Report Datenbankprojekt: Distributed Streaming EKF SLAM

Jestram, Johannes
jestram@posteo.de

Livert, Benedikt
benedikt.livert@gmx.de

Paranskij, Mark
mark.paranskij@gmail.com

January 31, 2020

Contents

1 Einleitung

In verschiedenen technischen Disziplinen ist es nützlich, auf Grundlage des bisherigen Verhaltens eines Systems eine Voraussage über dessen künftigen Zustand treffen zu können. Ein Beispiel dafür ist die Navigation: Mithilfe der bisherigen Bewegungsrichtung und der aktuellen Geschwindigkeit lässt sich die Position des Objektes zu einem zukünftigen Zeitpunkt schätzen. Um eine möglichst verlässliche Vorhersage treffen zu können, müssen verschiedene Aspekte des betreffenden Systems gemessen werden. Da Messungen jedoch stets - in unterschiedlichem Maß - fehlerbehaftet sind, sollten diese Messfehler auch in die Betrachtung des Systemzustandes einbezogen werden. Weiterhin unterscheidet sich die Komplexität der Methoden zur Vorhersage des zukünftigen Zustandes je nach betrachtetem System. Möglicherweise wird der Zustandsübergang des Systems durch komplexe Funktionen mit vielen Parametern dargestellt. Zuletzt erscheint es auch als sinnvoll, die getroffene Vorhersage anhand neuer Messungen zu überprüfen und gegebenenfalls anzupassen. Eine Methode, um solche Zustandsübergänge vorherzusagen und zu überprüfen, ist der sogenannte Kalman Filter [1].

Ein weiteres, vor allem durch die moderne Robotik geprägtes, Problem ist das Kartographieren einer Umgebung. Zum einen muss der autonome Roboter stets seine eigene Position innerhalb der unbekannten Umgebung kennen, zum anderen soll er eben diese Umgebung erkunden und aufzeichnen. Die Erkundung der Umgebung basiert in der Regel auf der Messung mit eingebauten Sensoren, daher ist die Positionsbestimmung der Erkannten Objekte stets relativ zur eigenen Position. Andererseits verortet sich der messende Roboter innerhalb der soeben kartierten Umgebung. Die Genauigkeit der eigenen Positionsbestimmung und der erkundeten Objekte in der Umgebung hängen also in beide Richtungen unmittelbar zusammen. Beide Messungen unterliegen jedoch Fehlern, möglicherweise gibt es sogar Aussetzer bei der Erfassung der Sensordaten. Die Problemstellung erinnert an ein Henne-Ei Problem. Diese Problemstellung wird im englischen als 'Simultaneous Localization and Mapping' (SLAM) bezeichnet.

Es ist möglich, den Kalman Filter zur Lösung des SLAM Problems zu nutzen [2]. Wie im Folgenden beschrieben werden wird, ist die Berechnung der Vorhersagen und deren Vergleiche mit den Messungen ein rechenaufwändiger Algorithmus.

1.1 Motivation

In diesem Projekt wird als Datengrundlage der Victoria Park Datensatz betrachtet. Diesem Datensatz liegt die Bewegung eines Fahrzeugs durch besagten Park zugrunde. Der Zustand des Fahrzeugs wird durch eingebaute Sensoren sowie GPS regelmäßig gemessen. Darüber hinaus ermittelt das Fahrzeug mittels eines Lasersensors vor ihm liegende Objekte.

Basierend auf diesen Messungen soll der EKF/SLAM Algorithmus implementiert werden. Da die reine Implementierung des Verfahrens basierend auf dem gegebenen Datensatz bereits erfolgt ist [3], wird es im Rahmen dieses Projektes darum gehen, wie ein solche Algorithmus parallelisiert werden kann. Um das EKF-SLAM Verfahren skalierbar zu machen, ist es folglich interessant dafür notwendigen Berechnungen zu parallelisieren. Es gilt zum Einen darum zu analysieren, inwiefern die Berechnungen selbst parallelisierbar sind. Weiterhin sind in den Anwendungsgebieten des Kalman Filter auch Szenarien denkbar, in denen Daten von mehr als einem System (z.B. Fahrzeug) verarbeitet werden sollen [4]. Für solche Szenarien stellt sich die Frage, ob und wie Daten von mehreren Fahrzeugen möglichst effektiv verarbeitet werden können.

