UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Personenverfolgung durch die Kombination von 2D-Entfernungs- und RGB-D-Daten

Bachelorarbeit
zur Erlangung des Grades
Bachelor of Science
im Studiengang Computervisualistik

vorgelegt von

Sebastian Stümper

Betreuer: Dipl.-Inform. Nicolai Wojke, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Betreuer: Dipl.-Inform. Viktor Seib, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Erstgutachter: Prof. Dr.-Ing. Dietrich Paulus, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau
Zweitgutachter: Dipl.-Inform. Nicolai Wojke, Institut für Computervisualistik, Fachbereich Informatik, Universität Koblenz-Landau

Koblenz, im Dezember 2013

Kurzfassung

Bei der Mensch-Maschine-Interaktion spielt die Verfolgung und Identifizierung von Personen eine wichtige Rolle. Im Rahmen dieser Arbeit ist für den Serviceroboter „Lisa “, der Arbeitsgruppe „Aktives Sehen “, ein Framework erstellt worden, um verschiedene Verfahren zur Erkennung, Verfolgung und Identifizierung von Personen zu kombinieren. Zuerst wird mittels 2D Laserscan eine Beindetektion durchgeführt um Hypothesen für Personen aufzustellen. Diese Annahme muss noch durch eine Analyse der Kinect-Punktewolke bestätigt werden. Nach erfolgreicher Bestätigung wird ein Online-Boosting auf RGB-Daten zur Identifizierung durchgeführt. Die Beindaten werden zudem mit einem linearen Kalman-Filter für die Schätzung der Personenbewegung genutzt. Durch die Kombination von Kalman-Filter mit Beindetektion und Online-Boosting soll Personenverfolgung ermöglicht werden. Des Weiteren soll eine Verwechslung von Personen - durch kurzzeitige Verdeckung oder fehlerhaftes assoziieren von Beinen - verhindert werden.

Abstract

In the man-machine interaction tracking and identification of individuals plays an important role. In this work, a framework for the service-robot „Lisa “,of the Active Vision Group, has been created to combine different methods for the detection, tracking and identification of individuals. First leg detection is performed to establish hypotheses for people using a 2D-laserscan. This assumption needs to be confirmed by an analysis of the Kinect point cloud. After successful confirmation online-boosting on RGB-data is performed for identification. The leg data will also be used with a linear Kalman filter to estimate the movement of people. Through the combination of of Kalman filter with leg detection and online-boosting people tracking should be enabled. Further receiving an interchange of persons should - by brief occlusion or faulty associate of legs - can be prevented.

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat und von dieser als Teil einer Prüfungsleistung angenommen wurde. Alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Die Vereinbarung der Arbeitsgruppe für Studien- und Abschlussarbeiten habe ich gelesen und anerkannt, insbesondere die Regelung des Nutzungsrechts.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.	ja ☐	nein ☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	ja ☐	nein ☐

Koblenz, den 18. Dezember 2013

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

In diesem Kapitel steht im ersten Abschnitt die Motivation und die Ziele der Arbeit. Im zweiten Abschnitt wird die Gliederung der Arbeit beschrieben.

1.1 Motivation und Ziele

Immer mehr Roboter halten Einzug in unser alltägliches Leben, um uns Aufgaben abzunehmen und unser Leben komfortabler zu gestalten. Derzeit handelt es sich dabei um einfache Roboter, welche für einen speziellen Zweck entworfen wurden, um zum Beispiel Staub zu saugen oder den Rasen zu mähen. Um aber einer großen Anzahl an Robotern im Haus entgegen zu wirken sollen Roboter allerdings mehr als nur eine Aufgabe können. Sie sollen am besten alles können.

Dieser Entwicklung folgt die RoboCup@Home-Liga. Forscher aus der ganzen Welt nehmen daran teil um gegenseitig ihrem aktuellen Entwicklungsstand in bestimmten Challenges zu messen, aber vor allem um Ihre Erfahrung und Entwicklung mit anderen zu teilen. Ein Teil dieser Entwicklung ist, neben einer guten Navigation und Interaktion im Haushalt, auch die Personenerkennung und -wiedererkennung. In der RoboCup@Home-Liga wird eine solche Personenerkennung in verschiedenen Challenges benötigt, speziell in der Challenge *Follow Me*, bei der ein Serviceroboter einer zu Beginn unbekannten Person (Operator) durch unbekanntes Gebiet folgen soll. Hierbei werden verschiedene Hürden gestellt, welche es zu meistern gilt. Eine dieser Hürden ist das absichtliche Kreuzen des Weges zwischen Operator und Roboter durch eine weitere Person. In einer solches Situation darf der Roboter seinen Operator nicht verlieren und plötzlich jemand anderem folgen. Der Roboter hat die Aufgabe zu erkennen, dass es sich um zwei verschiedene Personen handelt, wobei die Schwierigkeit darin besteht, dass der eigentliche Operator beim Kreuzen des Weges eine gewisse Zeit nicht sichtbar ist.

Ziel dieser Arbeit ist es ein System zu entwickeln, welches in der Lage ist Personen über ihre Beinen zu detektieren und mit einem Online-Boosting Verfahren auf Grauwertbildern zu erlernen und zu erkennen. Die Beine eines Menschen werden hierbei über ein geometrisches Modell - ein Halbkreis - in einem 2D-Laserscan auf 40 cm Höhe gesucht. Gefundene Beinpaare werden in einem Multi-Target-Tracker verarbeitet. Der Multi-Target-Tracker arbeitet auf Basis des linearen Kalman-Filters für Poseschätzung und ist erweitert mit Track-Management und Datenassoziation. Damit fehlerhafte Detektionen weniger Auswirkung haben müssen Beinpaare über einen längeren Zeitraum sichtbar sein. Gleichzeitig wird für jedes Target ein Online-Boosting erstellt, welches auf Grauwertbildern arbeitet und mit welchem es möglich ist Personen wieder zuerkennen. Das System soll schnell arbeiten und nach einer kurzen Lernphase Personen robust auseinander halten können.

Die vorhandene 2D-Personendetektion im Robbie-Framework soll überarbeitet und nach ROS portiert werden. Es soll ein Multi-Target-Tracker integriert und Modelle für Sensoren und Bewegung erstellt werden. Es wird eine einfach 3D-Personendetektion implementiert und ein Verfahren für Online-Lernen von Merkmalen integriert. Das System wird komplett in ROS implementiert.

Zum Schluss sollen die Ergebnisse des neuen Verfahrens evaluiert werden. Es soll die Erkennung von Personen getestet werden sowie ein Vergleich zwischen der Personenverfolgung mit Kombination von Beindaten und Online-Boosting gemacht werden und der Personenverfolgung, welche ausschließlich auf Beindaten basiert, durchgeführt werden. Zudem soll die Geschwindigkeit des Systems analysiert werden.

1.2 Aufbau der Arbeit

Die Arbeit gliedert sich in 5 Kapitel. Nach der Einleitung wird in Kapitel 2 auf die theoretischen Grundlagen eingegangen. Hier wird erklärt, wie die Beindetektion, das Online-Boosting und der lineare Kalman-Filter funktionieren. Abschnitt 2.1 behandelt die Beindetektion in einem horizontalen 2D-Laserscan, Abschnitt 2.2 das Online-Boosting in Grauwertbildern und Abschnitt 2.3 den linearen Kalman-Filter.

Kapitel 3 behandelt die Implementation der Arbeit. Es wird ROS sowie die erstellten oder integrierten Nodes, Bibliotheken und Messages vorgestellt. Abschnitt 3.1 führt ROS ein. Abschnitt 3.2 zeigt einen Systemüberblick mit allen Nodes, Bibliotheken und Messages. Es werden die Bibliothen Simple-Person-Detection, GMPHDF und Online-Boosting vorgestellt. Zudem die People-Tracking-Node und Leg-Detection-Node, die People-Tracking-Message und Leg-Detection-Message er-

läutert. In Abschnitt 3.3 wird der Systemablauf erklärt und an einem Beispiel gezeigt.

Kapitel 4 behandelt die Evaluation des Systems. Es wird die Erkennung sowie die Geschwindigkeit analysiert. Abschnitt 4.1 erklärt das Testszenario, die Testumgebung und die Konfiguration des Systems. In Abschnitt 4.2 wird das System analysiert und ein Vergleich zu einer Personenverfolgung ohne Bilddatenassoziation gemacht. Abschnitt 4.3 fasst die Ergebnisse zusammen.

Kapitel 5 enthält das Fazit und den Ausblick.

