Fachbereich Elektrotechnik und Informatik

Hochschule Niederrhein
University of Applied Sciences

Projektfach WS 16/17

SLAM - Simultaneous Localization and Mapping

eingereicht von: Andre Kürten
Matrikelnummer: 838311
Sebastian Schilling
Matrikelnummer: 971726

betreut durch: Prof. Dr.-Ing. Thomas Nitsche
Hochschule Niederrhein

Krefeld, der 8. Juni 2017

Kurzfassung

Dieser Bericht fasst die Arbeiten und Ergebnisse des Wahlprojektfaches des Masterstudiengangs Informatik der Hochschule Niederrhein zusammen. Ziel des Projektes ist die Erstellung einer Umgebungskarte eines Roboters in einer für ihn unbekannten Gegend. Mithilfe dieser generierten Karte, soll es dem Roboter möglich sein zu navigieren. Die oben genannten Anforderung lassen sich durch das SLAM-Problem beschreiben. Für das Projekt wurden verschiedenen Algorithmen ausprobiert und miteinander verglichen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

1 Einführung

1.1 Was ist SLAM?

In der Robotik steht das Akronym SLAM für Simultanesous Localization and Mapping. Dabei beschreibt SLAM das Problem eines Roboters, eine Karte von seiner unbekannten Umgebung zu erstellen und gleichzeitig mit dieser anhand von Sensordaten zu navigieren (auch SLAM-Problem genannt). Um das Problem zu lösen werden in der Regel probabilistische und nicht deterministische Ansätze verfolgt. Zur Lösung eines SLAM-Problems müssen u.a. folgende Unterprobleme gelöst werden:

- Positionsgewinnung der Landmarks
- Zuordnung von Sensordaten zu den Landmarks
- Entscheidung ob Sensordaten einen bereits verfassten Landmarks gehören.
- Minimierung des entstandenen Fehlers

Konkrete Ansätze zur Lösung des Problems werden Bayes Filter und Partikel Filter als probabilistische Modelle verwendet.

1.2 Bayes Filter

Im Allgemeinen ist ein Bayes - Filter ein probabilistischer Ansatz um eine unbekannte Wahrscheinlichkeitsdichtefunktion rekusiv über die Zeit hinweg zu bestimmen. In der Informatik und der Robotik ist ein Bayes - Filter ein Verfahren verschiedene, Wahrscheinlichkeiten für die Position und Orientierung eines Roboters zu berechnen, befolgt von einer Aktualisierung der wahrscheinlichsten Position innerhalb einer Karte. Im SLAM-Kontext wird zwischen Kalman- und Partikel - Filter unterschieden.

1.2.1 Kalman Filter

Vereinfacht kann man sagen, dass das Kalman - Filter die Störungen welche durch die Messgeräte verursacht werden entfernt. Das Filter bestimmt den aktuellen Systemzustand, rekursiv anhand der vorhergehenden gestörten Messungen. Für lineare Systeme

kann man sich auf das Kalman - Filter beschränken. In der Realität sind die Systeme meist jedoch nichtlinear.Um dieses Problem dennoch in den Griff zu bekommen benutzt man den *Extended - Kalman - Filter* (EKF) welcher auch für nichtlineare Systeme funktioniert. Das Filter besteht generell aus zwei Schritten:

1. Vorhersagen
2. Korrigieren

Beim Vorhersagen wird eine Annahme über den Systemzustand zum nächsten Zeitpunkt, anhand des bekannten Verhaltens des Systems getroffen. Bei der Korrektur wird der tatsächliche Systemzustand anhand der Messvorrichtung bestimmt, und anschließend den Zustand anhand der Abweichung der beiden Zustände korrigiert. Die nachstehende Abbildung zeigt exemplarisch den Kalman - Filter Algorithmus. Durch den rekusiven

1: **Extended_Kalman_filter**$(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$**:**

2: $\bar{\mu}_t = g(u_t, \mu_{t-1})$

3: $\bar{\Sigma}_t = G_t \, \Sigma_{t-1} \, G_t^T + R_t$

4: $K_t = \bar{\Sigma}_t \, H_t^T (H_t \, \bar{\Sigma}_t \, H_t^T + Q_t)^{-1}$

5: $\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$

6: $\Sigma_t = (I - K_t \, H_t) \, \bar{\Sigma}_t$

7: $return \; \mu_t, \Sigma_t$

Abb. 1.1: Kalman - Filter - Algorithmus

ansatz ist der Kalman - Filter auch für Echtzeitsysteme geeignet.