Dieses Projekt zielt folglich darauf ab, die parallele Implementierung des EKF SLAM Algorithmus umzusetzen und zu evaluieren. Dabei wird die Software für parallele und verteilte Verarbeitung von Datenströmen Apache Flink[1] eingesetzt.

1.2 Extended Kalman Filter

Der Kalman Filter wurde im Jahr 1960 von Rudolph E. Kalman im Rahmen seines Papers "A New Approach to Linear Filtering and Prediction Problems" [1] vorgestellt. Er ist ein Ansatz um Zustände zu schätzen. Der Kalman Filter minimiert die geschätzte Fehlervarianz. Der grundlegende Algorithmus funktioniert wie folgt: Zunächst wird im sog. Prediction-Step der Zustand eines Systems zu einem Zeitpunkt t auf Grundlage vorhergegangener Zustandsmessungen und einer Zustandsübergangsfunktion geschätzt. Diese Schätzung wird dann im update-step (auch Correction-Step) nach dem Erhalt neuer Messdaten korrigiert und der Algorithmus beginnt von vorne.

Eine attraktive Eigenschaft dieses Algorithmus ist seine rekursive Natur; zur Berechnung des geschätzten neuen Zustands sind nur die Daten der vorherigen Messung nötig, da sich in dieser alle vorherigen Messungen zusammengefasst wiederfinden. Der Extended Kalman Filter unterscheidet sich

[1] `https://flink.apache.org/`

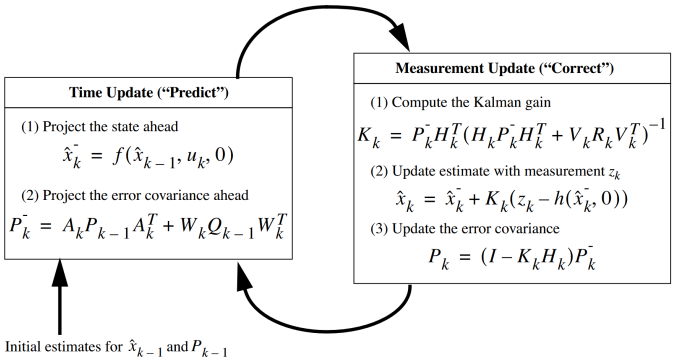

Figure 1: Extended Kalman Filter Algorithmus [5].

vom Diskreten Kalman Filter darin, dass die Zustandsübergänge durch nichtlineare Funktionen abgebildet werden. Die Nichtlinearität verkompliziert den Algorithmus, wie im folgenden gezeigt wird.

1.2.1 Prediction-Step

Der Zustand eines Systems kann über beliebig viele Parameter dargestellt werden. Um einen Zustand $\hat{x}_t$ in einen Zustand $\hat{x}_{t+1}$ zu überführen, muss für jeden Parameter eine Funktion definiert werden, die diesen Übergang abbildet. Im falle eines Fahrzeuges erfolgt dies über die Erstellung eines Bewegungsmodells. Auf Grundlage der Sensorik und vorhandenen Messwerte muss das Bewegungsmodells den Zustand, korrekt und dargestellt in die gewünschten Metriken, berechnen.

Doch das Bewegungsmodell ist nicht der einzige Einflussfaktor für den Zustandsübergang: Messfehler und äußere Einflussfaktoren sorgen für Ungenauigkeiten. Diese Ungenauigkeiten werden im Extended Kalman Filter berücksichtigt, wie im folgenden erläutert wird.