Kapitel 2

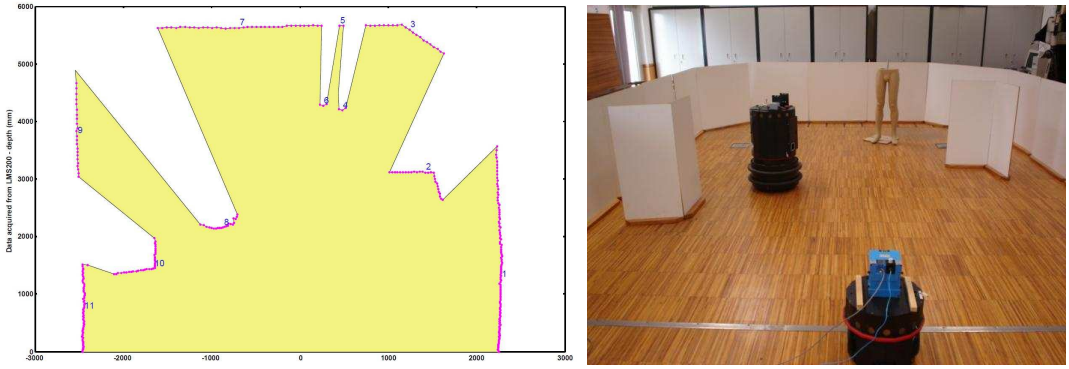
Theoretische Grundlagen

In diesem Kapitel sollen die theoretischen Grundlagen der 2D Beindetektion, des Online-Boosting und des linearen Kalman-Filters vermittelt werden. Das Kapitel gliedert sich inhaltlich in drei Abschnitte. Abschnitt 2.1 befasst sich mit der Beindetektion auf 2D-Entfernungsdaten eines Laserentfernungsmessers. Abschnitt 2.2 behandelt das AdaBoost von Freund und Schapire [FS97], sowie der Erweiterung zum Online-Boosting von Grabner et al. [GB06]. Im letzten Abschnitt 2.3 wird dann der lineare Kalman-Filter erklärt.

2.1 Beindetektion

Bei der Beindetektion wird mit Hilfe eines 2D-Laserscanners ein horizontaler 2-dimensionaler Bereich der Umgebung wahrgenommen. Bei einem 2D-Laserscanner befindet sich meist ein Spiegel im Ursprung der Messung und lenkt einen Laserstahl in eine bestimmte Richtung. Durch die Drehung des Spiegel wird der Laserstrahl in verschiedene Richtungen abgelenkt. Trifft der Laserstrahl auf ein Hindernis wird dieser reflektiert und die Entfernung zwischen Hindernis und Laserscanner ist bestimmbar. Die gemessenen Daten entsprechen dadurch aufeinander folgenden Punkten, deren Abstand und Winkel zum Laserscanner bekannt sind - es handelt sich dabei um Polarkoordinaten.

Die so vorliegenden Daten müssen zuerst segmentiert werden. Dazu eignet sich aufgrund der Daten das *Jump-Distance-Clustering* [SS08], auch Point-Distance-Based-Methods (PDBS) gennant [PN05]. Dieser Ansatz ist schnell und einfach zu implementieren. Die zugrunde liegende Idee ist einfach: Solange der euklidische Abstand zwischen zwei benachbarten Punkten nicht größer als ein Schwellwert ϑ ist, gehören diese Punkte in dasselbe Cluster. Erst wenn der Schwellwert überschritten wird, wird auch ein neues Cluster angelegt. Ein Beispiel für die Bildung von Cluster ist in Abbildung 2.1 (b) zu sehen. Im nächsten Schritt der Beindetektion werden



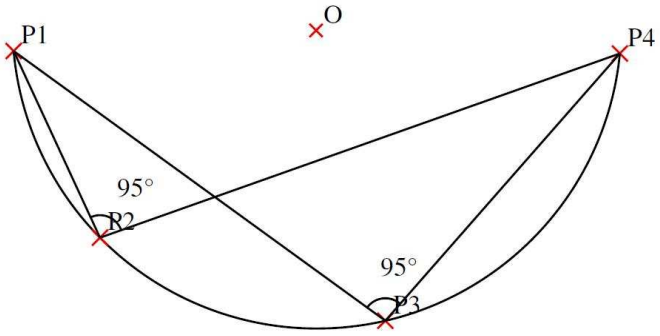
(a) segmentierter Laserscan (b) reale Szene

Abbildung 2.1: Segmentierung von 2D-Laserdaten [PN05]

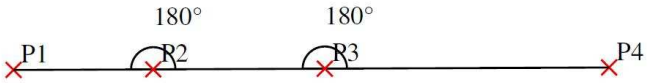
die gebildeten Cluster auf lokale Minima hin untersucht [BBCT05]. In Abbildung 2.1 (b) sind die Segmente 2, 4, 6, 8 und 10 lokale Minima. Es ist zu erkennen, dass die Beine der Puppe (Segmente 2 und 4) noch als Segmente erhalten bleiben, große Teile der Wände hingegen in diesem Schritt verworfen werden. Fehlerhafte Detektionen der Segmente 2, 8 und 10 sind aber noch vorhanden.

Im folgenden Schritt werden die Beine nun auf ein geometrisches Modell hin untersucht. Als Modell dient hier neben der Größe auch die Form des Segments. Eine effektive Methode für ein geometrisches Modell ist die Klassifizierung der Segmente in Halbkreise und Geraden. Xavier et al. [XPCR05] bestimmen dazu den inneren Winkel eines Segmentes mit der sogenannten *Internal Angel Variance*. Hierbei wird der Winkel zwischen zwei Geraden ausgehend vom Anfangs- und Endpunkt zu den weiteren Punkten des Segments bestimmt. In Abbildung 2.2 (a) wird dies anhand eines Beispiels mit 4 Punkten dargestellt. Der Punkt $P1$ ist der Startpunkt und der Punkt $P4$ der Endpunkt des Segments. Für jeden Punkt (hier $P2$ und $P3$) wird nun der Winkel der beiden aufgespannten Geraden bestimmt. Liegt der bestimmte Winkel zwischen 90° und 135° handelt es sich um ein Kreissegment. Liegt der Winkel allerdings bei ungefähr 180° handelt es sich um ein gerades Segment, siehe dazu Abbildung 2.2 (b). Mit diesem Modell lassen sich nun die fehlerhaften Segmente 2, 8 und 10 aus Abbildung 2.1 (a) aussortieren.

Im letzten Schritt werden die übrig gebliebenen Segmente noch zusammengefasst. Dafür wird für jedes Segment ein zweites gesucht, welches in einem gewissen Radius liegt. Ansonsten wird das Segment verworfen. Es wird dabei die Voraussetzung gemacht, dass bei einer Person immer beide Beine sichtbar sind. Dabei kommt es zwar zu dem Problem, dass seitlich zum Roboter stehende Personen nicht erkannt werden können, allerdings ist dies bei einer Interaktion mit dem Roboter eher unwahrscheinlich, da sich in diesem Fall eine Person in die Richtung des Roboters ausrichten wird.



(a) Kreissegment



(b) Liniensegment

Abbildung 2.2: Beispiel für Kreis- und Liniensegment [XPCR05]

2.2 Online-Boosting

Boosting hat seine Wurzeln in der PAC (**P**robability **A**pproximately **C**orrect) Lerntheorie von Valiant [Val84]. Die Idee mehrere schwache PAC-Lerner zu einem starken zu „boosten“ wurde erstmals von Kearns und Valiant [KV94] beschrieben. Es wurde versucht die Ergebnisse von einem schwachen Lernalgorithmus zu verbessern, indem man diesen wiederholt auf leicht veränderte Daten anwendet. Die ersten Versuche von Buchstabenerkennung mit frühen Boosting Algorithmen wurden von Drucker, Schapire und Simard in [DSS93] durchgeführt. Im folgenden wird nun der AdaBoost Algorithmus von Freund und Schapire [FS97] [FSA99] vorgestellt, sowie auf die Erweiterung von Grabner und Bischof hin zum Online-Boosting [GB06].

2.2.1 AdaBoost

Der AdaBoost Algorithmus wurde 1995 von Freund und Schapire [FS97] vorgestellt und löste viele praktische Probleme früherer Verstärkungsalgorithmen. Ziel des AdaBoost Verfahrens ist es durch die Linearkombination vieler schwacher Klassifikatoren einen starken Klassifikator zu bilden. Dabei können solange schwache Klassifikatoren zusammengefasst werden bis ein gewünschter kleiner Trainingsfehler erreicht ist. Die einzelnen schwachen Klassifikatoren müssen, bei einem binären Problem, nicht wesentlich besser sein als einfaches Raten. Der Fehler eines schwachen Klassifikators muss $< 50\%$ sein. Die Anzahl der T schwachen Klassifikatoren

werden mit einem Trainingsdatensatz gelernt und dann zu einem starken kombiniert. Ein Datensatz oder Trainingssatz ist eine Menge von Beispielen $\langle \boldsymbol{x}_1, y_1 \rangle, \ldots, \langle \boldsymbol{x}_n, y_n \rangle$, wobei jedem Merkmalsvektor $\boldsymbol{x}_i$ ein Ausgangswert y_i zugeordnet ist. AdaBoost lernt für gegebene schwache Klassifikatoren iterativ in einer gewissen Anzahl von Durchläufen, der Anzahl der Klassifikatoren, $t = 1, ..., T$ mit dem Datensatz. Eine der Grundideen des Algorithmus ist es, eine Verteilung bzw. Menge von Gewichten über den Trainingsdatensatz zu erhalten. Das Gewicht dieser Verteilung für das Beispiel i in Schritt t wird mit $D_t(i)$ bezeichnet. Zunächst sind alle Gewichte gleich. Mit jedem Schritt werden die Gewichte neu bestimmt, falsch klassifizierte Beispiele erhalten ein höheres Gewicht, so dass die schwachen Klassifikatoren gezwungen werden sich auf die schwierigen Beispiele in dem Trainingsdatensatz zu konzentrieren.

Das Ziel des schwachen Klassifikators ist es, eine schwache Hypothese

$$h_t^{weak} : X \to \{-1, +1\} \tag{2.1}$$

zu finden. Die Güte einer schwachen Hypothese wird bestimmt durch ihren Fehler e (für *error*) :

$$\epsilon_t = Pr_{i \sim D_t}[h_t^{weak} \neq y_i] = \sum_{i: h_t(x_i) \neq y_i} D_t(i) \tag{2.2}$$

Der Fehler wird hierbei in Bezug zum Gewicht $D_t(i)$ gemessen. Der Fehler entspricht dann der Wahrscheinlichkeitsdichte für diesen Klassifikator.

