hochschule mannheim

Realisierung und Stabilisierung eines Kalmanfilters für einen 1D Radarsensor und zusätzliche Objektverfolgung mittels DB-Scan für einen 3D Radarsensor

Jacob Gärtner, Nicolas Boskamp und Lara Elvira Gómez

Projektlabor

Maschinelles Lernen

im Studiengang Informationstechnik der Fakultät Informationstechnik

Vorgelegt von	Jacob Gärtner	(2160954)
	Lara Elvira Gómez	(2160965)
	Nicolas Boskamp	(2160964)
am	18. Februar 2022	
Lehrbeauftragter	Dr. -Ing Wei Yap Tan	

Schriftliche Versicherung laut Studien- und Prüfungsordnung

Hiermit erklären wir, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Mannheim, 18. Februar 2022

Jacob Gärtner, Lara Elvira Gómez und Nicolas Boskamp

Inhaltsverzeichnis

1 Kalman Filter für 1D-Radarsensor **4**

1.1 Aufgabenstellung Kalman Filter für 1D-Radarsensor 4

1.2 Realisierung Kalman Filter für 1D-Radarsensor 5

1.2.1 Initialisierung 5

1.2.2 Die *Step()*-Methode 7

1.2.3 Optimierung der Initialisierung von Q und R für 1D-Radarsensor 9

1.2.4 Ergebnisse des Kalman-Filters bei verschiedenen Bewegungsarten 11

1.2.5 Adaption für wechselnde Bewegungsarten in Echtzeit 13

2 Objektverfolgung mit 3D-Radarsensor **15**

2.1 Aufgabenstellung Objektverfolgung mit 3D-Radarsensor 15

2.2 Realisierung zur Rekonstruktion der Geschwindikeit aus der radialen Komponente 16

2.3 Realisierung des DBScan-Verfahrens 17

2.3.1 DBScan Basisalgorithums 17

2.3.2 Erweiterung des DBScan Algorithums durch Einbeziehung von Clusterschwerpunkten 18

2.4 Realisierung des Kalman-Filters für ein 3D-Radarsensor 22

2.4.1 Initialisierung 22

2.4.2 Die *Step()*-Methode 22

2.4.3 Optimierung der Initialisierung von Q und R für 3D-Radarsensor 23

2.4.4 Ergebnisse der Anwendung des 3D Kalman Filters 25

A Statische-Bewegung **IV**

B Rechteckige-Bewegung **V**

C Beschleunigung-Bewegung **VI**

D Geschwindigkeit-Bewegung **VII**

Kapitel 1

Kalman Filter für 1D-Radarsensor

Zur Verbesserung der Messgenauigkeit bei verschiedenen Bewegungsarten eines 1D-Radarsensors soll ein Kalman-Filter implementiert werden. Dabei werden zwei Messzustände erfasst. Der erste ist die Geschwindigkeit und der zweite der Abstand. In den folgenden Kapiteln werden zunächst die Aufgabenstellung und im Anschluss die Realisierung des Kalman-Filters vorgestellt.

1.1 Aufgabenstellung Kalman Filter für 1D-Radarsensor

In diesem Kapitel wird die Aufgabenstellung zum ersten Teil des Projektlabors erläutert. Es soll ein Kalman-Filter für einen 1D-Radarsensor implementiert werden. Mit dem Kalman-Filter soll eine Stabilität für verschiedene Bewegungsarten erreicht werden. Um das zu erzielen, sollen die Matrizen des Prozessrauschens (Q) und der Kovarianzmatrix des Messrauschens (R) systematisch an die verschiedenen Bewegungsarten angepasst werden. Dazu müssen geeignete Initialisierungswerte gefunden werden. Diese haben erheblichen Einfluss auf die Leistungsfähigkeit des Kalman-Filters. Neben der Verbesserung der Messgenauigkeit des 1D-Radarsensors soll in eineem letzten Schritt noch eine Strategie zur Erkennung der Bewegungsarten entwickelt werden, wodurch der Klaman-Filter für Echtzeitanwendungen angepasst wird.

1.2 Realisierung Kalman Filter für 1D-Radarsensor

In diesem Kapitel wird die Realisierung des Kalmanfilters für einen 1D-Radarsensor beschrieben. Für die Umsetzung wird eine Klasse Kalman mit einer *Init()*- und einer *Step()*-Methode implementiert.
Zunächst wird auf die Initialisierung der benötigten Parameter Q und R eingegangen. Im Anschluss wird die *Step()*-Methode, welche den grundlegenden Kalman-Algorithmus enthält, vorgestellt. Daraufhin wird die Optimierung der Kalman-Filter-Initialisierung beschrieben. In den letzten beiden Kapiteln werden die Ergebnisse der Optimierung für die verschidenen Bewegungsarten vorgestellt und der Kalman-Filter wird für eine Echtzeitfähigkeit bei wechselnden Bewegungsarten angepasst.

1.2.1 Initialisierung

In diesem Kapitel wird die grundlegende Initialisierung des Kalman-Filters vorgestellt. Dabei wird erläutert, wie die wichtigsten Größen initialisiert werden.
Im ersten Schitt wird die Übergangsmatrix *Phi* initialisiert, welche die Abhängigkeiten der Variablen beschreibt. Dabei wird neben dem Abstand und der Geschwindigkeit, welche gemessen werden noch die Beschleunigung eingeführt. Das wird gemacht, da es sich dabei um eine abhängige Größe handelt und somit der Kalman-Filter in seiner Performance verbessert werden kann. Da es sich dabei nicht um einen gemessenen Wert handelt, sondern um eine reine Schätzung über die Abhängigkeit, wird für die Beschleunigung ein extra Fehler e_g eingeführt. Dazu wird ein Vektor e_s aufgestellt, welcher in den ersten beiden Dimensionen mit Null initialisiert wird und in der dritten Dimension, welche für die Beschleunigung steht, mit e_g. Um die Übergangsmatrix aufzustellen, werden die Formeln 1.1 bis 1.3 definiert.

$$D[k] = D[k-1] \quad + \quad V[k-1] * t \quad + \quad \frac{1}{2} A[k-1] * t^2 \tag{1.1}$$

$$V[k] = \quad V[k-1] \quad + \quad A[k-1] * t \tag{1.2}$$

$$A[k] = \quad A[k-1] + e_g[k] \tag{1.3}$$

Die Zeitschritte Δt betragen 0,01 Sekunden. Daraus ergibt sich folgende Initialisierung für *Phi*:

$$\Phi = \begin{bmatrix} 1 & 0,01 & 0,00005 \\ 0 & 1 & 0,01 \\ 0 & 0 & 1 \end{bmatrix}$$

Neben der Initialisierung für Φ gilt es *Stat_k* und *Pred_err_k* zu initialisieren. Bei *Stat_k* handelt es sich um den Zustandsvektor, welcher vom Kalman-Filter als Schätzung des wahren Zustands angenommen wird. Er enthält die Schätzung der gemessenen Größen Geschwindigkeit und Abstand sowie der Beschleunigung und wird zu Beginn mit geschätzten Werten initialisiert.
Pred_err_k ist die Kovarianzmatrix des Prädikationsfehlers und wird mit einer 3x3 Einheits-Matrix mit sehr hohen Werten initialisiert. Die Initialisierung wird so gewählt, da zu Beginn große unkorrelierte Fehler erwartet werden.
Anschließend werden die beiden Größen Q und R initialisiert. Q steht für das Prozessrauschen und beschreibt zusätzliche Unsicherheiten im Modell, die nicht durch andere Parameter modelliert werden. Bei R handelt es sich um die Kovarianzmatrix des Messrauschens. Diese beschreibt die Abhängigkeit der Messfehler. Falls diese nicht korreliert sind ergibt sich eine Diagonal-Matrix. Die Initialisierung von R und Q haben erheblichen Einfluss auf die Leistung des Kalman-Filters. Um geeignete Werte für die Initialisierung dieser beiden Größen zu finden wird eine Funktion zur Optimierung geschrieben. Diese wird in Kapitel 2.4.4 beschrieben.

1.2.2 Die *Step()*-Methode

In der Klasse Kalman Filter gibt es eine Methode *Step()*, welche für jeden Zeitschritt t, aufgerufen wird. Dabei handelt es sich um den Algorithmus des Kalman-Filters. Der Algorithmus wird anhand des Ablaufdiagramms in Abbildung 1.1 erläutert.

