Echtzeit-Zustandsschätzung von multiplen humanoiden Robotern auf Basis von Unscented Kalman Filtern

Bachelorarbeit

Jonas Kuball

Matrikelnummer: 2924454

16. Oktober 2017

Fachbereich Mathematik / Informatik
Studiengang Informatik

1. Gutachter: Dr. Tim Laue
2. Gutachter: Prof. Dr. Rolf Drechsler

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Bremen, den 16. Oktober 2017

.....

(Jonas Kuball)

INHALTSVERZEICHNIS

KAPITEL 1

HINTERGRUND

Diese Arbeit entstand im Rahmen des Roboterfußballteams B-Human (Kapitel 1.1.3). B-Human ist ein studentisches Projekt der Universität Bremen und wird in Kooperation mit dem Deutschen Zentrum für Künstliche Intelligenz (DFKI) geführt. Das Team nimmt regelmäßig an internationalen Wettbewerben des RoboCups (Kapitel 1.1) in der Standard Platform League (Kapitel 1.1.1) teil und konnte bereits sechs mal – zuletzt 2017 – den Weltmeistertitel erringen.

1.1 RoboCup

Der RoboCup ist ein 1995 vorgestellter und seit 1997 jährlich ausgetragener Wettbewerb im Roboterfußball. [5] Motivation und Ziel des Wettbewerbs ist es, ein Standardproblem für die Künstliche Intelligenz (KI) bereitzustellen, um die Forschung im selbigen Bereich voranzutreiben. Als langfristiges Ziel wird ein Spiel zwischen Robotern und Menschen angestrebt – in welchem die Roboter gewinnen. Dieses soll sich aber erst 2050 abspielen. [7]

Die Entscheidung, den RoboCup auszuarbeiten, wurde hauptsächlich auf die Kritik an den damals existenten Standardproblemen gestützt: Prominente KI-Herausforderungen wie das Schachspiel führten zwar zu mächtigen Algorithmen, abstrahieren allerdings stark von der Wirklichkeit, sodass große Schwierigkeiten der echten Welt nicht betrachtet werden.

Das Fußballspiel eignet sich aus vielerlei Gründen als Umgebung zur KI-Forschung, allem voran die Dynamik, die dadurch entsteht, dass zwei Teams nicht miteinander kompatible Ziele verfolgen. [5]

1.1.1 Standard Platform League

Schnell hat sich der RoboCup zu etwas Großem entwickelt und wurde in verschiedene Ligen aufgeteilt. Diese Bachelorarbeit ist im Rahmen der Standard Platform League (SPL) entstanden. Die SPL zeichnet sich dadurch aus, dass sie als reiner Softwarewettbewerb bezeichnet werden kann – jedes Team spielt mit dem gleichen Roboter [6] und allein die Programmierung

entscheidet. Dieser Roboter ist der *NAO*, beschrieben in Kapitel 1.1.2.

1.1.2 Der *NAO*

Der in der Standard Platform League eingesetzte Roboter ist der *NAO* (Abb. 1.1). Der *NAO* ist ein humanoider Roboter der Firma SoftBank Robotics (früher Aldebaran Robotics) und kam 2009 auf den Markt. Zum aktuellen Zeitpunkt gibt es bereits fünf überarbeitete Versionen. [8] Die aktuelle Version (V5) ist 58 cm groß und wiegt 5400 g. [9]

Dem *NAO* steht diverse Sensorik zur Verfügung, darunter zwei Kameras, vier Mikrofone und einige taktile Messgeräte. Die Kameras sitzen nicht in den Augen, sondern auf der Stirn und auf Mundhöhe. Dies ist besonders im Roboterfußball wichtig, um den Ball beim Dribbeln im Sichtfeld zu behalten.

Im Kopf hat der *NAO* eine ATOM Z530 CPU mit 1,6 Gigahertz Taktrate, einen Gigabyte Arbeitsspeicher und zwei Gigabyte Flash-Speicher. Zur Langzeitspeicherung steht eine acht Gigabyte Micro SDHC zur Verfügung.

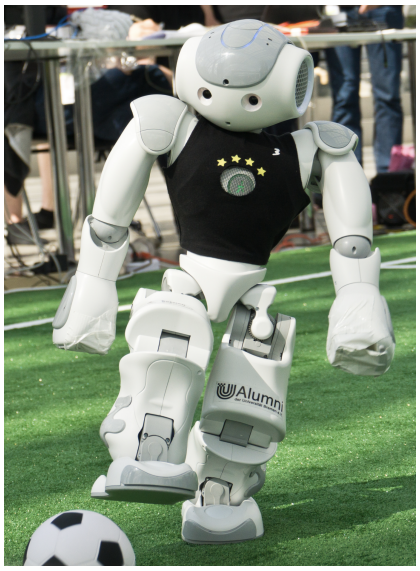


Abbildung 1.1 Der *NAO* in Aktion (RoboCup Leipzig, 2016)

1.1.3 B-Human

B-Human ist ein hauptsächlich aus Studenten bestehendes Roboterfußballteam der Universität Bremen und des Forschungsbereichs Cyber-Physical Systems des Deutschen Forschungszentrum für Künstliche Intelligenz (DFKI). [10] B-Human ist seit 2008 erfolgreich in der SPL (Kapitel 1.1.1) tätig, hat bereits sechs mal die Weltmeisterschaft gewonnen und bleibt in der RoboCup German Open sowie in der RoboCup European Open ungeschlagen.

KAPITEL **2**

EINLEITUNG

Bildverarbeitung auf dem *NAO* ist grundsätzlich ungenau, da die Kameras unter Bildrauschen leiden. Diese Ungenauigkeit wird dadurch verstärkt, dass alle Bildverarbeitungsalgorithmen selbst auf der heutzutage schwachen CPU des *NAO*s echtzeitfähig sein müssen. Verlässt man sich zu jedem Zeitschritt nur auf die aktuellen Sichtungen des Roboters, wird viel wichtige Information einfach ignoriert. Ebenfalls kann man davon ausgehen, dass sich ein gesichteter Roboter nicht allzu weit vom Fleck bewegt, wenn man für einige Sekunden wegguckt. Diese Probleme addressiert die Zustandsschätzung des Hindernismodells (Vgl. Kapitel 3.5.4).

2.1 Motivation und Ziele

B-Human's aktuelle Softwarelösung zum Errechnen des Hindernismodells ist 2014 im Zuge der Bachelorarbeit von Florian Maaß [4] enstanden und fungiert zu einigen Teilen als Inspirationsquelle des in dieser Arbeit entwickelten Lösungsansatzes.

Seit der Bachelorarbeit von Andre Mühlenbrock aus 2016 [2] erkennt B-Human in der Bildverarbeitung nicht nur die Position eines Roboters, sondern auch seine Orientierung auf dem Feld. Diese Orientierung wird aktuell nicht modelliert.

Florian Maaß benutzt den Extended Kalman Filter (EKF) zur Zustandsschätzung der Roboter. Ein bereits modellierter EKF gilt allgemeinhin als schwer erweiterbar. [1] Ein vergleichbarer Algorithmus zur Zustandsschätzung ist der Unscented Kalman Filter (UKF). (Kapitel 3.4.3)

Der UKF hat die gleichen Ein- und Ausgabeeinheiten wie der EKF – ist allerdings weitaus besser erweiterbar, was der Hauptmotivationsteil dieser Arbeit ist: Falls B-Human den UKF in Zukunft verändern will, ist es sehr einfach umsetzbar.

Ziel dieser Arbeit ist es nun, ein Modul zur Hindernismodellierung auf Basis des B-Human-Frameworks (Kapitel 3.5) zu entwickeln. Dieses Modul soll auch auf dem *NAO* echtzeitfähig sein und vergleichbare, wenn nicht sogar bessere Ergebnisse erzielen als das bisherige.

2.2 Verwandte Arbeiten

Hindernismodellierung ist kein junges Thema in der Robotik und wird prinzipiell für jeden autonomen mobilen Roboter benötigt. Die Forschung ist in diesem Bereich dementsprechend weit fortgeschritten. Für diese Arbeit gelten allerdings andere Regeln als für viele andere Roboter: Die Hardware ist beschränkt und die Teams haben keinerlei Möglichkeit zu entscheiden, welche Sensoren in den Roboter eingebaut werden. Auf Lasersensoren, Stereokameras und Vergleichbares muss verzichtet werden.

Ein möglicher und erfolgreicher Ansatz ist die Bildung einer sogenannten *Occupancy Map*. Die *Darmstadt Dribblers* haben dies bereits 2011 umgesetzt. [15] Das Spielfeld wird in ein probabilistisches Hindernisgitter aufgeteilt und durch eine Fusion von mehreren Informationsquellen gefüllt. Genutzt wird die Bildverarbeitung (Roboter-Erkennung), statisches Wissen (beispielsweise Torpfosten) und dynamisches Wissen (Daten von Teammitgliedern). Der Ansatz der *Darmstadt Dribblers* sticht insofern heraus, dass sie Bildinformation mit anderer Information kombiniert. Die Nutzung von statischem und dynamischen Wissen setzt allerdings eine korrekte Selbstlokalisierung des Roboters voraus, da das Ergebnis bei fehlerhafter Lokalisierung stark verfälscht wird.

Auch ausserhalb des Roboterfußballs werden *Occupancy Maps* erstellt. Jens-Steffen Gutmann et al. haben 2008 einen Ansatz [16] entwickelt, der es dem humanoiden Roboter *QRIO* erlaubt hat, sich sicher in seiner Umgebung zu bewegen. Im Gegensatz zum *Occupancy Grid* im Roboterfußball wird die dritte Dimension genutzt, damit der Roboter Hindernisse möglicherweise übersteigen oder unterschreiten kann. Für B-Human ist dies aber von keinerlei Relevanz, da uns die Höhe eines Hindernisses nicht interessiert.