1.2.2 Partikel Filter

Der Partikel Filter wird auch Sequenzielle Monte-Carlo-Methode (SMC-Methode) genannt. Ziel ist es, die gerade aktuelle, aber unbekannte Wahrscheinlichkeitsdichte auf den Zustandsraum zu schätzen. Daraus kann dann der wahrscheinlichste Systemzustand abgeleitet werden.
Häufig ist es der Fall, dass der Ort von Objekten zu Beginn nicht bekannt ist und diese erst durch Messungen ermittelt werden müssen. Die Messungen werden aber immer eine gewisse Fehlerrate vorweisen. Dh. die Messungen geben den wahren Zustand nicht korrekt wieder. Es ist aber möglich, anhand der Messungen den Zustand zu schätzen. Während der Kalman Filter von einer normalverteilten Störgröße ausgeht, geht der Partikel Filter von einer nichtlinearen Störgröße aus. Hier können mehrere Maxima auftreten und den Systemzustand wider spiegeln.

Beim Partikel Filter wird eine Wolke von sogenannten Partikeln produziert, die im Anfangszustand gleichverteilt auf der Karte auftreten. Jeder Partikel beschreibt einen Zustand, der aus einem Gewicht und einem Punkt auf der Karte besteht. Der Partikel Filter besteht aus vier Schritten:

1. Fortschreiten (Bewegung)
2. Addieren eines Fehlers auf die Zustände
3. Messen und gewichten der Partikel
4. Resampeln

Der erste Schritt ist das Fortschreiten des Partikels. Diese Schätzung wird für jeden Partikel ausgeführt. Im zweiten Schritt wird ein künstlich ausgewählter Fehler auf jede Bewegung aus Schritt eins hinzugefügt. Dies ist notwendig um das Systemrauschen, das bei den realen Messwerten auftritt, nachzubilden. Im dritten Schritt wird überprüft, wie realistisch die einzelnen Partikel zu den realen Werten passen. Passen sie sehr gut, so erhalten sie ein hohes Gewicht. Ansonsten wird ihnen ein niedriges Gewicht zugewiesen. Im letzten Schritt, dem Resampling, findet eine Selektierung der Partikel statt. Partikel mit hohen Gewichten werden behalten und Partikel mit niedrigen Gewichten werden verworfen.

Der Partikel Filter kann auch dazu verwendet werden, die Position eines Roboters mit gegebener Karte zu ermitteln. Dazu werden auf der gestellten Karte die Partikel im Anfangszustand gleichverteilt verteilt. Danach wird überprüft, ob die Messungen des Roboters mit der Karte übereinstimmen. Ist die der Fall, so werden diese Partikel hoch gewichtet. Im nächsten Schritt werden von den hohen gewichteten Partikel mehr Partikel gestreut. Die weniger gewichteten Partikel werden verworfen oder von dort aus weniger Partikel gestreut.
Nach wenigen Schritten kann eine relative gute Positionsbestimmung trotz verrauschten Messwerten erfolgen.

Kapitel 2

2 Datenformate

Es gibt verschiedene Möglichkeiten einen Roboter mittels Sensoren zu versehen. Unter anderem gibt es 2D-Laser, 3D-Laser oder auch stereo Kameras. Wir haben uns im Rahmen des Projektes nur mit 2D-Laser Daten auseinander gesetzt und stellen hier das gängigste Datenformate vor. Für die Algorithmen haben wir ein eigenes Datenformat, abgeleitet von den bestehenden, benutzt.

2.1 Eigenes Datenformat

Während unseren Experimenten haben wir festgestellt, dass viele Beispieldaten eine eigne Logik der Daten besitzen. Deshalb haben wir uns auf ein eigenes Datenformat geeinigt. Jede Zeile stellt einen Scandurchlauf dar. Die ersten drei Werte beschreiben die Odometrie: x-Wert, y-Wert und den Winkel Theta. Die vierte Zahl gibt die Anzahl der darauf folgenden Messwerte wieder. Eine Kommentarzeile wird durch eine # eingeleitet. Abbildung 2.1 zeigt einen Auszug einer Zeile aus einer Beispieldatei.

