lokalisationsverfahren zu stellenden Anforderungen beschrieben, welche in den daran anschließenden Vergleich der bisher vorgestellten Lokalisationsverfahren (s. Kap. 2.3) einfließen. Das daraus gewählte Verfahren wird schließlich weiter ausgearbeitet.

### 3.1.1. Anforderungen

Wie bereits in Kap. 1 erwähnt, wird das zu erarbeitende Lokalisationsverfahren im Rahmen des *AQopterI8*-Projekts entwickelt. Daraus ergibt sich aus mehreren Gründen eine hohe Robustheit als Anforderung an das Lokalisationsverfahren:

**Plattform Quadrokopter** Während bei einem bodengebundenen Roboter davon ausgegangen werden kann, dass er seine Position nur ändert, wenn er seinen Antrieb einsetzt, muss bei Quadrokoptern stets mit Ungenauigkeiten und Drift gerechnet werden. Ein Quadrokopter ist äußeren Einflüssen, insb. Wind bzw. Luftzirkulationen, deutlich stärker ausgesetzt als ein auf dem Boden stehender Agent. Selbst bei Windstille muss die Positionsregelung eines Quadrokopters immer wieder einen kleinen Drift ausgleichen, was zu minimalen Bewegungen führt.

Desweiteren bedeutet jede Bewegung eines Quadrokopters auch eine Neigung der Plattform, was Sensormessungen mit festen Richtungen (z.B. senkrecht zum Boden oder in der Rotorebene) verfälscht. Diese Messungen könnten bspw. per Sensorfusion mit einem Neigungssensor verbessert werden.

**Aspekt LowCost** Neben einer Verfälschung von Sensorwerten durch z.B. unberücksichtigte Neigungen der Plattform sind v.a. die Eigenschaften der Sensoren selbst für deren Zuverlässigkeit ausschlaggebend. Das Lokalisationsverfahren muss auch mit Sensorik funktionieren, die nicht immer hochgenau und ohne jeden Drift arbeitet. Der Einsatz deutlich besserer Sensorik ist in diesem Fall keine Lösung, da der erhebliche finanzielle Mehraufwand den Zielen des *AQopterI8*-Projekts widerspricht. Außerdem ist ein Lokalisationsverfahren, dass ausschließlich mit hochgenauen Messwerten funktionsfähig ist, per se nicht robust.
Aus diesem und dem vorherigen Punkt dieser Liste geht hervor, dass das zu wählende Lokalisationsverfahren auch mit eher ungenauen und unzuverlässigen Messungen umgehen können muss.

**Geplantes Einsatzgebiet** In der Einleitung dieser Arbeit wurde bereits zitiert, dass das *AQopterI8*-Projekt entwickelt wird, um z.B. Rettungskräften eine weitere Möglichkeit zur Suche nach Menschen in brennenden Gebäuden oder eingestürzten Minenschächten zur Verfügung zu stellen (vgl. Kap. 1). Aus diesem Grund werden hohe Ansprüche an die Autonomie des Quadrokopters gestellt, welche insb. durch eine zuverlässige Lokalisation (vgl. Kap. 2.3) gewährleistet werden muss. Es erscheint daher sinnvoll, ein Lokalisationsverfahren zu wählen, dass das Problem der globalen Lokalisation (s. Kap. 2.3) lösen kann. So ist gewährleistet, dass der Quadrokopter in der Lage ist, sich auch nach einem Verlust seiner Positionsschätzung (was nie komplett ausgeschlossen werden kann, sei es durch eine sich verändernde Umgebung oder Ungenauigkeiten im Lokalisationsverfahren) erneut zu lokalisieren.

**Ressourcenbedarf** Das Lokalisationsverfahren muss möglichst sparsam mit den Ressourcen (Speicher und Rechenzeit) des Quadrokopters umgehen, da aufgrund der fliegenden Plattform Quadrokopter Gewicht und Größe der mitgeführten Hardware begrenzt ist. Ein schnelles und sparsames Lokalisationsverfahren ermöglicht den Einsatz etwas leistungsschwächerer Hardware, welche wiederum weniger Platz, Gewicht, Kühlung und Energie benötigt, was dem Flugverhalten des Quadrokopters (Schnelligkeit, Manövrierfähigkeit, Flugdauer, etc.) zugutekommt.

Neben diesen Aspekten muss darauf geachtet werden, dass sich das gewählte Verfahren gut für eine Occupancy Grid Map, wie sie im *AQopterI8*-Projekt genutzt wird (vgl. Schmitt [2012]), umsetzen lässt.

### 3.1.2. Auswahl eines geeigneten Lokalisationsverfahrens

Es werden lokale Globalisationsverfahren aus den in Kap. 3.1.1 genannten Anforderungen ausgeschlossen werden. Dabei soll auch die EKF-Lokalisation ausgeschlossen, da sie zwar auf eine globale Lokalisation ausgebaut werden kann (s. Kap. 2.3.2), dies aber mit deutlichem Aufwand verbunden ist. Zudem eignet sich EKF primär für Landmarken-basierte Karten, außerdem kann eine Repräsentation der Positionsschätzung mittels einer Gaußschen Verteilung z.B. in der Nähe von Wänden oder Ecken problematisch sein. Hier besteht insb. in komplexen Innenräumen die Gefahr, dass eine Positionsschätzung durch z.B. Rundungs- oder Messfehler durch eine nahegelegene Wand „springt“ (sich der Quadrokopter also plötzlich auf der anderen Seite der Wand befindet).
Damit bleiben noch zwei Verfahren: Lokalisation per Histogramm oder Monte-Carlo-Lokalisation bzw. Lokalisation per Partikelfilter. Grundsätzlich sollten beide Verfahren geeignet sein, da sie sich für Occupancy Grid Maps sehr gut eignen, eine globale Lokalisation ermöglichen, sehr viele Verteilungsformen annähern können (was zur Robustheit beiträgt) und (im Vergleich zu z.B. Kalman Filter-basierten Verfahren) relativ einfach zu implementieren sind. Allerdings bietet ein Partikelfilter noch einige Vorteile gegenüber einer Histogrammlokalisation:

**Effizienz** Wie bereits in Kap. 2.3.3 beschrieben stellt ein Histogramm keine sehr effiziente Möglichkeit dar, Positionsschätzungen anzunähern. Bei einem Partikelfilter kann hingegen (insb. über die Anzahl der Partikel) die benötigte Rechenleistung auf Kosten der Genauigkeit - selbst während der Laufzeit - deutlich beeinflusst werden.

**Erweiterbarkeit** Aufgrund der Speicherauslastung ist es bei einer Lokalisation per Histogramm mit großem Aufwand verbunden, das Verfahren effizient auf sehr große oder gar dreidimensionale Karten auszuweiten. Ein Partikelfilter kann hingegen deutlich einfacher von der Karte getrennt implementiert werden, wodurch in der Zukunft evtl. nötige Erweiterungen etwas einfacher umzusetzen sind.

**Verbreitung** Partikelfilter sind aufgrund ihrer Einfachheit und Effizienz weit verbreitet (vgl. Kap. 2.3.4), wodurch eine große Menge an Informationen (insb. im Internet) diesbezüglich zur Verfügung steht. Außerdem kann damit gerechnet werden, dass Partikelfilter aufgrund ihrer Verbreitung stets weiterentwickelt und verbessert werden, was eine evtl. nötige zukünftige Verbesserung des in dieser Arbeit entwickelten Systems vereinfacht. Nicht zuletzt basieren einige sehr effektive SLAM-Verfahren (z.B. FastSLAM, s. Kap. 2.4.3) auf Partikelfiltern, was eine Erweiterung eines Partikel-basierten Lokalisationsverfahrens zu einem SLAM-Verfahren ebenfalls etwas beschleunigen kann.

Damit fällt die Wahl schlussendlich auf ein Partikel-basiertes Lokalisationsverfahren. Dieses wird im folgenden Kapitel entworfen.

## 3.2. Struktur

### 3.2.1. Übersicht

Die grundlegende Struktur des Lokalisationsverfahrens verdeutlicht Abb. 3.1.

Zentrales Element des Verfahrens ist der sog. *ParticleController*. Dieser verwaltet, wie der Name

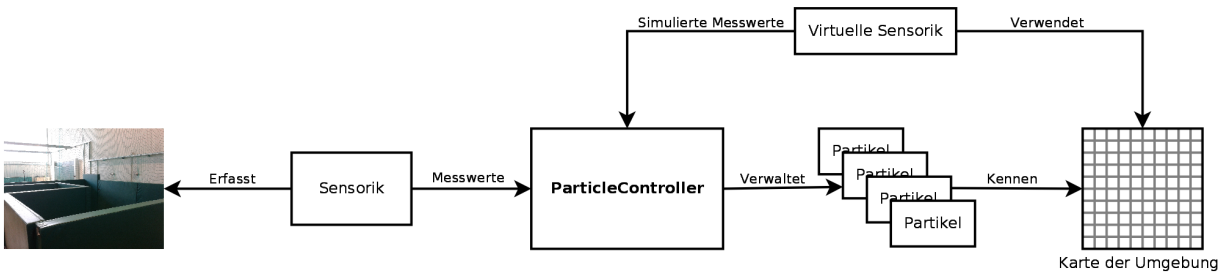



**Abbildung 3.1.:** Grobe Struktur des Lokalisationsverfahrens

schon andeutet, alle existierenden Partikel. Das bedeutet, dass er neue Partikel erzeugt, bestehende Partikel weiterentwickelt und zu schlechte Partikel wieder löscht. Jedes Partikel steht dabei für eine mögliche Pose des Quadrokopters. Diese müssen regelmäßig auf Plausibilität überprüft werden. Dazu nutzt der ParticleController eine Karte der Umgebung, in der er für jedes Partikel berechnen kann, welche Abstandsmessungen der Sensorik erwartet werden. Durch einen Vergleich der erwarteten Sensormessungen mit den tatsächlichen Messungen kann dann festgestellt werden, wie weit das betroffene Partikel der wahren Pose des Quadrokopters entsprechen könnte.

Sollten für ein Partikel mehrere Messungen in Folge nicht stimmen, wird es gelöscht. Ein neues Partikel wird dann in der Nähe eines bestehenden, plausiblen Partikels platziert. Sollte ein solches nicht existieren, wird das neue Partikel stattdessen zufällig in der Karte platziert.
Neben Abstandsmessungen benötigt der ParticleController auch Bewegungsinformationen. Sobald diese erhalten wurden, werden sie auf alle Partikel angewendet, diese bewegen sich also so, wie durch die gemessene Bewegung vorgegeben. Der Grund hierfür ist, dass nie bekannt ist, welche Partikel der wahren Pose des Quadrokopters entsprechen, weshalb alle so behandelt werden, als ob sie stimmen würden. Aus diesem Grund bewegen sich alle Partikel so, wie es der Quadrokopter wahrgenommen hat (im Prinzip handelt es sich hierbei um eine sehr kurzfristige Koppelnavigation für viele veschiedene Poses). Der Vorgang des Anwendens von Bewegungsinformationen und Abstandsmessungen auf alle Partikel wird im weiteren Verlauf der Arbeit als *Weiterentwicklung* der Partikel in die nächste Generation bezeichnet.
Die folgenden Kapitel vertiefen einzelne Aspekte des Lokalisationsverfahrens weiter. Alle IN KAPITÄLCHEN gesetzten Begriffe sind Parameter, die vor oder während der Laufzeit des Lokalisationsverfahrens definiert werden können. Sie werden im Folgenden als gegeben betrachtet. Eine genauere Erläuterung dieser Parameter findet sich in Anh. A.

### 3.2.2. Partikel

Zusätzlich zur Pose werden jedem Partikel $p_i$ zwei wichtige Kenngrößen zugeordnet: *Vertrauen* $|p_i|$ und *Alter* $a_i$. Jedes Partikel beginnt mit einem Alter von $0$, welches sich bei jedem Erreichen der nächsten Generation um $1$ erhöht, und einem Vertrauen von $0.5$. Das Alter erlaubt es, Partikel anhand einer festen Grenze in „jung“ und „alt“ zu teilen, wodurch Berechnungen vom Alter eines Partikels abhängig gemacht werden können (s. Abb. 3.3).
Deutlich wichtiger als das Alter eines Partikels ist sein Vertrauen (häufig auch als „Gewicht(ung)“ bezeichnet). In dieser Arbeit ist das Vertrauen als ein Wert in $[0, 1]$ definiert, wobei $1$ das bestmögliche Vertrauen ist (Partikel mit einem Vertrauen deutlich über $0.5$ werden als „gute“ Partikel bezeichnet). Anschaulich gesagt beschreibt das Vertrauen, für wie wahrscheinlich es der ParticleController hält, dass ein Partikel der wahren Pose des Quadrokopters entspricht. Sollte das Vertrauen sehr niedrig sein, wird das Partikel gelöscht und durch ein neues ersetzt, da der ParticleController davon ausgeht, dass das Partikel definitiv nicht richtig ist.

Die Aktualisierung des Vertrauens bei der Weiterentwicklung eines Partikels ist in Kap. 3.2 dargestellt.

Das ursprüngliche Vertrauen (welches sich im Intervall $[0, 1]$ befindet) wird durch drei Werte mo-

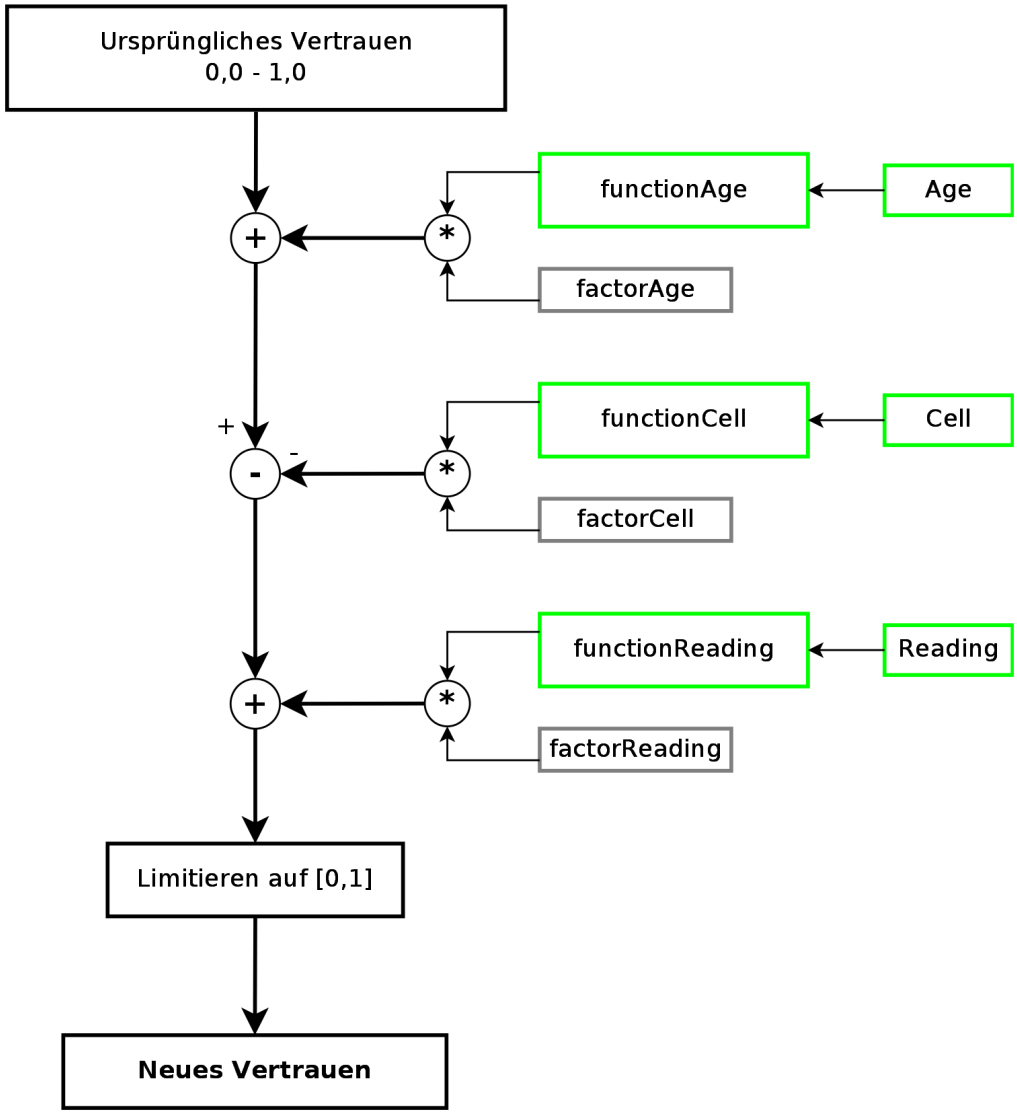



**Abbildung 3.2.:** Diagramm zur Vertrauensentwicklung jedes Partikels

difiziert (von oben nach unten): Dem Alter des Partikels, der Wahrscheinlichkeit, dass die Zelle der Karte, in der sich das Partikel befindet, belegt ist, sowie dem Verhältnis von simulierten und echten Abstandsmessungen. Diese Werte stammen aus speziellen Funktionen (s. nächste Kapitel), welche durch FACTORAGE, FACTORCELL und FACTORREADING gewichtet werden. Dies ermöglicht es, den Einfluss der einzelnen Funktionen auf das Vertrauen zu steuern.