Sehr interessant ist ebenfalls ein Ansatz von Dirk Schulz et al. aus dem Jahr 2001. [17] Ein mobiler Roboter trackt die Menschen in seiner Umgebung mithilfe eines Partikelfilters. Partikelfilter können, anders als Kalman Filter, mehrere Objekte gleichzeitig tracken und ebenfalls jede mögliche Verteilung darstellen, nicht nur Gaussverteilungen. Unglücklicherweise steigt die Komplexität von Partikelfiltern exponentiell in der Menge der zu trackenden Objekte. Aufgrund der begrenzten Rechenleistung des *NAO*-Roboters (Kapitel 1.1.2) kommt so ein Ansatz daher nicht in Frage.

KAPITEL **3**

GRUNDLAGEN

Zum Verständnis dieser Arbeit werden einige Informationen vorausgesetzt. Diese werden in den folgenden Kapiteln beschrieben, beginnend mit einigen mathematischen Grundlagen. (3.1, 3.2) Es folgen Koordinatensysteme (3.3) und Zustandsschätzung. (3.4) Abschließend werden die wichtigsten Aspekte des B-Human-Frameworks umrissen. (3.5)

3.1 Normalverteilungen

Normalverteilungen (auch Gaußverteilungen, nach Carl Friedrich Gauß) sind Wahrscheinlichkeitsdichtefunktionen. Wahrscheinlichkeitsdichtefunktionen sind integrierbare Funktionen $f : \mathbb{R} \to \mathbb{R}$ mit

$$\int_{-\infty}^{\infty} p(x)\mathrm{d}x = 1 \tag{3.1}$$

Eine Normalverteilung $\mathcal{N}$ lässt sich durch zwei Parameter darstellen, einen Erwartungswert μ und einer Varianz σ^2. Letztere wird in der Regel als quadrierte Standardabweichung σ dargestellt. Die Wahrscheinlichkeitsdichtefunktion ist wie folgt definiert. (Formel 3.2) [12]

$$\mathcal{N}\left(\mu, \sigma^2\right)(x) = \frac{1}{\sigma\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{3.2}$$

Der Fall $\mathcal{N}(0, 1)$ wird als Standardnormalverteilung bezeichnet. (Vgl. Abb. 3.1)

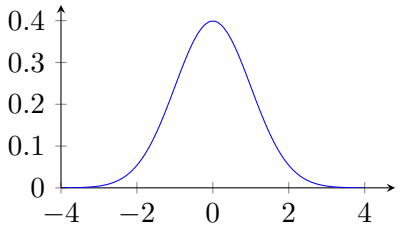


Abbildung 3.1 Die Standardnormalverteilung $\mathcal{N}(0, 1)$

3.1.1 Mehrdimensionale Normalverteilungen

Die eindimensionalen Normalverteilungen (3.1) lassen sich auf eine beliebige Anzahl von Dimensionen generalisieren. [12] Analog zur 1D-Verteilung gibt es bei der nD-Verteilung einen Erwartungswert μ, welcher in diesem Fall ein n-dimensionaler Vektor ist. Die Varianz Σ wird als $n \times n$-Kovarianzmatrix dargestellt.

Die Wahrscheinlichkeitsdichtefunktion ist wie folgt definiert. (Formel 3.3) [12]

$$\mathcal{N}_n(\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^n \det \Sigma}} \mathrm{e}^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)} \tag{3.3}$$

Im zweidimensionalen Bereich mit $\mu = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$, $\Sigma = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$ ist das die Standardnormalverteilung. (Abb. 3.2)

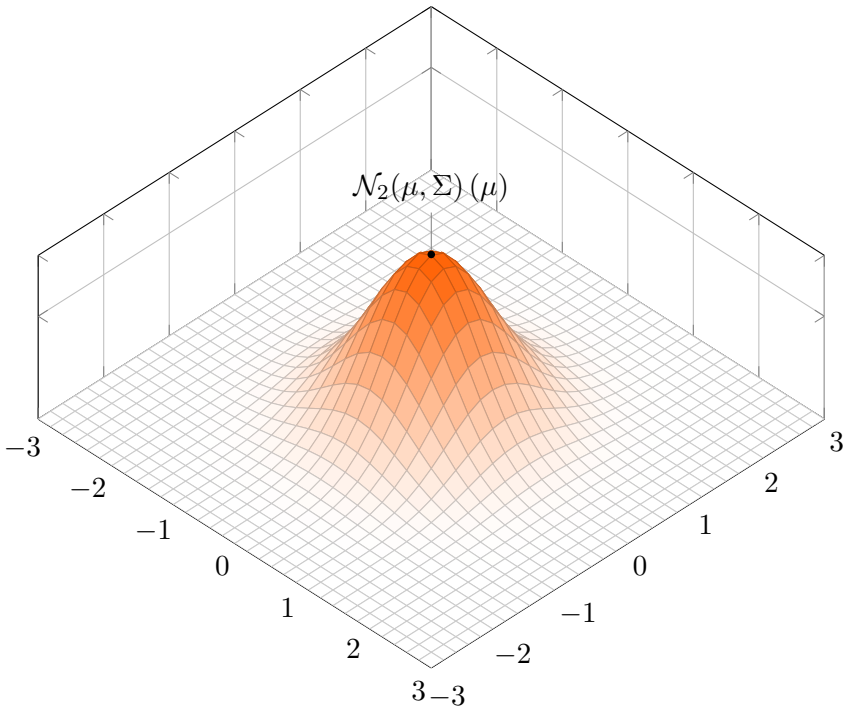


Abbildung 3.2 Die zweidimensionale Standardnormalverteilung

3.2 Gaussian Mixture Models

Als Gaussian Mixture Model (GMM) wird eine Sammlung an gleichdimensionalen Gaussverteilungen bezeichnet. Jede Gaussverteilung $\mathcal{N}(\mu_s, \Sigma_s)$ bekommt zusätzlich eine Gewichtung π_s zugeordnet, mit $\sum \pi_s = 1$. Die Wahrscheinlichkeitsdichtefunktion ist eine gewichtete Akkumulation der Wahrscheinlichkeitsdichtefunktionen der einzelnen Gaussverteilungen:

$$\mathcal{N}\left(\{\pi_s, \mu_s, \Sigma_s\}_{s=1..n}\right)(x) = \sum_{s=1}^{n} \pi_s \mathcal{N}(\mu_s, \Sigma_s)(x) \tag{3.4}$$

Um die Normalverteilung zu finden, die für einen gegebenen Punkt x die stärkste Zuständigkeit besitzt, muss die Komponente gefunden werden, bei der die gewichtete Wahrscheinlichkeitsdichtefunktion den größten Wert liefert (Häufig als `predict` bezeichnet):

$$\mathcal{N}_{\text{best}}(\mu, \Sigma) = \arg\max_s \frac{\pi_s \mathcal{N}(\mu_s, \Sigma_s)(x)}{\sum_{s=1}^{n} \pi_s \mathcal{N}(\mu_s, \Sigma_s)(x)} \tag{3.5}$$

3.3 Koordinatensysteme

Wenn wir Punkte in einer Szene beschreiben möchten, dann tun wir das in der Regel in Relation zu einem Ursprung. Diesen Ursprung bezeichnen wir als Koordinatensystem. Ein Koordinatensystem A besteht aus drei Achsen und einem Ursprung im Raum.

Die drei Achsen A_x, A_y und A_z stehen orthogonal zueinander und bestehen aus freien Vektoren der Länge 1. Der Ursprung ist ein Punkt A_W.

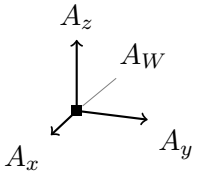


Abbildung 3.3 Ein generisches Koordinatensystem A

Einen Punkt in der Welt in Relation zum Koordinatensystem A bezeichnen wir als $p^{(\mathrm{A})}$. Für je zwei Koordinatensysteme A und B gilt, dass die Matrix $T_{\mathrm{B}\leftarrow\mathrm{A}}$ konstruiert und die Eigenschaft besitzt, Punkte in der Welt von einem Koordinatensystem in ein anderes zu transformieren.

$$p^{(\mathrm{B})} = T_{\mathrm{B}\leftarrow\mathrm{A}} \cdot p^{(\mathrm{A})} \tag{3.6}$$

Es gilt weiterhin, dass die Inverse der Transformationsmatrix die Umkehrmatrix beschreibt.

$$T_{\mathrm{B}\leftarrow\mathrm{A}}{}^{-1} = T_{\mathrm{A}\leftarrow\mathrm{B}} \tag{3.7}$$

3.3.1 Koordinatensysteme im B-Human-Framework

In dieser Arbeit werden hauptsächlich drei Koordinatensysteme verwendet: Feldkoordinaten, Roboterkoordinaten und Bildkoordinaten.

Das Feldkoordinatensystem ist am Mittelkreis des Feldes befestigt. Das Roboterkoordina-

tensystem befindet sich exakt zwischen den Füßen des Roboters und ist als Projektion des Nackengelenkes geschnitten mit der Ebene des Fussbodens zu sehen. Das Bildkoordinatensystem bezieht sich auf die Pixel im Kamerabild des Roboters und hat seinen Ursprung in der oberen linken Ecke.

Im B-Human-System sind alle benötigten Transformationsmatritzen ($T_{\text{Field}\leftarrow\text{Robot}}$, $T_{\text{Robot}\leftarrow\text{Image}}$) vorhanden und mit leichter Fehlerbehaftung korrekt kalibriert.

3.4 Zustandsschätzung

Wie in Abschnitt 2 bereits eingeleitet, kann nicht zu jedem Zeitpunkt jeder Roboter im Bild erkannt werden. Dennoch haben sie zu jedem Zeitpunkt einen Zustand, den es zu verfolgen (tracken) gilt. Dafür gibt es diverse Algorithmen zur Zustandsschätzung.

3.4.1 Das Predict-Update-Schema

Zustandsschätzungsalgorithmen sind in der Regel nach dem gleichen Schema aufgebaut, dem Predict-Update-Schema. Der Belief $bel(x_t)$ des zu trackenden Zustandes wird intern gespeichert und zu jedem Zeitschritt angepasst.