2.2 Beispieldaten

Einige Sensor Beispieldaten können von folgender Webseite heruntergeladen werden: http://www.mrpt.org/robotics_datasets

```
1  #(Odemetry) x y theta (Laser) #num [values]
2  -2.424026 4.800517 2.590948 181 0.642000 0.645000
     0.645000 0.644000 0.644000 0.643000 0.642000
   0.649000 0.648000 0.646000 0.645000 0.652000
   0.651000 0.660000 0.659000 0.659000 0.667000
   0.668000 0.667000 0.674000 0.675000 0.675000
   0.683000 0.691000 0.692000 0.702000 0.702000
   0.710000 0.710000 0.718000 0.726000 0.735000
   0.743000 0.743000 0.760000 0.760000 0.771000
   0.780000 0.797000 0.797000 0.815000 0.824000
   0.833000 0.841000 0.858000 0.872000 0.880000
   0.910000 0.917000 0.933000 0.950000 0.967000
   0.992000 1.018000 1.024000 1.052000 1.077000
   1.104000 1.130000 1.156000 1.191000 1.226000
   1.252000 1.284000 1.336000 1.380000 1.421000
   1.467000 1.476000 1.469000 1.455000 1.447000
   1.437000 1.430000 1.442000 1.513000 1.600000
   1.681000 1.796000 1.922000 2.068000 3.150000
   3.134000 3.385000 3.757000 4.210000 4.799000
   5.584000 6.775000 7.435000 8.183001 8.183001
   8.183001 8.183001 8.183001 8.183001 8.183001
   8.183000 8.183000 8.183001 8.183001 6.860000
   6.397001 6.403001 6.420000 6.112000 6.090000
   5.285000 4.937000 4.947000 4.667000 4.243000
   3.900000 3.712000 3.544000 3.392000 3.250000
   3.129000 3.007000 2.903000 2.808000 2.712000
   2.631000 2.549000 2.472000 2.402000 2.335000
   2.276000 2.216000 2.163000 2.110000 2.065000
   2.019000 1.973000 1.937000 1.903000 1.859000
   1.824000 1.798000 1.764000 1.741000 1.714000
   1.688000 1.661000 1.634000 1.618000 1.591000
   1.574000 1.550000 1.532000 1.515000 1.499000
   1.482000 1.465000 1.448000 1.440000 1.423000
   1.415000 1.406000 1.389000 1.381000 1.372000
   1.363000 1.347000 1.346000 1.339000 1.330000
   1.322000 1.314000 1.305000 1.305000 1.298000
   1.298000 1.289000 1.282000 1.282000 1.275000
   1.276000 1.278000 1.272000 1.264000
```

Abb. 2.1: Eine Datenzeile im eigenem Datenformat

Kapitel 3

3 Algorithmen

Da es so schien als gäbe es eine große Auswahl bereits fertig implementiert und getestete Algorithmen und Frameworks haben wir uns nach Absprache mit dem Betreuer dafür entschieden uns auf diese fertige Software zu beschränken. Ebenfalls haben wir uns auf Entfernungsdaten aus einem Laserscanner beschränkt, und etwaige Bildverarbeitung außen vor gelassen. Anschließen ist die Software aufgelistet bei denen wir die meisten Erfolge verbuchen konnten.

3.1 DP-Slam

DP-Slam ist ein Algorithmus der von Austin Eliazar und Ronald Parr von der Duke University zur Verfügung gestellt wird. Das Programm liegt in der Programmiersprache C vor und akzeptiert als Eingangsdaten unbearbeitete 2D-Laser Daten sowie Odometriewerte. Der Sourcecode kann unter folgender URL heruntergeladen werden: https://openslam.org/dpslam.html.
DP-Slam basiert auf dem Partikel-Filter und nutzt die Verbundwahrscheinlichkeit zwischen Karte und Roboter Position aus.

Wir konnten das Programm ohne Fehler kompilieren und haben uns zuerst die mitgelieferte Demo angeschaut. Wie oben beschrieben, wird bei dem Algorithmus eine Karte erstellt (siehe Abbildung 3.1 und 3.2). Zur Erstellung der Karte kommt es nach einer bestimmten Anzahl von Iterationsschritten. Somit erhält man über die Zeit verschiedene Karten, die die abgescannte Gegend darstellen.

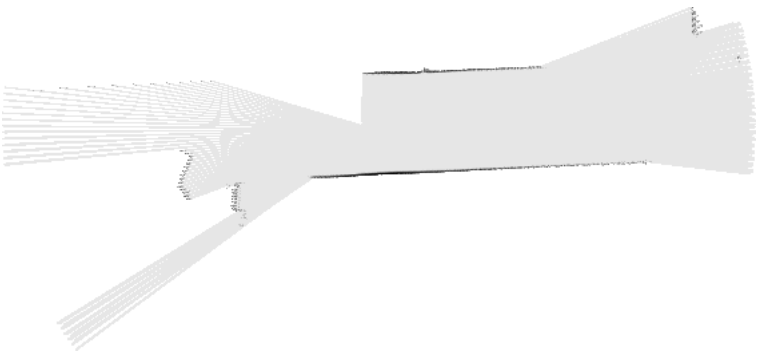

Abb. 3.1: Erste Karte nach erstem Durchlauf. Zu sehen ist ein Korridor.

Abb. 3.2: Karte nach mehreren Durchläufen. Zu sehen ist der Grundriss einer ganzen Etage.

Als nächsten Schritt haben wir versucht, das Programm mit Input-Daten von Dritten ans Laufen zu bekommen. Dazu wurde der Quellcode etwas verändert um die Daten einlesen zu können. Als Input-Daten wurden die Daten vom *tinySLAM*-Algorithmus genommen. Dieser Schritt hat erstaunlich gut geklappt. Als Output erschienen die zu erwarteten Karten, die mit dem *tinySLAM*-Algorithmus übereinstimmten.
Wir haben die Verarbeitung der Laser-Daten weiter verändert, sodass der Algorithmus auch unser eigenes Datenformat einlesen konnte. Zum Testen haben wir weitere unbearbeitete Laser-Daten getestet.