1.2.2 Update-Step

Der im Prediciton-Step geschätzte Zustand wird nun basierend auf neuen, externen Beobachtungen des Zustandes korrigiert. Jedoch ist es nicht so, dass die Schätzung einfach auf die äußere Beobachtung gesetzt wird. Die Korrektur wird gedämpft um den sog. Kalman Gain. Dieser berechnet sich sowohl aus Messunsicherheiten, als auch eine Kovarianzmatrix, die die Abhängigkeiten zwischen den einzelnen Parametern des Zustands abbildet. WARUM LINEARISIERUNG Weiterhin muss das nichtlineare

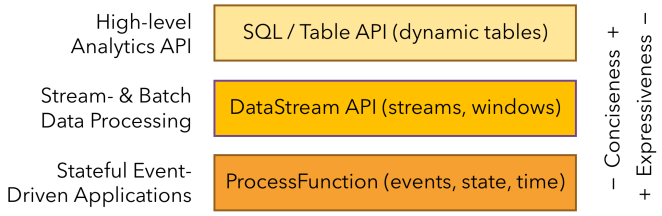

Figure 2: APIs auf verschiedenen Abstraktionsniveaus [**?**].

1.3 Simultaneous Localization and Mapping

2 Methodik

2.1 Verwendete Software

Im folgenden werden zunächst die dem Projekt zugrunde liegenden Konzepte detailliert vorgestellt. Danach wird erklärt, welche softwareseitigen Werkzeuge zur Umsetzung benutzt werden und warum. An dieser Stelle wird der Fokus auf den für das Projekt relevanten Funktionalitäten der jeweiligen Software liegen. Anschließend werden wir ausführlich auf die bereitgestellten Daten eingehen und deren Spezifika erläutern, da dieses Wissen Grundlage der Implementierung des Algorithmus ist. Als nächstes werden wir ausgewählte Designentscheidungen der Implementierung vorstellen. Die Ergebnisse bzw. die Ausgabe unserer Implementierung werden im Anschluss dokumentiert, analysiert und bewertet.

2.1.1 Apache Flink

Flink ist eine Software zur verteilten und parallelen Verarbeitung von Stream- oder Batchdaten. Es ist aus dem Stratosphere Projekt[2] [6] der TU-Berlin hervorgegangen und ein Top-Level-Projekt der Apache Foundation [3]. Die Software ist per Open Source Lizenz verfügbar und aufgrund der hohen Performanz [6] ein beliebtes Werkzeug für verteilte und skalierbare Datenverarbeitung.

Flink basiert auf der Java Virtual Machine (JVM) und ist in Scala und Java implementiert [4]. Der Funktionsumfang [Abb. 2] umfasst die paral-

[2] `http://stratosphere.eu`
[3] `https://apache.org`
[4] `https://github.com/apache/flink`

lele und verteilte Verarbeitung von Datenströmen, sowie einmaligen verschiedene Application Programming Interfaces (APIs).

Da das Ziel dieses Projektes die Anwendung von EKF-SLAM auf mehrere Fahrzeuge in Echtzeit ist, wird vor allem die DataStream API genutzt. Diese liefert vorgefertigte Funktionalitäten für viele übliche Aufgaben in der Verarbeitung von Datenströmen. Diese umfassen unter anderem die automatisierte parallelisierung der Verarbeitung von Datenströmen. Weiterhin ist es möglich, eingehende Daten mit sog. Schlüsseln (engl. keys) zu versehen und die weitere Verarbeitung der Daten im Folgenden abhängig von dem zugeordneten Key zu mache. Dies erfolgt über die Klassen KeyedDataStream und KeyedDataPoint. Ein KeyedDataStream verarbeitet generische Datentupel, welche jeweils mit einem Zeitstempel und einem Schlüssel versehen sind. Dabei liefern KeyedDataPoint eine Art Blaupause für den Inhalt eines einzelnen Datentupels.

Mit Hilfe solcher geschlüsselter Datenströme ist es möglich, für jeden Schlüssel einen eigenen Zustand (engl. state) zu definieren. Folglich ist es mit Schlüsseln und Status möglich, Daten und darauf basierende Berechnungen nach deren Quelle oder Zugehörigkeit klar voneinander zu trennen.

2.1.2 InfluxDB

InfluxDB[5] ist eine für Zeitreihendaten optimierte Datenbank. Die Interaktion erfolgt in einer Anfragesprache ähnlich zu SQL. Es existieren APIs für verschiedene Programmiersprachen, unter anderem Java.