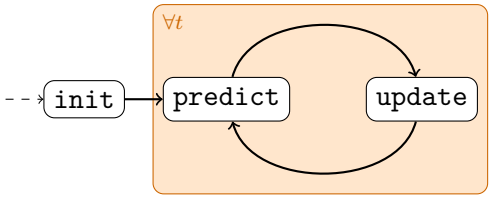


Abbildung 3.4 Das Predict-Update-Schema

i. `predict` verändert $bel(x_t)$ entsprechend eines internen Dynamikmodelles, welches die Zustandsänderung vorhersagt. Die Ungenauigkeit des Zustandes wächst.

ii. `update` wendet eine vorliegende Messung auf $bel(x_t)$ an. Die Ungenauigkeit sinkt. Wenn keine Messung vorliegt, kann dieser Schritt übersprungen werden – die Unsicherheit akkumuliert sich in diesem Fall. Hier wird zusätzlich der Kontrollfaktor angewandt, mit welchem wir eine Möglichkeit haben, Vorwissen in die Messung einfließen zu lassen. Ein Beispiel für solches Vorwissen wäre ein Signal, welches wir gleichzeitig an den Bewegungsmotor geschickt haben.

3.4.2 Kalman Filter

Der Kalman Filter ist die wohl am weitesten entwickelte Methode zur Zustandsschätzung auf Basis des Bayes Filters. [1] Er wurde 1960 von Rudolf Emil Kálmán als Technik zur Filterung und Vorhersage von linearen Gaußschen Systemen vorgestellt. [14]

Der Belief $bel(x_t)$ wird beim Kalman Filter parametrisch als (mehrdimensionale) Normalverteilung (Kapitel 3.1.1) dargestellt. Der Erwartungswert μ_t ist somit der Zustand. Die Kovarianzmatrix Σ_t quantifiziert die Ungenauigkeit der Schätzung.

Die exakte Funktionsweise des Kalman Filters wird in Abbildung 3.5 beschrieben.

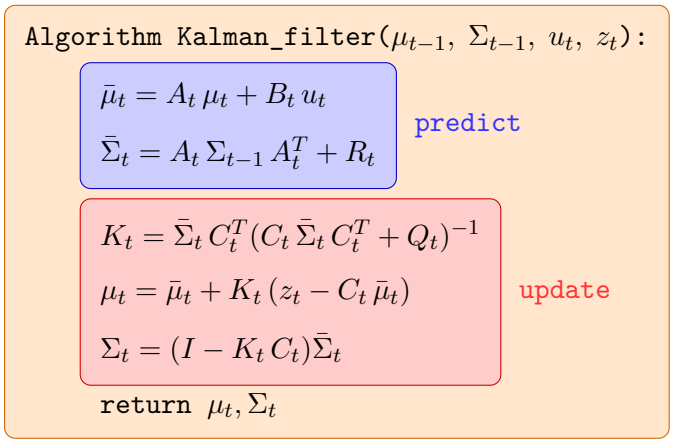


Abbildung 3.5 Der Kalman Filter in Pseudocode [1]

i. Im `predict`-Schritt wird ein Zustand per Dynamikmodell vorhergesagt. A_t und B_t sind Matritzen der Größe $n \times n$ und $n \times m$, wobei n die Länge des Zustandsvektors beschreibt und m die Länge des Kontrollvektors. Aufgrund dieser Matritzen nimmt der Kalman Filter ein lineares Dynamikmodell an. R_t beschreibt die Kovarianz einer n-dimensionalen Normalverteilung, welche die Ungenauigkeit der Vorhersage modelliert. Der Erwartungswert dieser Verteilung liegt bei 0. A_t, B_t und R_t müssen von Außen festgelegt werden und repräsentatieren domänenspezifisches Wissen.

ii. Im `update`-Schritt wird die Messung z_t angewandt. Diese setzt sich wie folgt zusammen.

$$z_t = C_t\,x_t + \delta_t$$

C_t ist eine Matrix der Größe $n \times k$, wobei k die Dimension des Messvektors z_t ist. δ_t ist ist eine Stichprobe (Sample) aus einer n-dimensionalen Normalverteilung mit dem Erwartungswert 0, welcher die Messungenauigkeit modelliert. Die Kovarianz dieser Messungenauigkeitsverteilung ist Q_t.

Der erste Zustand $bel(x_0)$ muss als valide Normalverteilung initialisiert werden.

3.4.3 Unscented Kalman Filter

Der Unscented Kalman Filter (UKF) ist eine Erweiterung des Kalman Filters (Kap. 3.4.2) und umgeht die Annahme, dass das Dynamikmodell des zu trackenden Objektes linear ist. Das Dynamik- und das Messmodell werden im UKF nicht wie im Kalman Filter durch Matritzen dargestellt, sondern als Funktionen g und h. (Abb. 3.6)

Algorithm Unscented_Kalman_filter(μ_{t-1}, Σ_{t-1}, u_t, z_t):

$$\mathcal{X}_{t-1} = \begin{pmatrix} \mu_{t-1} & \mu_{t-1} + \gamma\sqrt{\Sigma_{t-1}} & \mu_{t-1} - \gamma\sqrt{\Sigma_{t-1}} \end{pmatrix}$$

$$\bar{\mathcal{X}}_t^{*} = g(u_t, \mathcal{X}_{t-1}) \quad \text{Dynamikmodell}$$

$$\bar{\mu}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{X}}_t^{*[i]}$$

$$\bar{\Sigma}_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{X}}_t^{*[i]} - \mu_t\right)\left(\bar{\mathcal{X}}_t^{*[i]} - \mu_t\right)^T + R_t$$

predict

$$\mathcal{X}_t = \begin{pmatrix} \bar{\mu}_t & \bar{\mu}_t + \gamma\sqrt{\bar{\Sigma}_t} & \bar{\mu}_t - \gamma\sqrt{\bar{\Sigma}_t} \end{pmatrix}$$

$$\bar{\mathcal{Z}}_t = h(\bar{\mathcal{X}}_t) \quad \text{Messmodell}$$

$$\hat{z}_t = \sum_{i=0}^{2n} w_m^{[i]} \bar{\mathcal{Z}}_t^{[i]}$$

$$S_t = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)\left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)^T + Q_t$$

$$\bar{\Sigma}_t^{x,z} = \sum_{i=0}^{2n} w_c^{[i]} \left(\bar{\mathcal{X}}_t^{[i]} - \bar{\mu}_t\right)\left(\bar{\mathcal{Z}}_t^{[i]} - \hat{z}_t\right)^T$$

$$K_t = \bar{\Sigma}_t^{x,z} S_t^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t\left(z_t - \hat{z}_t\right)$$

$$\Sigma_t = \bar{\Sigma}_t - K_t S_t K_t^T$$

update

return μ_t, Σ_t

Abbildung 3.6 Der Unscented Kalman Filter in Pseudocode [1]

Der Algorithmus wendet das Dynamik- und das Messmodell an mehreren repräsentativen Stellen der Zustandsverteilung an. Diese repräsentativen Stellen sind die Sigmapunkte $\mathcal{X}$. Repräsentative Sigmapunkte einer Normalverteilung sind der Erwartungswert μ und Punkte, die sich auf jeder Dimensionsachse genau ein Sigma (eine Standardabweichung) vom Erwar-

tungswert entfernt befinden. Somit gibt es mit dem Mittelpunkt genau $2n + 1$ Sigmapunkte. (Vgl. Abb. 3.7) Dieses Verfahren nennt sich *Unscented Transform*. [1]

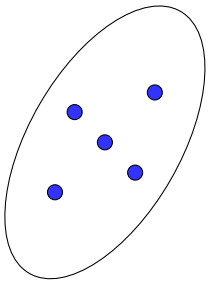


Abbildung 3.7 Beispielhafte Darstellung repräsentativer Sigmapunkte einer zweidimensionalen Normalverteilung

Die Standardabweichung lässt sich per Cholesky-Zerlegung $LL^T = \Sigma$ ermitteln. [12] Analog zum eindimensionalen Fall (der Quadratwurzel) wird diese im Algorithmus (Abb. 3.6) als $\sqrt{\Sigma}$ dargestellt.

3.5 B-Human-Framework

Das B-Human-Framework ist eine Sammlung an Werkzeugen und Software, die darauf ausgelegt ist, mit dem *NAO* Fußball in der Standard Platform League zu spielen. Dies beinhaltet Fernwartungs- und Simulationssoftware, sowie eine C++-Schnittstelle zum Entwickeln eigener Logik. [11]

Besonders relevant ist für diese Arbeit die Simulationssoftware SimRobot (Kapitel 3.5.1) und das in C++ realisierte Modulsystem (Kapitel 3.5.2).

3.5.1 SimRobot

SimRobot ist eine generische Robotersimulationssoftware, die – in erster Version – bereits 1990 entwickelt wurde. [3] SimRobot wurde bereits für viele verschiedene Robotersimulationen eingesetzt, besonders aber im Bereich RoboCup.

B-Human benutzt SimRobot als essentielles Werkzeug zum Entwickeln neuer Software. Es ist sowohl möglich, eine beliebige Anzahl an Robotern in einer 3D Umgebung zu simulierenm als sich auch mit einem realen Roboter zu verbinden, um mit ihm zur Laufzeit zu kommunizieren. Besonders bei der Fehlersuche ist dies ein sehr großer Vorteil.

Ein weiterer großer Aspekt der für diese Arbeit äußerst wichtig ist, ist die Möglichkeit *Debugzeichnungen* anzulegen, um interne Daten angemessen zu visualisieren. Ein Beispiel dafür ist in Abb. 3.9 zu sehen.