Alle bereits getesteten Datensätze beinhalten jeweils 181 Laserwerte pro Messung. Die nächste Herausforderung bestand darin, das Programm soweit abzuändern, dass es nicht nur mit 181 Laser-Daten lief, sondern auch mit deutlich weniger Werten. Hierzu wurde an verschiedenen Stellen der Code abgeändert, jedoch endete es immer mit Speicherzugriffsfehlern. Nachdem man die ersten Speicherzugriffsfehler behoben hatte, taten sich immer wieder neue Speicherzugriffsfehler auf. Hier war der Arbeitsaufwand nicht abzuschätzen, ob die Behebung der Fehler nicht immer wieder neue Probleme hervorrief. Um das Problem zu umgehen wurde ein anderer Ansatz gewählt.
Und zwar sollten die Reduzierung der Werte nicht dadurch geschehen, dass die Werte einfach wegfallen, sondern es sollte weiterhin mit 181 Werten gearbeitet werden. Jedoch sollten die Werte, die ignoriert werden sollten, mit einer ganz geringen oder ganz großen Distanz angegeben werden. Als kleinste Distanz konnte leider nicht Null gewählt werden, da der Algorithmus dies nicht akzeptierte. Leider war das Ergebnis nicht zufrierend

stellend. Die kleinen Werte wurden leider als Hindernis interpretiert und die großen Werte waren an den Stellen wo eigentlich Hindernisse waren als fehlerhaft anzusehen. Eine mögliche Lösung wäre die Approximation des auszulassenden Wertes. Dieser könnte zum Beispiel der Mittelwert aus Vorgänger und Nachfolger sein. Dies wurde aber aus zeitlichen Gründen nicht mehr umgesetzt.

3.2 tinySLAM

tinySLAM ist ein Open-Source Projekt der Wissenschaftler Bruno Steux und Oussama El Hamzaoui von der Mines ParisTech Universität. Entwicklet in weniger als 200 Zeilen C-Code und nur Laserdaten als Input. Diese Charakteristika erschien uns als Ideal da C gut auf andere Platformen zu portieren, hardwarenah und schnell ist. Ebenso weil die Anforderung sich auf Laserdaten zu beschränken ebenfalls erfüllt war. Beziehen kann man die Software unter *https://openslam.org/tinyslam.html.*
Nach der Kompilierung der Software untersuchten wir als erstes das mitgelieferte Testprogramm mit den dazugehörigen Testdaten. Das Testprogramm lieferte erste vielversprechende Ergebnisse. Neben einer Animation (Abb. 3.3) zur Laufzeit, stellt es nach Abschluss auch fertige Karten des Raumes (Abb. 3.4 / 3.5) zur Verfügung. In der Abbildung 3.3 kann man grafisch erkennen wie der Algorithmus arbeitet. Von seinem Startpunkt aus fährt er einmal im Kreis und erfasst alles soweit wie er kann. Im nächsten Durchlauf wird der Bereich untersucht wo potenziell keine Hindernisse erfasst wurden, und fährt erneut im Kreis. Als nächsten Schritt wollten wir das Testprogramm mit einem von uns ausgesuchten Datensatz testen. Um das zu bewerkstelligen schauten wir uns zuerst die verwendeten Daten einmal an.
Dabei stellten wir fest, dass wir bis auf den ersten Wert einer jeden Zeile gar keinen Wert, den zuvor ausgemachten Werten (Odometrie, Winkel, Entfernungsdaten etc.) zuordnen konnten. Zudem ist die Anzahl der Werte pro Zeile auch stark erhöht und unterliegen teilweise sehr großen Schwankungen von 2.4×10^{-3} bis 1.2×10^{4}. Ohne zusätzlich Informationen zu den verwendeten Testdaten war es uns nicht möglich diese zu deuten. Nichts desto trotz haben wir einfach einmal geschaut wie sich das Programm verhält wenn wir die Software mit anderen Datensätze (Abb. 2.1) ausführen. Die Software lief im Gegensatz aller Erwartungen mit den andren Testdaten ohne Fehlermeldungen durch. Allerdings ohne Animation zur Laufzeit und fehlerhaften Endresultaten (Abb. 3.6 und 3.7), was schon erahnen lässt, das die Testdaten nicht mit der Software kompatibel ist. Es scheint auf den Bilder so, als würde sich der Roboter nicht von der Stelle bewegen. Nach genauerem Vergleich mit den neuen Testdaten und den Daten welche bei tinySLAM an Bord waren, fiel uns auf das keine Informationen zur Odometrie und Ausrichtung des Roboters in den Daten vorhanden war. Das fehlen dieser Information, lässt darauf schließen dass diese Information aus den Daten zur Laufzeit berechnet werden. Da wir aber auch keine Information über die Ursprungsdaten haben, war es

uns auch mit Hilfe des Quellcodes nicht möglich die Berechnung dieser Information nachzuvollziehen, und entschieden uns diesen Ansatz aufzugeben, da Einarbeitungszeit und mögliche Resultate nicht einzuschätzen waren.