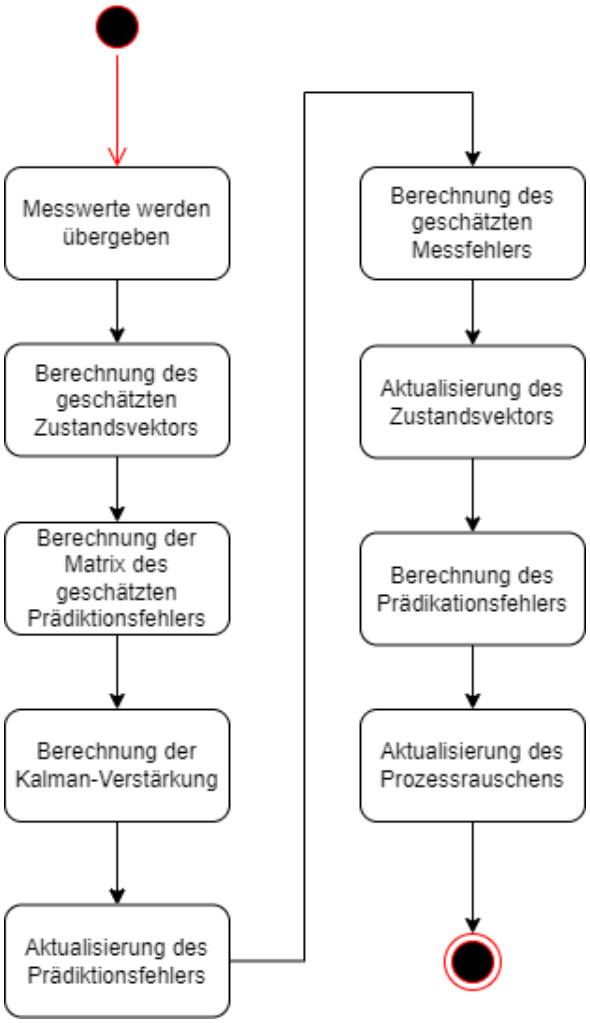


Abbildung 1.1: UML zum Kalman Filter

Bei dem Aufruf der *Step()*-Methode wird zunächst der Messvektor übergeben, welcher die aktuellen Messwerte für den Abstand und die Geschwindigkeit beeinhaltet. Im ersten Schritt wird der geschätzte Zustandsvektor berechnet. In Formel 1.4 ist zu sehen, dass dieser sich aus der Übergangsmatrix, dem vorherigen Zustandsvektor und der Schätzung des Modellfehlers errechnet. Damit wird der geschätze Zustandsvektor im ersten Schritt aktualisiert.

$$\hat{S}^P[k] = \Phi[k-1]\hat{S}[k-1] \tag{1.4}$$

Anschließend wird die Matrix des Prädikationsfehlers berechnet, wie in Formel 1.5 zu sehen ist. Dieser errechnet sich aus der Übergangsmatrix, dem Prädikationsfehler, der transponierten Übergangsmatrix und dem Prozessrauschen.

$$\hat{P}^P[k] = \Phi[k-1]\hat{P}[k-1]\Phi^T[k-1] + Q[k-1] \tag{1.5}$$

Das Kernelement des Kalman Filters, stellt die Kalman-Verstärkung dar. Diese stellt einen Faktor dar. Darin gehen der Prädikationsfehler und das Messrauschen ein. Die Kalman-Verstärkung wird verwendet, um die Korrektur für die Prädikation zu errechnen. Die Formel hierfür ist in 1.6 dargestellt. Da, um eine Division mit einer Matrix zu realisieren, die Invertierung der Matrix notwendig ist, kann es bei dieser Berechnung innerhlab des Invertierungsschritts zu einer Division durch null kommen. Um das zu vermeiden, wird statt der Inversen eine Pseudoinverse gebildet.

$$K[k] = \frac{\hat{P}^P[k]H^T[k]}{R[k] + H[k]\hat{P}^P[k]H^T[k]} \tag{1.6}$$

Anschließend wird die Schätzung des Prädikationsfehlers aktualisiert, welche in Formel 1.7 zu sehen ist. Dafür wird die Kalman-Verstärkung mit einer Verknüpfungsmatrix H, welche das Dimensionsproblem der gemessenen und geschätzten Werte löst, multipliziert und von einer Einheitsmatrix abgezogen. Damit wird die Korrektur des soeben geschätzten Prädikationsfehlers skaliert.

$$\hat{P}[k] = (I - K[k]H[k])\hat{P}^P[k-1] \tag{1.7}$$

Danach wird der Messfehler aus dem Zustand, der Messung und der Verknüpfungmatrix H berrechnet. Die Verknüpfungsmatrix wird mit der Zustandsmatrix multipliziert und von den Messwerten abgezogen, wie in Formel 1.8 dargestellt.

$$e_m^p[k] = m[k] - H[k]\hat{S}^P[k] \tag{1.8}$$

Der geschätzte Messfehler wird durch die Kalmanverstärkung verstärkt oder gedämpft und auf den geschätzten Zustansvektor aufaddiert, wie in Formel 1.9 zu sehen ist. Dieser Zustandsverktor ist der Vektor, der zum Schluss von der *Step()*-Methode zurückgegeben wird und die aktuelle Schätzung enthält.

$$\hat{S}[k] = \hat{S}^P[k] + K[k]e_m^p[k] \tag{1.9}$$

Damit ist ein Durchlauf der Step Funktion abgelaufen. Mit dem nächsten Zyklus wird der nächste Zeitpunkt t betrachtet.

1.2.3 Optimierung der Initialisierung von Q und R für 1D-Radarsensor

Wie in Kapitel xy beschrieben, ist eine gute Möglichkeit den Kalman Filter zu optimieren, eine geeignete Initialisierung von Q und R zu wählen. Um ideale Initialisierungs-Parameter für Q und R zu finden, wird ein Skript zur Automatisierung geschrieben. Dazu wird eine rekusive Funktion implementiert. Der Ablauf dieser Funktion ist in dem Ablaufdiagramm in Abbildung 1.2 dargestellt.
Zunächst wird eine Startinitialisierung für Q und R gewählt. Außerdem wird der bisher beste Fehlerwert mit einem hohen Wert initialisiert. Zusätzlich wird eine rekursive Tiefe festgelegt, über welche das Fenster, sowie die Schrittweite, mit welcher pro Durchlauf nach neuen Parametern gesucht wird, festgelegt wird.
Daraufhin wird die rekursive Funktion aufgerufen und die initialisierten Werte sowie ein generierter Datensatz für den Kalman Filter werden der Funktion übergeben. Nach dem die Schrittweite berechnet ist, werden drei Schleifen durchlaufen. Die erste ist für die optimierung von Q zuständig. Die beiden inneren Schleifen sind für die beiden Parameter auf der Diagonalen der R-Matix da. Die R-Matrix wird diagonal initialisiert, da davon ausgegangen wird, dass die Messfehler unkorreliert sind und somit sehr viel Rechenzeit gespart werden kann.

So werden die Initialisierungswerte in einem definierten Fenster durchlaufen. Für jeden der Initialisierungwerte wird ein Mean Square Error gebildet um die Abweichung der Geschwindigkeit und des Abstands zwischen der Ground Trouth und dem Output des Kalman Filters zu berechnen.
Die beiden Fehler werden gleichgewichtet zusammen addiert. An dieser Stelle wäre es möglich einen der beiden Fehler stärker zu gewichten. Interessant wäre eine Untersuchung, ob sich in bestimmten Fällen eine stärkere Gewichtung eines Parameters positiv auf die Performance auswirken kann. Denkbar wäre beispielsweise eine Geschwindigkeitsmessung, bei der die Geschwindigkeit stärker zu gewichten wäre, als die Distanz, da sie für diesen Fall mehr Relevanz enthält.
Sobald in einem Optimierungsschritt der gesamte Fehler besser ist, als der bisherige beste Fehler, werden die Initialisierungswerte als aktuell beste Parameter festgelegt und das daraus resultierende und vom Kalman Filter optimierte Ergebnis geplottet. Neben dem Plot für die Ground Trouth und der Kalman Ausgabe werden auch die Fehler für den Abstand und die Geschwindigkeit in einem Diagramm dargestellt. Für den besten Wert, der innerhalb eines Fensters gefunden wird, wird im nächsten Rekursionsschritt ein neues kleineres Fenster um diesen Wert festgelegt. Darin wird in kleineren Schritten nach besseren Werten gesucht, um fine tuning vorzunehmen.