3.5.2 Modul-Framework

Das Herzstück der ganzen B-Human Software ist das Modulsystem. *Module* sind Datenverarbeitende Klassen, Daten sind *Repräsentationen*. Module können Repräsentationen anfordern und bereitstellen. Ein globaler Modulmanager ist dafür zuständig, dass die Funktionalität der Module zu jedem Zeitschritt in der richtigen Reihenfolge aufgerufen wird. Solange keine Schleifen entstehen, kann dieser Abhängigkeitsgraph beliebig komplex werden. [11] Eine starke Besonderheit dieses Systems ist es, dass zur Laufzeit Module ausgewechselt – und Parameter angepasst werden können. Das ist während eines offiziellen Spiels nicht erlaubt, während der Entwicklungszeit ist es aber extrem hilfreich.

Module sind hierbei ganz normale C++-Klassen. Für jede Repräsentation die das Modul bereitstellt, wird eine Update-Funktion vorausgesetzt, die ebenjene mit aktuellen Daten füllt. Repräsentationen sind einfache C++-Structs und können prinzipiell jedmögliches Attribut beinhalten. Die einzige Beschränkung hierbei ist eine korrekte Implementierung des Streamingoperators.

3.5.3 Architektur

Der B-Human-Prozess auf dem Roboter ist in drei Threads aufgeteilt, welche mit der Außenwelt kommunizieren. (Vgl. Abb. 3.8) Die zwei Threads, auf denen Module ausgeführt werden, sind *Cognition* und *Motion*. Der *Debug*-Thread existiert, um die anderen beiden zeitkritischen Threads nicht auszubremsen.

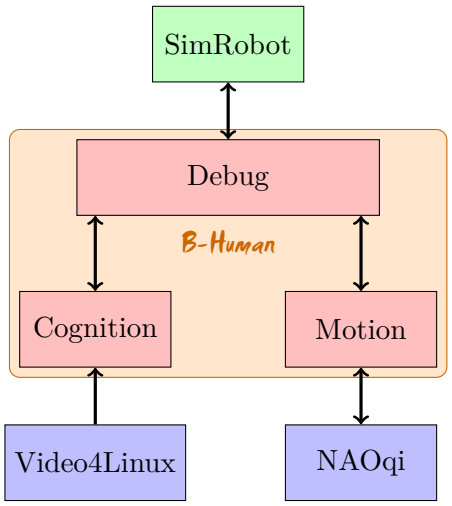


Abbildung 3.8 Die Architektur der B-Human-Software

Der Motion-Thread ist für die Kommunikation des B-Human-Systems und dem Roboterkörper zuständig. Über die von SoftBank Robotics zur Verfügung gestellte Schnittstelle NAOqi werden Sensordaten und Gelenkwinkel ausgelesen. Module, die für Gelenksteuerung zuständig sind, werden hier mit einer Wiederholungsrate von 100 Hertz ausgeführt.

Im Cognition-Thread werden Module mit einer Wiederholungsrate von 60 Hertz ausgeführt, welche Merkmale aus Kamerabildern extrahieren und welche ebenjene Merkmale über die Zeit modellieren. Ebenfalls wird das Verhalten im Cognition-Thread errechnet, auf dessen Basis der Motion-Thread dann Gelenksteuerungen errechnet.

3.5.4 Das *ObstacleModel*

Die in dieser Arbeit bereitgestellte Repräsentation ist das *ObstacleModel*. Das *ObstacleModel* realisiert einzig und allein eine Liste von Hindernissen (*Obstacles*). Hindernisse sind ebenfalls Repräsentationen. Idealerweise beinhaltet das *ObstacleModel* eine exakte Abbildung aller Hindernisse im Umgebungsbereich des Roboters. (Beispiel in Abb. 3.9).

Der Ausschnitt zeigt den internen Weltzustand eines Roboters, der kurz vor dem Tor des gegnerischen Teams steht. Links im Bild überlappen zwei Roboterzeichnungen – der tatsächlich echte Aufenthaltsort des Roboters und die von B-Human berechnete Selbstlokalisierung. In der Mitte wurde in der Roboterkamera ein Elfmeterpunkt erkannt. Rechts im Bild sieht man nun eine Ausgabe des *ObstacleModels* (Kapitel 3.5.4).

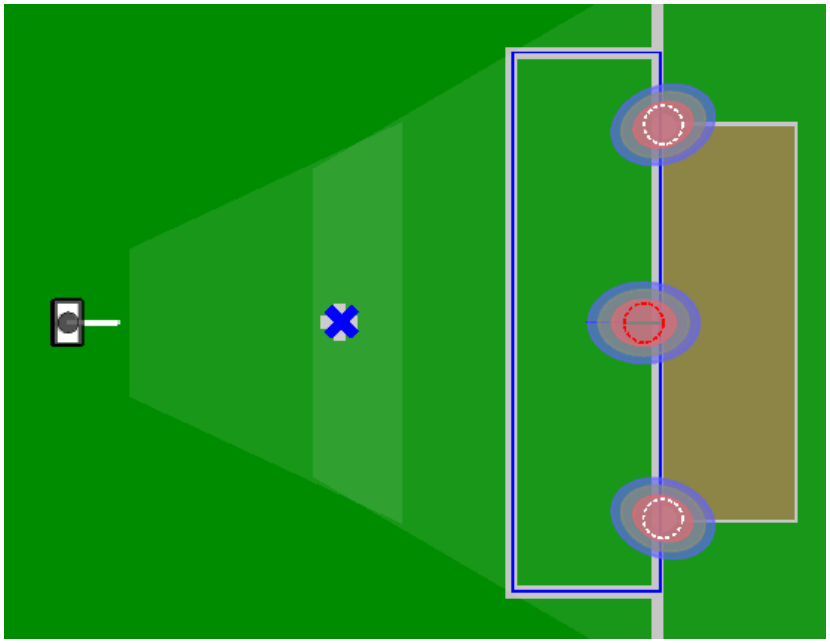


Abbildung 3.9 Visualisierung eines beinahe optimalen *ObstacleModels* vor dem Tor

Eingezeichnet sind dort die errechneten Kovarianzen der gesichteten Hindernisse (zwei Torpfosten und ein Torwart) und die erkannte Orientierung des Torwarts als grauer Strich. Die Farben der Zentren beschreiben den erkannten Typ des Hindernisses – Weiß für Torpfosten, Rot für Gegnerroboter.

Hindernisse werden intern in Roboterkoordinaten (Kap. 3.3.1) gespeichert und verarbeitet, sodass der Roboter – selbst, wenn er fehlerhaft lokalisiert ist – weiß, was in seiner Umgebung los ist.

KAPITEL 4

UMSETZUNG

4.1 Überblick

Ein einziger Ausführungszyklus des *ProbabilisticObstacleModelProviders* lässt sich auf sechs Methoden zurückführen und somit in einem einfachen Graphen (Abb. 4.1) darstellen.

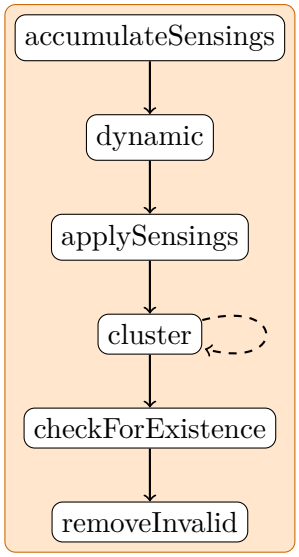


Abbildung 4.1 Übersicht eines Update-Zyklus

1. **accumulateSensings**: Zu Beginn werden neue Hindernis-Sichtungen des *PlayersPerceptors* angesammelt und relevante Information gespeichert. Das Modul ist an dieser Stelle so erweiterbar, dass andere Quellen für Messungen (Vgl. Kapitel 6) hinzugezogen werden können.
2. **dynamic**: Hier wird der Dynamikschritt (Predictschritt des UKFs) des Unscented Kalman Filters (UKF) für jedes Hindernis ausgeführt. In der ersten Iteration des Moduls existieren noch keine getrackten Hindernisse. (Kapitel 4.4)
3. **applySensings**: Die vorher akkumulierten Hindernismessungen werden nun einzeln in bereits getrackte Hindernisse eingepflegt (Updateschritt des UKFs). Die Zuordnung

wird mit von Gaussian Mixture Models (GMMs) realisiert. (Kapitel 4.5) Messungen, die zu keinem bisher existenten Hindernis zugeordnet werden können, werden als getracktes Hindernissen aufgenommen.

4. **cluster**: Es tritt durchaus der Fall auf, dass eine Messung nicht korrekt zugewiesen und dementsprechend ein neues internes Hindernis angelegt wird. Das bedeutet, dass das Selbe reale Hindernis mit zwei verschiedenen Hyphothesen beschrieben wird. In dieser Methode werden Hindernisse, die sich zu ähnlich sind, verschmolzen. (Kapitel 4.6) Dieser Schritt kann bis zur Konvergenz aufgerufen werden, um die aus der Verschmelzung entstandenen Hypothesen weiter einzuordnen.

5. **checkForExistence**: Hypothesen, die lange nicht gesehen worden sind, werden aussortiert. Das gleiche gilt für Hindernisse, die eine gewisse Anzahl an Bildern nicht gesehen worden sind, aber eigentlich im Bild sein sollten. (Kapitel 4.7)

6. **removeInvalid**: Aus Optimierungsgründen werden Hindernisse, die in den vorangegangen Methoden als aussortiert markiert worden sind, tatsächlich erst hier gelöscht.

4.2 Sichtungen

Das Bildverarbeitungssystem von B-Human liefert zu jedem Zeitschritt alle Hindernisse, die der Roboter im Bild erkennt. Das Modul, welches diese Hindernisse liefert wird allgemeinhin als *PlayersPerceptor* bezeichnet, obgleich es sich nicht auf Roboter beschränkt. Diese Hindernisse sind in der Regel Roboter, können aber auch Beine eines Menschen (bspw. die des Schiedsrichters) oder Torpfosten sein.

Der *PlayersPerceptor* ist außerdem in der Lage, die Teamzugehörigkeit und die Orientierung eines Roboters auf dem Feld zu erkennen. Diese Informationen funktionieren aber nur, wenn tatsächlich ein Roboter und kein allgemeines Hindernis erkannt wird.