Jedes zu speichernde Datentupel muss mit einem Zeitstempel versehen sein. Wie in Flink ist auch hier möglich, eingehende Daten anhand eines Schlüssels zu unterscheiden. Daten können in unterschiedliche Datenbanken gespeichert werden und innerhalb einer Datenbank nach Messreihen unterschieden werden. Einzelne Messungen werden in sog. Feldern gespeichert. Anzahl und Typ des Inhalts der einzelnen Felder können durch den Nutzer festgelegt werden. Dank der genannten Funktionen bietet InfluxDB eine leistungsfähige Lösung für die anwendungsspezifische Speicherung von Echtzeitdaten.

2.2 Victoria Park Datensatz

Die Inhalte dieses Datensatzes [6] wurden im Jahr 2006 im Victoria Park in Sydney gesammelt. Es wurde die Bewegung eines Fahrzeugs durch den

[5] `https://www.influxdata.com/products/influxdb-overview`

[6] `https://www.mrpt.org/Dataset_The_Victoria_Park`

Park aufgezeichnet und währenddessen vom Fahrzeug Objekte in der nahen Umgebung erfasst. Die Dauer der Aufzeichnungen ist rund 25 Minuten.

2.2.1 Messungen

Es existieren drei verschieden Quellen für Messungen. Die internen Sensordaten liefern die Odometrie. Diese setzt sich aus dem Lenkwinkel α, sowie der Geschwindigkeit v, gemessen am linken Hinterrad, zusammen. Außerdem wurde regelmäßig die Position des Fahrzeuges via GPS bestimmt. Die Messung der in der Umgebung befindlichen Objekte erfolgt mittels eines Lasersensors.

Neben den rohen Messungen existieren unterschiedlich aufgearbeitete Datensätze. Diese unterscheiden sich in der Anzahl der enthaltenen Datenpunkte sowie der Formatierung der Werte. Die Odometriedaten werden stets gemeinsam im Format Zeitstempel, Lenkwinkel, Geschwindigkeit angegeben. Es existiert eine Datei mit rund 66.000 Messungen und eine mit rund 4.000.

Auch die GPS-Daten werden in mehreren Versionen geliefert. Generell ist festzustellen, dass die GPS-Messungen in regelmäßigen Abständen von 200ms geliefert werden. Jedoch sind teils starke zeitliche Lücken in den Messungen vorhanden, sodass über einen längeren Zeitraum keine Positionsbestimmung erfolgt. Auch enthält der Datensatz einige Ausreißer. Diese erkennt man daran, dass das Auto, falls diese Messungen korrekt wären, von einem zum nächsten Punkt eine viel zu große Distanz zurücklegen würde. Beide genannten Mängel wurden auch von den Autoren des Datensatzes erkannt und benannt.

Der Lasersensor ist über der vorderen Stoßstange des Fahrzeugs positioniert und vermisst einen 180-Grad Winkel. Dieser Blickwinkel ist wird mit 361 Messpunkten von rechts nach links (in Fahrtrichtung) abgetastet. Für jeden einzelnen Messpunkt speichert der Laser den Abstand des gemessenen Objektes. Die Messungen sind für Objekte, die sich höchstens 80m entfernt befinden, hinreichend präzise.

2.2.2 Probleme mit den Daten

Im Laufe des Projektes stellte sich der Umgang mit den Daten als das zeitaufwändigste Problem dar. Vor allem die unterschiedlichen Versionen desselben Datensatzes sorgten für regelmäßige Probleme.

Auf der Website des Datensatzes [7] befindet sich ein Datensatz bestehend aus 4832 Datenpunkten, der laut Dokumentation mit hilfe eines externen

[7] `https://www.mrpt.org/Dataset_The_Victoria_Park`

Tools, des Rawlog Viewers [8], als Textdatei extrahiert und visualisiert werden kann. Diese 4832 Datenpunkte setzen sich zur Hälfte aus Inkrementen zwischen zwei Messungen in Richtung der X-Achse, Y-Achse und der Drehung des Fahrzeuges zusammen, sowie 2416 Lasermessungen. Problematisch bei diesem Datensatz ist hierbei jedoch die Tatsache, dass die komplexe Geometrie zwischen Geschwindigkeitsmessung und Laser- und GPS-Messung am Fahrzeug nicht näher dokumentiert war, was zu einem hohen Maß an Ambiguität bezüglich der Verarbeitung des Datensatzes geführt hat.