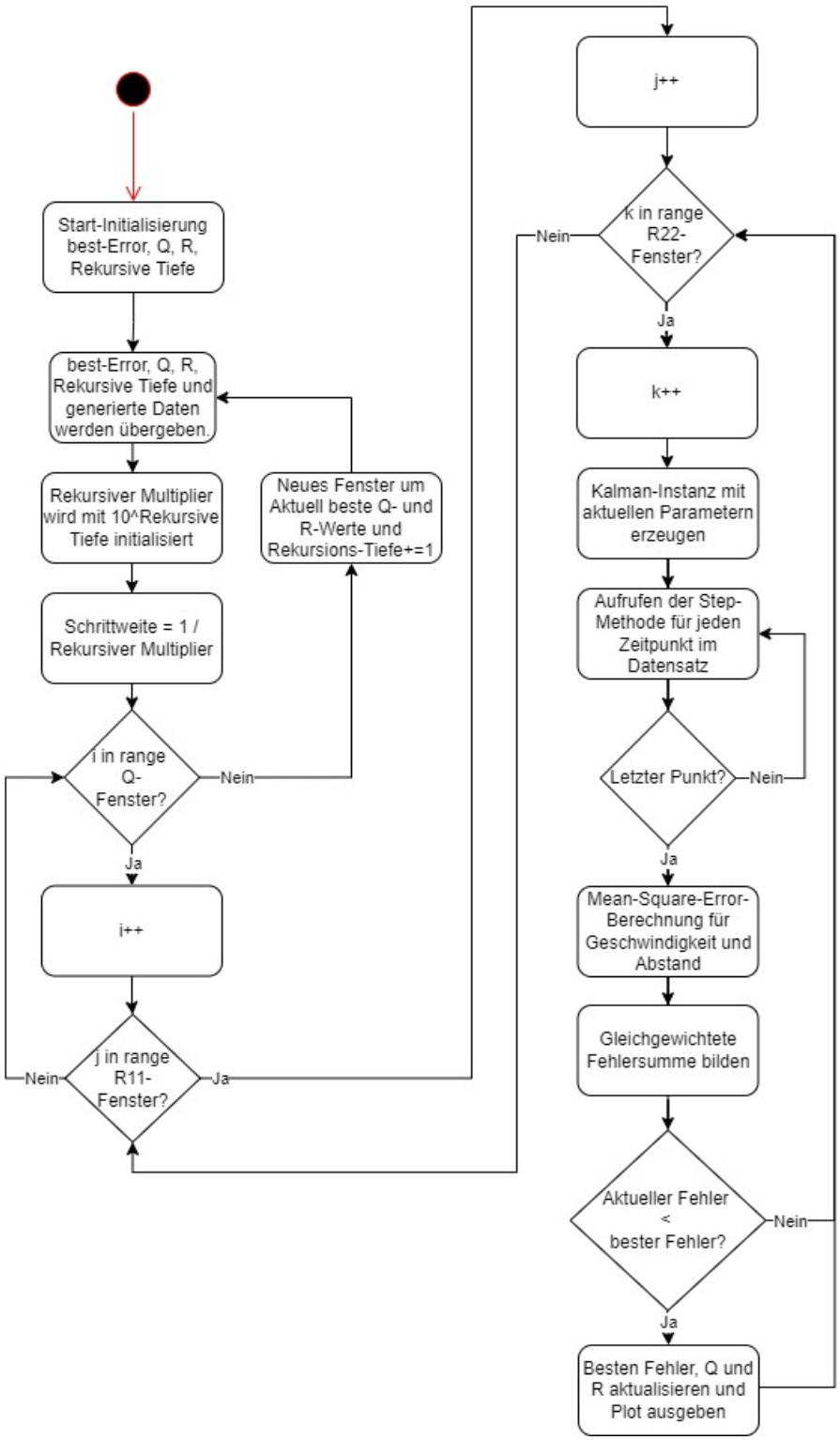


Abbildung 1.2: UML zur Kalman Filter Optimierung

1.2.4 Ergebnisse des Kalman-Filters bei verschiedenen Bewegungsarten

Der Generator für Radarsensorwerte unterscheidet zwischen fünf Bewegungstypen. Die Arten umfassen eine statische Bewegung, eine Sinusbewegung, eine Rechteckigebewegung, eine Bewegung mit konstanter Geschwindigkeit und eine Bewegung mit konstanter Beschleunigung.
Für die verschiedenen Bewegungsarten werden wie in Kapitel 1.2.3 beschrieben, jeweils die besten Initialisierungswerte für Q und die R ermittelt. Die Ergebnisse sind Tabelle 1.1 zu entnehmen. Mit dem ermittelten Wert q wird die Q-Matrix wie in Abbildung 2.5 (a) gezeigt initialisiert. Mit den Werten $r1$ und $r2$ wird in der R-Matrix, die Diagonale initialisiert. Diese ist in Abbildung 2.5 (b) dargestellt.

$$Prozessrauschen = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & q \end{bmatrix} \quad \text{(a)} \qquad Messrauschen = \begin{bmatrix} r1 & 0 \\ 0 & r2 \end{bmatrix} \quad \text{(b)}$$

Abbildung 1.3: Verwendete Parameter zum Prozessrauschen (a) und zum Messrauschen (b)

Bewegungstyp	Prozessrauschen	Messrauschen	
	q	r1	r2
Sinus	40	0.2	1
Statisch	1081.33	-93.85	120.82
Rechteckigebewegung	593.05	-0.81	4.67
Konstante Beschleunigung	1081.33	-93.85	120.82
Konstante Geschwindigkeit	885.82	-27.42	25.99

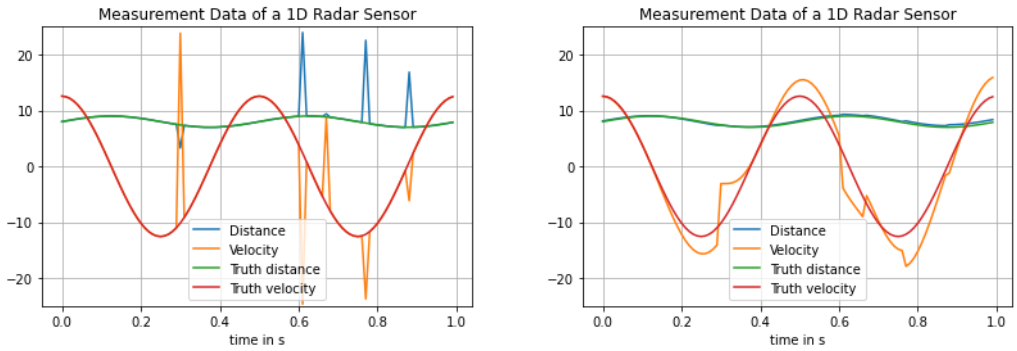
Tabelle 1.1: Initialisierungsparameter Q und R für unterschiedliche Bewegungsarten

In Tabelle 1.2 werden die Initialisierungswerte mit einem sporadic Error von fünf und einem Sporadic Error von zehn miteinander verglichen. Der Tabelle ist zu entnehmen, dass in der Regel bei mehr Sporadic Errors die Verbesserung deutlicher wird. Dies ist darauf zu führen dass durch den Mean Square Error die Peaks stärker gewichtet werden. Somit kann mit mehr Peaks in den Messwerten prozentual eine deutlichere Verbesserung durch die Kalmanfilterung erzielt werden.

Bewegungstyp	Error mit Sporadic Error 5		Error mit Sporadic Error 10	
	Vor Kalman	Nach Kalman	Vor Kalman	Nach Kalman
Sinus	23.11	7.47	40.9	9.35
Statisch	19.64	3.35	10.83	1.64
Rechteckigebewegung	24.61	8.97	19.51	9.04
Konstante Beschleunigung	11.48	2.00	27.05	2.56
Konstante Geschwindigkeit	17.74	4.44	46.37	13.07

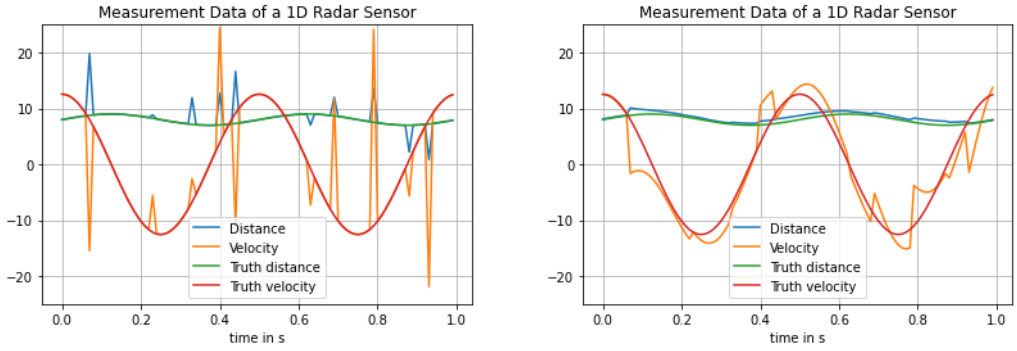
Tabelle 1.2: Vergleich zwischen Sporatic Error fünf und zehn

In den Abbildungen 1.4 (a) und (b) sind die Messwerte und Ergebnisse für eine Sinusbewegung unter Verwendung der Parameter aus Tabelle 1.1 und einem Sporadic Error von fünf dargestellt. Diese werden mit den Abbildungen 1.5 verglichen, welche ein Sporadic Error von zehn haben.