Das modifizierte Vertrauen wird schließlich auf das gültige Intervall $[0, 1]$ begrenzt. womit das

neue Vertrauen berechnet wurde.

Alle Funktionen haben gemeinsam, dass sie nur aus linearen Verläufen zusammengesetzt sind. Dies beschleunigt ihre Berechnung gegenüber z.B. logistischen Funktionsverläufen, liefert aber dennoch qualitativ akzeptable Werte.
Die drei verwendeten Funktionen sind folgendermaßen aufgebaut:

#### 3.2.2.1. functionAge

Diese Funktion erzeugt einen Bonus abhängig vom Alter des Partikels. Die Überlegung dahinter ist, dass ältere Partikel bereits mehreren Überprüfungen standgehalten haben, sie also grundsätzlich eine etwas höhere Wahrscheinlichkeit haben, richtig zu sein. Der Bonus, den sie durch diese Funktion erhalten, trägt dazu bei, sie gegenüber kleineren Unstimmigkeiten etwas widerstandsfähiger zu machen; mit älteren Partikeln wird also etwas toleranter umgegangen, um zu verhindern, dass gute Positionsschätzungen aufgrund kurzzeitiger Unstimmigkeiten verworfen werden.
Die Funktion ist nach oben begrenzt (s. Abb. 3.3), damit ein hohes Alter nicht dazu führt, dass Partikel „unsterblich“ werden. Sie wird durch folgende Formel beschrieben:

$$functionAge(x) = \min(x, \text{FUNCTIONAGETHRESHOLD}) \tag{3.1}$$

$$\text{FUNCTIONAGETHRESHOLD} \geq 0 \tag{3.2}$$

Dabei beschreibt FUNCTIONAGETHRESHOLD die Altersgrenze, durch die Partikel in jung und alt (alt, falls $a_i \geq$ FUNCTIONAGETHRESHOLD) geteilt werden.

#### 3.2.2.2. functionCell

Da ein Quadrokopter - oder allgemeiner jeder Agent - nicht durch Wände gehen oder fliegen kann, wird für jedes Partikel berücksichtigt, ob es sich laut Karte an einem Ort befindet, der nicht durch eine Wand o.ä. blockiert ist. Sollte dies dennoch der Fall sein, wird das Vertrauen für das betroffene Partikel deutlich gesenkt. Befindet sich das Partikel an einer freien Stelle wird dies nicht als Bonus gewertet, da das der Normalfall sein sollte. Problematisch ist dieses Vorgehen, wenn in der Karte eine Wand eingezeichnet ist, die in der Realität nicht existiert. Dies wird jedoch in Kauf genommen, da es im Zweifelsfall sicherer ist, eine gute Positionsschätzung zu verwerfen,

als sich an einer nicht zugänglichen Stelle zu lokalisieren.

Zu beachten ist, dass die Funktion (s. Abb. 3.4) positive Werte zurückliefert, diese jedoch *negativ* in das Vertrauen einfließen (vgl. Abb. 3.2). Hiermit soll noch deutlicher gemacht werden, dass es sich bei *functionCell* ausschließlich um einen Malus handelt.

$cellFree$ (kurz $c_{free}$) und $cellOccupied$ (kurz $c_{occ}$) sind Grenzwerte, mit denen bestimmt werden kann, ob eine Zelle der Karte frei oder belegt ist. Da jede Zelle eine Wahrscheinlichkeit besitzt, mit der sie belegt ist (vgl. Schmitt [2012]), muss dieser Wert etwas diskretisiert werden. Dazu wird festgelegt, dass eine beliebige Zelle $c_i$ mit einer Wahrscheinlichkeit $p(c_i)$ dann als frei interpretiert wird, wenn $p(c_i) \leq c_{free}$, und sie dann als belegt interpretiert wird, wenn $p(c_i) \geq c_{occ}$. Zwischen $cellFree$ und $cellOccupied$ kann ein Bereich liegen, in dem keine sinnvolle Annahme über die Zellenbelegung getroffen werden kann. Aus diesem Grund wird im betroffenen Bereich des Graphen ein Anstieg von $cellFree$ zu $cellOccupied$ verwendet, sodass sich auch Zellen mit unbekannter Belegung negativ auf das Vertrauen eines Partikels auswirken, gleichzeitig aber ein kontinuierlicher Übergang zwischen beiden Grenzwerten vorliegt.

Folgende Formel liegt der Funktion zugrunde:

$$functionCell(x) = \begin{cases} 0 & für \quad x \leq c_{free} \\ \frac{x - c_{free}}{c_{occ} - c_{free}} & für \quad c_{free} < x < c_{occ} \\ 1 & für \quad x \geq c_{occ} \end{cases} \tag{3.3}$$

#### 3.2.2.3. functionReading

Der wichtigste Aspekt der Lokalisation sind Abstandsmessungen. Wie bereits in Kap. 3.2.2 dargestellt ist es nötig, die realen Sensormessungen mit den erwarteten zu vergleichen und das Verhältnis mit einer Zahl zu bewerten, die dann in das Vertrauen des Partikels einfließt. Dazu wird folgendes Verfahren benutzt:

- Gegeben sind $n$ reale Abstandsmessungen $s_1..s_n$
  sowie $n$ erwartete Abstände $a_1..a_n$

- Berechne $n$ absolute Abweichungen: $\Delta_i = |s_i - a_i|, \quad i = 1..n$
  Es werden absolute Abweichungen benutzt, da relative Abweichungen (per Quotient) dazu führen würden, dass auf weite Entfernungen große Abweichungen vom erwarteten Wert genauso bewertet würden wie kleine Abweichungen auf kurze Entfernungen. Damit wür-

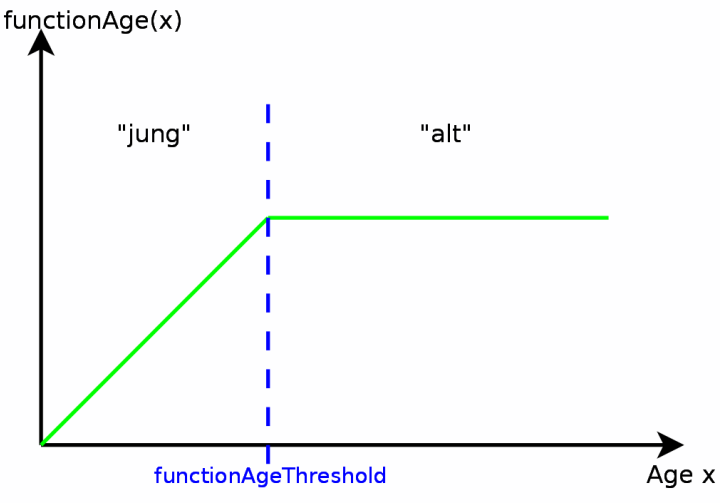



**Abbildung 3.3.:** functionAge

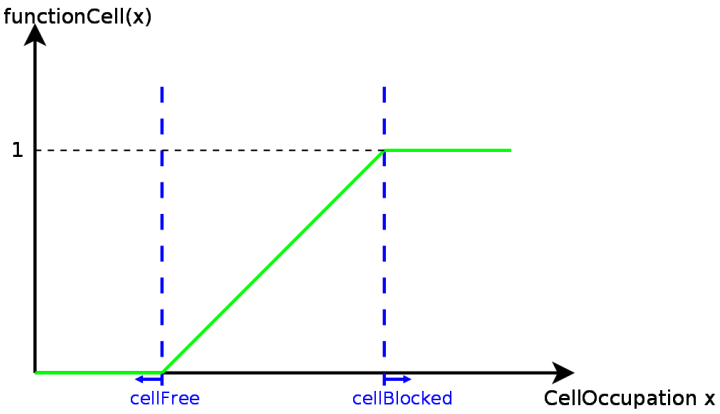



**Abbildung 3.4.:** functionCell

den allerdings Abstandsinformationen teilweise geringere Beachtung finden, da es durchaus von Bedeutung sein kann, ob sich in größerer Entfernung wie erwartet ein Hindernis befindet oder nicht.

- Entfernen der FUNCTIONREADINGBADSENSORS größten Werte aus $\Delta$ führt zu $\Delta'$ sowie $n' = n -$ FUNCTIONREADINGBADSENSORS
  Dieser Schritt dient dazu, eine gewisse Anzahl von Fehlmessungen zu verwerfen, wie sie z.B. durch Sonneneinstrahlung, vorbeigehende Personen oder schlicht defekte Sensorik verursacht werden können.

- Berechnen des durchschnittliches Fehlers: $x = \frac{\sum \Delta'}{n'}$

Der durchschnittliche Fehler $x$ wird nun durch *functionReading* in einen Wert umgewandelt (s. Abb. 3.5), der anschließend in das Gesamtvertrauen des Partikels einfließt. Solange $x <$ FUNCTIONREADINGOFFSET ist, wird die reale Messung als zur erwarteten Messung passend interpretiert, was das Partikelvertrauen positiv beeinflusst. Durch ein geeignetes FUNCTIONREADINGOFFSET kann auch die Messungenauigkeit der verwendeten Abstandssensorik berücksichtigt werden, da sowohl $x$ als auch FUNCTIONREADINGOFFSET Längen sind, die direkt im Bezug auf die realen Umstände interpretiert werden können: $x = 37cm$ bedeutet bspw., dass die Abstandsmessungen im Schnitt 37cm von den erwarteten Werten abweichen (vgl. dazu auch Anh. A).

Es ist zu beachten, dass dieses Verfahren insofern etwas problematisch ist, als dass die Funktion

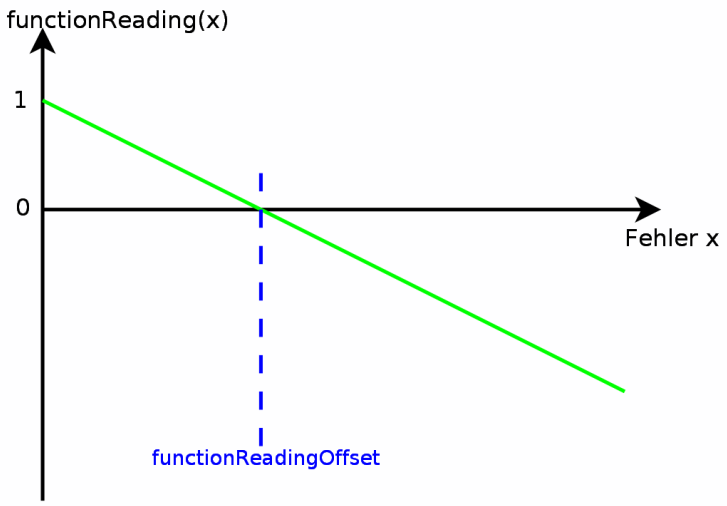



**Abbildung 3.5.:** functionReading

die Messwerte nicht auf Plausibilität überprüft. Sollte also der Fall eintreten, dass zufällig alle Abstandsmessungen gleichzeitig fehlerhaft sind, überträgt sich dieser Fehler durch die Funktion weiter auf die Partikel, was im schlimmsten Fall zum Aussterben aller Partikel führen kann. Dieser Fall wird im Rahmen dieser Arbeit als sehr unwahrscheinlich betrachtet, da der verwendete Quadrokopter insgesamt 28 Sensoren zur Abstandsmessung benutzt (s. Kap. 4.1).

### 3.2.3. Weiterentwicklung der Partikel

Sobald sich der Quadrokopter von Punkt *A* nach Punkt *B* bewegt hat, wird in *B* eine Abstandsmessung durchgeführt. Diese wird zusammen mit der gemessenen Bewegung von *A* nach *B* an den ParticleController übergeben, welcher mit diesen Informationen alle Partikel weiterentwickelt.
Abb. 3.6 verdeutlicht das Konzept der Weiterentwicklung. Zu Beginn wartet der ParticleController auf die benötigten Bewegungs- und Abstandsinformationen. Dann wird für jedes Partikel zunächst geprüft, ob dessen Vertrauen noch akzeptabel ist ($|p_i| \geq$ MINCONFIDENCE). Sollte dies nicht der Fall sein wird das Partikel gelöscht, da davon ausgegangen werden kann, dass es die wahre Pose des Quadrokopters nicht annähert. Ansonsten wird das Vertrauen des Partikels aktualisiert. Dies beinhaltet alle in Kap. 3.2.2 vorgestellten Aspekte, also z.B. den Vergleich zwischen realen Abstandsmessungen und den für dieses Partikel vermuteten Abstandsmessungen.
Nachdem alle vorhandenen Partikel weiterentwickelt wurden, werden solange neue Partikel erzeugt, bis die gewünschte (als ideal vermutete) Anzahl erreicht wurde. Dabei werden GROUPPROBABILITY (s. Anh. A) der neuen Partikel in der Nähe guter Partikel platziert, die restlichen werden zufällig verteilt. Zufällig platzierte Partikel werden mit $|p_i| := 0.5$ erzeugt, während in der Nähe guter Partikel platzierte Partikel mit einem höheren Vertrauen erzeugt werden. Dies bewirkt, dass sich der Algorithmus auf die vielversprechendsten Vermutungen konzentriert. Gleichzeitig sorgen die zufällig platzierten Partikel dafür, dass stets auch berücksichtigt wird, dass evtl. alle momentanen Vermutungen falsch sind.
Nachdem die gewünschte Anzahl an Partikeln erreicht wurde wird das durschnittliche Vertrauen $C$ aller Partikel berechnet. Dies ist eine wichtige Kenngröße, da ein großes $C$ bedeutet, dass der ParticleController sich sehr sicher ist, eine korrekte Lokalisation durchgeführt zu haben. Dieses Vertrauen könnte als einfaches arithmetisches Mittel berechnet werden. Da jedoch neu platzierte Partikel in den ersten Generationen noch große Schwankungen des Vertrauens aufweisen können (durch schlechte oder besonders gute Platzierung), wird stattdessen ein gewichtetes arithmeti-

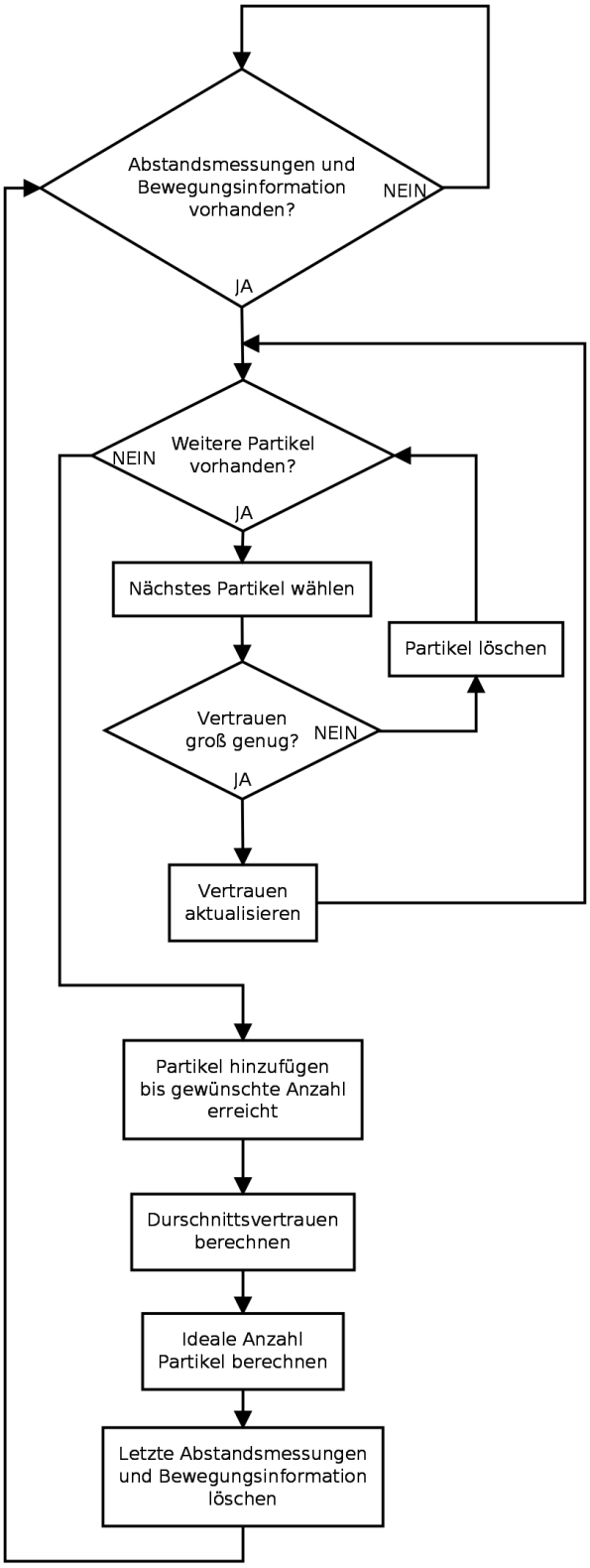



**Abbildung 3.6.:** Schema der Weiterentwicklung aller Partikel

sches Mittel verwendet, welches alte Partikel stärker als junge Partikel gewichtet.
Abschließend wird die ideale Anzahl an Partikeln berechnet. Sobald der ParticleController eine Lokalisation durchführen konnte, konzentriert sich der Großteil der Partikel auf einige wenige Poses. Da es jedoch nicht nötig ist, eine Pose mit mehreren hundert Partikeln anzunähern, wird die ideale Partikelzahl bei hohem Durschnittsvertrauen $C$ gesenkt. Dies senkt gleichzeitig die benötigte Rechenzeit. Es ist zu beachten, dass die berechnete ideale Anzahl niemals zu klein (z.B. weniger als 10 Partikel) wird, da sonst auch eine gefundene Pose nicht gut verfolgt werden kann. Gleichzeitig darf MAXPARTICLES nicht überschritten werden, was die maximal erlaubte Anzahl an Partikeln angibt (und damit auch die benötigte Rechenzeit begrenzt).