In Algorithmus 1 ist der generelle Ablauf des AdaBoost dargestellt. Nach der Initialisierung wird in jedem Schritt ein neuer Klassifikator erstellt. Dieser Klassifikator lernt dann den Trainingsdatensatz (Zeile 3). Dabei gibt es zwei Ansätze, die Gewichtung der Beispiele zu berücksichtigen. Es kann aus der Gesamtmenge der Beispiele eine bestimmte Anzahl Beispiele, entsprechend der jeweiligen Gewichtung $D(i)$, gezogen und diese dann zum Lernen genutzt werden, oder es wird jedes Trainingsbeispiel entsprechend seinem Gewicht direkt bewertet.

Im Anschluss wird die schwache Hypothese h_t^{weak} erstellt (Zeile 4) und dann der dazugehörige Fehler ϵ_t für diesen Klassifikator bestimmt (Zeile 5). Mit Hilfe des Fehlers kann dann der Gewichtungsfaktor α bestimmt werden. Je kleiner der Fehler *epsilon* ist, desto größer ist der Gewichtungsfaktor und umgekehrt. Man kann sagen, dass α die Wichtigkeit des Klassifikators h_t^{weak} darstellt. Des weiteren muss der Fehler $\epsilon \neq 0$ und $\epsilon < \frac{1}{2}$ sein, da $ln(0)$ nicht definiert ist und $ln(x) < 0$ mit $x < 1$ ist und somit negativ wäre.

Im letzten Schritt werden dann die Gewichte der einzelnen Trainingsbeispiele aktualisiert. Wird ein Beispiel i nicht richtig klassifiziert, so erhöht sich das Gewicht $D_t(i)$, wird ein Beispiel richtig klassifiziert, so verringert sich das Gewicht $D_t(i)$. Die Stärke der Änderung hängt dabei von dem aktuellen Fehler ϵ_t ab.

Algorithm 1 AdaBoost

Require: $(\boldsymbol{x}_1, y_1), ..., (\boldsymbol{x}_m, y_m)$, mit $\boldsymbol{x}_i \in X, y_i \in Y = \{-1, +1\}$

1: Initialisiere $D_1(i) = 1/m$
2: **for** $t = 1, ..., T$ **do**
3: lerne schwachen Klassifikator mit Betrachtung der Wahrscheinlichkeitsverteilung D_t
4: erstelle schwache Hypothese $h_t^{weak} : X \to \{-1, +1\}$
5: bestimme Fehler $\epsilon_t = Pr_{i \sim D_t}[h_t^{weak} \neq y_i] = \sum_{i:h_t(x_i) \neq y_i} D_t(i)$
6: bestimme Faktor $\alpha = \frac{1}{2} \ln \left(\frac{1-\epsilon_t}{\epsilon_t} \right)$
7: Aktualisiere die Wahrscheinlichkeitsverteilung

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{,wenn } h_t^{weak}(\boldsymbol{x}_i) = y_i \\ e^{\alpha_t} & \text{,wenn } h_t^{weak}(\boldsymbol{x}_i) \neq y_i \end{cases} = \frac{D_t(i) exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$$

mit Z_t als Normalisierungsfaktor.
8: **end for**

Ensure: Finale Hypothese: $H^{strong}(\boldsymbol{x}) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right)$

Zum Schluss erhält man dann den starken Klassifikator $H^{strong}(\boldsymbol{x})$. Der starke Klassifikator enthält die schwachen Klassifikatoren h_t^{weak} entsprechend ihrem Gewichtsfaktor α.

Analyse des Trainingsfehlers

Die theoretische Fähigkeit des AdaBoost ist es, den Trainingsfehler zu reduzieren. Es kann gezeigt werden, dass mit der Anzahl der schwachen Klassifikatoren der Trainingsfehler (also der Anteil der Fehler auf dem Trainingssatz) des starken Klassifikators exponentiell sinkt. Dazu kann der Trainingsfehler ϵ_t eines schwachen Klassifikators h_t^{weak} als $\epsilon_t = \frac{1}{2} - \gamma_t$ beschrieben werden. Da eine „ratende“ Hypothese für jedes Beispiel bei einem binären Problem nur richtig oder falsch sein kann, hat diese eine Fehlerquote von $\frac{1}{2}$. γ_t misst wie viel besser die Vorhersage von dem schwachen Klassifikator h_t^{weak} im Vergleich zum Raten ist. Freund und Schapire [FS97] zeigten, dass der Trainingsfehler des starken Klassifikators (der finalen Hypothese) $H^{strong}(x)$ nach oben beschränkt ist.

$$\prod_t \left[2\sqrt{\epsilon_t(1-\epsilon_t)} \right] = \prod_t \sqrt{1 - 4\gamma_t^2} \leq \exp\left(-2 \sum_t \gamma_t^2 \right) \tag{2.3}$$

Ist nun jede schwache Hypothese etwas besser als Raten, sodass $\gamma_t \geq \gamma$ für ein $\gamma \geq 0$ ist, dann sinkt der Trainingsfehler exponentiell schnell. Eine ähnliche Ei-

genschaft erreichten schon frühere Verstärkungsverfahren, allerdings musste dafür eine solche Untergrenze γ zu Beginn schon bekannt sein. Dieses Wissen über eine untere Grenze ist in der Praxis nicht vorhanden oder nur schwer anzugeben.

Analyse des Generalisierungsfehlers

Freund und Schapire zeigten im Hinblick auf den Trainingsfehler, die Anzahl der Trainingsbeispiele m, die VC-Dimension d des Hypothesenraums der schwachen Hypothese und der Anzahl der schwachen Klassifikatoren T die Grenze des Generalisierungsfehlers auf. Sie verwendeten Techniken von Baum und Haussler [BH89], um zu erläutern, dass der Generalisierungsfehler mit hoher Wahrscheinlichkeit der Gleichung 2.4 entspricht.

$$\hat{Pr}[H^{strong}(\boldsymbol{x}) \neq y] + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right) \tag{2.4}$$

Dabei ist $\hat{Pr}[\cdot]$ die empirische Wahrscheinlichkeit des Trainingsbeispiels. Diese Grenze legt nahe, dass Boosting zu *overfitting* neigt, wenn z.B. T zu groß wird. Einige Experimente [Bre98] [DC96] [Qui96] haben aber gezeigt das Boosting kein overfitting zeigt, sogar wenn es mehrere tausend Iterationen gelaufen ist. Es konnte sogar gezeigt werden, dass AdaBoost den Generalisierungsfehler auf Grundlage der Testdaten weiter minimiert, auch wenn der Fehler auf den Trainingsdaten schon null erreicht hatte. Der linken Teil der Abbildung 2.3 zeigt je eine Kurve für Trainings- und Testdaten auf Basis von Quinlans C4.5 Entscheidungsbaum Lernalgorithmus [Qui93] mit „Buchstaben“ als Test- und Trainingsdaten. Diese Ergebnisse scheinen den Grenzen der Gleichung 2.4 zu widersprechen.

Als Antwort für dieses empirische Ergebnis stellte Schapre et al. [SFBL98] auf Basis der Arbeit von Bartlett [Bar98], die *Margin-Theory* auf. Der Abstand (Margin) eines Beispiels $(\boldsymbol{x}, y)$ ist definiert als

$$margin(\boldsymbol{x}, y) = \frac{y \sum_t \alpha_t h_t^{weak}(\boldsymbol{x})}{\sum_t \alpha_t}. \tag{2.5}$$

Der Margin liegt im Intervall $[-1, +1]$ und ist nur dann positiv, wenn das Beispiel x richtig klassifiziert wurde. Desweitern kann der Wert des Margins als Messung für die Sicherheit der Schätzung verstanden werden, mit dem das Beispiel klassifiziert wurde. Schapire et al. zeigen, dass ein größerer Abstand zu einer höheren Obergrenze des Generalisierungsfehlers führt. Insbesondere ist der Generalisierungsfehler mit hoher Wahrscheinlichkeit maximal

$$\hat{Pr}[margin(\boldsymbol{x}, y) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right) \tag{2.6}$$

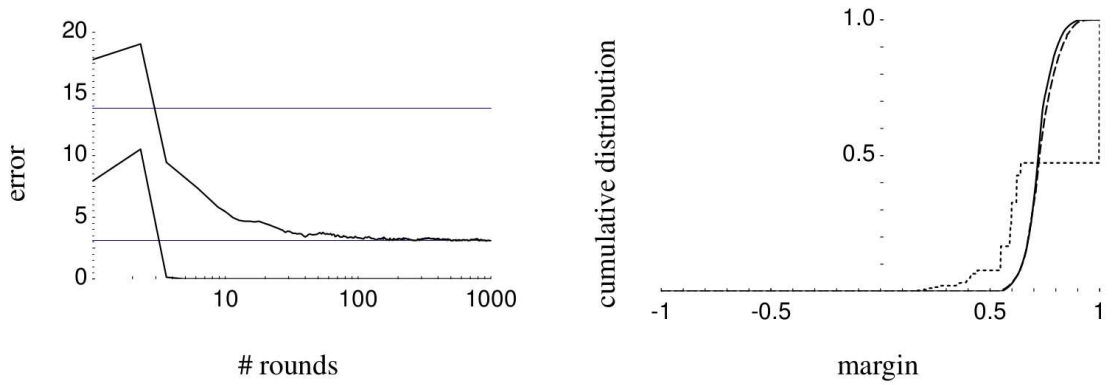


Abbildung 2.3: Generalisierungsfehler und Margin. *links:* Kurven für den Fehler in Trainings- (unten) und Testdaten (oben), *rechts:* Margin nach 5 (kurze Striche), 100 (lange Striche) und 1000 (durchgehende Kurve) Iterationen. [FSA99]

für ein $\theta > 0$. Diese Grenze ist unabhängig von der Anzahl der Iterationen T. Zusätzlich haben Schapire et al. bewiesen, dass Boosting auf die Verringerung des Abstandes zielt, da es sich auf Beispiele mit den kleinsten Abständen konzentriert (egal ob positiv oder negativ). Die Wirkung auf die Abstände kann empirisch bewiesen werden, zum Beispiel auf der rechten Seite der Abbildung 2.3. Sie zeigt die kumulative Verteilung der Abstände der Trainingsdaten mit „Buchstaben“. In diesem Fall erhöht das Boosting die Abstände weiter, auch wenn der Trainingsfehler schon Null erreicht hat, was zu einem Rückgang des Fehlers im Testdatensatz führt.