Diese Sichtungen liefern wichtige Informationen, allen voran die Position auf dem Feld inklusive der Breite im Bild, dargestellt durch die Punkte im Bild, die am weitesten Außen sind. Über eine Koordinatenprojektion (Vgl. 3.3, 3.3.1) können diese Positionen in Roboterkoordinaten umgerechnet werden.

$$p^{(\text{Robot})} = T_{\text{Robot}\leftarrow\text{Image}} \cdot p^{(\text{Image})} \tag{4.1}$$

Die Koordinatenprojektion von Bild- auf Feldkoordinaten wird ungenauer, je weiter die Sichtung vom Roboter entfernt ist. Sichtungen, die eine gewisse Distanz überschreiten – im Laufe der Entwicklungsphase haben sich hier drei Meter bewährt – werden vom *ProbabilisticObstacleModelProvider* komplett ignoriert, da sie keinerlei Relevanz für den Roboter und die sichere Bewegung in seiner Umgebung haben.

4.3 Filterdesign

Wenn man einen Kalmanfilter anwenden möchte, geht allem voran die Entscheidung, welche Attribute des Objektes modelliert werden sollen. Die Entscheidung fiel auf fünf Attribute. Daraus ergibt sich dann die Zusammensetzung des Zustandsvektors, dargestellt in Abb. 4.2.

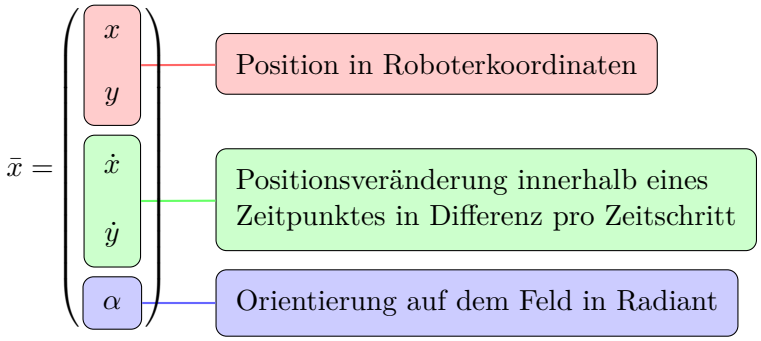


Abbildung 4.2 Aufbau des Zustandsvektors

Diese fünf Attribute lassen sich in drei Kategorien einteilen: Position, Positionsveränderung und Orientierung.

1. Die Position des Hindernisses wird in zwei Koordinaten abgespeichert. Die z-Koordinate interessiert für Umgebungsmodelierung nicht, da ein Roboter sich grundsätzlich am Boden aufhalten sollte. Die Koordinaten sind im Roboter-Koordinatensystem dargestellt. (Vgl. Kapitel 3.3)

2. Die Positionsveränderung wird ebenfalls in zwei Werten abgespeichert: Die Differenz der x- bzw. y-Koordinate in Differenz pro Zeitschritt. Aus diesen Werten lässt sich die Bewegungsrichtung und Bewegungsgeschwindigkeit ableiten. Da die Roboter omnidirektional laufen können, kann man aus diesem Richtungsvektor nicht die Orientierung des Roboters herauslesen.

3. Die Orientierung α ist ein einfacher Wert in Radiant. Die Entscheidung fiel gegen ein weiteres Attribut der Winkelveränderung $\dot{\alpha}$, da es sehr unwahrscheinlich ist, dass ein Roboter in einer Drehung betrachtet wird. Roboter laufen zwar omnidirektional – aber normalerweise schräg, ohne ihre Rotation zu verändern. Eine unnötige Vergrößerung des Zustandsvektors würde außerdem zu längeren Ausführungszeiten des Moduls führen.

4.4 Dynamik

Zu jedem Zeitschritt muss der Dynamikschritt des Kalmanfilters für jedes getrackte Hindernis durchgeführt werden. Das in dieser Arbeit eingesetzte Dynamikmodell besteht aus zwei Aufgaben.

1. Aufrechnung der Odometrie[1] des Roboters. Die Odometrie wird vom B-Human-System errechnet und bereitgestellt.
2. Anwendung der Geschwindigkeit auf die Position.

Nach dem Aufrechnen der Odometrie bildet sich nun noch folgendes Dynamikmodell:

$$\vec{p}_t = \vec{p}_{t-1} + \dot{\vec{p}}_{t-1} \cdot \Delta t \tag{4.2}$$

$$\alpha_t = \alpha_{t-1} \tag{4.3}$$

mit $\vec{p}_t = \binom{x_t}{y_t}$ und $\dot{\vec{p}}_t = \binom{\dot{x}_t}{\dot{y}_t}$. Da $\dot{\vec{p}}$ als Differenz pro Zeitschritt definiert ist, ist Δt immer 1 und kann vernachlässigt werden. Wie in Formel 4.3 zu sehen, wird keine Änderung in der Orientierung angenommen. Darüber mehr in Kapitel 6.

4.5 Zuordnung

Für jedes vom *PlayersPerceptor* gelieferte Hindernis muss eine Entscheidung getroffen werden: Gehört diese Sichtung zu einem bisher bekannten Hindernis oder war es bisher unbeachtet und muss als neues Objekt getrackt werden? Um diese Entscheidung angemessen beantworten zu können, wird sich der Mathematik der Gaussian Mixture Models (GMMs) bedient.

Jedes bereits bekannte interne Hindernis beschreibt exakt eine zweidimensionale Normalverteilung: Die Position beschreibt den Erwartungsvektor μ und die Kovarianz der Position des Hindernisses beschreibt die Kovarianz der Verteilung Σ. Alle Normalverteilungen zusammen können als GMM betrachtet werden.

Mithilfe der `predict`-Funktion (Vgl. Formel 3.5) des gebildeten GMMs lässt sich nun das Hindernis ermitteln, welches die Sichtung am besten beschreibt. Die daraus resultierende Wahrscheinlichkeit beschreibt allerdings den Zusammenhang der Sichtung zu allen anderen Hindernissen und skaliert dementsprechend antiproportional zur Anzahl ebenjener. Das macht es unmöglich, aus diesem Wert mehr Information hinauszulesen als die, zu welcher Gaussverteilung die Sichtung wahrscheinlich gehört. Dementsprechend wird diese Wahrscheinlichkeit nicht direkt genutzt, sondern nur die Gaussverteilung.

[1] Als Odometrie bezeichnet man den Versatz der Position und Orientierung des Roboters innerhalb des letzten Zeitschritts

Durch die Wahrscheinlichkeitsdichtefunktion dieser einzelnen Verteilung $\mathcal{N}_{best}$ (Vgl. 3.1) lässt sich nun ein Faktor zwischen 0 und 1 errechnen, indem wir die Wahrscheinlichkeit der Sichtung s mit dem Maximalwert normieren:

$$p\left(s \in \mathcal{N}_{best}(\mu, \Sigma)\right) = \frac{\mathcal{N}_{best}(\mu, \Sigma)\,(s)}{\mathcal{N}_{best}(\mu, \Sigma)\,(\mu)} \tag{4.4}$$

Überschreibt dieser Wert nun einen parametrisierten Schwellwert, kann sicher angenommen werden, dass die Sichtung zum Hindernis gehört. Weiterhin haben wir nun einen Faktor, mit dem wir die Sicherheit des Zusammenhangs quantifizieren können. Mit diesem Faktor wird die Abweichung Q_t für das Messmodell des Unscented Kalman Filters gewichtet.

4.6 Clustering

Für den Fall, dass die Zuordnung (4.5) fehlschlägt und zwei interne Hindernisse in der Realität das Selbe beschreiben, werden regelmäßig alle Hindernisse durchlaufen und gegebenenfalls verschmolzen. Dazu wird die Distanz zwischen allen Paaren errechnet.

Um die Distanz zwischen zwei Hindernissen quantifizieren zu können, wird eine Metrik benötigt. Die Entscheidung fiel auf die die Mahalanobis-Distanz [13], welche eine Distanz zwischen zwei Punkten – gegeben einer Kovarianzmatrix – berechnet. Die Kovarianzmatrix Σ setzt sich als Mittel von Σ_1 und Σ_2 zusammen.

$$D_M(\mu_1, \mu_2, \Sigma) = \sqrt{(\mu_1 - \mu_2)^T \cdot \Sigma^{-1} \cdot (\mu_1 - \mu_2)} \tag{4.5}$$

Paare, die nun eine gewisse Distanz unterschreiten und bei denen mindestens ein Teil sehr jung ist, werden per quadratischer Ausgleichsrechnung [12] zusammengefügt.

Dieser Schritt kann bis zur Konvergenz aufgerufen werden. Darauf kann aber verzichtet werden, um Rechenzeit einzusparen.

4.7 Existenzanalyse

Schlussendlich müssen Elemente aussortiert werden, die vermutlich nicht mehr im Umgebungsbereich des Roboters existieren oder insgesamt nur Fehlmessungen waren. Ein Hindernis wird genau dann gelöscht, wenn eine der folgenden drei Kriterien erfüllt wird:

1. Das Hindernis hat sich durch das Dynamikmodell oder durch die Odometrie des Roboters aus der effektiven Reichweite des Moduls bewegt

2. Das Hindernis wurde eine fest definierte Zeitspanne nicht mehr gesehen

3. Das Hindernis wurde eine fest definierte Zeitspanne (kürzer als bei Punkt 2.) nicht mehr gesehen, obwohl es im Bild hätte zu sehen sein müssen

Ein Objekt sollte genau dann zu sehen sein, wenn das Ergebnis einer Koordinatenprojektion (Vgl. 3.3, 3.3.1) noch im Bild liegt:

$$p^{(\text{Image})} = T_{\text{Image}\leftarrow\text{Robot}} \cdot p^{(\text{Robot})} \tag{4.6}$$

Die Implementierung von Verdeckung eines Roboters durch einen Anderen erscheint nicht zweckgemäss. Ein Roboter kann problemlos vernachlässigt werden, falls er hinter einem anderen zu sein scheint, da er für den Nahbereich nicht mehr relevant ist. Sobald der verdeckte Roboter wieder zu sehen ist, beginnt die Zustandsschätzung ein weiteres Mal.