#### 3.2.3.1. Bewegungsmodell

Zur Modellierung der Bewegung wird das sog. *Odometry Motion Model* genutzt, wie es u.a. in Thrun et al. [2005] vorgestellt wird. Dabei wird auf eine Pose zunächst eine Rotation $\delta_{rot1}$ angewendet. Dann wird eine Translation um $\delta_{trans}$ in Blickrichtung durchgeführt, gefolgt von einer zweiten Rotation $\delta_{rot2}$ am erreichten Punkt (vgl. Abb. 3.7). Der Vorteil dieser Bewegungsrepräsentation ist ihre Einfachheit und Flexibilität. Zusätzlich können künstliche Fehler in jedem Bewegungsschritt hinzugefügt werden, welche eine charakteristische Streuung der Partikel ergeben. Dies ist in Anh. C dargestellt.

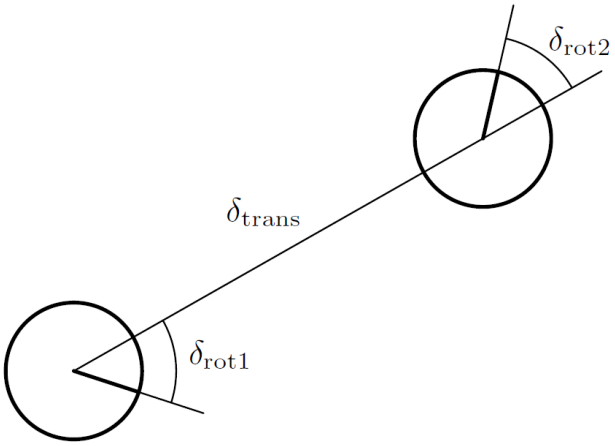



**Abbildung 3.7.:** Das Odometry Motion Model. Quelle: Thrun et al. [2005]

#### 3.2.3.2. Sensormodell

Ein geeignetes Sensormodell wurde bereits in Schmitt [2012] vorgestellt. Dabei wird jeder Sensor durch einen Kegel mit definierter maximaler Reichweite repräsentiert. In der Karte werden alle Zellen bestimmt, die im Sensorkegel liegen, und die Entfernung vom Sensor zur nahesten belegten Zelle berechnet. Wie in Abb. 3.8 dargestellt führt dies dazu, dass am Rand des Kegels liegende Hindernisse so gemessen werden, als ob sie sich in der Mitte des Sensorkegels befinden. Damit kommt man dem realen Verhalten von Ultraschall- oder Infrarotsensoren sehr nahe. Weder den realen noch den simulierten Abstandsmessungen wird ein künstlicher Fehler hinzu-

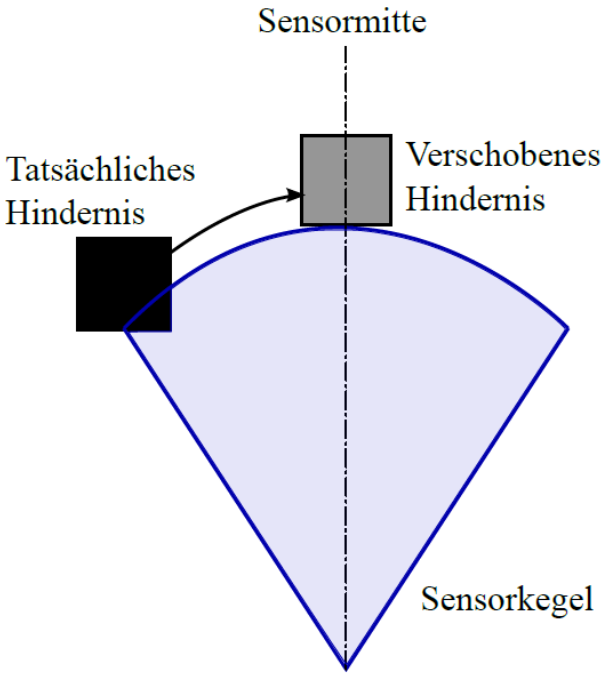



**Abbildung 3.8.:** Verschiebung von Hindernissen in die Kegelmitte. Quelle: Schmitt [2012]

gefügt, da die realen Messungen bereits fehlerbehaftet sind und die für den Vergleich von realen und künstlichen Messungen zuständige Funktion (s. Kap. 3.2.2.3) eine Fehlertoleranz besitzt. Ein künstlicher Fehler würde diese Toleranz evtl. schon alleine komplett auslasten, wodurch reale Fehler nichtmehr ausgeglichen werden könnten.

# 4. Implementierung

In diesem Kapitel wird die Umsetzung des zuvor vorgestellten Konzepts erläutert. Dazu wird zunächst das bestehende System erläutert, auf das diese Arbeit aufbaut. Darauf folgt eine Vorstellung der Softwarestruktur sowie einzelner Details. Die Software wird dann in das bestehende System eingebunden. Abschließend wird die erstellte Benutzeroberfläche erklärt.

## 4.1. Bestehendes System

Das Software-System des *AQopterI8*-Projekts besteht aus zwei Teilen: Zum einen der Software, die auf dem Quadrokopter läuft, zum anderen der Software der „Bodenstation“, die alle Daten des Quadrokopters empfängt und verarbeitet. Außerdem erlaubt sie das Senden von Befehlen an den Quadrokopter. Die Bodenstation besteht aus einem gewöhnlichen Windows-Rechner, der per Bluetooth mit dem Quadrokopter kommuniziert.
Wie bereits in Kap. 3.2 erläutert, müssen für den Lokalisationsalgorithmus Abstands- und Bewegungsmessungen bereitgestellt werden. Dazu wird einerseits das in Benz [2013] entwickelte System zur Abstandsmessung verwendet. Es fusioniert in acht Richtungen insgesamt 16 Infrarot- sowie 12 Ultraschall-Sensoren. Damit wird eine Reichweite von bis zu 5m erreicht, die erzielte Genauigkeit hängt dabei u.a. von der Oberflächenbeschaffenheit des detektierten Objekts und dem Einfallwinkel der Signale ab.
Zur Bewegungsmessung wird ein sog. *Optical Flow-Sensor* (dt. optischer Fluss, kurz OF) eingesetzt, dessen Integration in das *AQopterI8*-Projekt in Strohmeier [2012] beschrieben ist. Ein OF-Sensor ist eine auf den Boden gerichtete Kamera, deren Signal so analysiert wird, dass damit Bewegungen des Bildausschnittes berechnet werden können.
Die verwendete Sensorik sendet ihre Messwerte auf Anfrage an die Bodenstation, wodurch diese Arbeit vollständig in das Bodenstations-System integriert werden kann. Das dortige Programm

wurde mit C++ und Qt, einer Bibliothek u.a. zur Entwicklung von Benutzeroberflächen, entwickelt (s. Abb 4.1).

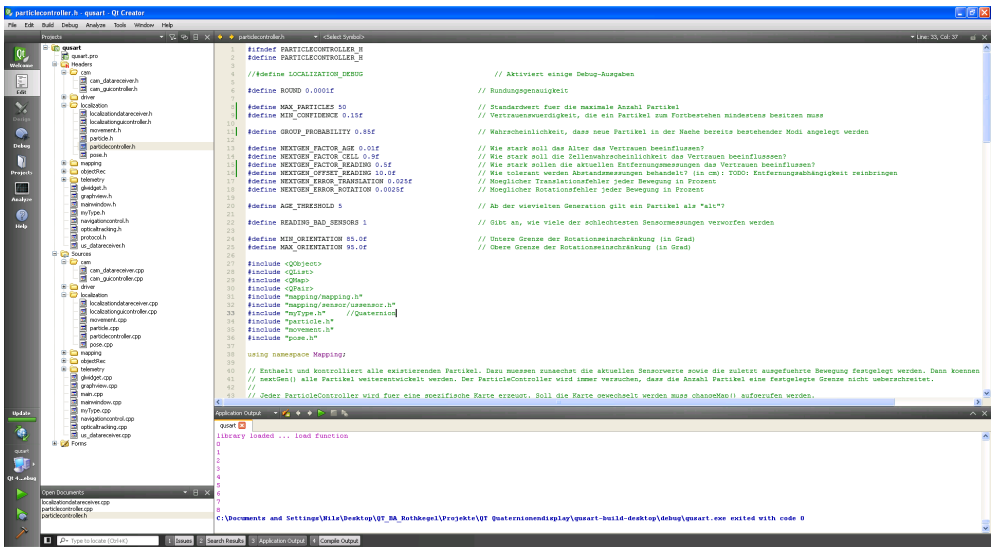


**Abbildung 4.1.:** Qt-Entwicklungsumgebung

## 4.2. Strukturierung der Software

Bei der Programmierung des Lokalisationsalgorithmus und der zugehörigen Benutzeroberfläche wurde versucht, eine logische Aufteilung und Kapselung der implementierten Funktionalität zu erreichen. Dazu wurden folgende Klassen angelegt:

**Pose** Repräsentiert eine Pose. Enthält bereits einige Methoden, um eine Pose durch Bewegungen zu beeinflussen.

**Movement** Entspricht einer Bewegung der in Kap. 3.2.3.1 vorgestellten Form.

**Particle** Eine Instanz dieser Klasse repräsentiert ein Partikel. Es verfügt über alle relevanten Eigenschaften wie Alter und Vertrauen und kann bewegt werden.

**ParticleController** Diese (als Singleton zu verwendende) Klasse beinhaltet den eigentlichen Lokalisationsalgorithmus und verwaltet alle existierenden Partikel (vgl. Kap. 3.2). Dem ParticleController müssen die zur Weiterentwicklung der Partikel benötigten Messwerte

übergeben werden. Außerdem können hier alle Parameter des Lokalisationsverfahrens eingestellt werden.

**LocalizationGUIController** Diese Klasse ist das Bindeglied zwischen dem ParticleController und der Benutzeroberfläche. Einerseits werden alle relevanten Daten des ParticleControllers an die Benutzeroberfläche weitergereicht (und dort entsprechend dargestellt), andererseits werden alle Benutzereingaben ausgewertet, validiert und an den ParticleController weitergegeben.

**LocalizationDataReceiver** Diese Klasse baut auf einer Klasse namens „DataReceiver“ auf und empfängt und verarbeitet die vom Quadrokopter empfangenen Daten. Diese werden dann an den ParticleController weitergereicht.

Der ParticleController wurde so konzipiert, dass er von der Benutzeroberfläche vollständig getrennt funktionsfähig ist. Er kann seine benötigten Daten außerdem aus einer beliebigen Quelle erhalten. Momentan sind dies die Nutzeroberfläche (simulierte Daten) und der `LocalizationDataReceiver` (tatsächliche Quadrokopterdaten).
Abb. 4.2 gibt nochmals einen Überblick über den Zusammenhang der implementierten Klassen.

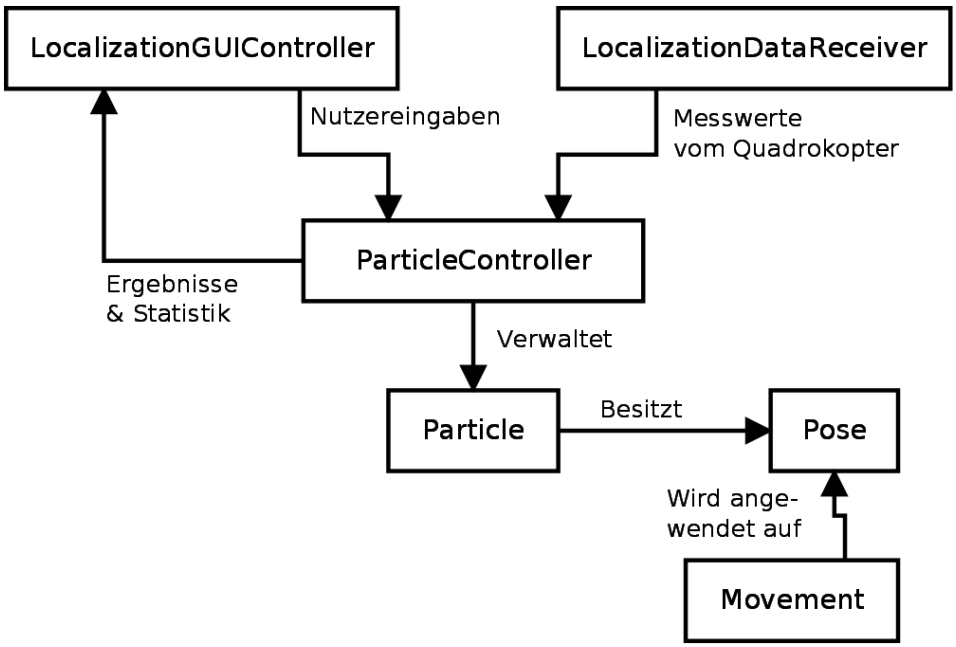



**Abbildung 4.2.:** Zusammenhang der implementierten Klassen

### 4.2.1. Sensormodell

Während der Entwicklung der Software wurde deutlich, dass die bestehende Möglichkeit zur Simulation von Abstandsmessungen zu langsam arbeitete. Auf einem Atom N270-Prozessor bedeutete dies, dass bereits bei ca. 20 Partikeln (was 160 Abstandsmessungen entspricht, da acht Richtungen) eine Laufzeit von mehr als einer halben Sekunde benötigt wurde. Deshalb wurde entschieden, die Abstandssensorik stattdessen durch mehrere Linien anzunähern. Dazu wurde ein Algorithmus entwickelt, der mit einer Linie die nächste belegte Zelle ermitteln kann (s. Kap. B). Ein Sensorkegel wurde dann jeweils mit fünf dieser Linien repräsentiert: Eine in der Mitte, jeweils eine am äußersten Rand des Kegels, die verbleibenden beiden genau zwischen den bisherigen drei Linien (s. Abb. 4.3). Die kürzeste gemessene Entfernung wurde dann als simulierter Messwert verwendet. Dadurch wurde der Algorithmus ungefähr um zwei Größenordnungen, also

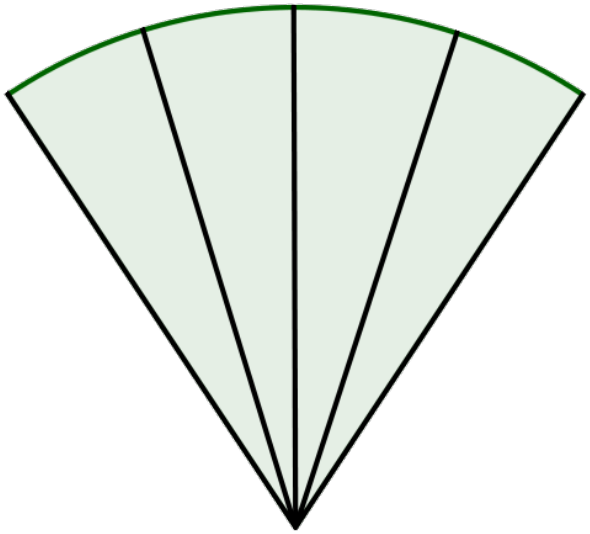


**Abbildung 4.3.:** Annäherung eines Sensorkegels durch Strahlen

deutlich, beschleunigt. Die verlorene Genauigkeit kann vernachlässigt werden, da nur die Entfernung zum nahesten Objekt von Interesse ist. Selbst wenn ein oder zwei Linien durch einen „Spalt“ in der Karte eine viel zu hohe Entfernung messen, sollte dies durch die verbleibenden Linien aufgefangen werden.