3.3 Sonstige

In diesem Kapitel werden Framework / Software aufgelistet welche untersucht wurden, aufgrund von schlechter Dokumentation die Ergebnis nicht interpretiert, oder auf Grund veralteter Softwarekomponenten (teilweise nicht mehr existent) gar nicht erst zu einem fertigen Programm kompiliert werden konnten. Zudem kam es auch zu unlösbaren Laufzeitfehlern.

Das Grundlegende Problem dabei war, die Entscheidung zu treffen wie lange man sich mit dem Problem und deren Lösung beschäftigt, obgleich man sich keineswegs sicher sein kann, das dass Resultat wirklich für unser Vorhaben geeignet ist.

3.3.1 MRPT

MRPT steht für Mobile Robot Programming Toolkit und ist ein umfassendes Framework. Das Framework ist OpenSource und es stehen eine gute Dokumentation bzw. Tutorials auf deren Website zur Verfügung. Das Toolkit und die Dokumentation kann unter folgender URL eingesehen werden: http://www.mrpt.org/.
Von diesem Framework haben wir uns am Anfang viel versprochen, da es eine gute Dokumentation und einen großen Leistungsumfang vorweisen konnte. Für unser SLAM-Problem mit 2D-Laser Daten kommen zwei Algorithmen des Frameworks infrage: icp-slam und rbpf-slam. Die Algorithmen liefen, aber ohne grafische Oberfläche, was den Gebrauch und die Auswertung schwierig machte.

Grund des Ausschlusses:
Nicht kompilierbar. Abhängigkeit zur alten Paketquellen, die nicht mehr existieren.

3.3.2 FALKOlib

FALKOlib (Fast Adaptive Laser Keypoint Orientation-invariant) ist eine Library, in der zwei neue *Keypoint-Detectors* implementiert wurden.
Später stellte sich zudem herraus dass die Software "lediglichïnteressante Schlüsselpunkte herrausfindet. Eine lösung für SLAM-Problem ist nicht vorgesehen, es dient nur zur Erweiterung eines Roboters der für ein SLAM-Problem eingesetzt wird.

Grund des Ausschlusses:
Keine Lösung des gewünschten Problems, zudem nicht kompilierbar.

3.3.3 SLoM - Sparse Least Squares on Manifolds

Das SLoM - Framework wollten wir genauer untersuchen, da damit geworben wurde das verschiedene Datenformate(LIDAR,Bilder etc.), und damit vielleicht auch unterschiedliche Ausführungen von Entfernungsdaten verarbeitet werden können. Der Algorithmus löst das SLAM-Problem mittels *Least Squares on Manifolds* - Methode.

Grund des Ausschlusses:
Nicht kompilierbar.

3.3.4 RFS-SLAM

RFS-SLAM ist ebenfalls ein komplettes Framework wo man eigene Modelle zu speziellen Problemen definieren kann. RFS-SLAM löst das SLAM-Problem mittels *Random Finite Set* - Estimation.

Grund des Ausschlusses:
Laufzeitfehler.

3.3.5 Linear SLAM

Linear SLAM ist ein Algorithmus für große Feature-Vektoren und Graphen. Der Algorithmus verwendet die Methode der kleinsten Quadrate. Als Eingabeparameter werden lokale Submaps akzeptiert.

Grund des Ausschlusses:
Eigenes Input- und Output-Material. Das Output-Material muss grafisch aufbereitet werden, da nur Koordinaten zurückgegeben werden.