KAPITEL **5**

EVALUATION

Zur Auswertung habe wurden mehrere Experimente durchgeführt, um das Modul bewerten zu können – auch im direkten Vergleich zum alten *ObstacleModelProvider* aus Florian Maaß' Bachelorarbeit. [4]

5.1 Experiment 1 – Ausführungsgeschwindigkeiten

Ein großes Ziel dieser Arbeit ist die Echtzeitfähigkeit auf dem *NAO*. Das folgende Experiment soll dies sicherstellen.

i. **Versuchsaufbau**: Ein Roboter wird auf dem Mittelkreis platziert und schaut in Richtung Tor. Exakt einen Meter vor dem Roboter wird ein Hindernis aufgestellt; in diesem Fall ein weiterer Roboter. Es wird sichergestellt, dass der Versuchsroboter dieses Hindernis als ebensolches erkennt. Beide Module werden nun exakt eine Minute (3600 Zyklen) lang ausgeführt und Ausführungsgeschwindigkeiten gemessen.

ii. **Ergebnisse**: Das in dieser Arbeit entstandene Modul ist nur marginal langsamer als das bisher Eingesetzte. (Abbildung 5.1) Unter den Voraussetzungen dieses Experimentes ist ein Anstieg von 30% zusätzlicher Ausführungszeit zu verbuchen.

Modul	**min.**	**max.**	**avg.**
Alt	0,047	0,205	0,068
Neu	0,059	0,213	0,092

Abbildung 5.1 Ausführungsgeschwindigkeiten von Experiment 1 (in ms)

iii. **Auswertung & Fazit**: Die Echtzeitfähigkeit ist immer noch gewährleistet. Der Gewschwindigkeitsverlust ist zu verkraften, da das alte Modul eine Teilmenge des neuen Moduls ist.

5.2 Experiment 2 - Positionsgenauigkeit

Neben der Echtzeitfähigkeit muss ebenfalls die Genauigkeit des Trackings ausgewertet werden. Um eine Metrik zur Bewertung der Genauigkeit zu haben, werden die folgenden Experimente in einer Simulation dargestellt. Damit ist zu jedem Zeitpunkt die Wirklichkeit (*Ground Truth*) bekannt.

5.2.1 Experiment 2.1 - Statischer Roboter, mobiles Hindernis

Zuerst wird evaluiert, wie gut mobile Hindernisse getrackt werden, ohne, dass sich der trackende Roboter bewegt.

i. **Versuchsaufbau**: Ein Roboter wird auf dem Mittelkreis platziert und schaut in Richtung Tor. Exakt zwei Meter vor ihm läuft ein anderer Roboter mit konstanter Geschwindigkeit durch das Bild. Es wird die Distanz zur *Ground Truth* gemessen. Der messende *NAO* bewegt sich nicht.

ii. **Ergebnisse**: Die Messungen lassen sich in Abb. 5.2 und Abb. 5.3 ablesen. Beide beschreiben die gleichen Daten, letztere zeigt einen detaillierten Ausschnitt.

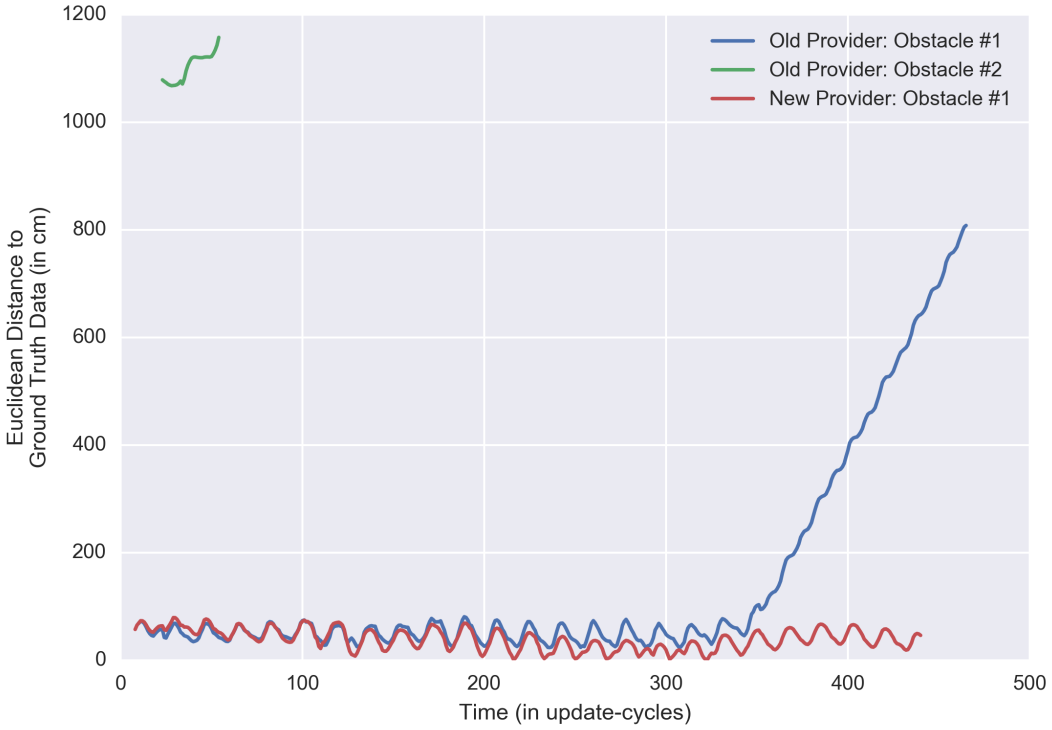


Abbildung 5.2 Auswertung von Experiment 2.1

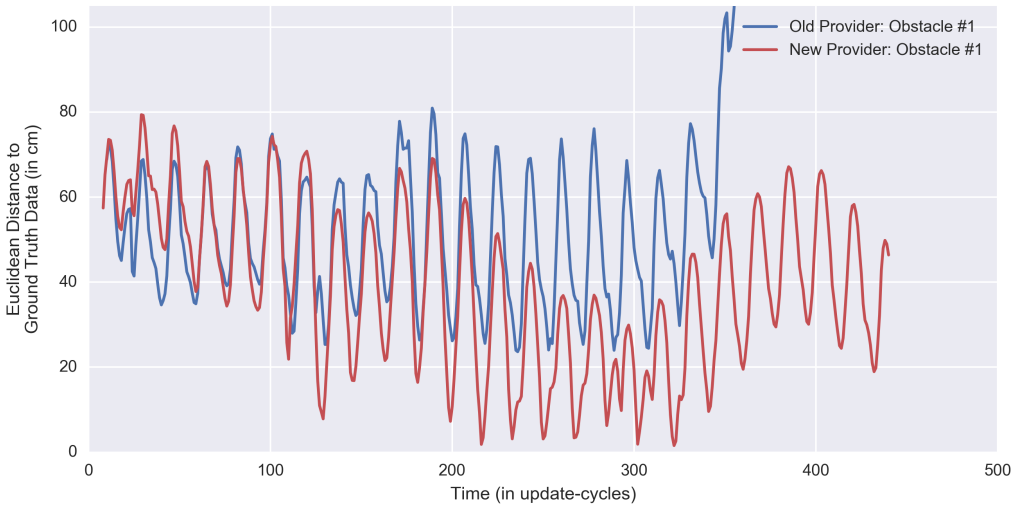
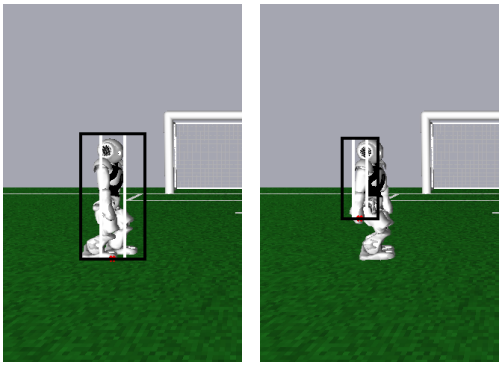


Abbildung 5.3 Auswertung von Experiment 2.1 im Detail

iii. **Auswertung & Fazit**: In Abb. 5.2 lassen sich zwei Eigenheiten ablesen, die das in dieser Arbeit entwickelte „neue“ Modul dem „alten“ Modul voraus hat.

Obgleich nur ein einzelnes Hindernis zu tracken war, hat der alte *ObstacleModelProvider* kurzzeitig ein zweites Hindernis modelliert. Dies lässt sich durch einen Fehler in der Bildverarbeitung erklären. (Abb. 5.4) Transformiert man die Bildkoordinaten des fehlerhaft erkannten Roboters in Feldkoordinaten, so liegen diese weit von der *Ground Truth* entfernt. Das Nichtauftauchen dieses Fehlers im neuen Modul ist darauf zurückzuführen, dass ein Hindernis erst als Hindernis betrachtet wird, wenn es mehrere Bilder hintereinander erkannt wird.

Ebenfalls in Abb. 5.2 zu sehen ist ein starker Fehler ab etwa 350 Zyklen. Dieser Umstand ist damit zu erklären, dass der getrackte Roboter das Bild verlassen hat und somit die Position des Hindernisses allein vom Dynamikmodell des EKF bzw. des UKF abhängt. Hier erreicht das „neue“ Modul eine weit bessere Leistung.



(a) Korrekt **(b)** Fehlerhaft

Abbildung 5.4 Fehler in der Robotererkennung

5.2.2 Experiment 2.2 - Mobiler Roboter, statisches Hindernis

Nun muss ausgewertet werden, wie sich das Modul bei einem laufenden Roboter verhält.

i. **Versuchsaufbau**: Zwei Roboter werden sich mit einem Abstand von drei Metern gegenüber gestellt. Einer der beiden Roboter fungiert als Hindernis, der andere trackt ebenjenes. Der trackende Roboter wird nun – um Kollisionen zu vermeiden – einen halben Meter weiter zur Seite bewegt und läuft nun geradeaus, bis das Hindernis nicht mehr im Blick ist.

ii. **Ergebnisse**: Die Messungen lassen sich in Abb. 5.5 ablesen.