## 4.3. Einbindung in bestehendes System

Auf das bestehende System musste kaum Einfluss genommen werden. In der Kontrollsoftware wurde ein neuer Reiter „Localization“ angelegt, der alle relevanten Elemente zur Bedienung des Lokalisationsverfahrens enthält. Ferner musste dafür gesorgt werden, dass der ParticleController alle benötigten Datenpakete vom Quadrokopter erhält. Alle Erweiterungen der Kontrollsoftware konnten durchgeführt werden, ohne Änderungen am bestehenden Code vornehmen zu müssen. Die Quadrokopter-Firmware wurde weder verändert noch erweitert, da bereits alle benötigten Datenpakete (Abstands- und Bewegungsmessungen) geschickt werden können.

## 4.4. Beschreibung der Benutzeroberfläche

Die implementierte Benutzeroberfläche ist in Abb. 4.4 dargestellt. Die einzelnen Bedienelemente

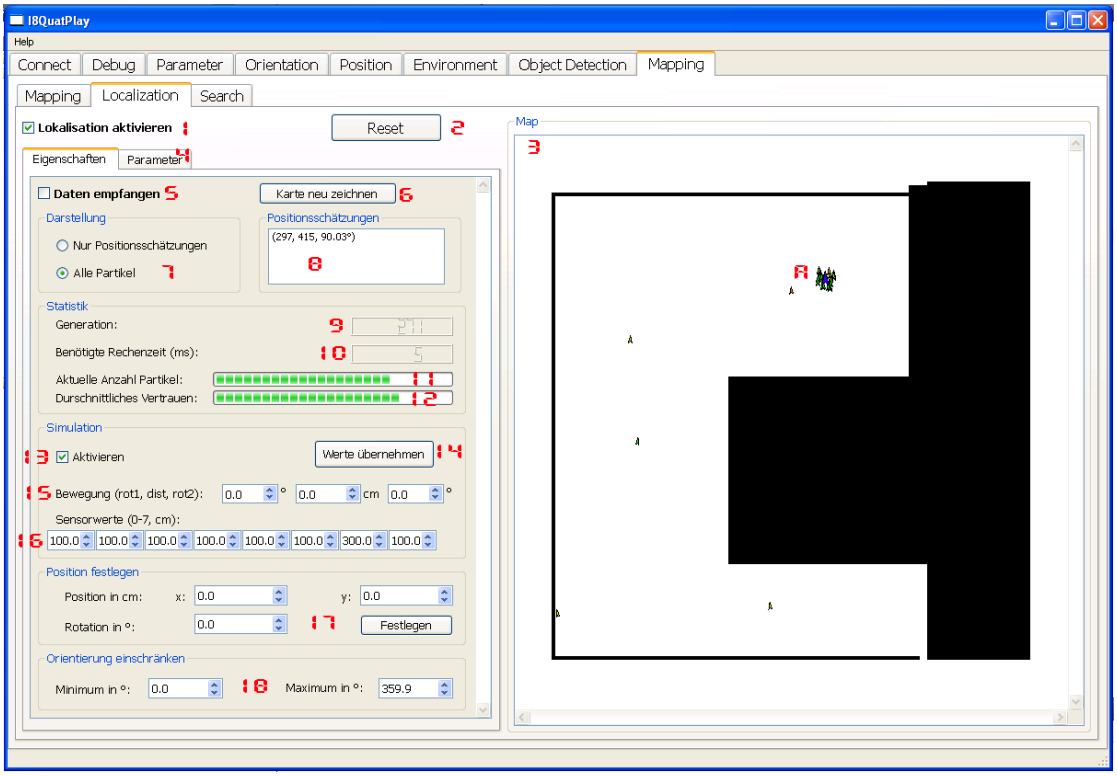



**Abbildung 4.4.:** Screenshot der implementierten Benutzeroberfläche

sind durch rote Nummern nachträglich gekennzeichnet worden. Die folgende Liste erläutert die nummerierten Elemente:

**1:** Aktiviert die Bedienelemente des Lokalisationsverfahrens. Mit dieser Checkbox können die Einstellungen vor unbeabsichtigten Änderungen geschützt werden.

**2:** Der Reset-Button setzt den ParticleController sowie die entsprechende Darstellung auf den Anfangszustand zurück. Dadurch werden alle Partikel gelöscht und die aktuelle Generation wieder auf 0 gesetzt.

**3:** In diesem Bereich werden die Karte der Umgebung sowie die Partikel und Positionsschätzungen abgebildet. Mit dem Mausrad kann der Bildausschnitt vergrößert oder verkleinert werden, mit einem Mausklick lässt sich der Bildausschnitt verschieben.

**4:** Der Reiter „Parameter“ enthält alle Parameter des Lokalisationsverfahrens. Diese können dort sowohl betrachtet als auch verändert werden.

**5:** Sobald diese Option aktiviert ist, empfängt der ParticleController automatisch aktuelle Messungen vom Quadrokopter und führt die Lokalisation selbstständig durch. Zuvor muss unter „Connect“ eine Verbindung hergestellt worden sein und unter „Debug“ müssen die Pakete „IR US Fused“ und „Visual Odometry“ ausgewählt worden sein.

**6:** Dieser Button veranlasst das Programm dazu, die Karte neu zu zeichnen, was z.B. nach dem Laden einer neuen Karte sinnvoll ist.

**7:** Über die Radioboxen lässt sich die Darstellung im rechten Abschnitt des Fensters beeinflussen. „Alle Partikel“ stellt nicht nur Positionsschätzungen, sondern auch jedes einzelne Partikel dar, was je nach Grafikkarte und Partikelzahl das Programm ausbremsen kann.

**8:** Diese Liste enthält - zusätzlich zur grafischen Darstellung im rechten Teil des Fensters - eine Auflistung aller momentanen Positionsschätzungen, also aller Poses, von denen das Lokalisationsverfahren meint, dass sie sehr wahrscheinlich der wahren Pose des Quadrokopters entsprechen. Die Werte stehen für x-Koordinate, y-Koordinate und Orientierung.

**9:** Gibt an, in der wievielten Generation sich der ParticleController befindet. Dies ist identisch zur Zahl der bisher durchgeführten Iterationen.

**10:** Gibt eine gemittelte Rechenzeit in Millisekunden an. Es wird nur das eigentliche Lokalisationsverfahren berücksichtigt, die Benutzeroberfläche wird in die Messung nicht einbezogen.

**11:** Dieser Balken stellt dar, wie viele Partikel (der maximal erlaubten) momentan benutzt werden (vgl. Kap. 3.2.3). Ein voller Balken bedeutet, dass so viele Partikel wie erlaubt genutzt werden, bei einem halben Balken werden nur halb so viele Partikel wie erlaubt genutzt.

**12:** Hier wird das durchschnittliche Vertrauen aller momentan existierenden Partikel dargestellt. Ein voller Balken entspricht einem Vertrauen von 1, ein leerer Balken einem durchschnittlichen Partikel-Vertrauen von 0. Je voller der Balken ist, desto mehr glaubt das Lokalisationsverfahren also, die wahre Pose des Quadrokopters angenähert zu haben.

**13:** Mit dieser Checkbox werden die Benutzeroberflächenelemente aktiviert (s. Punkt 14-16 dieser Aufzählung), mit denen dem ParticleController manuell Werte übergeben werden können. Gleichzeitig wird dafür gesorgt, dass ein evtl. bisher stattfindender Datenempfang deaktiviert wird.

**14:** Mit einem Klick auf diesen Button werden die eingegebenen Werte an den ParticleController geschickt.

**15:** Hier wird die zuletzt durchgeführte Bewegung festgelegt.

**16:** Hier werden die einzelnen Abstandsmessungen festgelegt. Sensor 0 zeigt in Blickrichtung, dann wird im Uhrzeigersinn durchnummeriert.

**17:** In diesem Abschnitt kann dem Lokalisationsverfahren die wahre Position des Quadrokopters mitgeteilt werden. Dies ist zu jedem Zeitpunkt möglich.

**18:** Evtl. ist es sinnvoll, die Orientierung des Quadrokopters künstlich einzuschränken (da er z.B. mit gleichbleibender Orientierung fliegt). Je stärker die Einschränkung, desto effektivier arbeitet das Lokalisationsverfahren. Allerdings kann dann kein Rotationsdrift des Quadrokopters berücksichtigt werden. Sollte der Quadrokopter also dazu neigen, im Laufe der Zeit seine Orientierung zu ändern, ist eine Einschränkung nicht sinnvoll.

**A:** Hier ist eine Ansammlung von Partikeln sichtbar. Jedes Partikel wird durch einen Pfeil dargestellt, die Richtung des Pfeils gibt dabei die Orientierung des Partikels (oder genauer seiner Pose) an. Grüne Partikel besitzen ein sehr gutes, gelbe ein mittelmäßiges und rote ein schlechtes Vertrauen. Sobald der ParticleController der Ansicht ist, eine erfolgreiche

Lokalisation durchgeführt zu haben, wird die ermittelte Positionsschätzung ebenfalls eingezeichnet. Sie wird als etwas größeres, blaues Partikel dargestellt.

Zu beachten ist, dass eine Orientierung von 0° einer Ausrichtung nach rechts entspricht.

# 5. Evaluierung

In diesem Kapitel wird das implementierte Lokalisationsverfahren evaluiert. Dazu werden zunächst manuell Werte per Benutzeroberfläche eingegeben und das Verhalten des Lokalisationsverfahrens simuliert und analysiert. Anschließend wird der Vorgang mit realen Sensorwerten wiederholt.
In den folgenden Kapiteln werden nicht alle Generationen abgebildet, da in manchen Generationen nichts bedeutendes passiert. Auf der dieser Arbeit beigelegten CD sind alle Messreihen vollständig enthalten.

## 5.1. Evaluierung mit simulierten Sensorwerten

Zunächst wurde die in Abb. 5.1 dargestellte Karte erzeugt. Diese wurde für alle folgenden Evaluierungen genutzt, um vergleichbare Ergebnisse zu erzielen. Zu beachten ist, dass die Karte für das Lokalisationsverfahren etwas problematisch ist, da die obere Hälfte des Raumes aus drei ähnlich großen Quadraten besteht. Zudem enthält der Raum durch den einfachen Aufbau keine komplexen Merkmale wie schräge Wände, Säulen o.ä. Die geflogene Trajektorie wird durch die rote Linie dargestellt. An jedem Wegpunkt (1-9) wurden die in Tabelle 5.1 aufgeführten Werte in die Benutzeroberfläche eingegeben (wobei Sensor 0 nach oben zeigt, danach wird im Uhrzeigersinn nummeriert). Dabei wurde jede Iteration mittels Screenshot dokumentiert. Der Quadrokopter bewegt sich von Wegpunkt zu Wegpunkt, wobei seine Orientierung konstant (in Abb. 5.1 nach oben) ist. Die folgenden Kapitel dokumentieren das Verhalten des Lokalisationsverfahrens mit verschiedenen Parametern.

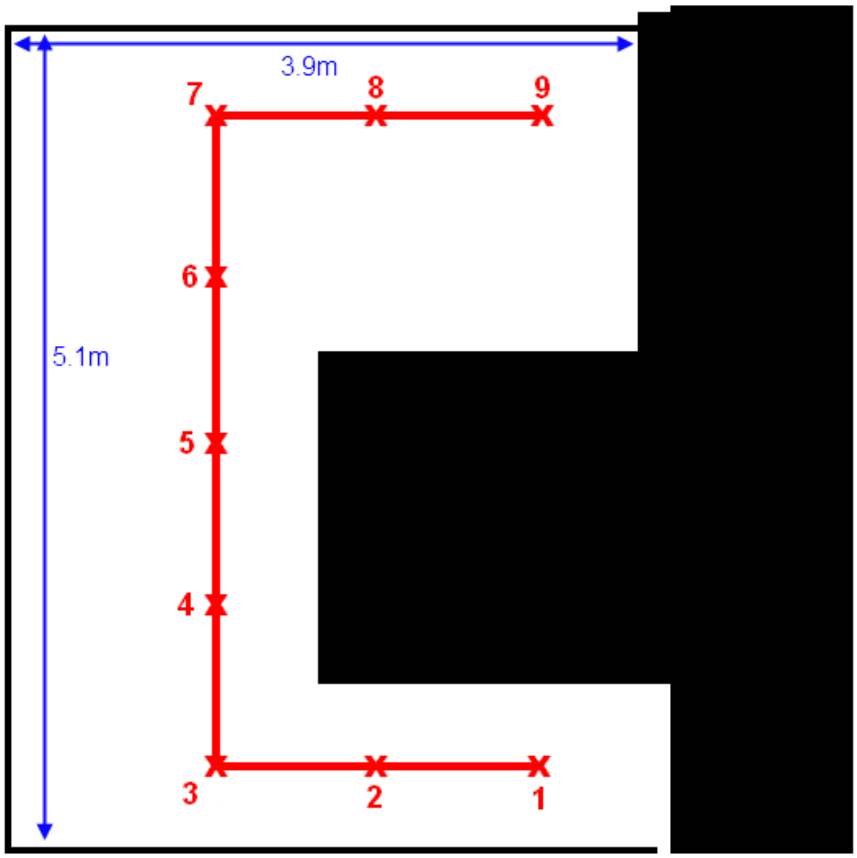



**Abbildung 5.1.:** Die für die Evaluierung genutzte Karte

| Wegpunkt | Sensor 0 | Sensor 1 | Sensor 2 | Sensor 3 | Sensor 4 | Sensor 5 | Sensor 6 | Sensor 7 |
|---|---|---|---|---|---|---|---|---|
| 2 | 50 | 60 | 150 | 50 | 50 | 50 | 150 | 60 |
| 3 | 210 | 100 | 150 | 50 | 50 | 50 | 125 | 130 |
| 4 | 350 | 70 | 70 | 70 | 150 | 130 | 125 | 130 |
| 5 | 260 | 70 | 70 | 70 | 245 | 130 | 125 | 130 |
| 6 | 160 | 170 | 120 | 85 | 220 | 130 | 125 | 130 |
| 7 | 60 | 65 | 195 | 165 | 220 | 130 | 125 | 65 |
| 8 | 60 | 65 | 165 | 145 | 135 | 230 | 195 | 65 |
| 9 | 60 | 65 | 65 | 70 | 135 | 145 | 195 | 65 |

**Tabelle 5.1.:** An das Lokalisationsverfahren übergebene Abstandsmesswerte (in cm)

### 5.1.1. Lokalisation mit bekannter Startpose

Zunächst wird das Verhalten bei bekannter Startpose überprüft. Dazu wurde dem Lokalisationsverfahren zu Beginn über die entsprechenden Bedienelemente der Benutzeroberfläche mitgeteilt, dass sich der Quadrokopter an Wegpunkt 1 befindet.