Zusammenfassung AdaBoost

AdaBoost hat viele Vorteile. Es ist schnell, einfach und leicht zu programmieren. Zudem muss man keine Parameter einstellen, mit Ausnahme der Iterationen T. Es wird kein Wissen über die schwachen Klassifikatoren benötigt und kann daher mit nahezu allen Methoden zum Finden einer schwachen Hypothese kombiniert werden. Desweiteren gibt es einige theoretische Garantien bei ausreichenden Daten und schwachen Klassifikatoren, welche mäßig genaue, aber zuverlässige schwache Hypothesen liefern. Andererseits ist AdaBoost abhängig von ausreichenden Daten und den Klassifikatoren. Sind die schwachen Hypothesen zu komplex oder zu schwach, oder die Daten zu schlecht, kann AdaBoost nicht durchgeführt werden.

2.2.2 Erweiterung des AdaBoost

In diesem Abschnitt geht es um die Erweiterung des AdaBoost von Schapire und Freund durch Grabner und Bischof [GB06] [GGB06]. AdaBoost ist ein offline Erkenner. Es benötigt also eine Trainingsphase für die Objekterkennung, um dann mit Hilfe des vorher Gelernten die Erkennung durchzuführen. Desweiteren besitzt es nicht die Fähigkeit während der Laufzeit Änderungen von Objekten zu berücksichtigen und kontinuierlich zu lernen. Diese Fähigkeit ist aber wichtig um z.B. zu *tracken* und mögliche Variationen eines Zielobjektes zu erkennen. Das Online-AdaBoost wurde 2006 von Grabner und Bischof vorgestellt und erweitert den bereits vorgestellten AdaBoost um diese fehlende Fähigkeit. Desweiteren wurde AdaBoost um die Fähigkeit der Merkmalsselektion erweitert nach der Idee von Tieu und Viola [TV00]. Im folgenden wird zuerst die Merkmalsselektion und danach der Online-AdaBoost vorgestellt.

Merkmalsselektion

Tieu und Viola [TV00] erweiterten den AdaBoost um die Fähigkeit der Merkmalsselektion. Die Idee davon ist es, dass jeder schwache Klassifikator zu genau einem Merkmal f (*feature*) gehört. Zudem gibt es eine Menge von Merkmalen $\mathcal{F}$ (*feature pool*). Da dieser Feature-Pool sehr groß sein kann, benutzt der Algorithmus nur eine Teilmenge $\mathcal{F}_{sub} = f_1, \ldots, f_k$. Das Lernen ist ähnlich zum AdaBoost. Mit jeder Iteration wählt der Algorithmus ein neues Feature aus $\mathcal{F}$ und fügt es zu der Menge $\mathcal{F}_{sub}$ hinzu. Danach wird für jedes Feature aus $\mathcal{F}_{sub}$ eine schwache Hypothese gelernt auf Basis der Trainingsdaten $\langle(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n.y_n)\rangle$. Dann wird die beste schwache Hypothese h_t^{weak} gewählt und bildet mit dem Feature den neuen Klassifikator. Der Faktor α_t wird entsprechend dem Fehler ϵ_t bestimmt. Zum Schluss wird der starke Klassifikator gebildet. Die Anzahl der Iterationen entsprechen nun der Anzahl der gewählten Feature und schwachen Klassifikatoren.

Online-AdaBoost

Damit der AdaBoost Algorithmus online-fähig ist muss jeder der Schritte aus Algorithmus 1 online-fähig sein. Zudem fehlen dem Algorithmus die Trainingsbeispiele. Die Schwierigkeit liegt dabei nicht darin die schwachen Klassifikatoren oder den Faktor Online zu aktualisieren, sondern darin, das Gewicht $D_t(i)$ eines Beispiels zu bestimmen, da Online a priori die Schwierigkeit eines Beispiels nicht bekannt ist. Im Folgenden wird das Gewicht mit λ bezeichnet, da es sich um eine andere Gewichtung handelt als beim eigentlichen AdaBoost-Verfahren. Um dieses Problem zu lösen, nutzen Grabner und Bischof die Idee von Oza [OR01b]. Dabei wird das Gewicht eines Beispiels bestimmt, indem es mit der Menge aller bisherigen schwachen Klassifikatoren geschätzt wird. Oza [OR01a] konnte zeigen das wenn Offline-

und Online-Boosting den gleichen Datensatz haben, auch das Online-Boosting zum selben Ergebnis hin konvergiert wie Offline-Boosting, wenn die Anzahl der Iterationen $T \to \infty$ geht.

Der Online Algorithmus benötigt, dass zu Beginn die Anzahl der Hypothesen schon feststeht. Dafür ist es wichtig den Unterschied zum AdaBoost zu verstehen. Im Offline-Fall werden alle Trainigsbeispiele genutzt, um einen schwachen Klassifikator zu lernen. Hingegen müssen im Online-Fall mit einem Beispiel alle schwachen Klassifikatoren gelernt und der Faktor neu bestimmt werden.

Da der Ansatz von Oza nicht direkt für die Merkmalsselektion genutzt werden kann, haben Grabner und Bischof für den AdaBoost einen Selektor eingeführt.

Selektor Der Selektor besteht aus eine Menge $\mathcal{M}$ von schwachen Klassifikatoren $\mathcal{H}^{weak} = \{h_1^{weak}, \ldots, h_M^{weak}\}$. Aus dieser Menge $\mathcal{M}$ wählt der Selektor immer einen

$$h^{sel}(\boldsymbol{x}) = h_m^{weak}(\boldsymbol{x}) \tag{2.7}$$

wobei $h_m^{weak}(\boldsymbol{x})$ dem schwachem Klassifikator entspricht, dessen Hypothese dem gewünschten Optimierungskriterium am Ähnlichsten ist. In diesem Fall wird der Klassifikator h_i^{weak} mit dem geringsten Fehler ϵ_i aus der Menge $\mathcal{H}^{weak}$ ausgewählt, sodass $m = \arg\min_i e_i$ ist.

Der Selektor kann auch als Klassifikator verstanden werden, da dieser lediglich zwischen seinen eigenen schwachen Klassifikatoren wechselt. Einen Selektor zu lernen bzw. aktualisieren bedeutet, dass jeder schwache Klassifikator lernt und dann der Beste gewählt wird. Wie im Offline-Fall gehört auch hier jeder schwache Klassifikator aus $\mathcal{H}^{weak}$ zu genau einem Feature und jeder Selektor wählt seine Menge von Feature $\mathcal{M}$ aus einem globalen Feature-Pool $\mathcal{F}$, also $\mathcal{M} = \mathcal{F}_{sub} = \{f_1, \ldots, f_M | f_i \in \mathcal{F}\}$. Die dahinter stehende Idee ist, dass das Online-Boosting nun nicht mehr direkt auf den schwachen Klassifikatoren, sondern auf den Selektoren arbeitet.

Dazu ist auch eine kleine Anpassung des starken Klassifikators nötig. Wo beim AdaBoost der starke Klassifikator aus der Linearkombination der Menge von schwachen Klassifikatoren besteht, besteht er nun aus der Menge N der Selektoren (Gleichung 2.8). Die Funktionsweise des starken Klassifikators ändert sich damit nicht, da jeder Selektor nur einen Verweis auf einen schwachen Klassifikator darstellt. Ein weiterer Unterschied zum AdaBoost besteht darin, dass von Beginn an N Selektoren und somit auch immer N schwache Klassifikatoren für den starken Klassifikator zur Verfügung stehen.

$$H^{strong}(\boldsymbol{x}) = sign\left(\sum_{n=1}^{N} \alpha_n h_n^{sel}(\boldsymbol{x})\right) \tag{2.8}$$

In Abbildung 2.4 ist das Prinzip des Online-AdaBoost dargestellt und in Algorithmus 2 der veränderte Algorithmus. Der Algorithmus arbeitet nun wie folgt. Zuerst wird eine Menge N von Selektoren erstellt und jeder Selektor erhält seinen eigenen Feature-Pool $\mathcal{H}^{weak}$ mit zufälligen Features aus $\mathcal{F}$. Mit jedem Trainingsbeispiel $(x, y), y \in \{-1, +1\}$ wird nun jeder Selektor aktualisiert. Dazu ist es natürlich weiterhin wichtig anzugeben, ob ein Beipiel ein positives oder negatives ist, also ob das zu lernende Objekt im Beispiel enthalten ist oder nicht.