(a) (b)

Abbildung 1.4: Sinusbewegung nach und vor der Kalmanfilterung mit Sporadic Error von fünf



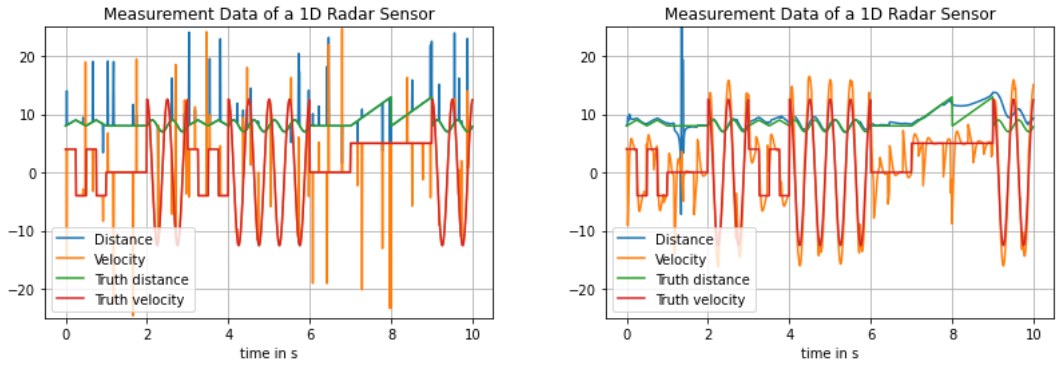
(a) (b)

Abbildung 1.5: Sinusbewegung nach und vor der Kalmanfilterung mit Sporadic Error von zehn

In Abbildung 1.5 wird im Vergleich zu Abbildung 1.4 bei mehr Peaks ein ähnlich gutes Gesamtergebnis erzielt. Allerdings ist zu beachten, dass die Verbesserung in Abbildung 1.5 (b) im Vergleich zu der in Abbildung 1.4 (b) prozentual ein deutlich besseres Ergebnis liefert. Bei einem Sporadic Error von 5 beträgt die Verbesserung nach Kalmanfilterung 209,4%. Bei einem Sporadic Error von 10 dagegen wird eine Verbesserung von 337,4% erzielt. Im Anhang können die Abbildungen zu den restlichen Bewegungsarten eingesehen werden.

1.2.5 Adaption für wechselnde Bewegungsarten in Echtzeit

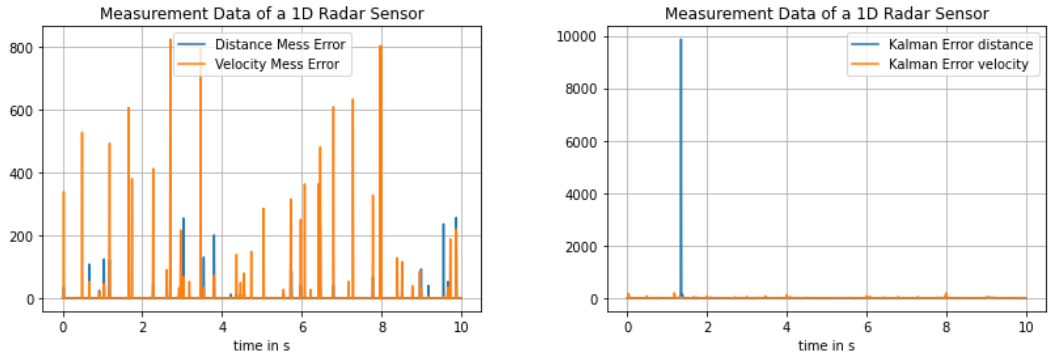
Der letzte Schritt der Implementierung des 1D-Kalman-Filters dient der Echtzeitfähigkeit bei Verwendung verschiedener Bewegungsarten. Dazu müssen die Parameter Q und R bei einem Wechsel der Bewegungsart neu initialisiert werden. Die Parameter für die verschiedenen Bewegungsarten liegen aus Kapitel 1.2.4 vor und werden in einem Dictionary in der Klasse Kalman gespeichert. Um den Wechsel der Bewegungsart zu erkennen, wird der geschätzte Messfehler über die Zeit betrachtet. Dazu wird eine Python-DQ (Double-ended queue) initialisert, in welcher die letzten fünf Messfehlerwerte gehalten werden. Aus diesen wird ein Mittelwert gebildet. Steigt dieser über einen definierten Schwellwert, wird eine Neuinitialisierung durchgeführt. Dazu wird aus dem Dictionary mit den Parametern zufällig eine Bewegungart ausgewählt, welche nicht der aktuell angenommenen entspricht und die entsprechenden Parameter für Q und R werden gesetzt. Entsprechen die gewählten Parameter nicht der aktuellen Bewegungsart, ist anzunehmen, dass es nach wenigen Schritten erneut zu einer Neuinitialisierung kommt. Die Ergebnisse dabei schwanken stark. In Abbildung 1.6 ist ein Beispiel zu sehen, bei welchem das sehr gut funktioniert.



(a) (b)

Abbildung 1.6: Messwerte (a) und Kalmanfilterung (b) für wechselnde Bewegungsarten

Allerdings tritt bei langen Messfolgen manchmal bei dem Abstand ein starker Peak auf, welcher eine falsche Neuinitialisierung veranlassen kann. Diese Peaks treten nach einer Überkorrektur des geschätzten Distanzwertes auf. Die Geschwindigkeit wird allerdings über die Bewegungsarten hinweg sehr zuverlässig geschätzt. Dabei wird in den meisten Fällen eine Verbesserung des quadratischen Fehlers der Messwerte um etwa die Hälfte erreicht. Der quadratische Fehler der Messwerte und der der Kalmanschätzung ist in den Abbildungen 1.7 (a) und (b) dargestellt.



(a) (b)

Abbildung 1.7: Quadratischer Error der Messwerte (a) und der Kalmanfilterung (b) für wechselnde Bewegungsarten

In Tabelle 1.3 sind die errechneten Fehlerwerte mit einer prozentualen Angabe zur Veränderung aufgeführt. Dabei hat sich in diesem Beispiel ein mittlerer quadratischer Fehler der Messwerte bei dem Abstand von 3,157 und bei der Geschwindikeit von 11,864 ergeben. Der mittlere quadratische Fehler der Kalmanschätzung zur Groundtruth beträgt beim Abstand 12,017 und bei der Geschwindigkeit 6,953. Der Fehler bei der Geschwindigkeit wird um etwa 41,3 % verbessert. Bei der Distanz ist der Fehler höher als bei den Messwerten. Das liegt daran, dass der quadratische Fehler des einzelnen Peaks aus Abbildung 1.7 (b) den Mittelwert des Fehlers stark anhebt. Abgesehen von diesem Peak ist in Abbildung 1.6 (b) zu sehen, dass die Filterung der falschen Messwerte gut funktioniert und die Peaks aus den Messwerten der Distanz zuverlässig entfernt werden.

Messwerte	Fehler der Messwerte	Fehler der Kalmanschätzung	Prozentuale Veränderung
Abstand	3,157	12,017	-280,64%
Geschwindigkeit	11,864	6,963	+41,31%

Tabelle 1.3: Vergleich der Messwerte und der Kalmanfilterung

Kapitel 2

Objektverfolgung mit 3D-Radarsensor

Zur Clusterbildung soll ein DBScan-Verfahren eingesetzt werden, welches bei einem 3D-Radarsensor verschiedene Objekte erkennt. Zur Verbesserung der Messgenauigkeit, soll zusätzlich ein Kalman-Filter implementiert werden.
In den folgenden Kapiteln werden zunächst die Aufgabenstellung und im Anschluss die Realisierung sowie die Ergebnisse vorgestellt.

2.1 Aufgabenstellung Objektverfolgung mit 3D-Radarsensor

In diesem Kapitel wird die Aufgabenstellung zum zweiten Teil des Projektlabors erläutert. Es soll eine Objektverfolgung eines 3D-Radarsensors realisiert werden. Dafür soll ein DBScan-Verfahren zur Clusterung verschiedener Objekte implementiert werden. Es sollen Ausreißer erkannt und als Rauschen rausgefiltert werden. Dabei soll das Modell mehrere Objekte mit unterschiedlichen Bewegungspfaden zuverlässig trennen können, auch wenn sich die Flugbahnen der Objekte treffen.
Im Anschluss an die Realisierung des DBScan-Verfahrens sollen die Messdaten durch ein Kalman-Filter angepasst werden. Dies soll zu der Verbesserung der Messergebnisse beisteuern.