Die Parameter (s. Anh. A) wurden folgendermaßen festgelegt:

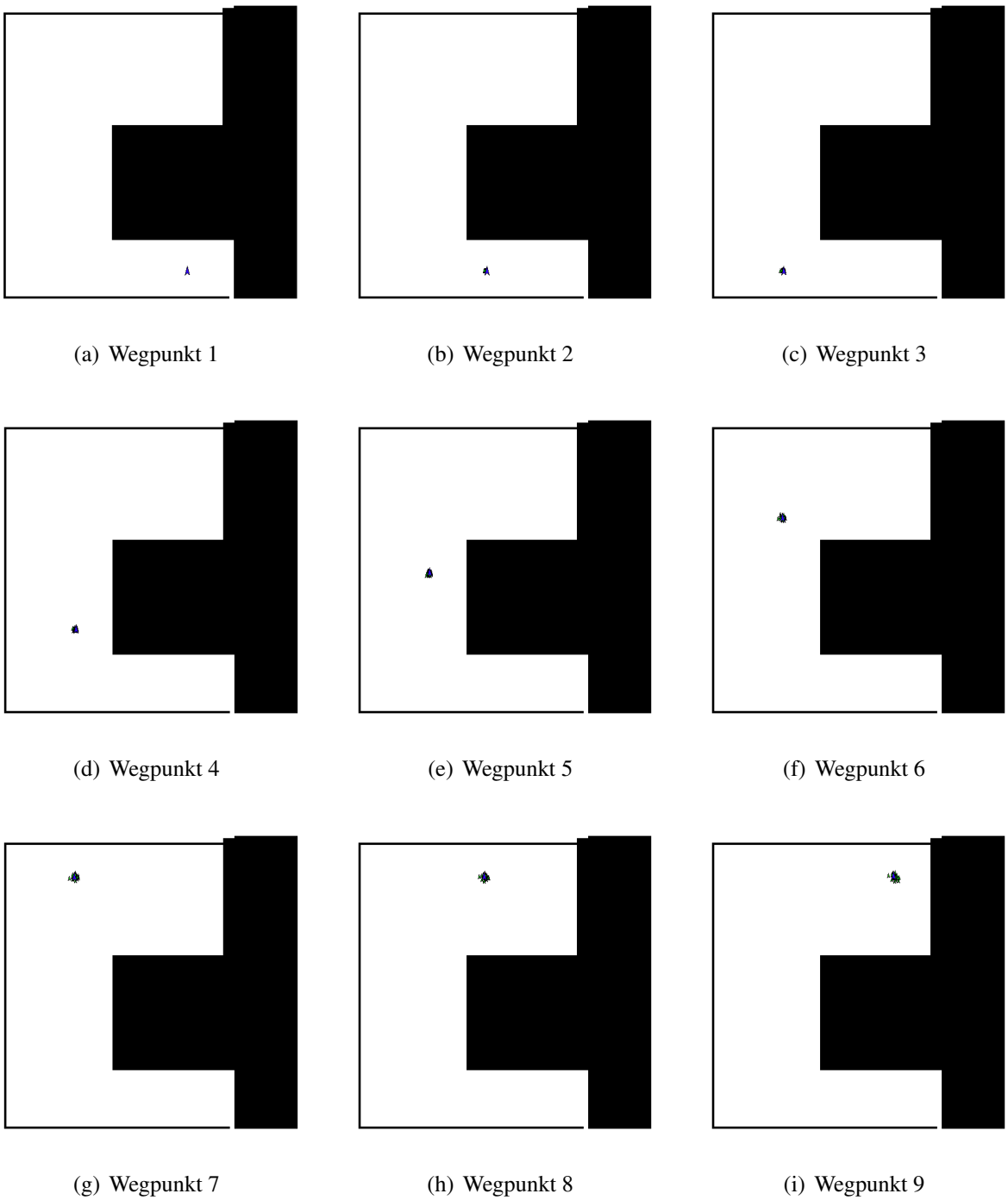
readingBadSensors = 1
maxParticles = 50
ageThreshold = 5
minConfidence = 0.25
groupProbability = 0.95
nextGenFactorAge = 0.01
nextGenFactorCell = 0.9
nextGenFactorReading = 1.0
nextGenOffsetReading = 7
nextGenErrorTranslation = 0.025
nextGenErrorRotation = 0.0025

#### 5.1.1.1. Geringer Bewegungsfehler

Es werden die Parameter aus Kap. 5.1.1 genutzt. Aus Abb. 5.2 wird ersichtlich, dass die Trajektorie gut verfolgt wird. In diesem Fall könnte es evtl. sinnvoll sein, die Toleranzen (insb. *nextGenOffsetReading*) zu verringern, sodass die Streuung der Partikel weiter vermindert wird.

#### 5.1.1.2. Mittlerer Bewegungsfehler

Es werden die in Kap. 5.1.1 beschriebenen Parameter genutzt, jedoch sind *nextGenErrorTranslation* und *nextGenErrorRotation* jeweils um Faktor 5 größer. Hier macht sich die Partikelstreuung bereits bemerkbar (s. Abb. 5.3). Kurzzeitig verliert das Lokalisationsverfahren seine Positionsschätzung, welche jedoch wieder erlangt werden kann.

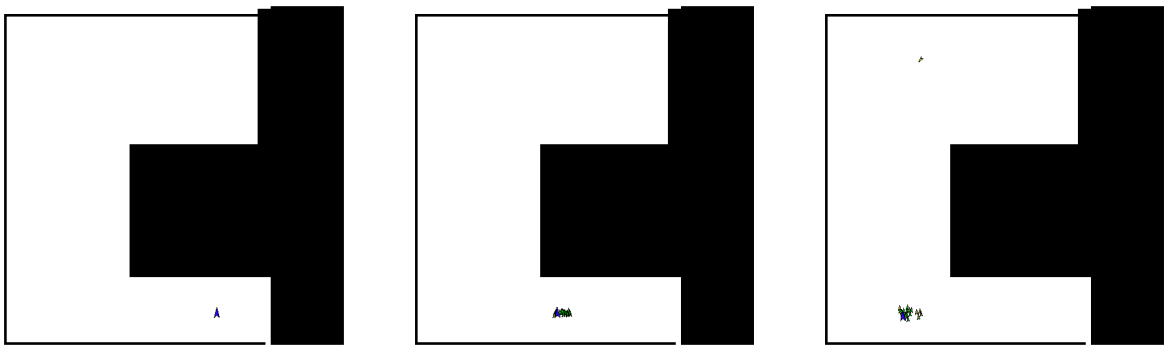


(a) Wegpunkt 1 (b) Wegpunkt 2 (c) Wegpunkt 3

(d) Wegpunkt 4 (e) Wegpunkt 5 (f) Wegpunkt 6

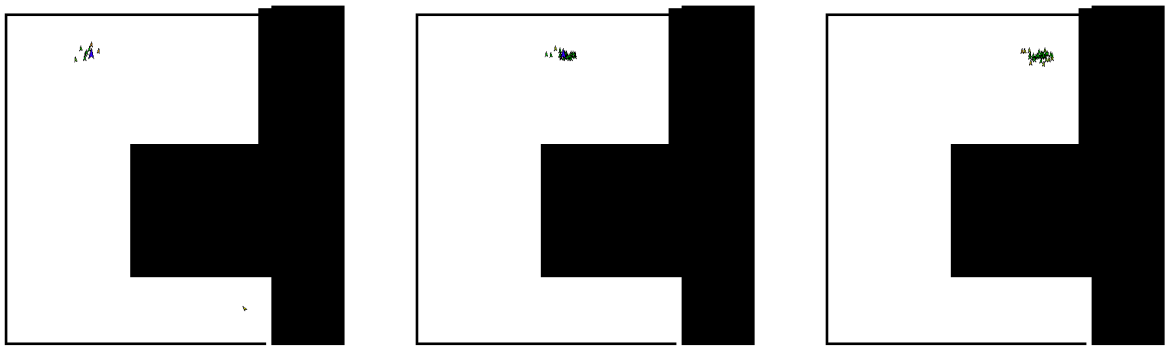
(g) Wegpunkt 7 (h) Wegpunkt 8 (i) Wegpunkt 9

**Abbildung 5.2.:** Partikelverhalten bei geringem Bewegungsfehler



(a) Wegpunkt 1 (b) Wegpunkt 2 (c) Wegpunkt 3

(d) Wegpunkt 4 (e) Wegpunkt 5 (f) Wegpunkt 6



(g) Wegpunkt 7 (h) Wegpunkt 8 (i) Wegpunkt 9

**Abbildung 5.3.:** Partikelverhalten bei mittlerem Bewegungsfehler

#### 5.1.1.3. Großer Bewegungsfehler

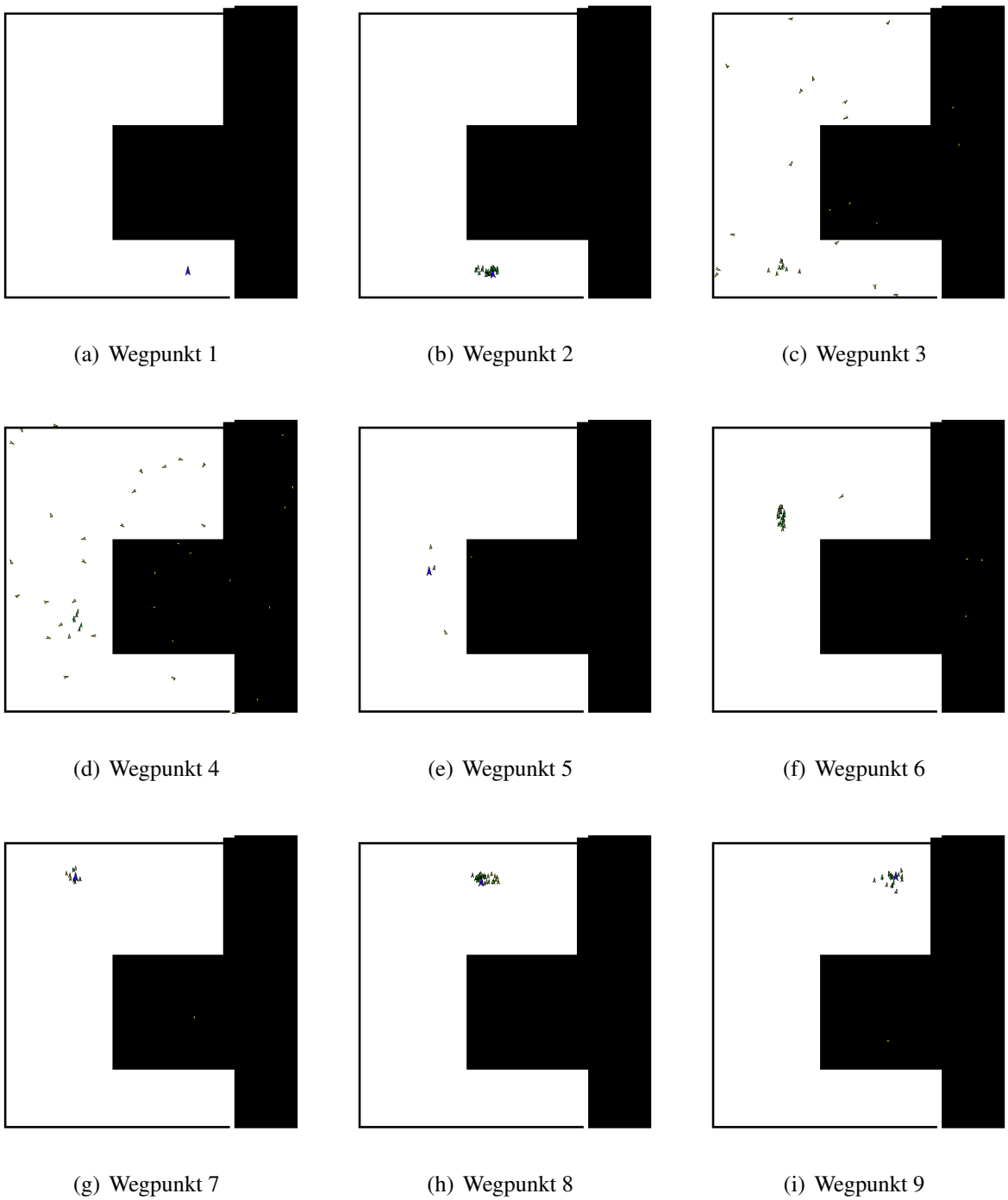
Es werden die Parameter aus Kap. 5.1.1 genutzt, jedoch sind *nextGenErrorTranslation* und *nextGenErrorRotation* jeweils um Faktor 10 größer. Hier macht sich die Partikelstreuung deutlich stärker bemerkbar (s. Abb. 5.4). Bereits bei Wegpunkt 3 befinden sich deutlich weniger Partikel an der korrekten Position als noch bei Wegpunkt 2. Bei einem derart großen Bewegungsfehler hat das Lokalisationsverfahren bereits leichte Schwierigkeiten. Dennoch kann die Trajektorie korrekt weiter verfolgt werden. In einem solchen Fall ist es sinnvoll, die Anzahl der Partikel zu erhöhen, wodurch trotz der Partikelstreuung eine gute Abdeckung des Bereichs der möglichen Pose erzielt wird.

#### 5.1.1.4. Fazit

Zusammenfassend kann gesagt werden, dass das Lokalisationsverfahren in der Lage ist, eine Trajektorie korrekt zu verfolgen. Auch bei beträchtlichen Bewegungsfehlern ist es noch möglich, kurzzeitige Unsicherheiten aufzufangen.

### 5.1.2. Lokalisation mit unbekannter Startpose

Im Anschluss an das Verfolgen einer Trajektorie mit bekannter Startpose wurde überprüft, ob der Algorithmus auch in der Lage ist, sich ohne bekannte Startpose korrekt zu lokalisieren. Dafür wurde wieder die gleiche Trajektorie wie in Kap. 5.1.1 genutzt und der Algorithmus, falls erforderlich, am letzten Wegpunkt so lange fortgeführt, bis eine Positionsschätzung bestimmt werden konnte.
Es werden die in Kap. 5.1.1 aufgeführten Parameter genutzt, allerdings wurde *maxParticles* auf 250 erhöht. Abb. 5.5 zeigt das Verhalten des Lokalisationsverfahrens. Die Lokalisation wird erfolgreich durchgeführt. Zwar konnte die richtige Pose erst nach Abfliegen der Trajektorie ermittelt werden, allerdings liegt das in der Natur des Lokalisationsverfahrens: Durch die zufällige Platzierung neuer Partikel ist das Verfahren darauf angewiesen, dass Partikel zufällig richtig platziert werden. Dies ist von der ersten Generation an theoretisch möglich, in diesem Fall wurden Partikel erst ca. ab der zehnten Generation richtig platziert.

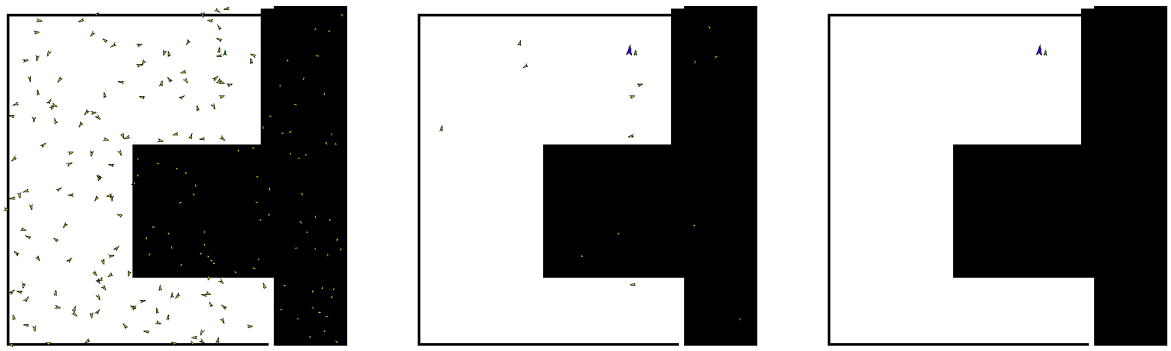


(a) Wegpunkt 1 (b) Wegpunkt 2 (c) Wegpunkt 3

(d) Wegpunkt 4 (e) Wegpunkt 5 (f) Wegpunkt 6

(g) Wegpunkt 7 (h) Wegpunkt 8 (i) Wegpunkt 9

**Abbildung 5.4.:** Partikelverhalten bei großem Bewegungsfehler

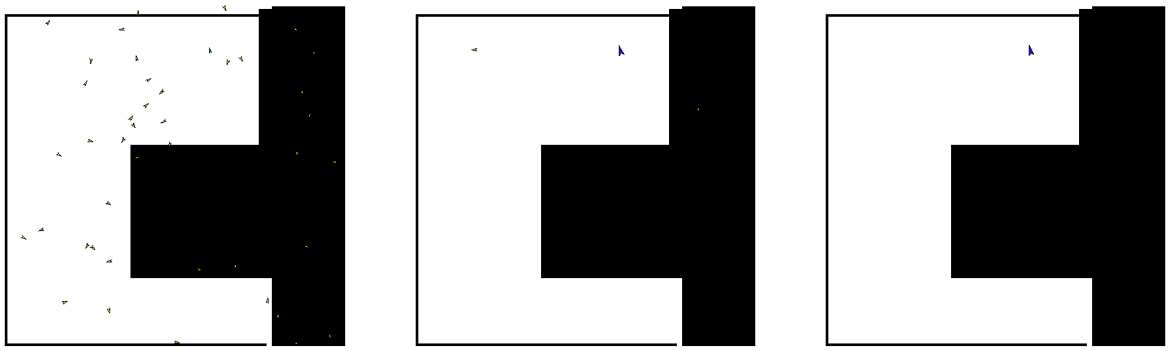


(a) Generation 10 (b) Generation 11 (c) Generation 12

**Abbildung 5.5.:** Lokalisation mit unbekannter Startpose

#### 5.1.2.1. Lokalisation mit 50 Partikeln

Es wird getestet, ob eine Lokalisation auch mit nur 50 Partikeln möglich ist. Bis auf *maxParticles* wurden alle Parameter wie in Kap. 5.1.2 beschrieben gesetzt. Wie in Abb. 5.6 zu sehen ist, konnte