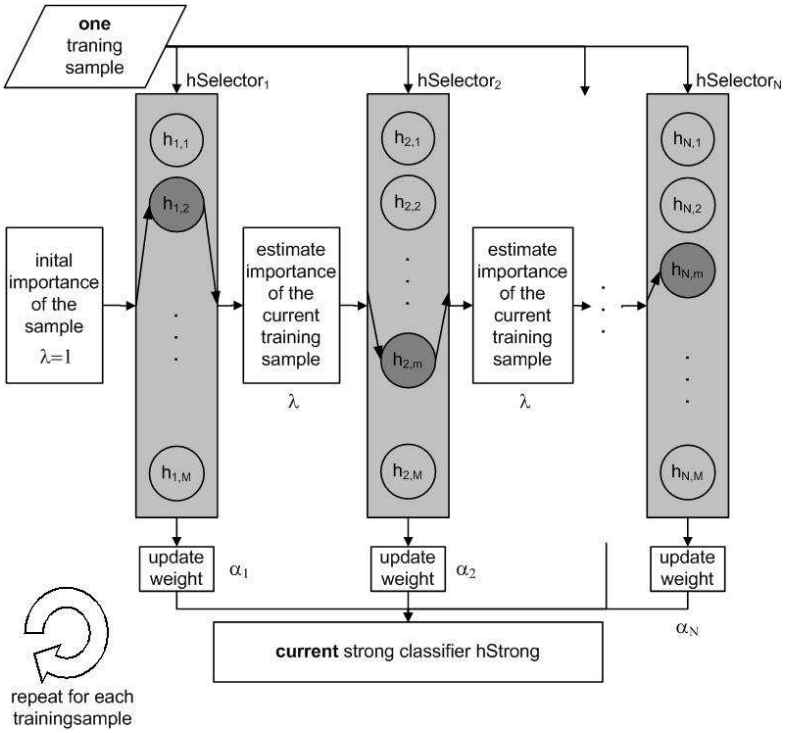


Abbildung 2.4: OnlineBoosting und Merkmalselektion [GB06]

Diese Aktualisierung wird, wie auch beim AdaBoost, entsprechend der Gewichtung λ durchgeführt. Ganz zu Beginn einer Iteration gilt $\lambda = 1$ (siehe dazu Abbildung 2.4 links weißer Kasten). Danach wählt der Selektor den besten schwachen Klassifikator (Zeile 11). Es wird jener schwache Klassifikator gewählt der den kleinsten Fehler im Selektor hat

$$\text{argmin}_m(\epsilon_{n,m}), \epsilon_{n,m} = \frac{\lambda_{n,m}^{wrong}}{\lambda_{n,m}^{corr} + \lambda_{n,m}^{wrong}}. \tag{2.9}$$

$\epsilon_{n,m}$ ist der Fehler des m-ten schwachen Klassifikators $h_{n,m}^{weak}$ des n-ten Selektors, gewichtet nach dem aktuellen Gewicht von korrekt $\lambda_{n,m}^{corr}$ und falsch $\lambda_{n,m}^{wrong}$ klassifizierten Beispielen. Danach wird die Gewichtung aktualisiert (Zeilen 17-20) und

der Faktor α_n (Zeile 16) bestimmt. Zum Schluss wird noch in jedem Selektor der schlechteste schwache Klassifzierer, also jener mit größtem Fehler ϵ aus dem Selektor aussortiert (Zeile 21) und durch einen neuen zufälligen aus dem Feature-Pool $\mathcal{F}$ ersetzt (Zeile 23). Dies soll dazu führen, das der Unterschied der einzelnen Selektoren vorhanden bleibt und diese sich nicht zu stark annähern. Zudem soll damit auf Veränderungen im Hintergrund eingegangen und diese korrigiert werden können.

Algorithm 2 Online-AdaBoost

Require: Trainingsbeispiel (x, y) mit $y \in Y = \{-1, +1\}$
Require: Starken Klassifikator $H^{strong}(x)$
Require: Gewichte $\lambda^{corr}_{n,m}, \lambda^{wrong}_{n,m}$
1: Initialisiere Gewicht $\lambda = 1$
2: **for** $n = 1, \ldots, N$ **do** ▷ für alle Selektoren
3: **for** $m = 1, \ldots, M$ **do** ▷ aktualisiere den Selektor h^{sel}_n
4: $h^{weak}_{n,m} = \text{update}(h^{weak}_{n,m}, \langle x, y \rangle, \lambda)$ ▷ aktualisiere alle Klassifikatoren
5: **if** $h^{weak}_{n,m}(x) = y$ **then** ▷ bestimme Fehler
6: $\lambda^{corr}_{n,m} = \lambda^{corr}_{n,m} + \lambda$
7: **else** $\lambda^{wrong}_{n,m} = \lambda^{wrong}_{n,m} + \lambda$
8: **end if**
9: $\epsilon_{n,m} = \frac{\lambda^{wrong}_{n,m}}{\lambda^{corr}_{n,m} + \lambda^{wrong}_{n,m}}$
10: **end for**
11: $m^+ = \text{argmin}_m(\epsilon_{n,m})$ ▷ wähle schwachen Klassifikator
12: $\epsilon_n = \epsilon_{n,m}; h^{sel}_n = h^{weak}_{n,m^+}$ ▷ mit kleinstem Fehler
13: **if** $\epsilon_n = 0 || \epsilon_n > \frac{1}{2}$ **then**
14: exit;
15: **end if**
16: $\alpha = \frac{1}{2} \ln\left(\frac{1-\epsilon_n}{\epsilon_n}\right)$ ▷ bestimme Faktor
17: **if** $h^{sel}_n(x) = y$ **then** ▷ aktualisiere Gewicht
18: $\lambda = \lambda * \frac{1}{2*(1-\epsilon_n)}$
19: **else** $\lambda = \lambda * \frac{1}{2*\epsilon_n)}$
20: **end if**
21: $m^- = \text{argmax}_m(\epsilon_{n,m})$ ▷ Entferne schlechtestes schwachen Klassifikator
22: $\lambda^{corr}_{n,m^-} = 1; \lambda^{wrong}_{n,m^-} = 1;$
23: neuen Klassifikator $h^{weak}_{n-m^-}$ ▷ ersetze mit neuem schwachen Klassifkator
24: **end for**
Ensure: Finale Hypothese: $H^{strong}(x) = sign\left(\sum_{n=1}^{N} \alpha_n h^{sel}_n(x)\right)$

2.3 Kalman-Filter

Dieser Abschnitt behandelt den linearen Kalman-Filter. Der Kalman-Filter ist eine Umsetzung des Bayes-Filter mit Momenten erster und zweiter Ordnung. Bevor in Abschnitt 2.3.3 auf den Kalman-Filter eingegangen wird, wird in Abschnitt 2.3.2 zuerst der Bayes-Filter erklärt. Zudem wird in Abschnitt 2.3.1 auf die nötigen Begriffe erläutert. Die Erklärungen zum Kalman-Filter, Bayes-Filter und den Begriffen sind aus [TBF05].

2.3.1 Begriffe

Zustand

Die Umwelt eines Roboters wird durch verschiedene Zustände beschrieben. Ein Zustand beinhaltet dabei eine Menge von verschiedenen betrachteten Aspekten. Diese Aspekte können z.B. eine Position oder Geschwindigkeit sein. Zustände können zudem statisch oder dynamisch sein. Wände und Bäume sind ein Beispiel für statische Zustände, da sich ihre Position nicht verändert. Personen hingegen sind dynamische Zustände, da diese sich frei bewegen können. Im weiteren Verlauf der Arbeit wird mit Zustand immer ein dynamischer Zustand betrachtet, sollte doch mal ein statischer Zustand betrachtet werden, dann wird dies explizit genannt.

Ein Zustand wird bezeichnet mit der Variablen $\boldsymbol{x}$, wird eine genauer Zeitpunkt eines Zustandes betrachtet so wird dieser mit $\boldsymbol{x}_t$ bezeichnet.

Messung

Neben dem vorhanden sein von Zuständen ist es auch wichtig diese Zustände wahrzunehmen. Dazu dienen *Messungen*. Mit Sensoren können *bestimmte Zustände* zu einem *bestimmten Zeitpunkt* wahrgenommen werden. Ein Sensor ist in diesem Fall nicht nur eine Hardwarekomponente, wie z.B. ein Laserscanner oder eine Kamera, sondern auch eine Softwarekomponente, wie eine Beindetektion. Eine Messung wird mit der Variablen $\boldsymbol{z}_t$ bezeichnet. Im Gegensatz zu einem Zustand $\boldsymbol{x}$ benötigt eine Messung immer einen Zeitpunkt t . Alle Messungen zwischen zwei Zeitpunkten $t_1 < t_2$ werden mit

$$\boldsymbol{z}_{t_1:t_2} = \boldsymbol{z}_{t_1}, \boldsymbol{z}_{t_1+1}, \boldsymbol{z}_{t_1+2}, ..., \boldsymbol{z}_{t_2} \tag{2.10}$$

in einer Notation zusammengefasst.