2.2 Realisierung zur Rekonstruktion der Geschwindikeit aus der radialen Komponente

Ein 3D-Radarsensor liefert Messdaten zu der aktuellen Position eines Objektes und seiner radialen Geschwindigkeit. Die radiale Geschwindikeit v_{rad} stellt einen skalierten Vektor zwischen dem Radarsensor und dem Objekt dar. Die eigentliche Geschwindigkeit ist auf diesen Richtungsvektor projeziert. Um die radial gemessene Geschwindigkeit in die wahre Geschwindigkeit transformieren zu können, wird eine Funktion benötigt. Die Realisierung eines ersten Ansatzes dieser Funktion wird in diesem Kapitel beschrieben.
Da die Transformation der radialen Geschwindigkeit v_{rad} in die wahre Geschwindigkeit über das kartesische Koordinatensystem nicht praktikabel ist, wird die Transformation über Polarkoordinaten realisiert. Dazu wird in einem ersten Ansatz die Zerlegung der radialen Geschwindigkeit v_{rad} in die x-,y- und z-Komponenten in einer Funktion implementiert.

Der Funktion werden die alte und die neue Position des Objektes sowie die Position des Radarsensors und die gemessene radiale Geschwindigkeit v_{rad} übergeben. Die Position des Sensors wird jeweils von der alten Position und der neuen Position des Objektes subtrahiert. Diese Vekoren werden in ein Polarkoordinatensystem transformiert, wodurch sich je ein Θ- und ein Φ-Winkel ergibt. Die Polarkoordinaten der alten Position werden von denen der neuen Position subtrahiert, woraus sich die Winkeländerungen ergeben. Anschließend wird wieder in das kartesische Koordinatensystem rücktransformiert. Der Rücktransformierte Vektor geteilt durch Δt enthält die aus v_{rad} errechneten x-,y- und z-Komponenten der Geschwindigkeit. Allerdings ist diese Berechnungsmethode sehr fehleranfällig.

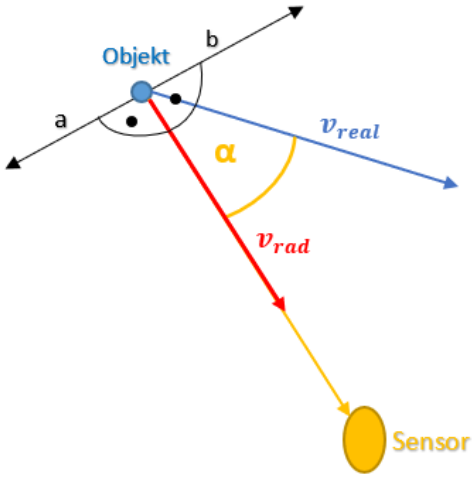


Abbildung 2.1: Aufteilung der radialen Geschwindigkeit

In Abbildung 2.1 ist schematisch die Zerlegung der Geschwindigkeit in einem 2D Modell dargestellt. Der rote Pfeil stellt die radiale Geschwindigkeit v_{rad} und der blaue die reale Geschwindikeit eines Objektes dar. Da der α-Winkel zwischen dem radialen Geschwindigkeitsvektor und dem wahren Geschwindikeitsvektor unbekannt ist, können Fehler auftreten, die abhängig von der Größe des Winkels sind. Ein kleiner Winkel führt zu einer sehr guten Zerlegung, da die reale Geschwindikeit der radialen sehr ähnlich ist. Bei großen Winkeln wird diese zunehmend schlechter, da die radiale Geschwindigkeit v_{rad} sich gegen null bewegt.
Eine radiale Geschwindikeit v_{rad} gleich null entsteht bei einer Flugrichtung senkrecht zur Ausrichtung des 3D-Sensors, wie in der Abbildung 2.1 durch die schwarzen Pfeile a und b dargestellt wird. In diesem Fall wird keine radiale Geschwindigkeit gemessen. Das Problem hierbei ist, dass der Sensor nicht unterscheiden kann in welche Richtung sich der Vektor ausrichtet und mit welcher realen Geschwindigkeit.
Außerdem ist ein weiteres Problem an diesem Ansatz, dass die Geschwindikeiten, die berechnet werden immer um Δt verschoben sind. Das bedeutet, dass die Geschwindikeit für den aktuellen Punkt über die Bewegungsrichtung vom letzten Punkt zum aktuellen hin berechnet wird.
Den wahren Geschwindigkeitsvektor exakt zu rekonstruieren ist nicht umsetzbar. Deshalb wird in einem zweiten Ansatz versucht die Geschwindigkeit im DBScan über die Clusterzentren anzunähren. Auf diesen Ansatz wird in Kapitel 2.3.2 näher eingegangen.

2.3 Realisierung des DBScan-Verfahrens

Um mehrere Objekte im dreidimensionalen Raum voneinander unterscheiden zu können, werden die Datenpunkte, bevor sie dem Kalmanfilter übergeben werden, mit dem DBScan Verfahren geclustert. Um die Datenpunkte clustern zu können ist es nicht nötig die Gesamtanzahl der Cluster beziehungsweise der Flugobjekte zu kennen. Dabei werden zwei Ansätze getestet. Im ersten Unterkapitel wird der Basisalgorithmus des DBScan beschrieben. In Unterkapitel 2.3.2 wird eine Erweiterung des DBScan-Algorithmus vorgestellt, in welcher aus aktiven Punkten ein Clusterschwerpunkt gebildet und über diese die Clusterentscheidung getroffen wird. Außerdem wird hierbei die Geschwindigkeit über die Clusterzentren berechnet.

2.3.1 DBScan Basisalgorithums

In diesem Kapitel wird der grundlegende DBScan Algorithmus vorgestellt. Für die Realisierung des DBScan-Verfahrens wird ein Script implementiert. Zunächst werden alle Punkte eines Datensatzes in einer Liste dem DBScan übergeben. Zu Beginn sind

alle Punkte als unbetrachtet anzusehen. Zusätzlich wird dem DBScan-Algorithmus ein ε übergeben, welches den Mindestabstand eines Punktes zu dem nächsten Punkt eines Clusters definiert. Der dritte Parameter, welcher dem Algorithmus übergeben wird gibt an, wie viele Punkte mindestens nötig sind, um ein Cluster zu bilden.
Zu Beginn des Algorithmus wird ein Container erzeugt, in welchem die Cluster gesammelt werden. Danach werden in einer Schleife alle übergebenen Punkte nacheinander betrachtet. Ein Punkt, welcher betrachtet wird, wird als solcher markiert. Für diesen Punkt werden aus den bisher unbetrachteten Punkten alle Nachbarn ausgewählt, die einen kleineren Abstand als ε haben. Diese werden in einer Liste gesammelt. Wenn die Liste nicht die Mindestanzahl der Punkte für ein Cluster enthält, wird der aktuelle Punkt als Rauschen markiert. Enthält diese Liste mehr Punkte als die konfigurierte Mindestanzahl der Clusterpunkte, wird aus diesen Punkten ein Cluster gebildet und für den aktuell betrachteten Punkt dieses Cluster gesetzt. Anschließend wird das Cluster dem anfangs erzeugten Container hinzugefügt.
Für alle benachbarten Punkte des aktuell betrachteten Punktes wird daraufhin eine Funktion aufgerufen mit der das Cluster rekursiv erweitert wird. Innerhalb dieser Funktion werden erneut Nachbarpunkte aus den bisher unbetrachteten Punkten gesucht. Falls diese einen kleineren Abstand als ε haben werden sie dem Cluster hinzugefügt.
Ein Problem hierbei ist, dass sich überschneidene Cluster möglicherweise nicht als verschiedene Cluster detektiert werden. Deshalb wird der DBScan-Algorithmus in einem zweiten Ansatz angepasst.

2.3.2 Erweiterung des DBScan Algorithums durch Einbeziehung von Clusterschwerpunkten

In einem zweiten Ansatz soll der DBScan-Algorithmus angepasst werden, um eine exaktere Schätzung der Geschwindigkeit und eine zuverlässigere Detektion von sich überschneidenden Clustern zu ermöglichen. Um dies zu erreichen wird für den neuen Ansatz für den DBScan eine Klasse implementiert, welche die aktiven Punkte, sowie die geclusterten Punkte hält. Der Ablauf wird anhand des Ablaufdiagramms in Abbildung 2.2 dargestellt.
Bei diesem Ansatz werden nur die neusten Punkte betrachtet. Diese werden als aktive Punkte bezeichnet. Die neu hinzugefügten Punkte werden nicht mehr in eine Liste geschrieben, sondern durch eine Python-DQ mit FIFO-Prinzip für die aktiven Punkte geschleust. Die DQ hat eine maximale Länge von 200.
Mithilfe von Clusterzentren die aus den aktiven Punkten bestimmt werden, können die Clusterschwerpunkte berechnet werden. In diese Berechnung fließen die sechs aktuellsten aktiven Punkte des jeweiligen Clusters ein. Um diese Punkte zwischenzuspeichern werden ebenfalls DQs mit einer maximalen Länge von sechs verwendet.