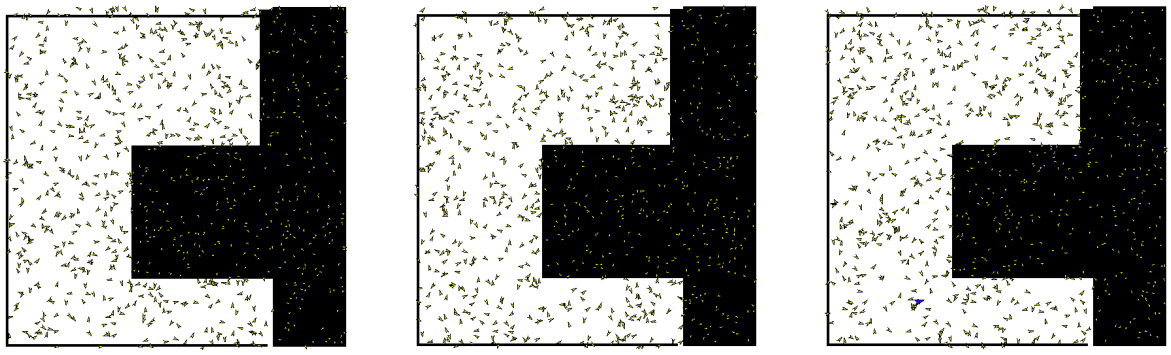
(a) Generation 131 (b) Generation 132 (c) Generation 133

**Abbildung 5.6.:** Lokalisation mit unbekannter Startpose, 50 Partikel

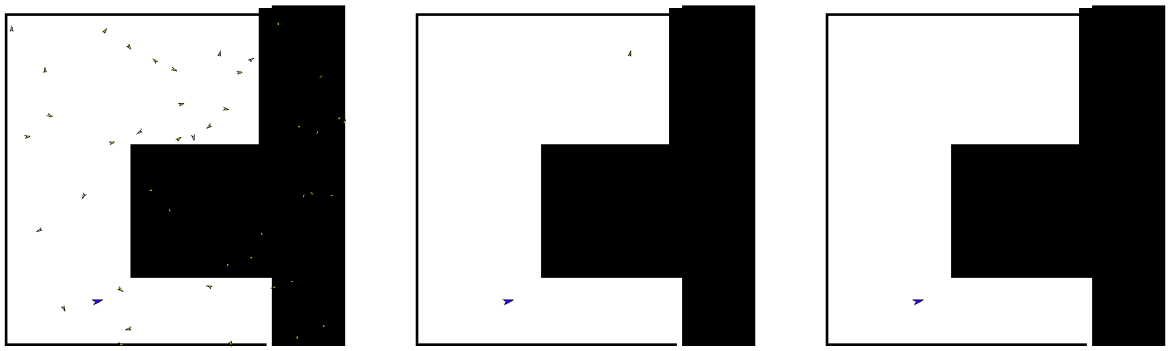
auch hier eine Lokalisation korrekt durchgeführt werden. Durch die geringe Partikelzahl dauerte dies allerdings bis etwa in die 130. Iteration. Allerdings muss in diesem Zusammenhang beachtet werden, dass eine geringe Partikelzahl auch eine kurze Rechendauer pro Iteration bedeutet.
Auf der dieser Arbeit beigefügten CD sind alle Screenshots der Evaluierungen auch unbearbeitet enthalten, wodurch die Benutzeroberfläche sichtbar ist. Dadurch lässt sich die durchschnittliche Dauer von Iterationen abschätzen. In diesem Fall benötigte eine Iteration ca. 10ms, was bedeutet, dass die hier benötigten 130 Iterationen in etwas über einer Sekunde bereits berechnet waren.

#### 5.1.2.2. Lokalisation mit 1000 Partikeln

Als Gegenstück zu Kap. 5.1.2.1 wird evaluiert, ob eine hohe Partikelzahl eine genauere oder schnellere Lokalisation bewirkt. Abgesehen von *maxParticles* sind die Parameter identisch zu Kap. 5.1.2. Abb. 5.7 zeigt, dass eine hohe Partikelzahl auch das Risiko von Falschlokalisationen



(a) Generation 6 (b) Generation 7 (c) Generation 8



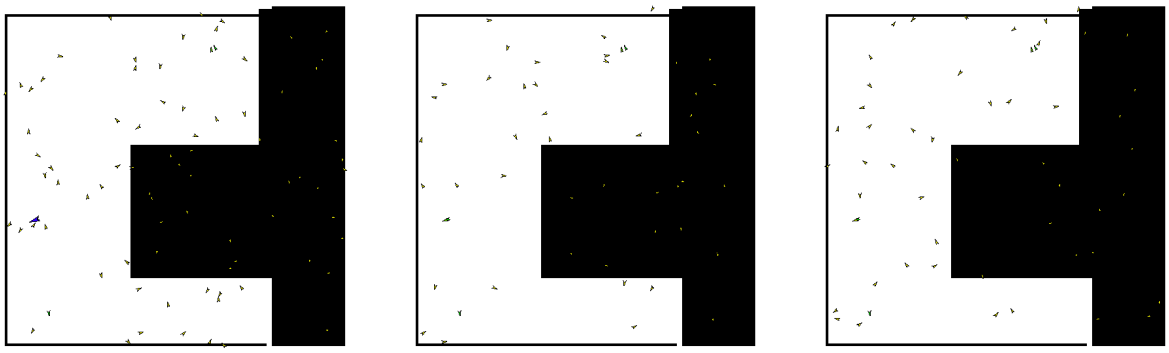
(d) Generation 9 (e) Generation 10 (f) Generation 11

**Abbildung 5.7.:** Lokalisation mit unbekannter Startpose, 1000 Partikel

erhöht. Allerdings muss beachtet werden, dass auch die gefundene Pose gut zu den Abstandsmessungen von Wegpunkt 9 passt, insofern ist die Falschlokalisation hier großteils durch die Karte bedingt. Dennoch wäre es besser, wenn das Lokalisationsverfahren in diesem Fall neben der gefundenen Pose auch die richtige bei Wegpunkt 9 findet und beide Poses gleichberechtigt weiterverfolgt werden. Ein komplexeres Lokalisationsverfahren könnte möglicherweise auch die bisherigen Bewegungen berücksichtigen, sodass eine der beiden Poses ausgeschlossen werden kann.

#### 5.1.2.3. groupProbability = 0.6

Mit den Parametern aus Kap. 5.1.2 und einer geänderten *groupProbability* soll ermittelt werden, welchen Einfluss dieser Parameter auf die Konvergenz des Lokalisationsverfahrens hat. Es wird



(a) Generation 18 (b) Generation 19 (c) Generation 20

**Abbildung 5.8.:** Lokalisation mit unbekannter Startpose, *groupProbability* = 0.6

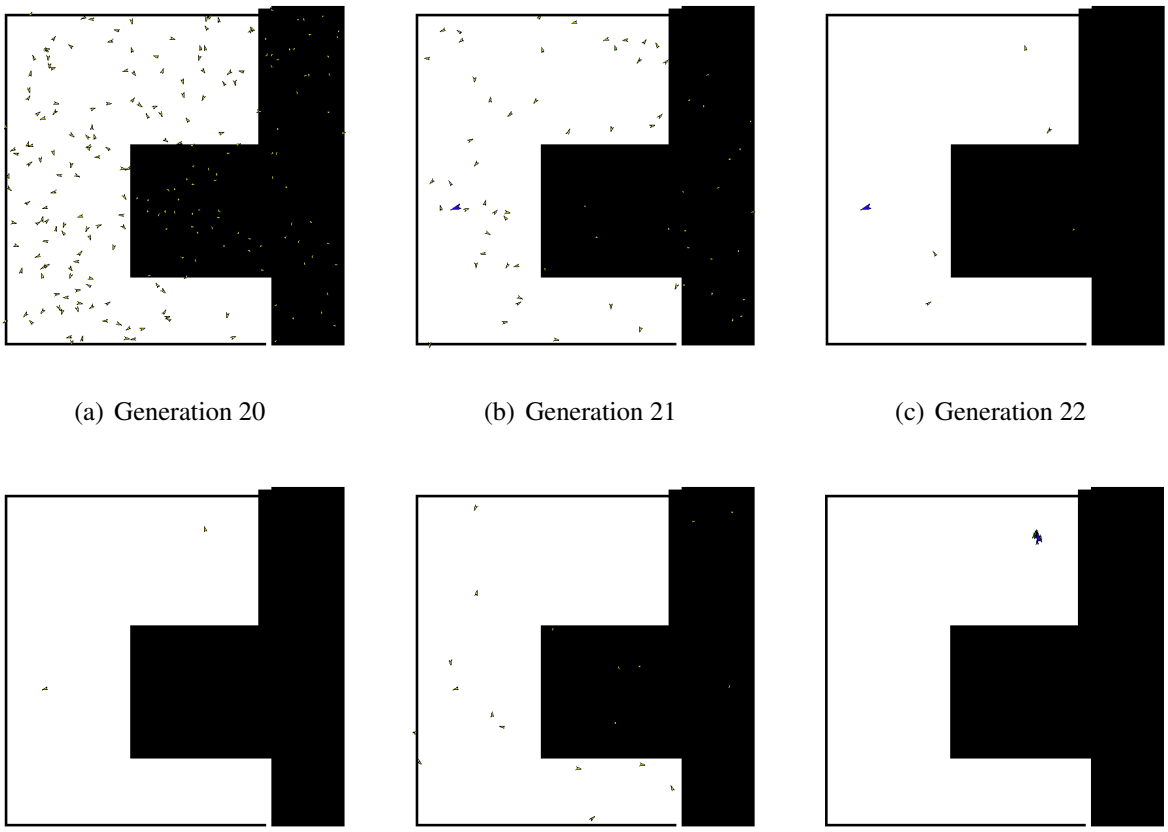
deutlich, dass das Lokalisationsverfahren nun kaum noch konvergiert und auch gefundene Positionsschätzungen schnell wieder verwirft (s. Abb. 5.8). Mit diesem Parameter könnte also dem in Kap. 5.1.2.7 beschriebenen Problem, dass sich das Verfahren zu schnell auf eine Pose konzentriert, begegnet werden. Allerdings ist der Wert hier offensichtlich zu klein gewählt, weshalb in Kap. 5.1.2.4 und Kap. 5.1.2.5 nochmal die Auswirkung einer Steigerung von *groupProbability* evaluiert wird.

#### 5.1.2.4. groupProbability = 0.7

Wie in Kap. 5.1.2.3 erwähnt folgt eine Evaluierung für eine höhere *groupProbability*. Es fällt auf, dass sich das Lokalisationsverfahren zunächst falsch lokalisiert (s. Abb. 5.9). Die Positionsschätzung wird jedoch wieder verworfen. Schlussendlich erfolgt dann in der 143. Generation die richtige Lokalisierung, wobei die späte Lokalisierung nicht auf *groupProbability*, sondern vielmehr auf die Anzahl der Partikel zurückzuführen ist (vgl. Kap. 5.1.2.2).

#### 5.1.2.5. groupProbability = 0.8

Der Parameter *groupProbability* wird im Vergleich zu Kap. 5.1.2.4 nochmal gesteigert. Das Lokalisationsverfahren lokalisiert sich zunächst falsch und verwirft diese Positionsschätzung auch



(a) Generation 20 (b) Generation 21 (c) Generation 22

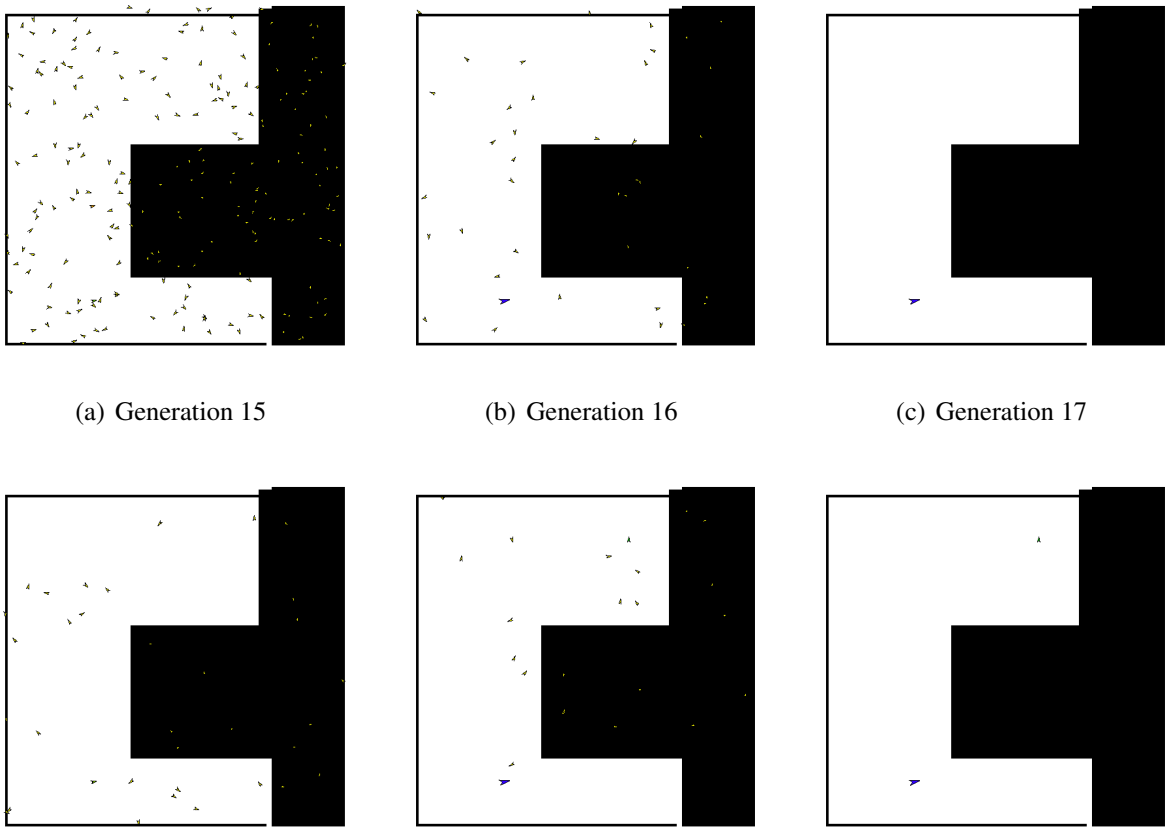
(d) Generation 23 (e) Generation 24 (f) Generation 143

**Abbildung 5.9.:** Lokalisation mit unbekannter Startpose, *groupProbability* = 0.7

kurzzeitig wieder (s. Abb. 5.10). Wenige Generationen später wird diese falsche Positionsschätzung jedoch wieder aufgenommen. Zu beachten ist, dass sich dennoch ein Partikel an der eigentlich richtigen Position befindet. Hier besteht Optimierungsbedarf (vgl. Kap. 5.1.2.2).

#### 5.1.2.6. Einschränkung der Orientierung

Abschließend wird die mögliche Partikelorientierung auf einen Bereich von 85°-95° eingeschränkt, da der Quadrokopter die Trajektorie mit gleichbleibender Orientierung abfliegt. Abb. 5.11 zeigt, dass eine Einschränkung der Orientierung das Lokalisationsverfahren deutlich beschleunigen kann. Die korrekte Pose wurde bereits vor Erreichen des letzten Wegpunktes gefunden. Außerdem werden viele fehlerhafte Positionsschätzungen dadurch ausgeschlossen.