Aktion

Eine *Aktion* ändert einen Zustand in der Welt durch aktives Handeln. Ein Beispiel für eine Aktion ist das Öffnen einer Tür durch den Roboter oder auch eine

durchgeführte Bewegung des Roboters. Aktionen sind also selbst durchgeführte Handlungen und damit bekannt und müssen nicht gemessen werden. Eine Aktion wird mit der Variablen $\boldsymbol{u}_t$ bezeichnet. Eine Aktion $\boldsymbol{u}_t$ entspricht dabei immer einer ausgeführten Aktion im Zeitintervall $[t-1;t[$. Eine Sequenz von Aktionen $\boldsymbol{u}_{t_1:t_2}$ wird wieder mit

$$\boldsymbol{u}_{t_1:t_2} = \boldsymbol{u}_{t_1}, \boldsymbol{u}_{t_1+1}, \boldsymbol{u}_{t_1+2}, ..., \boldsymbol{u}_{t_2} \tag{2.11}$$

zusammengefasst.

Probabilistische Gesetze

Mit Hilfe von Wahrscheinlichkeitsgesetzen ist es möglich eine Aussage für einen Zustand zum Zeitpunkt $\boldsymbol{x}_t$ zu machen. Es ist hier von Interesse zu wissen, wie wahrscheinlich ein Zustand $\boldsymbol{x}_t$ unter bestimmten Bedingungen ist. Dafür werden alle Vorzustände $\boldsymbol{x}_{0:t-1}$, Messungen $\boldsymbol{z}_{1:t-1}$ und Aktionen $\boldsymbol{u}_{1:t}$ betrachtet. Damit ergibt sich eine Wahrscheinlichkeitsverteilung der Form:

$$p(\boldsymbol{x}_t|\boldsymbol{x}_{0:t-1}, \boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}) \tag{2.12}$$

Der Zustand $\boldsymbol{x}_t$ wird auch *vollständiger Zustand* genannt, da dieser alle Aktionen und Messungen bis zum Zeitpunkt t zusammenfasst. Ist der Vorzustand $\boldsymbol{x}_{t\text{-}1}$ auch ein vollständiger Zustand, so enthält dieser ebenfalls alle Messungen $\boldsymbol{z}_{1:t-1}$ und Aktionen $\boldsymbol{u}_{1:t-1}$ bis zum Zeitpunkt $t-1$. Dadurch kann man den Zustand $\boldsymbol{x}_t$ auch beschreiben mit:

$$p(\boldsymbol{x}_t|\boldsymbol{x}_{0:t-1}, \boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}) = p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_t) \tag{2.13}$$

Diese Gleichung erfordert die *bedingte Unabhängigkeit*. Die bedingte Unabhängigkeit besagt, dass eine bestimmte Variable unabhängig von anderen Variablen ist, wenn diese Variablen durch eine Dritte beschrieben wird. Die bedingte Unabhängigkeit wird später von den Bayes-Filter noch benötigt. Die bedingte Wahrscheinlichkeit $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{u}_t)$ ist die **Wahrscheinlichkeit für einen Zustandsübergang**.

Möchte man die Wahrscheinlichkeit für eine bestimmte Messung $\boldsymbol{z}_t$ wissen, lässt sich dies ebenfalls mit einer probabilistischen Aussage darstellen:

$$p(\boldsymbol{z}_t|\boldsymbol{x}_{0:t}, \boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}) \tag{2.14}$$

Auch diese Aussage ist bedingt unabhängig unter der Annahme, das $\boldsymbol{x}_t$ ein vollständiger Zustand ist.

$$p(\boldsymbol{z}_t|\boldsymbol{x}_{0:t}, \boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}) = p(\boldsymbol{z}_t|\boldsymbol{x}_t) \tag{2.15}$$

Die bedingte Wahrscheinlichkeit $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ ist die **Wahrscheinlichkeit einer Messung**.

Belief

Der *Belief* (deutsch: Vorstellung) beruht auf der Tatsache, dass ein Zustand nicht korrekt gemessen werden kann, sondern immer einer Ungenauigkeit unterliegt. Der Belief ordnet dazu jedem möglichen Zustand $\boldsymbol{x}_t$ eine Wahrscheinlichkeit zu. Der Belief ist eine *posterior* Wahrscheinlichkeit auf Grundlage aller verfügbaren Daten:

$$bel(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t}, \boldsymbol{u}_{1:t}) \tag{2.16}$$

Diese posterior Wahrscheinlichkeit des Zustandes $\boldsymbol{x}_t$ zum Zeitpunkt t, basiert auf allen Messungen $\boldsymbol{z}_{1:t}$ und Aktionen $\boldsymbol{u}_{1:t}$. Diese Wahrscheinlichkeit setzt voraus, dass alle vorherigen Messungen vorhanden sind. Interessant wäre es aber auch zu wissen, wie wahrscheinlich ein Zustand ist ohne die letzte Messung $\boldsymbol{z}_t$. Dieses posterior wird bezeichnet mit:

$$\overline{bel}(\boldsymbol{x}_t) = p(\boldsymbol{x}_t|\boldsymbol{z}_{1:t-1}, \boldsymbol{u}_{1:t}) \tag{2.17}$$

Diese Wahrscheinlichkeit wird im Zusammenhang mit Zustandsfiltern als **Prädiktion** bezeichnet. $bel(\boldsymbol{x}_t)$ wird im selben Zusammenhang als **Korrektur** bezeichnet.

2.3.2 Bayes-Filter

Algorithm 3 Bayes-Filter

Require: bel($\boldsymbol{x}_{t\text{-}1}$) , $\boldsymbol{u}_t$, $\boldsymbol{z}_t$
1: **for** $\boldsymbol{x}_t$ **do**
2: $\quad \overline{bel}(\boldsymbol{x}_t) = \int p(\boldsymbol{x}_t|\boldsymbol{u}_t, \boldsymbol{x}_{t\text{-}1}) bel(\boldsymbol{x}_{t\text{-}1}) dx$
3: $\quad bel(\boldsymbol{x}_t) = \eta p(\boldsymbol{z}_t|\boldsymbol{x}_t) \overline{bel}(\boldsymbol{x}_{t\text{-}1})$
4: **end for**
Ensure: $bel(\boldsymbol{x}_t)$

Mit dem *Bayes-Filter* lässt sich der Belief auf Basis von Messungen und Aktionen berechnen. Der Bayes-Filter ist ein rekursiver Algorithmus, da der Belief $bel(\boldsymbol{x}_t)$ zum Zeitpunkt t aus dem Belief $bel(\boldsymbol{x}_{t\text{-}1})$ zum Zeitpunkt $t-1$ bestimmt wird. Algorithmus 3 zeigt den Basis Bayes-Filter in Pseudocode. Der Bayes-Filter Algorithmus besteht aus zwei entscheidenden Schritten. In *Zeile 2* wird die Aktion $\boldsymbol{u}_t$ verarbeitet. Dabei wird der Belief des Zustand $\boldsymbol{x}_t$ basierend auf dem Belief des Zustandes $\boldsymbol{x}_{t\text{-}1}$ und der Aktion $\boldsymbol{u}_t$ berechnet. Genauer, der Belief $\overline{bel}(\boldsymbol{x}_t)$ der dem Zustand $\boldsymbol{x}_t$ zugewiesen wird, ist das Integral des Produkts der zwei Verteilungen $p(\boldsymbol{x}_t|\boldsymbol{u}_t, \boldsymbol{x}_{t\text{-}1})$ - der Wahrscheinlichkeit das $\boldsymbol{u}_t$ eine Änderung von $\boldsymbol{x}_{t\text{-}1}$ zu $\boldsymbol{x}_t$ induziert - und $bel(\boldsymbol{x}_{t\text{-}1})$ - dem vorherigen Belief. Dieser erste Schritt entspricht der

oben genannten *Prädiktion*. Im zweiten Schritt wird dann die Messung verarbeitet. In *Zeile 3* wird dazu die Wahrscheinlichkeit für die Messung $\boldsymbol{z}_t$ mit dem Belief $\overline{bel}(\boldsymbol{x}_t)$ multipliziert, um den Belief $bel(\boldsymbol{x}_t)$ zu erhalten. Dies wird für alle Zustände $\boldsymbol{x}_t$ berechnet. Zudem wird jeder Belief noch mit η normalisiert. Als Ergebnis bleibt der endgültige Belief $bel(\boldsymbol{x}_t)$.

Um den Bayes-Filter zu nutzen ist desweiteren ein initialer Zustand $\boldsymbol{x}_0$ für den Zeitpunkt $t = 0$ nötig. Kennt man den Anfangszustand des Systems, so kann man den $bel(\boldsymbol{x}_0)$ bestimmen. Kennt man den Anfangszustand nicht, dann kann man diesen z.B. mit einer gleichmäßigen Verteilung initialisieren.

2.3.3 Lineare Kalman-Filter

Der Kalman-Filter gehört zu der Gruppe der Gaußfilter. Die Gaußfilter sind rekursive Zustandsschätzer und basieren auf dem Bayes-Filter. Die Gaußfilter basieren auf der Idee, den Belief durch eine mehrdimensionale Normalverteilung darzustellen.

$$p(\boldsymbol{x}) = det(2\pi\Sigma)^{-\frac{1}{2}} exp\{-\frac{1}{2}(\boldsymbol{x}-\mu)^T\Sigma^{-1}(\boldsymbol{x}-\mu)\} \tag{2.18}$$

Diese Dichte der Variablen x wird durch zwei Gruppen von Parametern gekennzeichnet: dem Mittelwert μ und der Kovarianz Σ. Der Mittelwert μ ist ein Vektor, der die gleiche Dimensionalität wie der Zustand $\boldsymbol{x}$ besitzt. Die Kovarianz ist eine quadratische, symmetrische und positiv-semidefinite Matrix. Die Dimension entspricht der Dimensionalität des Zustand $\boldsymbol{x}$ quadriert. Somit hängt die Anzahl der Elemente in der Kovarianz-Matrix quadratisch von der Anzahl der Elemente in dem Zustandsvektor ab.