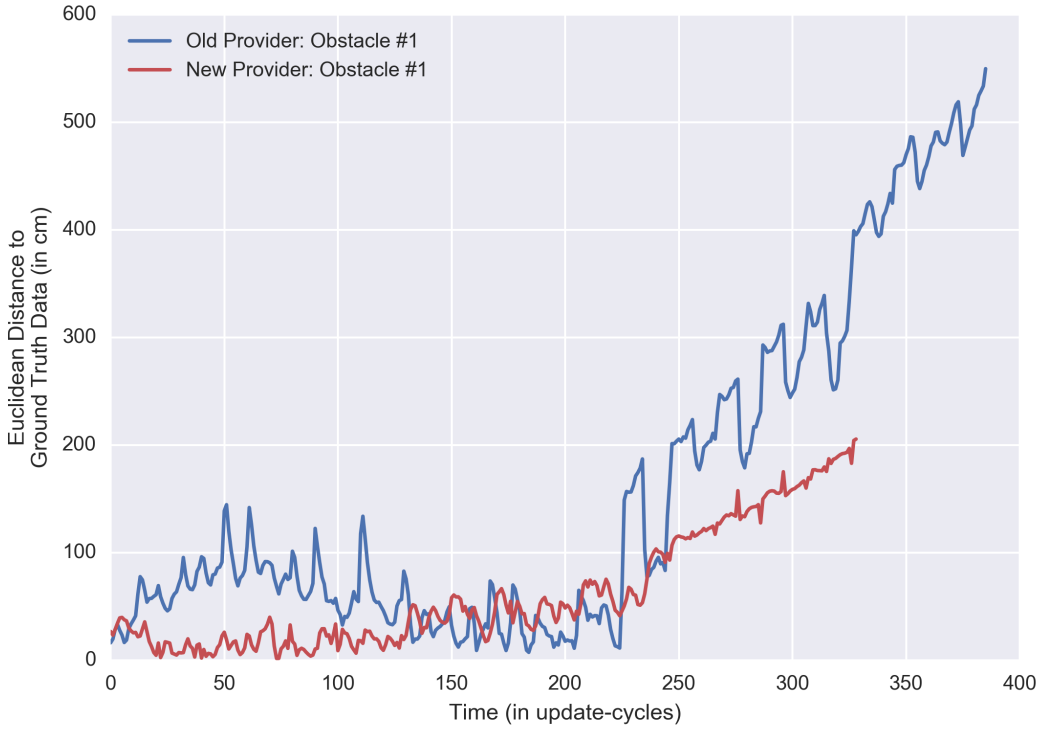


Abbildung 5.5 Auswertung von Experiment 2.2

iii. **Auswertung & Fazit**: Das in dieser Arbeit entwickelte Modul liefert auch hier besser Ergebnisse als das bisher Eingesetzte. Tatsächliche Besonderheiten lassen sich den Daten kaum entnehmen.

5.2.3 Experiment 2.3 - Mobiler Roboter, mobiles Hindernis

Das tatsächliche Spiel ist eine dynamische Szene, in der sich jeder Roboter bewegt. Diesen Umstand bezieht folgendes Experiment mit ein.

i. **Versuchsaufbau**: Zwei Roboter werden sich mit einem Abstand von sechs Metern gegenüber gestellt. Einer der beiden Roboter fungiert als Hindernis, der andere trackt ebenjenes. Der trackende Roboter wird nun – um Kollisionen zu vermeiden – einen halben Meter weiter zur Seite bewegt.

   Beide Roboter laufen nun geradeaus, bis der zu trackende Roboter nicht mehr im Bild ist.

ii. **Ergebnisse**: Die Messungen lassen sich in Abb. 5.6 ablesen.

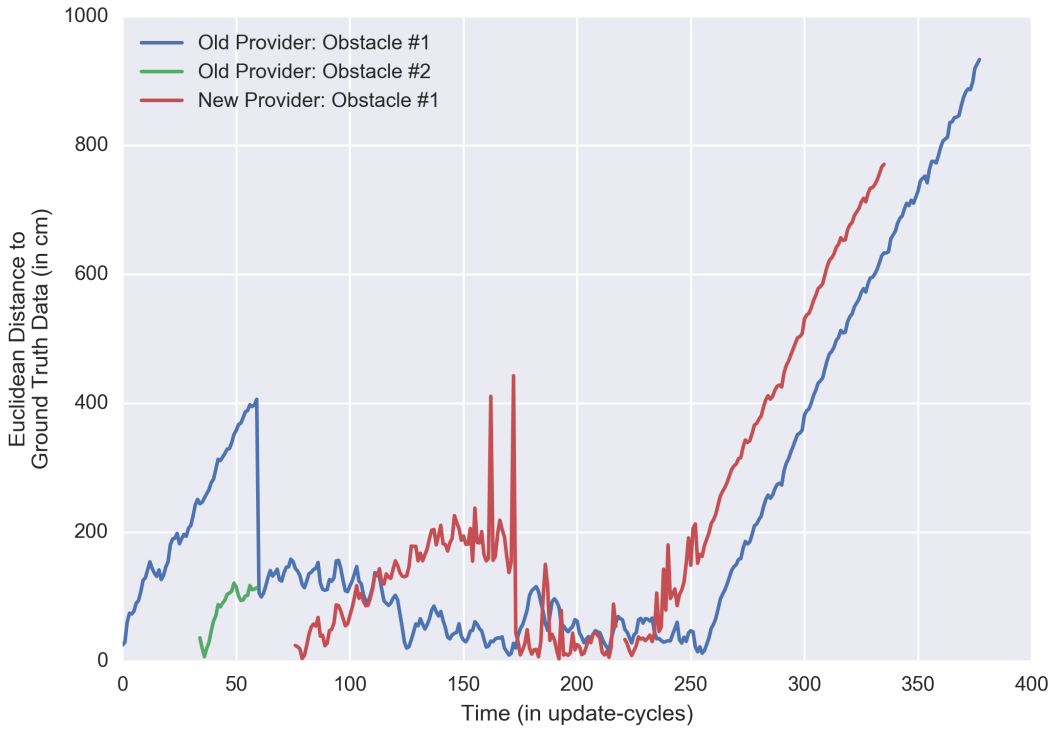


Abbildung 5.6 Auswertung von Experiment 2.3

iii. **Auswertung & Fazit**: In dem Fall, dass sich beide Roboter bewegen, liefert der neue Provider – anders als bei bisherigen Experimenten – keine deutlich besseren Ergebnisse. Dennoch sind die Ergebnisse vergleichbar und die Fehler tolerierbar.

   Der späte Einsatz des *ProbabilisticObstacleModelProviders* lässt sich durch die effektive Distanz von drei Metern erklären – jede Robotersichtung, die diese Distanz überschreitet wird ignoriert.

5.3 Experiment 3 - Orientierungsgenauigkeit

5.3.1 Experiment 3.1 - Mobiles Hindernis, konstante Orientierung

i. **Versuchsaufbau**: Ein Roboter wird auf dem Mittelkreis platziert und schaut in Richtung Tor. Exakt zwei Meter vor ihm läuft ein anderer Roboter mit konstanter Geschwindigkeit durch das Bild. Der Aufbau ist der Selbe wie in Experiment 2.1.

ii. **Ergebnisse**: Die Messungen lassen sich in Abb. 5.7 ablesen.

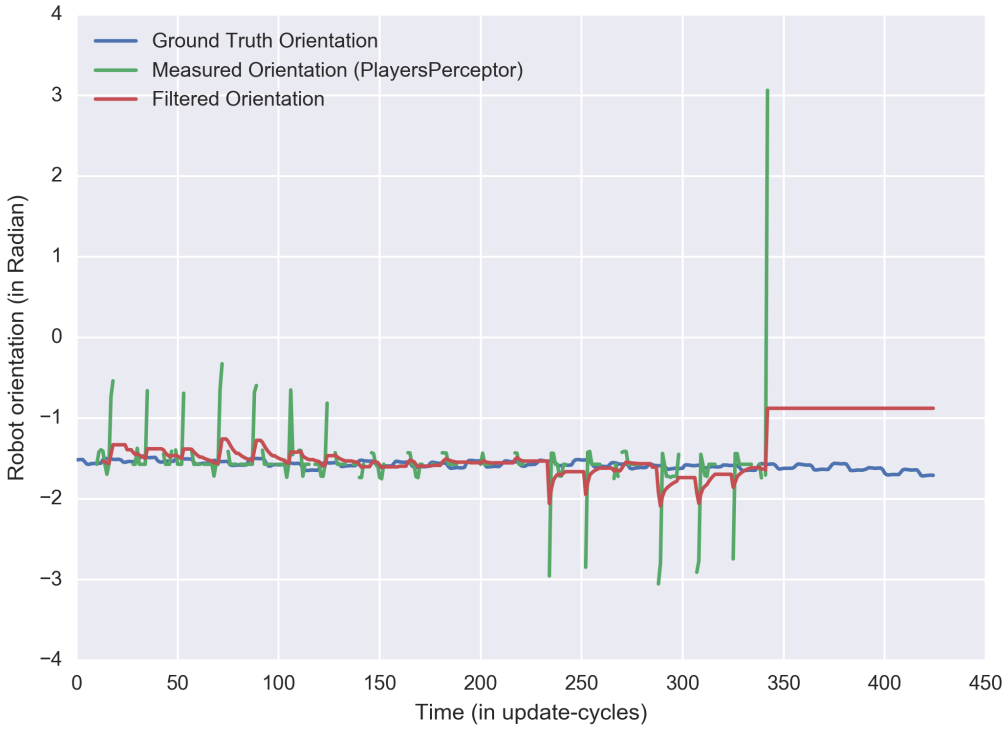


Abbildung 5.7 Auswertung von Experiment 3.1

iii. **Auswertung & Fazit**: Bei konstanter Orientierung des Roboters sind die gefilterten Werte gut. Trotz der starken Ausschläge durch Fehlerkennung des *PlayersPerceptors* bleibt die gefilterte Ausgabe nah an der *Ground Truth*.