(a) Generation 15 (b) Generation 16 (c) Generation 17

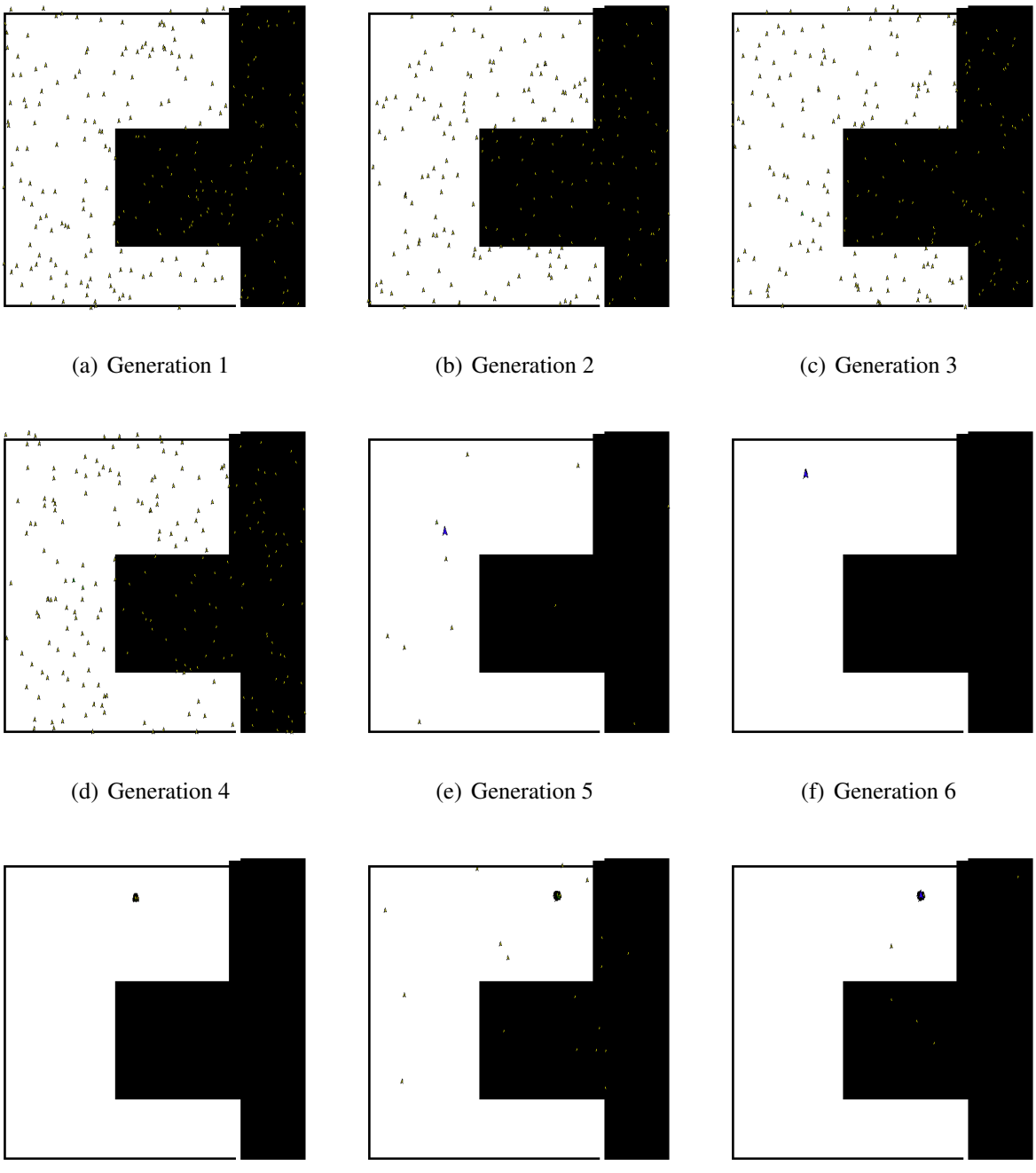
(d) Generation 34 (e) Generation 42 (f) Generation 53

**Abbildung 5.10.:** Lokalisation mit unbekannter Startpose, *groupProbability* = 0.8

#### 5.1.2.7. Fazit

Es konnte gezeigt werden, dass der Lokalisationsalgorithmus auch dann eine Lokalisation durchführen kann, wenn er keinerlei Informationen über die Startpose des Quadrokopters besitzt. Damit handelt es sich bei dem implementierten Verfahren, wie beabsichtigt (vgl. Kap. 3.1.1), um ein globales Lokalisationsverfahren (s. Kap. 2.3).

Aufgrund der durchgeführten Evaluierung scheinen insb. zwei Aspekte des Algorithmus überarbeitungswürdig zu sein: Zum einen konvergiert das Verfahren noch zu schnell (vgl. Kap. 5.1.2.2), ohne andere mögliche Poses zu berücksichtigen. Grundsätzlich ist schnelles Konvergieren wünschenswert, allerdings sollte sich das Verfahren bei mehreren Möglichkeiten nicht derart schnell festlegen, da sonst dauerhafte Falschlokalisationen auftreten. Zum andern muss ein Verfahren entwickelt werden, um gefundene Positionsschätzungen mit der Zeit zu verbessern. Wie z.B. in Kap. 5.1.2 gezeigt könnten auch gute Positionsschätzungen noch verbessert werden. Möglicher-



(a) Generation 1 (b) Generation 2 (c) Generation 3

(d) Generation 4 (e) Generation 5 (f) Generation 6

(g) Generation 7 (h) Generation 8 (i) Generation 9

**Abbildung 5.11.:** Lokalisation mit unbekannter Startpose, eingeschränkte Orientierung

weise wäre eine Lösung, neue Partikel nicht exakt auf guten Partikeln, sondern mit einer gewissen Streuung zu positionieren, sodass sie auch den Bereich um ein gutes Partikel betrachten. Außerdem könnte eine dynamische Parametrisierung (also eine automatisierte Parametrisierung zur Laufzeit) eine deutliche Verbesserung bringen, indem das Verfahren bei gefundenen Positionsschätzungen immer geringere Toleranzen zulässt, wodurch nach und nach auch „nur“ gute Partikel gelöscht und durch noch bessere ersetzt werden. Hier muss allerdings darauf geachtet werden, dass dies nicht wiederum zu verstärkten Falschlokalisationen führt.
Eine weitere Verbesserung ist bei der Platzierung zufälliger Partikel nötig, da noch zu viele Partikel auf belegten Zellen erzeugt werden.

## 5.2. Evaluierung mit realen Sensorwerten

Zu Beginn dieses Kapitels soll direkt erwähnt werden, dass aufgrund der verwendeten Sensorik eine reale, sinnvolle Evaluierung nicht durchgeführt werden konnte. Stattdessen konnte allerdings qualitativ gezeigt werden, dass das Lokalisationsverfahren zumindest ansatzweise auch mit realen Sensoren funktioniert.
Es wurde der in Abb. 5.12 dargestellte Versuchsaufbau genutzt. Mit Matten wurde der gleiche Raum wie in Kap. 5.1 aufgebaut. Der Quadrokopter wurde auf einen rollbaren Wagen gesetzt, wobei der OF-Sensor freie Sicht auf den Boden hatte (s. Abb. 5.13). Es war dann beabsichtigt, wie in Kap. 5.1 den Quadrokopter entlang einer vorgegebenen Trajektorie zu bewegen und an jedem Wegpunkt die realen Sensorwerte zu empfangen. Leider musste festgestellt werden, dass der Boden durch die aufgestellten Wände - trotz eingeschalteter Hallenbeleuchtung und zusätzlicher Scheinwerfer - für den OF-Sensor teilweise zu dunkel war, was einen erheblichen Drift verursachte. Dadurch erschien eine umfassende Evaluierung des Lokalisationsverfahrens durch das Abfahren einer Trajektorie mit realen Sensormessungen nicht sinnvoll, da der Drift des OF-Sensors an dunklen Stellen das Ergebnis deutlich verfälschte. Hier konnten auch die Abstandsmessungen keine Abhilfe schaffen, da sie bereits eine Ungenauigkeit aufwiesen, die ebenfalls durch das Lokalisationsverfahren ausgeglichen werden musste. Deshalb wurde der Quadrokopter schlussendlich an ein Stelle gestellt, die eine ausreichende Helligkeit aufwies, und eine globale Lokalisation durchgeführt.

**Abbildung 5.12.:** Der mit Matten aufgebaute Raum

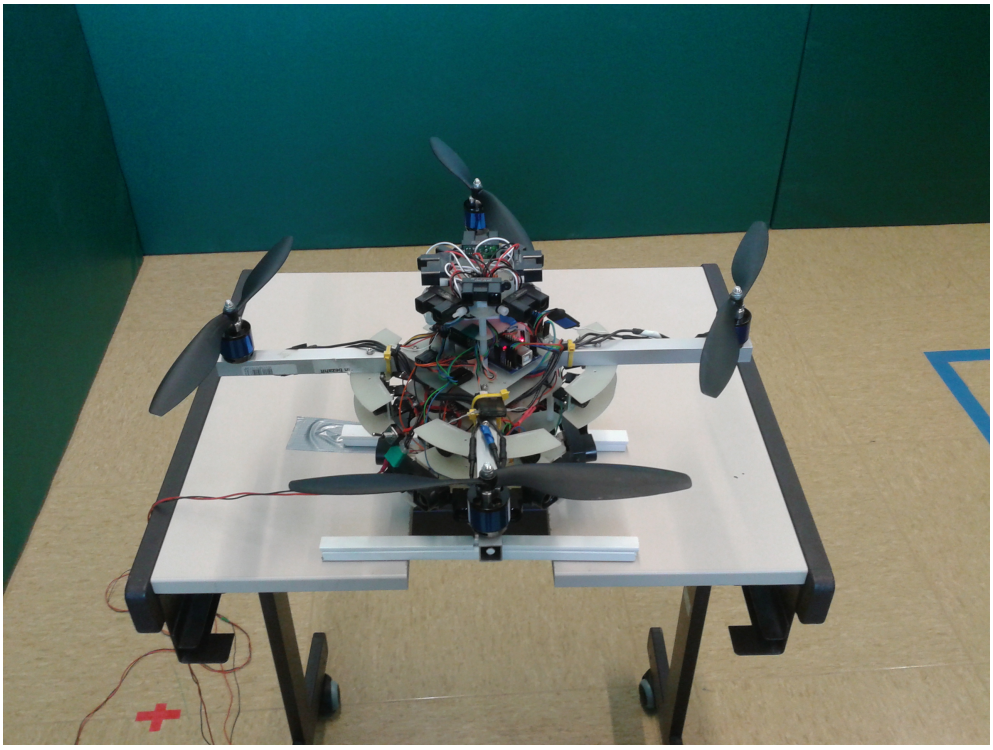


**Abbildung 5.13.:** Quadrokopter auf Rollwagen

Die Parameter (s. Anh. A) wurden folgendermaßen gewählt:

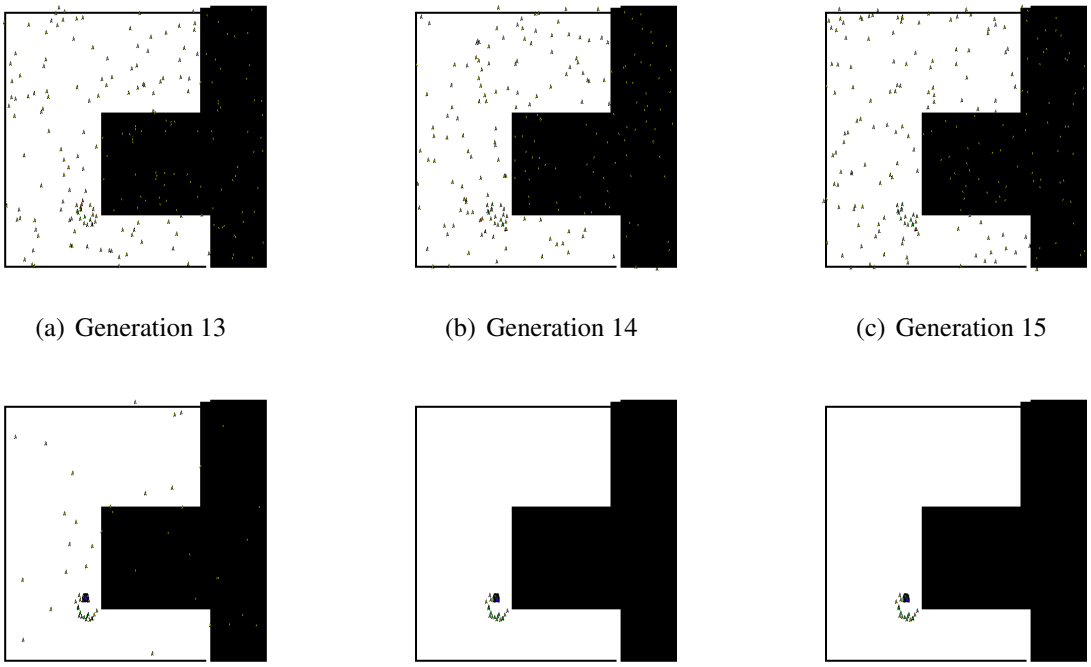
readingBadSensors = 1
maxParticles = 50
ageThreshold = 5
minConfidence = 0.15
groupProbability = 0.85
nextGenFactorAge = 0.01
nextGenFactorCell = 0.9
nextGenFactorReading = 0.5
nextGenOffsetReading = 10
nextGenErrorTranslation = 0.025
nextGenErrorRotation = 0.0025

### 5.2.1. Erster Versuch

Für den ersten Versuch wurde die mögliche Rotation auf 85°-95° eingeschränkt. Der Quadrokopter wurde während des Versuchs nicht bewegt, außerdem hat er keinerlei Kenntnis über seine wahre Pose. Die wahre Pose des Quadrokopters entspricht dem blauen Pfeil in Abb. 5.14(f). Wie in Abb. 5.14 zu sehen, führt die Einschränkung der möglichen Rotation schnell zu einer Ansammlung vertrauenswürdiger (grüner) Partikel. Der gefundenen Pose wird dann nach und nach mehr Vertrauen geschenkt. Allerdings ist auch ersichtlich, dass einige Partikel unterhalb der wahren Pose verbleiben. Dies spricht dafür, dass *nextGenOffsetReading* einige Centimeter niedriger hätte gewählt werden sollen.

### 5.2.2. Zweiter Versuch

Als Vergleich zu Kap. 5.2.1 wurde der Versuch ohne Rotationseinschränkung wiederholt. Wie in Abb. 5.15 zu sehen lokalisiert sich das Lokalisationsverfahren etwas unterhalb der wahren Pose. Dies ist wiederum mit einem zu hohen *nextGenOffsetReading* begründet, zusätzlich macht sich hier besonders *readingBadSensors* bemerkbar: Dadurch, dass die gefundene Pose etwas unterhalb einer Kante liegt, sollte für den nach rechts gerichteten Sensor ein großer Wert erwartet werden. Da der gemessene Wert jedoch deutlich niedriger ist, wird diese Messung aufgrund der hohen

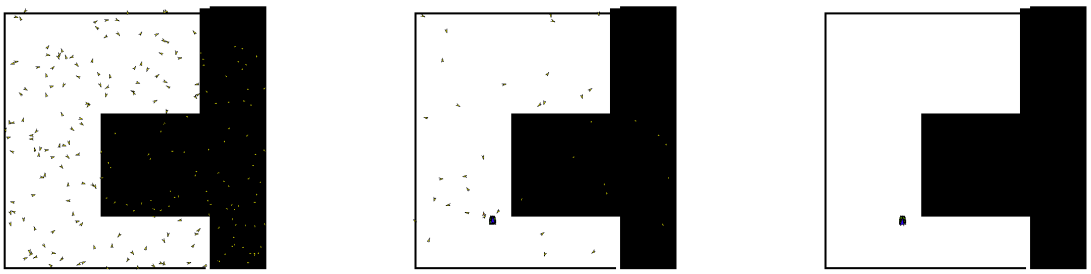


(a) Generation 13 (b) Generation 14 (c) Generation 15

(d) Generation 40 (e) Generation 41 (f) Generation 42

**Abbildung 5.14.:** Globale Lokalisation mit unbekannter Startpose, eingeschränkte Orientierung

Diskrepanz verworfen (vgl. Kap. 3.2.2.3). Alle anderen Sensorwerte sind von (durch *nextGenOffsetReading*) tolerierbaren Abweichungen betroffen, sodass das Lokalisationsverfahren keinen Anlass sieht, die Pose weiter zu verbessern. Statt jedoch *readingBadSensors* auf 0 zu verringern wäre hier, wie auch in Kap. 5.2.1, eine Verringerung von *nextGenOffsetReading* sinnvoller, da *readingBadSensors* insb. für starke kurzfristige Störungen (z.B. durch Messwertausreißer oder vorbeigehende Personen, s. Anh. A) zuständig ist.



(a) Generation 15 (b) Generation 25 (c) Generation 26

**Abbildung 5.15.:** Globale Lokalisation mit unbekannter Startpose

# 6. Fazit

In diesem Kapitel werden die Ergebnisse der Arbeit diskutiert und ein Ausblick auf die mögliche weitere Entwicklung gegeben.