Die Darstellung einer Gaußverteilung durch den Mittelwert und die Kovarianz ist die Repräsentation durch die Momente erster und zweiter Ordnung einer Wahrscheinlichkeitsverteilung.
Der Kalman-Filter wurde in den 1950er Jahren von Rudolph Emil Kalman erfunden. Mit dem Kalman-Filter ist es möglich, Vorhersagen und Korrekturen in linearen Systemen durchzuführen. Der Kalman-Filter implementiert den Belief für stetige Zustandsräume und ist nicht einsetzbar für diskrete oder hybride Zustandsräume. Im Kalman-Filter wird, wie oben bei den Gaußfiltern genannt, der Belief durch die Momente repräsentiert. Zu einem Zeitpunkt t besteht der Belief aus dem Mittelwert μ_t und der Kovarianz Σ_t. Zusätzlich zu der Markovannahme müssen noch folgende drei Eigenschaften erfüllt sein, damit der Posterior gaußverteilt ist.

1: Die Wahrscheinlichkeit $p(\boldsymbol{x}_t|\boldsymbol{u}_t, \boldsymbol{x}_{t\text{-}1})$ für den Folgezustand $\boldsymbol{x}_t$ muss durch eine lineare Funktion mit additivem gaußschen Rauschen beschrieben sein:

$$\boldsymbol{x}_t = \boldsymbol{A}_t\boldsymbol{x}_{t\text{-}1} + \boldsymbol{B}_t\boldsymbol{u}_t + \varepsilon_t \tag{2.19}$$

$\boldsymbol{x}_t$ und $\boldsymbol{x}_{t\text{-}1}$ sind die Zustandsvektoren und $\boldsymbol{u}_t$ ein Aktionsvektor, der mehrere Aktionen beinhalten kann.

$$\boldsymbol{x}_t = \begin{pmatrix} \boldsymbol{x}_{1,t} \\ \boldsymbol{x}_{2,t} \\ \vdots \\ \boldsymbol{x}_{n,t} \end{pmatrix} \quad und \quad \boldsymbol{u}_t = \begin{pmatrix} \boldsymbol{u}_{1,t} \\ \boldsymbol{u}_{2,t} \\ \vdots \\ \boldsymbol{u}_{m,t} \end{pmatrix} \tag{2.20}$$

$\boldsymbol{A}_t$ und $\boldsymbol{B}_t$ sind Matrizen. $\boldsymbol{A}_t$ ist eine quadratische Matrix der Größe $n \times n$ mit der Dimension n des Zustandsvektors $\boldsymbol{x}_t$. $\boldsymbol{B}_t$ ist eine Matrix der Größe $n \times m$ mit der Dimension m des Aktionsvektors $\boldsymbol{u}_t$. Mit den Multiplikationen $\boldsymbol{A}_t\boldsymbol{x}_{t\text{-}1}$ und $\boldsymbol{B}_t\boldsymbol{u}_t$ wird die *Zustandsübergangsfunktion* linear. Die Variable ε_t ist eine gaußsche Zufallsvariable, welche eine Zufälligkeit im Zustandsübergang modellieren soll. ε_t hat dieselbe Dimension wie $\boldsymbol{x}_t$. Der Mittelwert von ε_t ist null und die Kovarianz wird durch die Matrix $\boldsymbol{R}_t$ beschrieben und bezeichnet.

Es ist nun möglich die *Zustandsübergangswahrscheinlichkeit* durch Einsetzen der Gleichung 2.19 in die Gleichung 2.18 auszudrücken. Der Mittelwert des Posterior-Zustands ist durch $\boldsymbol{A}_t\boldsymbol{x}_{t\text{-}1} + \boldsymbol{B}_t\boldsymbol{u}_t$ gegeben und die Kovarianz durch $\boldsymbol{R}_t$:

$$p(\boldsymbol{x}_t|\boldsymbol{u}_t, \boldsymbol{x}_{t\text{-}1}) = det(2\pi\boldsymbol{R}_t)^{-\frac{1}{2}} exp\{-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{A}_t\boldsymbol{x}_{t\text{-}1} + \boldsymbol{B}_t\boldsymbol{u}_t)^T \boldsymbol{R}_t^{-1}(\boldsymbol{x} - \boldsymbol{A}_t\boldsymbol{x}_{t\text{-}1} + \boldsymbol{B}_t\boldsymbol{u}_t)\} \tag{2.21}$$

Dieser Zustandsübergang mit den Matrizen $\boldsymbol{A}_t, \boldsymbol{B}_t$ und $\boldsymbol{R}_t$ wird häufig auch im Begriff **Bewegungsmodell** zusammengefasst.

2: Die Wahrscheinlichkeit $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ für eine Messung $\boldsymbol{x}_t$ muss ebenfalls durch eine lineare Funktion mit additivem gaußschen Rauschen beschrieben sein:

$$\boldsymbol{z}_t = \boldsymbol{c}_t\boldsymbol{x}_t + \delta_t \tag{2.22}$$

$\boldsymbol{z}_t$ ist der Messvektor, der mehrere Messungen beinhalten kann.

$$\boldsymbol{z}_t = \begin{pmatrix} \boldsymbol{z}_{1,t} \\ \boldsymbol{z}_{2,t} \\ \vdots \\ \boldsymbol{z}_{k,t} \end{pmatrix} \tag{2.23}$$

Die Matrix $\boldsymbol{C}_t$ hat die Größe $k \times n$ mit k der Dimension der Messvektors $\boldsymbol{z}_t$. Der Vektor δ_t beschreibt die Messungenauigkeit, also das Sensorrauschen. Die

Verteilung von δ_t ist eine mehrdimensionale Gaußverteilung um den Mittelwert null und die Kovarianz $\boldsymbol{Q}_t$. Die Messwahrscheinlichkeit lässt sich durch folgende mehrdimensionale Normalverteilung darstellen:

$$p(\boldsymbol{z}_t|\boldsymbol{x}_t) = det(2\pi\boldsymbol{Q}_t)^{-\frac{1}{2}} exp\{-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{c}_t\boldsymbol{x}_t)^T {\boldsymbol{Q}_t}^{-1}(\boldsymbol{x} - \boldsymbol{c}_t\boldsymbol{x}_t)\} \tag{2.24}$$

Dieser Vorgang mit den Matrizen $\boldsymbol{C}_t$ und $\boldsymbol{Q}_t$ wird häufig auch mit dem Begriff **Beobachtungsmodell** bezeichnet.

3: Schlussendlich muss noch der initale Belief $bel(\boldsymbol{x}_0)$ normalverteilt sein. Dazu wird der Mittelwert des initalen Beliefs mit μ_0 und die Kovarianz mit Σ_0 bezeichnet:

$$bel(\boldsymbol{x}_0) = p(\boldsymbol{x}_0) = det(2\pi\Sigma_0)^{-\frac{1}{2}} exp\{-\frac{1}{2}(\boldsymbol{x}_0 - \mu_0)^T \Sigma_0^{-1}(\boldsymbol{x}_0 - \mu_0)\} \tag{2.25}$$

Diese drei Eigenschaften sind ausreichend, um sicherzustellen, dass der posterior Belief $bel(\boldsymbol{x}_t)$ zu jedem Zeitpunkt t gaußverteilt ist.

Der Kalman-Filter Algorithmus

Algorithm 4 Kalman-Filter

Require: μ_{t-1} , Σ_{t-1} , $\boldsymbol{u}_t$, $\boldsymbol{z}_t$

1: $\overline{\mu}_t = \boldsymbol{A}_t\mu_{t-1} + \boldsymbol{B}_t\boldsymbol{u}_t$
2: $\overline{\Sigma}_t = \boldsymbol{A}_t\Sigma_{t-1}\boldsymbol{A}_t^T + \boldsymbol{R}_t$
3: $\boldsymbol{K}_t = \overline{\Sigma}_t\boldsymbol{C}_t^T(\boldsymbol{C}_t\overline{\Sigma}_t\boldsymbol{C}_t^T + \boldsymbol{Q}_t)^{-1}$
4: $\mu_t = \overline{\mu}_t + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{C}_t\overline{\mu}_t)$
5: $\Sigma_t = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{C}_t)\overline{\Sigma}_t$

Ensure: μ_t , Σ_t

Der Algorithmus des Kalman-Filters ist in Algorithmus 4 dargestellt. Im Kalman-Filter wird der Belief $bel(\boldsymbol{x}_t)$ zu einem Zeitpunkt t durch den Mittelwert μ_t und die Kovarianz Σ_t repräsentiert. Als Eingabewerte für den Kalman-Filter dient der Mittelwert μ_{t-1} und die Kovarianz Σ_{t-1} des Zeitpunktes $t-1$. Um den Kalman-Filter zu aktualisieren wird zudem noch der Aktionsvektor $\boldsymbol{u}_t$ und der Messvektor $\boldsymbol{z}_t$ benötigt. Als Ergebnis liefert der Kalman-Filter den Belief $bel(\boldsymbol{x}_t)$, dargestellt durch den Mittelwert μ_t und die Kovarianz Σ_t.