Darüber hinaus befindet sich auf der originalen Seite für den Datensatz der Publikation [9] ein Datensatz im ASCII Format. Dieser ist aus 69941 Datenpunkten zusammengesetzt, von denen 61763 Odometrie Daten sind, die jeweils einen Zeitstempel, die Geschwindigkeit in Metern pro Sekunde und den Winkel der Vorderachse des Fahrzeuges beinhalten. Dazu kommen 948 Datenpunkte für GPS-Messungen, sowie 7230 Lasermessungen.

Ebenfalls, befindet im originalen Download [10] ein MatLab Datensatz, den wir mit Hilfe eines Python-Skriptes ausgelesen haben, welcher sich aus 61945 gleich zusammengesetzten Odometrie Datenpunkten, 4466 GPS-Messungen und 7249 Lasermessungen, insgesamt also 73660 Datenpunkten, ergibt.

Aufgrund eines Mangels an definitiver Information und inkohärenter Datensätze wurde der Projektfortschritt lange durch das auswerten dieser Datensätze hinausgezögert. Hierbei haben wir versucht durch die Visualisierung der Daten diese zu validieren. Dies war führte bei dem kleinen Datensatz, bestehend aus 4832 Datenpunkten, zu keinem Erfolg, da das Format der Inkremente nicht ausreichend dokumentiert ist, und wir erst mit viel Recherche und durch interne Informationen aus dem Rawlog Viewer [11] auf das zugrundliegende Bewegungsmodel des Datensatzes [12] gestoßen sind.

In visualisierter Form waren lediglich die GPS-Messungen des MatLab Datensatzes ähnlich der Form des bekannten Victoria Parks, wodurch wir uns entschlossen haben EKF aus Basis der GPS Daten auf diesem Datensatz anzuwenden um das Projekt voranzutreiben. Gesamtheitlich betrachtet machten die genannten Probleme mit dem Datensatz circa die Hälfte der Projektarbeit aus. Selbst am Ende des Projektes kam es noch dazu, dass wir verschiedene Eingangsdaten ausprobieren mussten, um valide und

[8] https://www.mrpt.org/list-of-mrpt-apps/rawlogviewer

[9] http://www-personal.acfr.usyd.edu.au/nebot/victoria_park.htm

[10] https://www.mrpt.org/Dataset_The_Victoria_Park

[11] https://www.mrpt.org/list-of-mrpt-apps/rawlogviewer

[12] https://www.mrpt.org/tutorials/programming/odometry-and-motion-models/probabilistic_motion_models

funktionierende Ergebnisse zu erzielen, da beispielsweise der ASCII Datensatz gravierend vom MatLab Datensatz abweicht und aus unserer Perspektive als falsch anzusehen ist. So weichen die GPS-Messungen der beiden Datensätze gänzlich voneinander ab und die Odometrie durch den gleichen EKF-Algorithmus verarbeitet, resultiert in komplett unterschiedlichen Ergebnissen.

2.3 Implementierung

2.3.1 Simulatiaon mehrerer Fahrzeuge

Da es in diesem Projekt auch um die Parallelisierung der Verarbeitung von Daten unterschiedlicher Fahrzeuge geht, haben wir den vorhandenen Datensatz mehrfach repliziert. So kann die Berechnung mehrerer Systeme und deren Zustände simuliert und mit Hilfe von Flink parallelisiert werden. Die replizierten Daten haben wir zur besseren Veranschaulichung verändert; beispielsweise wurde die Reihenfolge der Tupel umgekehrt oder nur Teile des vollständigen Datensatzes für unterschiedliche simulierte Fahrzeuge verwendet.