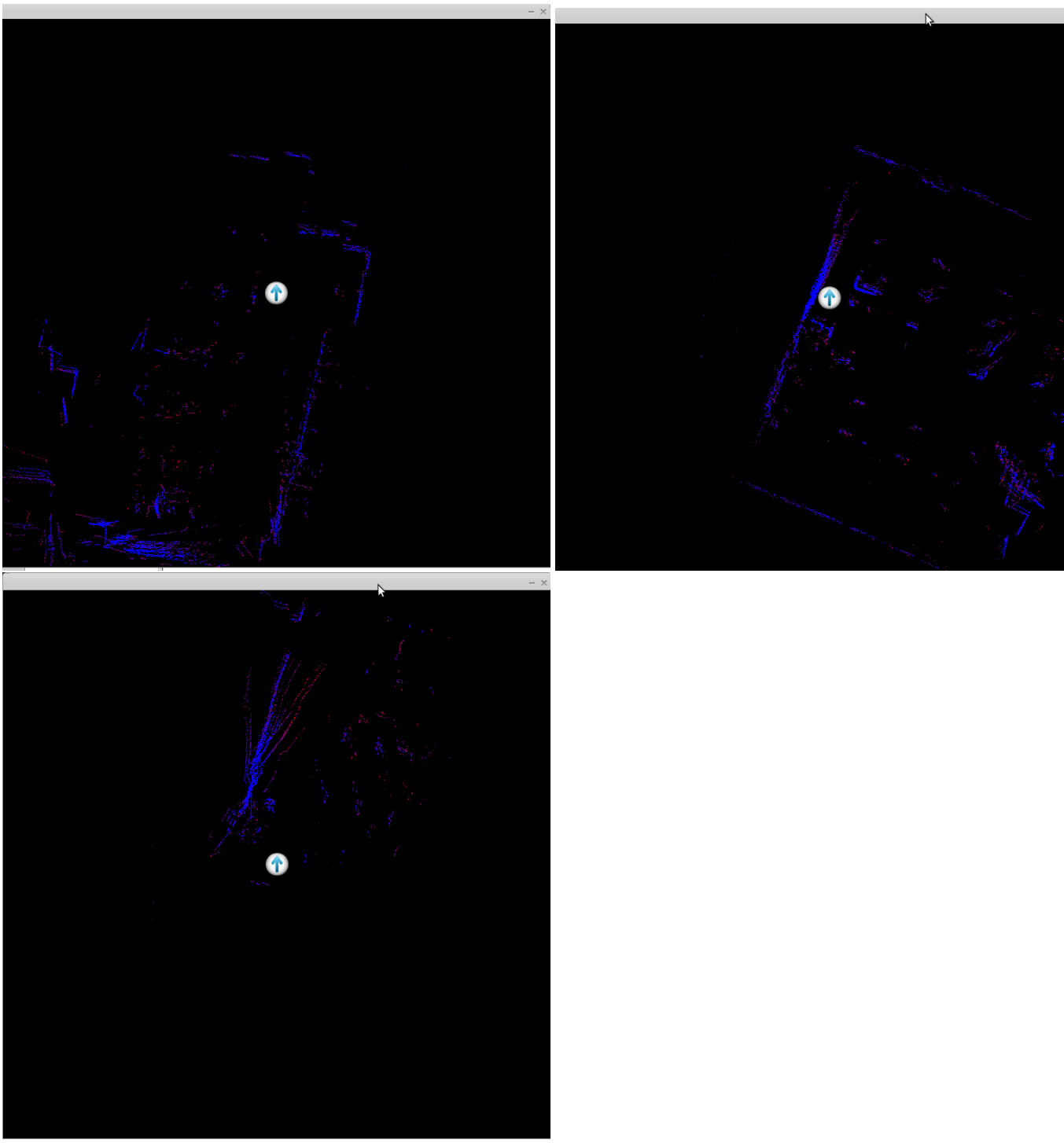

Abb. 3.3: Bildausschnitte während der Roboteranimation zur Laufzeit

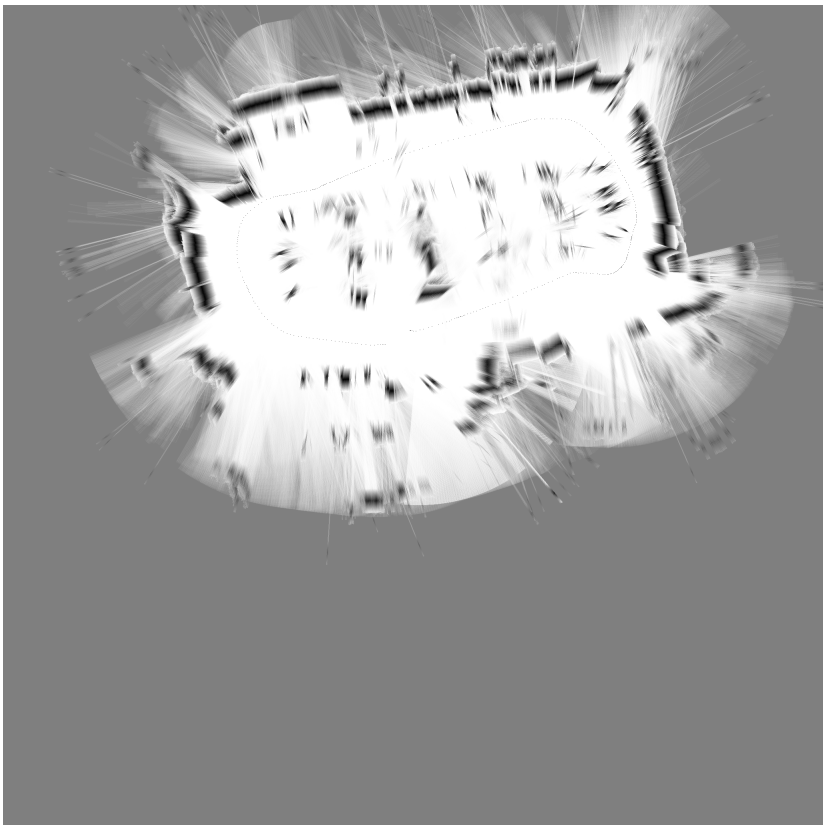

Abb. 3.4: Erste Karte nach erstem Durchlauf des Roboters