Außerdem wird innerhalb der Klasse ein Dictionary erstellt, indem die Clusterzentren im Verlauf der Zeit gespeichert werden. Neben der Anzahl der Punkte, die in das Clusterzentrum eingehen, werden wie in Ansatz eins die Mindestanzahl der Punkte, die für ein Cluster benötigt werden sowie ein ε initialisiert. Anders als bei dem ersten Ansatz werden hier keine ganze Datensätze auf einmal übergeben, sondern um Echtzeitfähigkeit zu ermöglichen, die einzelnen Datenpunkte kontinierlich eingespeist.
Für jeden neu hinzugefügten Datenpunkt werden in den aktiven Punkten Nachbarn gesucht. Es wird geprüft ob einer der Nachbarn schon einem Cluster zugeordnet wurde.

Sollte der Nachbarspunkt bereits in einem Cluster sein, wird das alte Clusterzentrum aus den aktiven Punkten ermittelt und der aktuelle Datenpunkt wird in die DQ der aktiven Punkte des Clusters geschoben. Das neue Clusterzentrum wird aus der DQ bestimmt und die Geschwindigkeit wird aus dem neuen und altem Clusterzentrum berechnet und aktualisiert.
Ist keiner von den Nachbarn bereits gecluster und die Anzahl der Nachbarn ist größer, als die Mindestanzahl der Punkte für ein Cluster, wird ein neues Cluster erzeugt und dieser Punkt und seine Nachbarn dem Cluster hinzugefügt. Das neue Clusterzentrum wird im Anschluss aus den aktiven Punkten bestimmt.
Sobald ein Punkt einem Cluster hinzugefügt wird, wird im Anschluss das berechnete Clusterzentrum aktualisiert und die Geschwindigkeiten werden in einem Container gespeichert.
Schlussendlich wird für jeden aktiven Punkt im Cluster überprüft, ob dieser noch in den aktiven Punkten vorhanden ist. Wenn dies nicht der Fall ist, werden er aus der DQ der aktiven Punkte des Clusters gelöscht. Im nächsten Schritt kann der nächste Datenpunkt übergeben werden. Sind sie noch aktiv müssen sie nicht gelöscht werden und der nächste Datenpunkt kann übergeben werden. Ist kein Datenpunkt mehr vorhanden endet der Algorithmus. So werden sich überschneidende Cluster nicht zusammengefasst, sondern sauber getrennt.

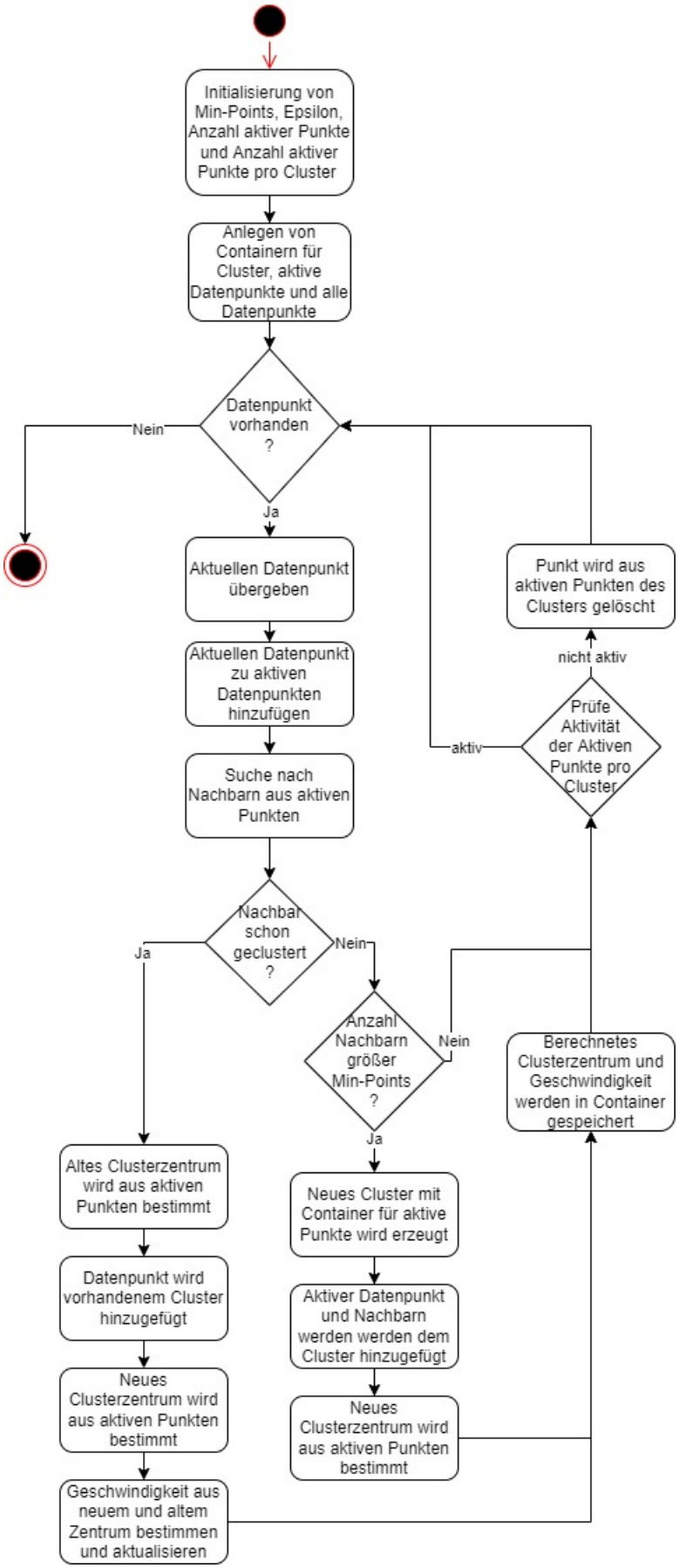
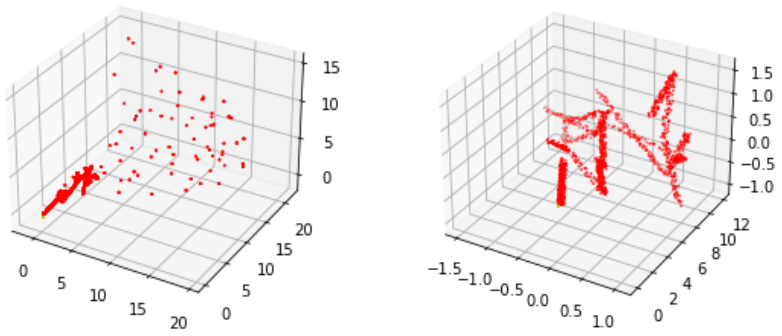


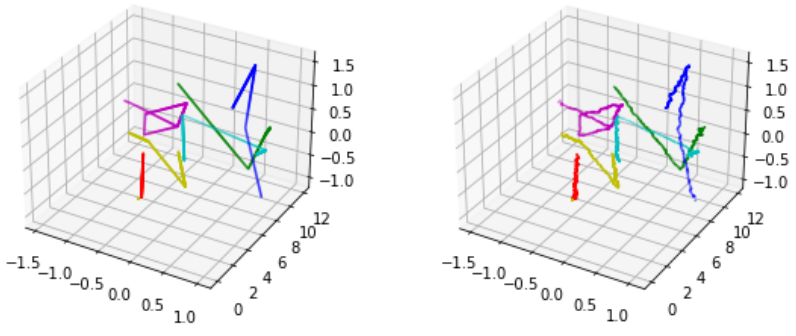
Abbildung 2.2: UML zum DBScan mit Clusterzentren

Die Ergebnisse des Algorithmus werden in Abbildungen 2.3 und 2.4 dargestellt. In Abbildung 2.3 (a) sind alle Datenpunkte aus dem Generator dargestellt und in (b) werden die Fehldetektionen nicht mit abgebildet. Darauf folgen die Abbildungen 2.4 (a) und (b), bei denen die Ground Truth und die anschließend geclusterten Fluglinien als Clusterzentren abgebildet sind. Um die Ergebnise während eines laufenden Echtzeitbetriebs darzustellen wird ein Video erstellt. Ein Beispielvideo ist unter der URL `https://www.youtube.com/watch?v=W2xaC0-6zTc` verfügbar. Anhand der Ergebnisse ist erkennbar, dass durch die Mittelwertbildung, welche bei der Berechnung der Clusterzentern erfolgt, eine deutliche Verbesserung der Genauigkeit der weiterzuverwendenden Datenpunkte erreicht wird. Anhand des Videos lässt sich außerdem erkennen, dass der Algorithmus in der Lage ist Objekte außeinander zu halten, selbst wenn ihre Pfade sich überschneiden. Auch wenn die Objekte sich gleichzeitig in einem Punkt treffen, kommt es nur vereinzelt zu falschen Zuordnungen der Datenpunkte.