2.3.2 Umsetzung in Apache Flink

Apache Flink unterscheidet Arbeitsaufträge in Streaming- und Batch-Aufträge. Diese unterscheiden sich in der Art und Weise der Parallelisierung der Berechnungen: bei Streams müssen beispielsweise Aggregationen mit einem Fenster versehen werden, da man sie bei einem potentiell unendlich langen Datenstrom nicht auf die gesamten Daten ausführen kann. Bei Batch-Daten ist das im Gegensatz möglich.

Da die Nutzung des Kalman Filters vor allem vor dem Hintergrund von kontinuierlich, über einen langen Zeitraum eintreffenden Datenströmen sinnvoll ist, fiel die Entscheidung auf einen Streaming-Auftrag.

Abgesehen von der Implementierung des EKF/SLAM Algorithmus bestand die Kernherausforderung beim Implementieren in der Parallelisierung der Berechnungen. Dazu mussten zum einen die Daten der unterschiedlichen Fahrzeuge auch bei der Verarbeitung dem entsprechenden Fahrzeug zugeordnet werden. Da jedes Fahrzeug einen eigenen, distinkten Zustand hat, dürfen die eingehenden Daten nicht vermischt werden. Zur Umsetzung dieser Unterscheidung dienten uns vor allem KeyedDataStreams. Jedes Fahrzeug verfügt über eine eigene ID. Diese ID wird dann zum Schlüssel des KeyedDataStreams, sodass die interne Zuordnung der Eingangsdaten eindeutig

und korrekt ist. Die Berechnungen können durch die Schlüssel also für jedes Fahrzeug parallelisiert werden.

Darüber hinaus

2.3.3 EKF Implementierung

Prediction Schritt und Bewegungsmodell Für den EKF ohne SLAM, der folglich eine GPS-Messung als Grundlage des Update-Step verwendet, initialisierten wir den Filter mit folgenden Werten und Funktionen:

Der Zustand des Fahrzeugs wird durch die x- und y-Koordinaten sowie den Winkel des Fahrzeuges zur x-Achse des Koordinatensystems angegeben.

$$\hat{x}_t = \begin{pmatrix} x_t \\ y_t \\ \phi_t \end{pmatrix} \tag{1}$$

Da sich der find the bug Sensor zur Geschwindigkeitsmessung am linken Hinterrad befindet, muss die Messung noch auf die Fahrzeugmitte zentriert werden.

$$\begin{aligned} v &= \frac{v_{Messung}}{1 - \tan(\alpha) * \frac{H}{L}} \\ &\text{mit} \\ v &= \text{Fahrzeuggeschwindigkeit} \\ v_{Messung} &= \text{Geschwindigkeit am linken Hinterrad} \\ H &= \text{Achsabstand Sensor zur Fahrzeugmitte} = 0,76m \\ L &= \text{Abstand Hinterachse zur Vorderache} = 2,83m \end{aligned} \tag{2}$$

Die Zustandsvorhersage $\hat{x}_{t+1}^-$ ergibt sich aus einem Bewegungsmodell, das anhand der vergangenen Zeit ΔT zwischen dem letzten Update- und dem jetzigen Prediction-Step die neue Position des Fahrzeuges schätzt.

$$\begin{aligned} \hat{x}_{t+1}^- = f(\hat{x}_t, u_t) &= \begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \phi_{t+1} \end{pmatrix} \\ &= \begin{pmatrix} x_t + \Delta T(v\cos(\phi_t) - \frac{v}{L}\tan(\alpha)) * (a\sin(\phi_t) + b\cos(\phi_t)) \\ x_t + \Delta T(v\sin(\phi_t) + \frac{v}{L}\tan(\alpha)) * (a\cos(\phi_t) - b\sin(\phi_t)) \\ \phi_t + \Delta(\frac{v}{L}\tan(\alpha)) \end{pmatrix} \end{aligned} \tag{3}$$

Während des EKF wird die Kovarianzmatrix P in jedem Durchlauf verändert. Wir initialisieren sie mit einem hohen Wert, sodass sie sich in-

nerhalb der ersten Durchläufe stark verändert.

$$P_{initial} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{4}$$

Für die Linearisierung des Bewegungsmodells, i.e. zur Aktualisierung von P, im EKF wird außerdem noch die Jakobi-Matrix des Bewegungsmodells benötigt.