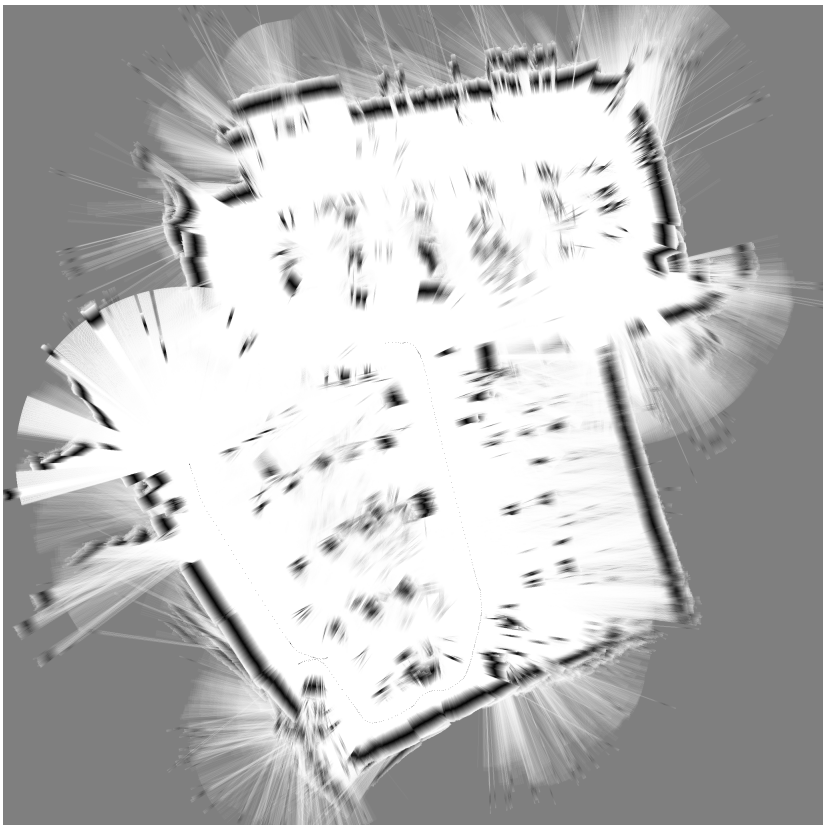

Abb. 3.5: Komplette Karte nach zweitem Durchlauf des Roboters

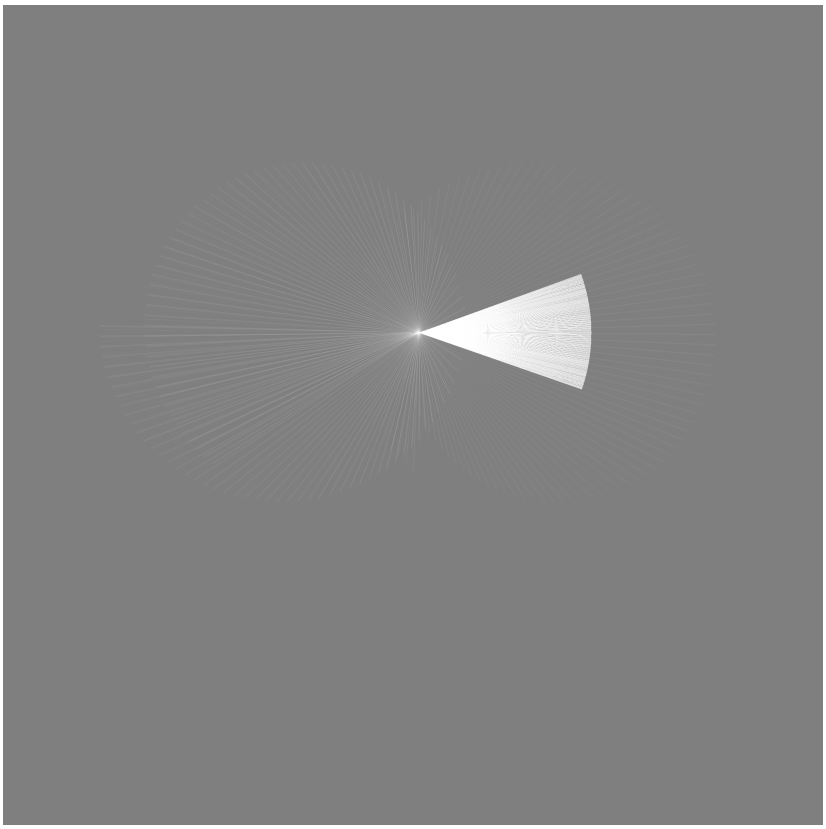

Abb. 3.6: Erste Karte nach erstem Durchlauf des Roboters (neue Daten)