(a) (b)

Abbildung 2.3: Alle Datenpunkte (a) und Datenpunkte ohne Fehldetektionen (b)



(a) (b)

Abbildung 2.4: Ground Truth (a) und berechnete Clusterzentren (b)

2.4 Realisierung des Kalman-Filters für ein 3D-Radarsensor

Im folgenden Kapitel wird die Anpassungen des Kalman Filters erläutert. Von dem im Kapitel 1.2 beschriebenen Kalman Filter wird die *Step()*-Funktion minimal geändert. Die Initialisierung muss von dem 1D- in ein 3D-Modell überführt werden. Das betrifft vor allem die Dimensionierung.

2.4.1 Initialisierung

Die Zustandsmatrix wird auf eine 3x3 Matrix erweitert und initialisiert. Sie enthält die Schätzung der gemessenen Größen Geschwindigkeit und Abstand sowie der Beschleunigung in x-, y-, und z-Richtung und wird mit geschätzten Werten initialisiert. Anders als beim 1D-Radarsensor wird nicht ein Zeitschritt Δt von 0,01 Sekunden sondern durch die Messrate von 30 Hz aus dem Generator ein Zeitschritt Δt von $0{,}0\bar{3}$ Sekunden gewählt. Daraus ergibt sich eine neue Initialisierung für die Übergangsmatrix Φ.

$$\Phi = \begin{bmatrix} 1 & 0,033 & 0,00056 \\ 0 & 1 & 0,033 \\ 0 & 0 & 1 \end{bmatrix}$$

Für die Verknüpfungsmatrix H, den Messerror e_m und den Schätzerror e_s für die Beschleunigung, werden 3x3 Matrixen initialisiert. Das Prozessrauschen wird mit dem Schätzerror und dessen Transponierter Matrix gesetzt.

2.4.2 Die *Step()*-Methode

Die *Step()*-Funktion erhält als Parameter die Werte des aktuellen Clusterzentrums sowie die errechnete Geschwindigkeit. Ansonsten wird die Berechnung wie in Kapitel 1.2.2 durchgeführt, wobei die Matrizen wie in Kapitel 2.4.1 beschrieben erweitert werden, um nun die Parameter für den dreidimensionalen Raum zu repräsentieren.

2.4.3 Optimierung der Initialisierung von Q und R für 3D-Radarsensor

Die für die Initialisierung relevanten Matrizen Q und R enthalten im dreidimensionalen Fall wesentlich mehr Einzelwerte als in Abbildung 1.2.3. Da die Berechnungszeit mit dem Algorithmus in Abbildung 1.2 exponentiell mit der Anzahl der zu optimierenden Paramter ansteigt ist dieser Algorithmus für die stark gestiegene Anzahl der Parameter nicht mehr praktikabel.
Aus diesem Grund wird der Algorithmus, wie in Abbildung 2.6 dargestellt, angepasst. In diesem Ansatz werden in der rekursiven Funktion nicht mehr alle möglichen Wertekombiniationen in einer Fensterbreite um die bisher besten Parameter ausprobiert. Stattdessen werden den bisher besten Parametern in jedem Schritt zufällige Werte in einem bestimmten Fenster aufaddiert und mit diesen Parametern der Kalman Filter initialisiert. Die Breite des Fensters wird dabei wie bisher durch die Rekursionstiefe bestimmt. Wie beim Ansatz zuvor wird bei jedem Iterationsschritt ein Errorwert berechnet. Dies wird wiederholt bis eine definierte Anzahl von Versuchen durchgeführt ist.
Die in diesem Rekursionschritt besten gefundenen Initialisierungswerte werden dann an den nächsten Rekursionsschritt übergeben. Durch diese Änderungen werden die möglichen Parameterkombinationen nicht mehr systematisch abgetastet. Dies hat zur Folge, dass die durch den Algorithmus gefundene beste Parameterkombination nicht mehr deterministisch bestimmt wird. Über die Rekursionstiefe und Anzahl der Versuche pro Rekursion kann die Rechenzeit und Genauigkeit des Algorithmus ohne Einfluss der Anzahl der zu optimierenden Parameter eingestellt werden.
Die optimale Initialisierung für Q und R, welche für den dreidimensionalen Fall gefunden wurde, ist im Folgenden dargestellt.

$$Prozessrauschen = \begin{bmatrix} -44580,0 & 21345,8 & 39199,3 \\ -34350,3 & 41257,8 & 58321,7 \\ -24745,9 & -13777,6 & 95015,6 \end{bmatrix}$$

(a)

$$Messrauschen = \begin{bmatrix} 4590,6 & 4871,9 & -8573,2 \\ 4871,9 & 67248,1 & -9083,9 \\ -8573,2 & -9083,9 & 16010,9 \end{bmatrix}$$

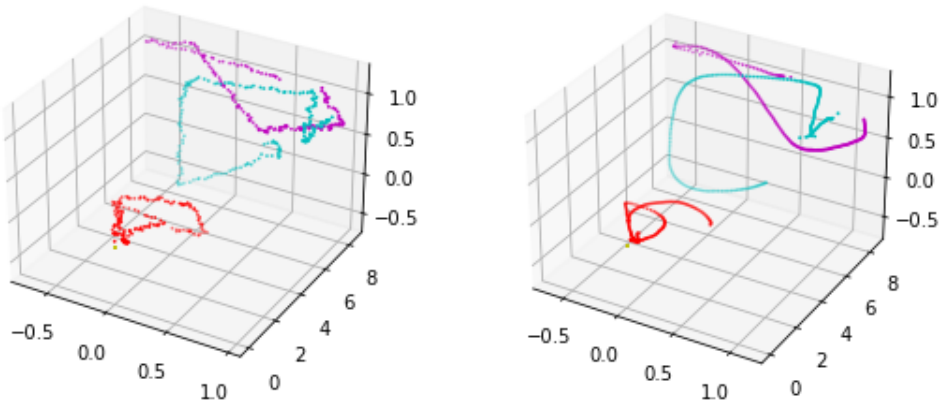
(b)

Abbildung 2.5: Verwendete Parameter zum Prozessrauschen (a) und zum Messrauschen (b)

Abbildung 2.6: Optimierung des Kalmann Filters über zufällige Parameterauswahl

2.4.4 Ergebnisse der Anwendung des 3D Kalman Filters

Nachdem wie im vorherigen Kapitel beschrieben Parameter für den Kalman Filter gefunden wurden, können diese eingesetzt werden um eine beispielhafte Filterung von bereits geclusterten Daten zu erzeugen. Diese ist in Abbildung 2.7 dargestellt. Darin ist zu erkennen, das der Filter Rauschen innerhalb der Datenpunkte weiter glättet und die scharfen Kanten innerhalb der Flugbahnen weicher macht. Um die Fähigkeit des Filters, das Rauschen zu minimiern, aufzuzeigen, wird das DBScan Verfahren für die Beispielbilder in Abbildung 2.7 mit weniger aktiven Punkten pro Cluster eingestellt, sodass die Ausgabe des DBScan Algorithmus immer noch ein erkennbares Rauschen aufweist.



(a) (b)

Abbildung 2.7: DBScan Ergebnis (a) und Ergebnis des Kalman Filters (b)

Abbildungsverzeichnis

1.1 UML zum Kalman Filter 7

1.2 UML zur Kalman Filter Optimierung 10

1.3 Verwendete Parameter zum Prozessrauschen (a) und zum Messrauschen (b) 11

1.4 Sinusbewegung nach und vor der Kalmanfilterung mit Sporadic Error von fünf 12

1.5 Sinusbewegung nach und vor der Kalmanfilterung mit Sporadic Error von zehn 12

1.6 Messwerte (a) und Kalmanfilterung (b) für wechselnde Bewegungsarten 13

1.7 Quadratischer Error der Messwerte (a) und der Kalmanfilterung (b) für wechselnde Bewegungsarten 14

2.1 Aufteilung der radialen Geschwindigkeit 16

2.2 UML zum DBScan mit Clusterzentren 20

2.3 Alle Datenpunkte (a) und Datenpunkte ohne Fehldetektionen (b) . . . 21

2.4 Ground Truth (a) und berechnete Clusterzentren (b) 21

2.5 Verwendete Parameter zum Prozessrauschen (a) und zum Messrauschen (b) 23

2.6 Optimierung des Kalmann Filters über zufällige Parameterauswahl . . 24

2.7 DBScan Ergebnis (a) und Ergebnis des Kalman Filters (b) 25

A.1 Statische Bewegung Messwerte und Ground-Truth IV

A.2 Statische Bewegung nach Kalmanfilterung IV

B.1 Rechteckige Bewegung Messwerte und Ground-Truth V

B.2 Rechteckige Bewegung nach Kalmanfilterung V

C.1 Konstante Beschleunigung Bewegung Messwerte und Ground-Truth . VI
C.2 Konstante Beschleunigung Bewegung nach Kalmanfilterung VI