5.3.2 Experiment 3.2 - Rotierendes Hindernis

i. **Versuchsaufbau**: Ein Roboter wird auf dem Feld platziert. Exakt zwei Meter vor ihm steht ein anderer Roboter und dreht sich um die eigene z-Achse. Die Erwartung ist kein überragendes Ergebnis, da keine Orientierungsveränderung modelliert wird. Der Fall tritt innerhalb eines Spiels aber auch sehr selten auf.

ii. **Ergebnisse**: Die Messungen lassen sich in Abb. 5.8 ablesen.

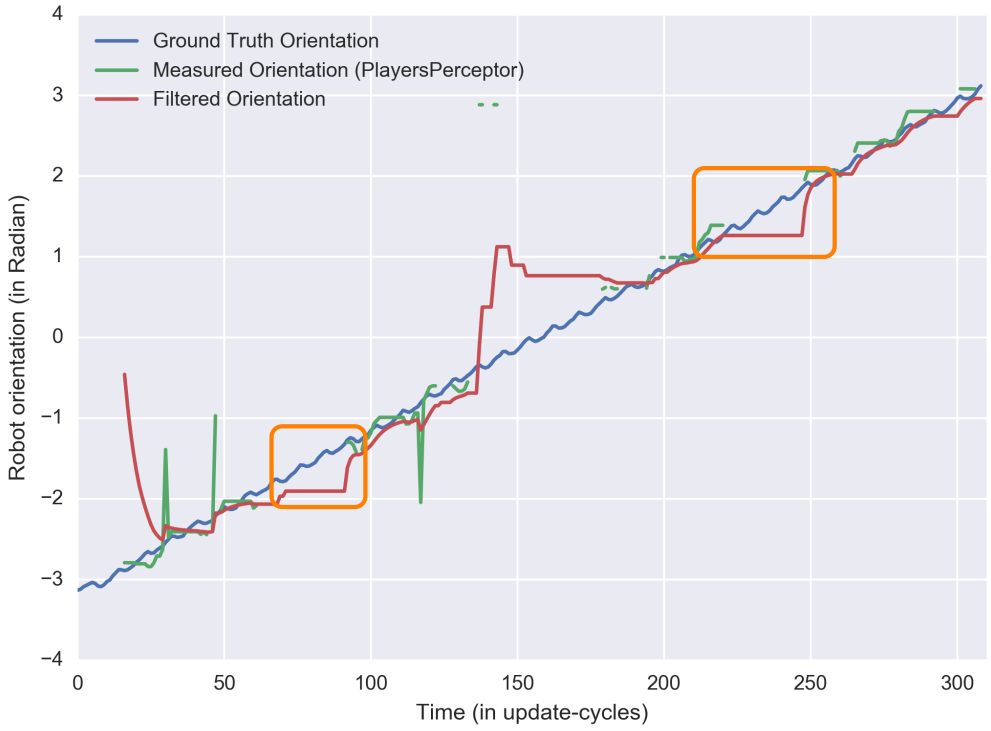


Abbildung 5.8 Auswertung von Experiment 3.2

iii. **Auswertung & Fazit**: Wie erwartet sind die Ergebnisse von der *Ground Truth* entfernt. Besonders auffällig sind zwei Ausschläge (orange markiert). Diese Ausschläge würden nicht auftreten, wenn der Zustandsvektor des Unscented Kalman Filters um $\dot{\alpha}$ erweitert wird. Wie wichtig der *Tradeoff* von Ausführungsgeschwindigkeiten zu Genauigkeit aber tatsächlich ist, müsste vorher evaluiert werden.

KAPITEL **6**

FAZIT UND AUSBLICK

Grundsätzlich zeigen die in der Evaluation (Kapitel 5) durchgeführten Experimente den Erfolg dieser Arbeit. Dennoch gibt es durchaus Optimierungsbedarf.

Besonders die Leistung des UKFs im Gegensatz zum vorher eingesetzten EKF wird hervorgehoben. Gerade Experiment 2 (Kapitel 5.2) zeigt, dass das Dynamikmodell die Zukunft um Längen besser voraussagt als das bisherige. Der damit einhergehende Verlust in Ausführungszeiten ist verkraftbar, wenn konstantere Ergebnisse geliefert werden können.

Wie stark es nun ins Gewicht fällt, dass die Orientierungsveränderung nicht modelliert wird, ist debattierbar. In den Experimenten hat die Orientierungsmodellierung aufgrund des fehlenden $\dot{\alpha}$-Parameters einen maximalen Fehler von etwa 30° erreicht. Zu vermerken wäre, dass die Erweiterung des Dynamikmodell aufgrund der periodischen Eigenschaft von Winkeln nicht trivial ist. (Vgl. Formel 6.1)

$$\alpha_t \neq \alpha_{t-1} + \dot{\alpha}_{t-1} \cdot \Delta t \tag{6.1}$$

Eine weitere Verbesserungsmöglichkeit des Moduls ist es, mehr Informationsquellen zurate zu ziehen. Der *NAO* hat noch mehr Sensoren, die möglicherweise in Frage kommen, um Hindernisse zu messen. Darunter fallen besonders die Ultraschallsensoren in der Brust des Roboters. Je nach Genauigkeit müssen die Varianzen dort aber sehr hoch angesetzt werden. Das Selbe gilt für alle taktilen Sensoren (Fußdrucksensoren, Schulterkontakte). Verarbeitet man diese Informationen, können bei guter Parametrisierung vermutlich noch bessere Ergebnisse geliefert werden.

Das mit dieser Arbeit synthetisierte *ObstacleModel* kann nun direkt zur Wegfindung im Nahbereich des Roboters eingesetzt werden. Eine weitere sinnvolle Nutzung wäre die Bildung eines globalen Weltmodells durch Zusammenlegung der Hindernismodelle mehrerer Roboter.

LITERATURVERZEICHNIS

[1] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic Robotics*. The MIT Press, 2005. – ISBN 0262201623

[2] Mühlenbrock, Andre ; Laue, Tim: Vision-based Orientation Detection of Humanoid Soccer Robots. In: *RoboCup 2017: Robot Soccer World Cup XXI*, Springer, Date TBD (Lecture Notes in Artificial Intelligence)

[3] Laue, Tim ; Röfer, Thomas: SimRobot - Development and Applications. In: Amor, Heni B. (Hrsg.) ; Boedecker, Joschka (Hrsg.) ; Obst, Oliver (Hrsg.): *The Universe of RoboCup Simulators - Implementations, Challenges and Strategies for Collaboration. Workshop Proceedings of the International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2008)*. Venice, Italy, 2008

[4] Maass, Florian: *Hindernismodellierung für Fußball spielende Roboter*, University of Bremen, Bachelorarbeit, 2014

[5] Kitano, Hiroaki ; Asada, Minoru ; Kuniyoshi, Yasuo ; Noda, Itsuki ; Osawa, Eiichi: *RoboCup: The Robot World Cup Initiative*. 1995

[6] RoboCup Standard Platform League: *Homepage*. `https://spl.robocup.org/`, 2017. – Nur online: Abgerufen am 27.08.2017

[7] RoboCup Federation: *Objective*. `http://www.robocup.org/objective`, 2017. – Nur online: Aufgerufen: 30.08.2017

[8] SoftBank Robotics: *Who is NAO?* `https://www.ald.softbankrobotics.com/en/robots/nao/`, 2017. – Nur online: Abgerufen am 27.08.2017

[9] SoftBank Robotics: *NAO- Technical overview*. `http://doc.aldebaran.com/2-1/family/robots/index_robots.html`, 2017. – Nur online: Abgerufen am 27.08.2017

[10] B-Human: *Homepage*. `https://www.b-human.de/`, 2017. – Nur online: Abgerufen am 28.08.2017

[11] Röfer, Thomas ; Laue, Tim ; Kuball, Jonas ; Lübken, Andre ; Maass, Florian ; Müller, Judith ; Post, Lukas ; Richter-Klug, Jesse ; Schulz, Peter ; Stolpmann, Andreas ; Stöwing, Alexander ; Thielke, Felix: *B-Human Team Report and Code Release 2016*. https://www.b-human.de/downloads/publications/2016/coderelease2016.pdf, 2016. – Nur online: Abgerufen am 29.08.2017

[12] Frese, Udo ; Schröder, Lutz: *Theorie der Sensorfusion*. 2016. – Vorlesungsskript an der Universität Bremen

[13] Mahalanobis, Prasanta C.: On the generalized distance in statistics. In: *Proceedings of the National Institute of Sciences (Calcutta)* 2 (1936)

[14] Kálmán, Rudolph E.: A New Approach to Linear Filtering and Prediction Problems. In: *Transactions of the ASME–Journal of Basic Engineering* 82 (1960), Nr. Series D, S. 35–45

[15] Kohlbrecher, Stefan ; Stumpf, Alexander ; Stryk, Oskar von: Grid-Based Occupancy Mapping and Automatic Gaze Control for Soccer Playing Humanoid Robots. (2011)

[16] Gutmann, Jens-Steffen ; Fukuchi, Masaki ; Fujita, Masahiro: 3D Perception and Environment Map Generation for Humanoid Robot Navigation. In: *The International Journal of Robotics Research* 27 (2008), Nr. 10, 1117-1134. http://dx.doi.org/10.1177/0278364908096316. – DOI 10.1177/0278364908096316

[17] Schulz, D. ; Burgard, W. ; Fox, D. ; Cremers, A. B.: Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In: *Proceedings 2001 ICRA. IEEE International Conference on Robotics and Automation (Cat. No.01CH37164)* Bd. 2, 2001. – ISSN 1050–4729, S. 1665–1670 vol.2