## 6.1. Ergebnisse

Wie in Kap. 5 beschrieben gestaltete sich eine reale Evaluierung etwas schwierig, da die benötigte Sensorik nur unzuverlässig funktionierte. Dennoch lässt sich zusammenfassend sagen, dass das implementierte Verfahren grundsätzlich funktioniert und sich auch für ungenauere Sensorik eignet. Es wurde somit eine Basis für eine Indoorlokalisation geschaffen, auf deren Grundlage ein zuverlässiges und genaues Lokalisationsverfahren möglich ist.
Die Evaluierung des Verfahrens mit simulierten Sensorwerten (s. Kap. 5.1) hat gezeigt, dass das Lokalisationsverfahren in der Lage ist, sich in verschiedenen Situationen korrekt zu lokalisieren. Auch die durchgeführte Evaluierung mit realen Sensorwerten konnte dies bestätigen (s. Kap. 5.2). Allerdings zeigten die Versuche auch, dass hinsichtlich der Parametrisierung noch Verbesserungspotential besteht.
Deutlich wichtiger scheint jedoch eine Verbesserung der bisher verwendeten Strukturen und Algorithmen. Das verwendete Bewegungsmodell funktioniert zwar, allerdings wäre es hier sinnvoll, einen künstlichen Drift der x- und y-Koordinaten zu implementieren, um so dem realen Verhalten des Quadrokopters näher zu kommen. Dies würde außerdem bewirken, dass Partikel auch dann streuen, wenn sie nicht bewegt oder neu erzeugt werden, statt wie bisher statisch an der gleichen Position zu verbleiben. Desweiteren sollte dafür gesorgt werden, dass sich verschiedene Parameter dynamisch während der Laufzeit an den momentanen Zustand des Lokalisationsverfahrens anpassen, sodass Positionsschätzungen schnelller gefunden und gefundene Schätzungen kontinuierlich verbessert werden. Außerdem ist es nötig, das Verfolgen mehrerer Positionsschätzungen

zu verbessern, was sich aber auch bereits durch eine dynamische Anpassung der Parameter ergeben kann.
Neben diesen unmittelbaren Verbesserungen ist außerdem denkbar, das Lokalisationsverfahren auf einer höheren Ebene intelligenter zu gestalten. Ein Beispiel hierfür wäre, für jedes Partikel auch eine Historie der bisherigen Poses zu verwalten, sodass Partikel schneller als unpassend identifiziert werden können (da die Historie bspw. zeigt, dass ein Partikel durch eine Wand geflogen sein müsste). Bei solchen Verbesserungen muss allerdings abgeschätzt werden, ob die erzielte Verbesserung des Lokalisationsverfahrens die zusätzlich benötigte Laufzeit rechtfertigt.

## 6.2. Ausblick

Ein funktionierendes Lokalisationsverfahren stellt den ersten Schritt zu einer umfassenden Autonomie eines Quadrokopters dar. Als nächster Schritt wäre z.B eine Vereinigung von Mapping und Lokalisationsverfahren sinnvoll, wodurch ein SLAM-Verfahren entstehen würde. Hier böte sich bspw. DP-SLAM (s. Kap. 2.4.3) an, welches auf einem Partikelfilter aufbaut und sich gut für Occupancy Grid Maps (s. Kap. 2.2.2) umsetzen lässt.
Ebenso wäre denkbar, das Lokalisationsverfahren aktiver zu gestalten, indem ein Teil der Steuerung dem Quadrokopter überlassen wird. Dadurch könnte dieser selbstständig Umgebungen anfliegen, die sich durch viele Details und komplexe Formen deutlich von anderen abheben, um so eine zuverlässige Lokalisation zu ermöglichen. Desweiteren könnte der Quadrokopter so fliegen, dass er offene Flächen meidet, um seine Position nicht zu verlieren.

# 7. Literaturverzeichnis

[Benz 2013] BENZ, Paul: *Implementierung und Evaluierung eines Systems zur Hinderniserkennung und Kollisionsvermeidung für Indoor-Quadrokopter*. Bachelorarbeit. 2013. – Julius-Maximilians-Universität Würzburg

[Cox und Leonard 1994] COX, Ingemar J. ; LEONARD, John J.: Modeling a dynamic environment using a Bayesian multiple hypothesis approach. In: *Artif. Intell.* 66 (1994), April, S. 311–344

[Eliazar und Parr 2003] ELIAZAR, Austin ; PARR, Ronald: *DP-SLAM: Fast, robust simultaneous localization and mapping without predetermined landmarks*. 2003

[Eliazar und Parr 2004] ELIAZAR, Austin ; PARR, Ronald: *Dp-Slam 2.0*. 2004

[Fox 1998] FOX, Dieter: *Markov localization - a probabilistic framework for mobile robot localization and navigation*, Universität Bonn, Dissertation, 1998

[Fox et al. 1999] FOX, Dieter ; BURGARD, Wolfram ; DELLAERT, Frank ; THRUN, Sebastian: Monte Carlo Localization: Efficient Position Estimation for Mobile Robots. In: *IN PROC. OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE (AAAI)*, 1999, S. 343–349

[Gageik 2013] GAGEIK, Nils: *Autonomous Quadrotor for Indoor Exploration*. Juni 2013. – URL `http://www8.informatik.uni-wuerzburg.de/en/wissenschaftforschung/aqopteri8/`

[Hähnel 2005] HÄHNEL, Dirk: *Mapping with mobile robots*, University of Freiburg, Dissertation, 2005. – URL `http://www.freidok.uni-freiburg.de/volltexte/1632/`

[Lu und Milios 1997] LU, F. ; MILIOS, E.: Globally Consistent Range Scan Alignment for Environment Mapping. In: *AUTONOMOUS ROBOTS* 4 (1997), S. 333–349

[Montemerlo et al. 2002] MONTEMERLO, Michael ; THRUN, Sebastian ; KOLLER, Daphne ; WEGBREIT, Ben: FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. In: *In Proceedings of the AAAI National Conference on Artificial Intelligence*, AAAI, 2002, S. 593–598

[Montemerlo et al. 2003] MONTEMERLO, Michael ; THRUN, Sebastian ; ROLLER, Daphne ; WEGBREIT, Ben: *FastSLAM 2.0: An improved particle filtering algorithm for simultaneous localization and mapping that provably converges*. 2003

[Schmitt 2012] SCHMITT, Norbert: *Intelligentes Mapping für Indoor-Quadrocopter*. Bachelorarbeit. 2012. – Julius-Maximilians-Universität Würzburg

[Strohmeier 2012] STROHMEIER, Michael: *Implementierung und Evaluierung einer Positionsregelung unter Verwendung des optisches Flusses*. Bachelorarbeit. 2012. – Julius-Maximilians-Universität Würzburg

[Thrun et al. 2005] THRUN, Sebastian ; BURGARD, Wolfram ; FOX, Dieter: *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. – ISBN 0262201623

[Welch und Bishop 2001] WELCH, Greg ; BISHOP, Gary: An Introduction to the Kalman Filter. (2001)

# A. Parameter

Eine kurze Erläuterung aller Parameter, mit denen das Verhalten des Lokalisationsalgorithmus beeinflusst werden kann:

**maxParticles** Maximal erlaubte Anzahl an Partikeln. Die tatsächliche Anzahl kann aufgrund von dynamischen Optimierungen niedriger sein.

**minConfidence** Sobald das Vertrauen eines Partikels unter diesen Wert sinkt wird es gelöscht.

**groupProbability** Gibt an, wie groß der Anteil neuer Partikel ist, die in der Nähe bereits bestehender, guter Partikel platziert werden.

**nextGenFactorAge** Gewichtet den Einfluss des Alters eines Partikels auf die Entwicklung seines Vertrauens (s. Kap. 3.2.2.1).

**nextGenFactorCell** Gewichtet den Einfluss der Zellbelegung eines Partikels auf die Entwicklung seines Vertrauens (s. Kap. 3.2.2.2).

**nextGenFactorReading** Gewichtet den Einfluss der Güte von Abstandsmessungen eines Partikels auf die Entwicklung seines Vertrauens (s. Kap. 3.2.2.3).

**nextGenOffsetReading** Noch tolerierte durchschnittliche Abweichung der erwarteten Abstandsmessungen von den tatsächlichen Messungen, um die Messung als „passend“ zu interpretieren (s. Kap. 3.2.2.3).

**nextGenErrorTranslation** Künstlicher Translationsfehler einer Bewegung (s. Kap. 3.2.3.1) in Prozent. Beispiel: Eine Translation von 2m kann bei *nextGenErrorTranslation = 0.1* um $\pm$20cm abweichen.

**nextGenErrorRotation** Künstlicher Rotationsfehler einer Bewegung (s. Kap. 3.2.3.1). Skaliert einen möglichen Fehler von -180° bis +180°, bspw. bedeutet *nextGenErrorRotation = 0.1*, dass jede Rotation mit einem Fehler in einem Bereich von -18° bis +18° behaftet sein kann.

**ageThreshold** Gibt an, ab der wievielten Generation ein Partikel als „alt“ interpretiert wird.

**readingBadSensors** Anzahl Abstandssensoren, deren Messungen vernachlässigt werden sollen (s. Kap. 3.2.2.3).

Teilweise wird in dieser Arbeit das Präfix „nextGen“ aus Gründen der Lesbarkeit weggelassen.

# B. Sichtlinien-Algorithmus

Wie in Kap. 4.2.1 beschrieben, wurde ein Sichtlinienalgorithmus zur Simulation von Abstandsmessungen entwickelt. Im Gegensatz zum Bresenham-Line-Following-Algorithmus (wie in Schmitt [2012] vorgestellt) lässt dieser Algorithmus keine Zellen aus und ist außerdem schnell zu berechnen. Sein einfaches Funktionsrinzip lässt vermuten, dass dieser Algorithmus bereits in der einschlägigen Literatur zu finden ist.

Zur Berechnung der getroffenen Zellen wird vom Startpunkt ausgehend der nächste Zellenrand, der von der Sichtlinie geschnitten wird, betrachtet. Der Schnittpunkt wird dann als neuer Startpunkt des Algorithmus betrachtet und das Verfahren beginnt von vorne (s. Abb. B.1). In der rechten unteren Ecke ist eine der Zellen im Detail dargestellt. Das rote Kreuz bezeichnet den Startpunkt der Sichtlinie, der schwarze Pfeil die Richtung, in der geprüft werden soll. Die blauen und grünen Linien sind für die Berechnung relevante Abstände des Startpunktes von den Zellenwänden. Der Algorithmus kann an mehrere Abbruchbedingungen geknüpft sein. Es kann eine maximale Sichtlinienlänge vorgegeben werden, die nicht überschritten werden darf, ebenso ist es möglich, auf ein Verlassen der Karte zu prüfen. Im Zusammenhang der Lokalisation werden zudem belegte Zellen als Abbruchbedingung gewertet.
Zur effizienten Berechnung ist es nötig, möglichst viele Variablen bereits vorab zu berechnen. Am bedeutendsten ist dabei die Orientierung der Sichtlinie, da sich dadurch bereits eine Einschränkung der zu prüfenden Zellenwände ergibt. Soll der Algorithmus bspw. vom Startpunkt aus in Richtung Nordosten prüfen, so kann - wenn man vom Inneren einer Zelle ausgeht - nur die nördliche und östliche Zellenwand getroffen werden.
Die Berechnung des Schnittpunktes von Sichtlinie und Zellenwand wurde in dieser Arbeit über Geradengleichungen bestimmt. Dazu wird zunächst die Orientierung der Sichtlinie in eine Geradensteigung umgerechnet. Da damit aber nur knapp 180° abgedeckt werden können wird in einer

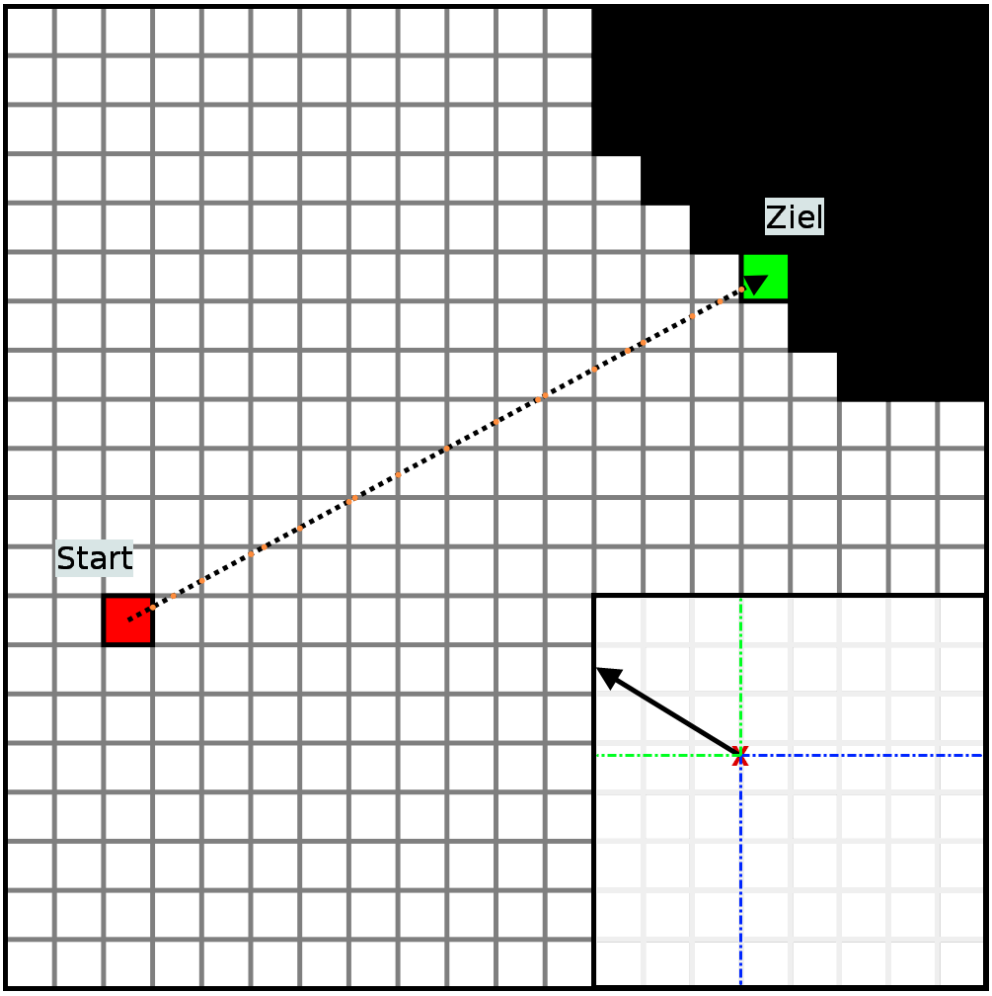



**Abbildung B.1.:** Vereinfachte Darstellung des Algorithmus

zusätzlichen Variablen gespeichert, ob die Sichtlinie in die westliche oder in die östliche Hälfte weist. Nun ist es möglich, die getroffene Zellenwand zu berechnen, ohne dafür trigonometrische Funktionen zu benötigen.

Am deutlichsten wird der Algorithmus durch den Code selbst, weshalb an dieser Stelle statt weiterer Ausführungen auf den sich auf der CD befindlichen Code verwiesen wird. Der Algorithmus befindet sich in `code/QT Quaternionendisplay/qusart/localization/particlecontroller.cpp` und heißt `float testMeasurement(int startX, int startY, float dir, float maxLength)`.

# C. Parametrisierung des Bewegungsmodells

Die folgenden Grafiken zeigen den Einfluss von Translations- und Rotationsfehler auf die Streuung der Partikel. Es wurden vom Start (jeweils links) aus immer fünf gleich weite, geradlinige Bewegungen nach rechts durchgeführt und alle Schritte in einem Bild kombiniert.

**Abbildung C.1.:** Insgesamt geringer Bewegungsfehler

**Abbildung C.2.:** Großer Translationsfehler

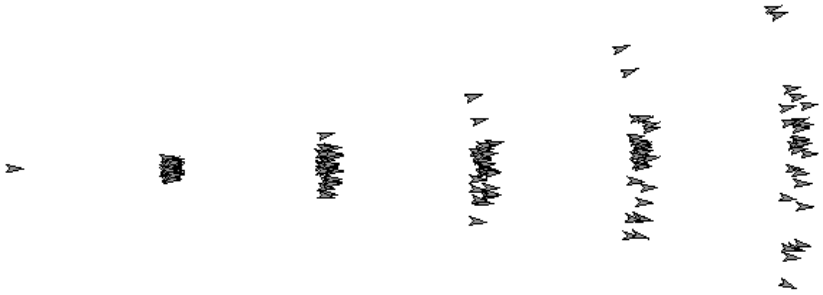


**Abbildung C.3.:** Großer Rotationsfehler

**Abbildung C.4.:** Großer Translations- und Rotationsfehler