$$\begin{aligned} F_{ekf} &= \frac{\delta f}{\delta \hat{x}} \\ &= \begin{pmatrix} 1 & 0 & -\Delta T(v\sin(\phi_t) + \frac{v}{L}\tan(\alpha) * a\cos(\phi_t) - b\sin(\phi_t)) \\ 0 & 1 & \Delta T(v\cos(\phi_t) - \frac{v}{L}\tan(\alpha) * a\sin(\phi_t) + b\cos(\phi_t)) \\ 0 & 0 & 1 \end{pmatrix} \end{aligned} \tag{5}$$

Update Schritt Im Update Schritt werden die Ergebnisse des Prediction Schritt mit Hilfe der GPS-Messung aktualisiert. Dazu muss zunächst das Beobachtungsmodell definiert werden; dessen Aufgabe ist es, die Messungen des GPS-Sensors auf das Format des Zustandsvektors abzubilden. Im vorliegenden Fall werden x- und y-Koordinate im passenden Format gemssen. Der Winkel ϕ wird durch den GPS-Sensor nicht gemessen und daher auch nicht im Beobachtsungsmodell abgebildet. Der auf Grundlage des Beobachtungsmodells berechnete Kalman Gain K_t wird aufgrund seiner Dimensionalität dennoch ϕ beeinflussen.

$$z_{ekf} = h(\hat{x}) = \begin{pmatrix} x_{gps} \\ y_{gps} \end{pmatrix} \tag{6}$$

Zwar ist das Beobachtungsmodell linear, da jedoch aufgrund des nichtlinearen Bewegungsmodells der Extended Kalman Filter verwendet wird, muss auch vom Beobachtungsmodell die Jakobi-Matrix gebildet werden.

$$H(\hat{x}) = \frac{\delta h}{\delta \hat{x}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \tag{7}$$

$$R = \begin{pmatrix} 0,5 & 0 \\ 0 & 0,5 \end{pmatrix} \tag{8}$$

2.4 SLAM Implementierung

Der Unterschied zum EKF ohne SLAM-Anteil liegt vor allem darin, dass der Update-Schritt nun auf Grundlage der in der Umgebung verorteten Objekte

stattfindet. Der Zustandsvektor $\hat{x}$ des Systems enthält nun zusätzlich zur Position und Fahrtrichtung des Fahrzeuges die Koordinaten jedes gemessenen Objektes - im Falle des Victoria Parks vor allem Bäume.

$$\hat{x} = \begin{pmatrix} x_{fahrzeug} \\ y_{fahrzeug} \\ \phi \\ x_{objekt-1} \\ y_{objekt-1} \\ ... \\ ... \\ x_{objekt-n} \\ y_{objekt-n} \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ \phi \\ x_L \\ y_L \end{pmatrix} \tag{9}$$

Ableiten der Objektposition aus den Lasermessungen Bezüglich der Lasermessungen, deren Tupel in jedem Datensatz vergleichbar aussehen und jeweils aus 361 einzelnen Abstandsmessungen mit einer Reichweite von 80 Metern bestehen, müssen entsprechend gesondert Berechnungen angestellt werden, damit man Bäume in Form von Koordinaten erhält. Dazu müssen zunächst ungültige Messpunkte, die eine zu geringe oder zu große Messentfernung aufweisen, sowie ungültige Messpunkte, die bei jeder Lasermessung zu Beginn der Messserie auftreten, auf 0 gesetzt werden, damit lediglich gültige Messungen übrig bleiben. Anschließend werden zusammenhängende Messpunkte, die einem individuellen Baum zugesprochen werden der eine bestimmte Distanz zum nächstgelegenen Baum aufweist, gruppiert um ein Cluster aus Messpunkten auf eine Koordinate mit Durchmesser per Baum reduzieren zu können. Dies geschieht mit den folgenden Formeln: FORMELN AUS DEM PAPER FÜR BÄUME EINFÜGEN.