D.1 Konstante Geschwindigkeit Bewegung Messwerte und Ground-Truth VII
D.2 Konstante Geschwindigkeit Bewegung nach Kalmanfilterung VII

Tabellenverzeichnis

1.1 Initialisierungsparameter Q und R für unterschiedliche Bewegungsarten 11
1.2 Vergleich zwischen Sporatic Error fünf und zehn 11
1.3 Vergleich der Messwerte und der Kalmanfilterung 14

Anhang A

Statische-Bewegung

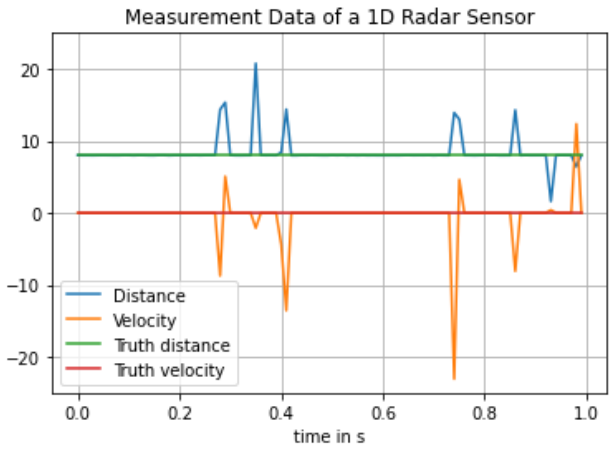


Abbildung A.1: Statische Bewegung Messwerte und Ground-Truth

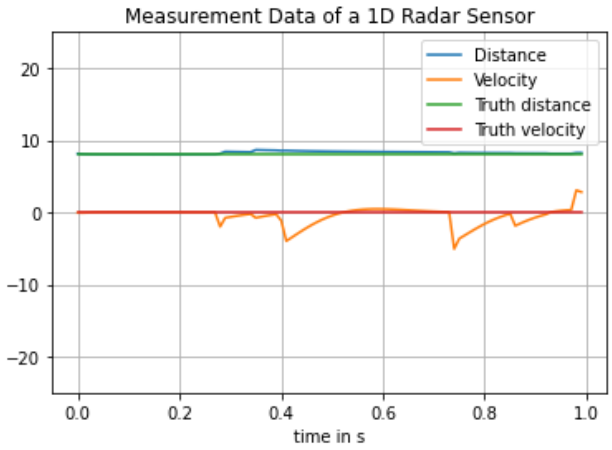


Abbildung A.2: Statische Bewegung nach Kalmanfilterung

Anhang B

Rechteckige-Bewegung

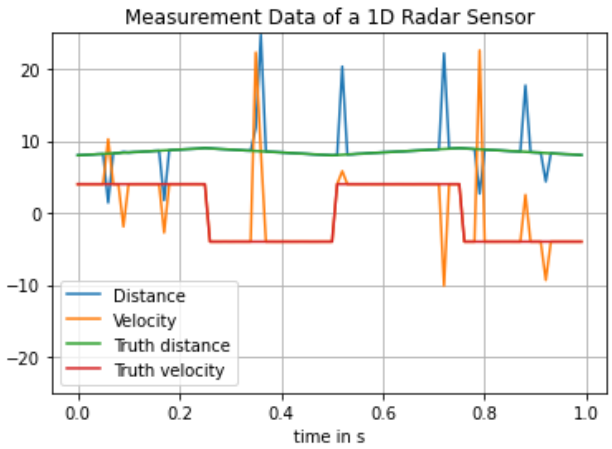


Abbildung B.1: Rechteckige Bewegung Messwerte und Ground-Truth

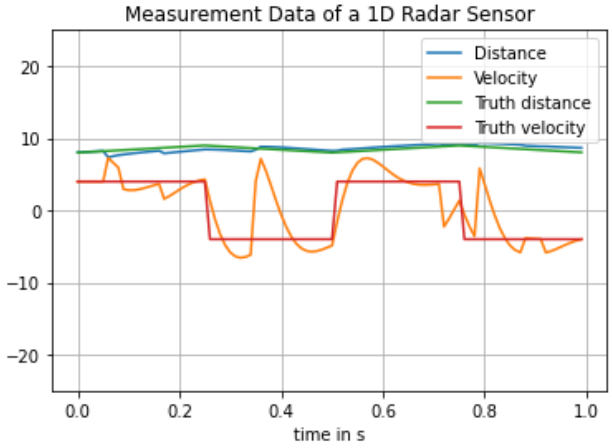


Abbildung B.2: Rechteckige Bewegung nach Kalmanfilterung

Anhang C

Beschleunigung-Bewegung

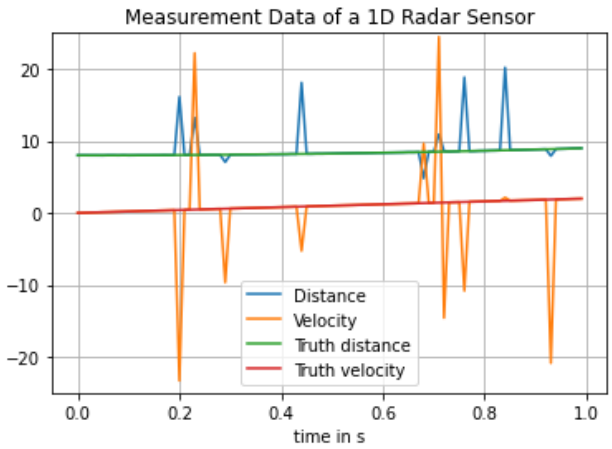


Abbildung C.1: Konstante Beschleunigung Bewegung Messwerte und Ground-Truth

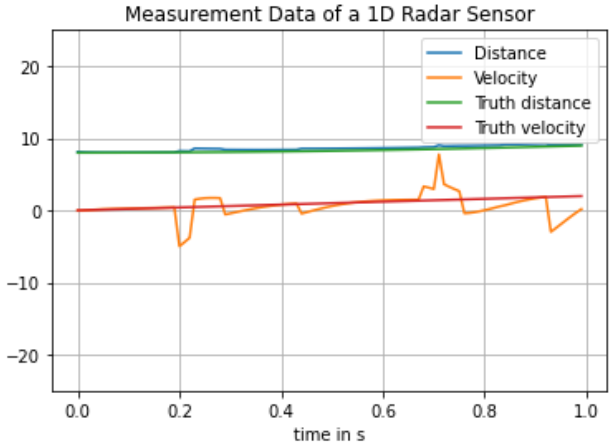


Abbildung C.2: Konstante Beschleunigung Bewegung nach Kalmanfilterung

Anhang D

Geschwindigkeit-Bewegung

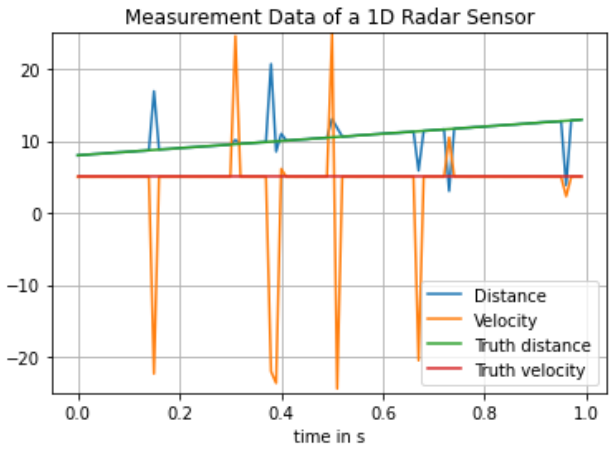


Abbildung D.1: Konstante Geschwindigkeit Bewegung Messwerte und Ground-Truth

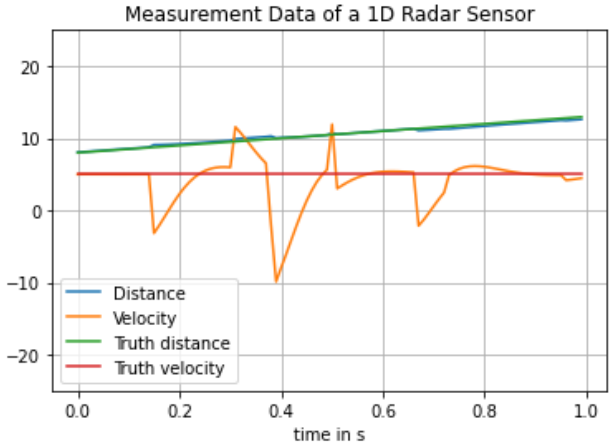


Abbildung D.2: Konstante Geschwindigkeit Bewegung nach Kalmanfilterung