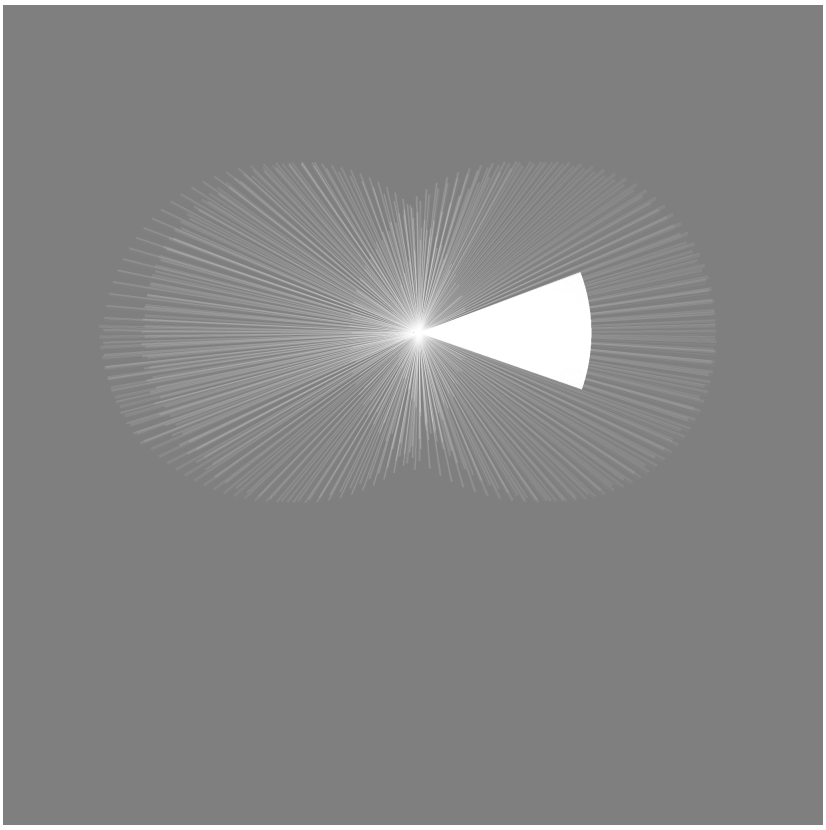

Abb. 3.7: Komplette Karte nach zweitem Durchlauf des Roboters (neue Daten)

Kapitel 4

4 Ausblick

Unsere Herangehensweise war es, existierende Algorithmen zu untersuchen und diese gegebenenfalls zu manipulieren. Somit haben wir viel Zeit investiert, die Algorithmen zu verstehen und die Test-Programme zu kompilieren. Die Fehlersuche in den einzelnen Programmen war ebenfalls sehr Zeit aufwendig.Im Nachhinein würden wir dieses Vorgehen nicht empfehlen.
Als weiteres Vorgehen würden wir vorschlagen, einen einfachen SLAM-Algorithmus selbst zu implementieren, und an echter Hardware zu Testen. Dadurch lernt man die Funktionsweise des SLAM-Problems kennen und kann das Programm ohne weiteres seinen Anforderungen anpassen.

Literaturverzeichnis

[1] Austin Eliazar, Ronald P.: DP-SLAM: Fast, Robust Simultainous Localization and Mapping Without Predetermined Landmarks. In: *IJCAI*, 2003

[2] Eliazar, Austin I. ; Parr, Ronald: DP-SLAM 2.0

[3] Liang Zhao, Shoudong H. ; Dissanayake, Gamini: Document for Linear SLAM MATLAB and C/C++ Source Code / University of Technology Sydney, Australia. 2013. – Forschungsbericht

[4] Mengelkoch, Marco: *Implementieren des FastSLAM Algorithmus zur Kartenerstellung in Echtzeit*, UniversitÃ€t Koblenz-Landau, Studienarbeit, 2007

[5] Riisgaard, S. ; Blas., M.: *SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping*. 2005

[6] *MRPT - Empowering C++ development in robotics*. http://www.mrpt.org/, . – Aufgerufen: 30.05.2017

[7] *Sequenzielle Monte-Carlo-Methode*. https://de.wikipedia.org/wiki/Sequenzielle_Monte-Carlo-Methode, . – Aufgerufen: 30.05.2017

[8] *Simultaneous Localization and Mapping*. https://de.wikipedia.org/wiki/Simultaneous_Localization_and_Mapping, . – Aufgerufen: 30.05.2017

[9] *Kalman Filter*. http://www.mi.hs-rm.de/~schwan/Projects/CG/CarreraCV/doku/kalman/kalman.htm#2, . – Aufgerufen: 08.06.2017

[10] *Extended Kalman Filter - SLAM*. http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam05-ekf-slam.pdf, . – Aufgerufen: 08.06.2017