Prediction-Step Das Bewegungsmodell des Fahrzeuges ändert sich im Vergleich zum EKF nicht. Die Funktion $f(\hat{x}_t, u_t)$ wird allerdings um Funktionen für die Abbildung eines Objektes vom Zeitpunkt t zum Zeitpunkt $t+1$ ergänzt. Die Objekte werden zur Vereinfachung als unbeweglich angenommen, daher sind die neue Abbildungsfunktion 10, wie auch die zugehörige Jakobi Matrix 11 trivial.

$$\hat{x}_{t+1}^{-} = f(\hat{x}_t, u_t) = \begin{pmatrix} x_{t+1} \\ y_{t+1} \\ \phi_{t+1} \\ x_{L_t} \\ y_{L_t} \end{pmatrix} \tag{10}$$

$$F_{slam} = \begin{pmatrix} F_{ekf} & 0 \\ 0 & I \end{pmatrix} \tag{11}$$

Jedoch ändert sich das Beobachtungsmodell z_{SLAM}, welches nun nicht mehr auf GPS Messungen, sondern auf den gemessenen Objekten im Umfeld des Fahrzeuges basiert.

$$\begin{aligned} z_{slam} &= \begin{pmatrix} z_\gamma^i \\ z_\beta^i \end{pmatrix} = h(\hat{x}, x_i, y_i) = \begin{pmatrix} \sqrt{q} \\ \arctan 2(\delta_y, \delta_x) - \phi_t \end{pmatrix} \\ &\text{mit} \\ q &= \delta^\intercal \delta \\ \delta &= \begin{pmatrix} \delta_x \\ \delta_y \end{pmatrix} = \begin{pmatrix} x_{L,t+1} - x_{L,t} \\ y_{L,t+1} - y_{L,t} \end{pmatrix} \end{aligned} \tag{12}$$

Im SLAM Fall ist also das Beobachtungsmodell im Gegensatz zum EKF mit GPS nicht linar. Demnach ist auch die Jakobi Matrix H_{SLAM} 13 komplexer als zuvor.

$$\begin{aligned} H(\hat{x}) = \frac{\delta h}{\delta \hat{x}} &= \begin{pmatrix} \frac{\delta h_\gamma}{\delta \hat{x}} \\ \frac{\delta h_\beta}{\delta \hat{x}} \end{pmatrix} = \begin{pmatrix} \frac{\delta z_\gamma}{\delta (x_i, y_i, \phi, \{x_L, y_L\})} \\ \frac{\delta z_\beta}{\delta (x_i, y_i, \phi, \{x_L, y_L\})} \end{pmatrix} \\ &= \begin{pmatrix} hi \\ hi \end{pmatrix} \end{aligned} \tag{13}$$

3 Evaluation

4 Zusammenfassung

5 Diskussion

6 Appendix

References

[1] R. E. Kalman, "A new approach to linear filtering and prediction problems," 1960.

[2] W. Burgard and K. Arras, "Introduction to Mobile Robotics: SLAM: Simultaneous Localization and Mapping," 2017.

[3] G. Jose and E. Nebot, "Simultaneous Localization and Map Building: Test case for Outdoor Applications."

[4] K. Juraszek, N. Saini, M. Charfuelan, H. Hemsen, and V. Markl, "Extended Kalman Filter for Large Scale Vessels Trajectory Tracking in Distributed Stream Processing Systems," in *Advanced Analytics and Learning on Temporal Data* (V. Lemaire, S. Malinowski, A. Bagnall, A. Bondu, T. Guyet, and R. Tavenard, eds.), vol. 11986, pp. 151–166, Cham: Springer International Publishing, 2020.

[5] G. Welch and G. Bishop, "An Introduction to the Kalman Filter," p. 81.

[6] A. Alexandrov, R. Bergmann, S. Ewen, J.-C. Freytag, F. Hueske, A. Heise, O. Kao, M. Leich, U. Leser, V. Markl, F. Naumann, M. Peters, A. Rheinländer, M. J. Sax, S. Schelter, M. Höger, K. Tzoumas, and D. Warneke, "The Stratosphere platform for big data analytics," *The VLDB Journal*, vol. 23, pp. 939–964, Dec. 2014.