Die Zeilen 1 und 2 zeigen die **Prädiktion** mit dem geschätzten Belief $\overline{bel}(\boldsymbol{x}_t)$ repräsentiert durch $\overline{\mu}_t$ und $\overline{\Sigma}_t$. Dieser Belief ist geschätzt durch das Einbeziehen des Aktionsvektors $\boldsymbol{u}_t$. Der Mittelwert $\overline{\mu}_t$ wird aktualisiert mit dem deterministischen Teil der Funktion 2.19. Die Aktualisierung der Kovarianz berücksichtigt dabei die

Tatsache, dass der Zustand durch die Matrix $\boldsymbol{A}_t$ auf den Vorzuständen basiert. Die Matrix $\boldsymbol{A}_t$ wird zweimal mit der Kovarianz multipliziert, da die Kovarianz eine quadratische Matrix ist.

In Zeile 3 wird das sogenannte *Kalman-Gain* $\boldsymbol{K}_t$ bestimmt. Mit dem Kalman-Gain wird festgelegt wie stark die Messung in die neue Zustandsschätzung mit einbezogen wird. In den Zeilen 4 und 5 wird aus dem Belief $\overline{bel}(\boldsymbol{x}_t)$ die endgültige Schätzung $bel(\boldsymbol{x}_t)$ bestimmt. Der Mittelwert μ_t wird bestimmt durch Anpassen in dem Verhältnis des Kalman-Gain und der Abweichung der aktuellen Messung $\boldsymbol{z}_t$. Zum Schluss wird noch die endgültige Kovarianz Σ_t bestimmt, ebenfalls unter Einbeziehung des Kalman-Gain und der Messung $\boldsymbol{z}_t$.

Beispiel

In Abbildung 2.5 wird der Kalman-Filter anhand einer ein-dimensionalen Lokalisation erklärt. Zu Beginn (a) gibt es eine initiale Schätzung (Prädiktion) der Position des Roboters. Es wird geschätzt, dass sich der Roboter am Wahrscheinlichsten in der Nähe von Position „8“ aufhält. Es ist zu erkennen, dass die Gaussglocke sehr flach verläuft, woraus zu sehen ist, dass der Roboter in einem großen Bereich um die Position „8“ sein kann. In (b) wird nun eine Messung durchgeführt. Die Messung besagt das sich der Roboter um die Position „6“ aufhält, diese Ungenauigkeit basiert z.B auf Sensorrauschen. Die Gaußglocke der Messung ist sehr steil und grenzt den Bereich in dem sich der Roboter befindet stark ein. In (c) kommt es nun zur Korrektur. Mit der Messung und der Schätzung wird die wahrscheinlichste Position des Roboters bestimmt. Es ist zu erkennen, dass die Position nun noch genauer bestimmt werden kann als durch die Messung. In (d) ist die Prädiktion zu sehen, nachdem sich der Roboter weiter nach rechts bewegt hat. Mit der Aktion und dem Bewegungsmodell ist die Gaußglocke nun wieder flacher geworden und der Roboter wird sich nun vermutlich im Bereich um die Position „21“ aufhalten. Schritt (e) zeigt nun wieder eine Messung. Diesmal wird die Position „25“ gemessen. (f) zeigt eine Korrektur und die wahrscheinlichste Position ist nun „23“. Es ist zu erkennen, dass die Genauigkeit in (f) nicht so hoch ist wie in (c). Dies liegt zum einen daran, dass die Messung in (e) nicht so genau wie jene in (b) ist und zudem die Prädiktion (d) und Messung (e) nicht so sehr übereinstimmen wie die Prädiktion (a) und Messung (b).

2.4 Zusammenfassung

In diesem Kapitel wurden die theoretischen Grundlagen dargelegt. Es wurde die Beindetektion aus 2D Entfernungsdaten, der AdaBoost, sowie die Erweiterung zum Online-Boosting und zum Schluss der lineare Kalman-Filter erklärt.

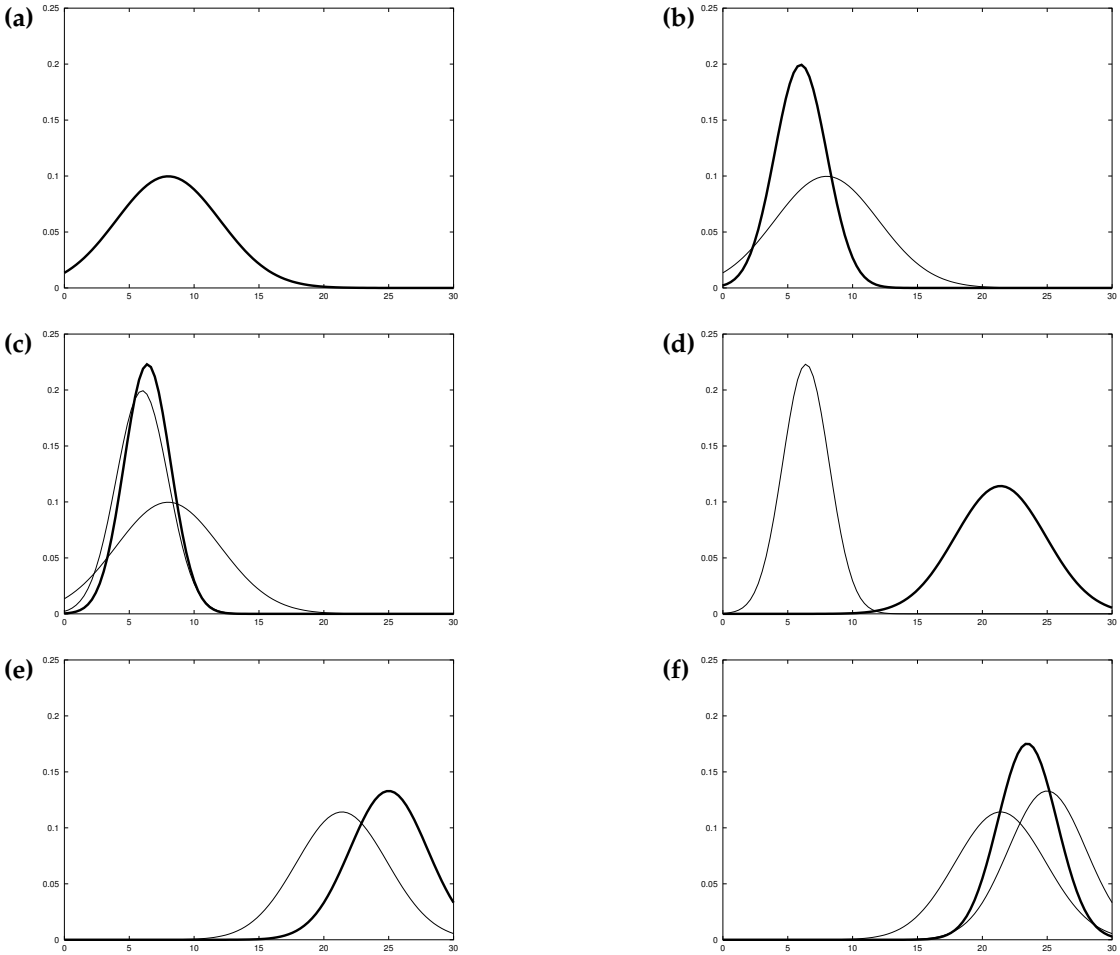


Abbildung 2.5: Beispiel für den Kalman-Filter im 1 dimensionalen Raum, (a) initale Belief, (b) eine Messung mit Unsicherheit (fettgedruckt), (c) Belief nach Verarbeitung der Messung, (d) Belief nach einer Bewegung nach rechts (was zu einer größeren Unsicherheit führt), (e) eine weiter Messung, (f) der resultierende Belief nach Verarbeitung der Messung, aus *Probabilistic Robotics* [TBF05]

Kapitel 3

Implementierung

In diesem Kapitel wird das entwickelte System vorgestellt. Im ersten Abschnitt wird ein Überblick auf das System und die Verbindung der einzelnen Verfahren gegeben. Im zweiten Abschnitt wird an einem Beispiel der Ablauf des Systems erklärt.

Das gesamte System besteht aus mehreren einzelnen Komponenten und wurde in C++ mit dem ROS-Interface implementiert [ros13a]. Dazu gehört eine ROS-Node für die Beindetektion (im Folgenden „Leg-Detection-Node “) und eine weitere ROS-Node für die Verfolgung und Erkennung (im Folgenden „People-Tracking-Node “).

3.1 Robot-Operating-System

Anders als es der Name vermuten lässt handelt es sich bei **ROS** nicht um ein eigenständiges Betriebssystem. ROS ist ein Meta-Betriebssystem mit dem man einen Roboter steuern kann. Dafür stellt es verschiedene Dienste zur Verfügung, wie z.B. Hardwareabstraktion, Gerätetreiber und Nachrichtenkommunikation zwischen Programmteilen. Zudem bietet ROS Werkzeuge um auch auf mehreren Computer gleichzeitig ausgeführt zu werden.

ROS ist modular aufgebaut. Es besteht aus mehreren *Packages*, welche einzelne Funktionalitäten bereitstellen. Ein Package kann eine ausführbare *Node*, *Messages* und *Service* zur Kommunikation mit anderen Nodes bereitstellen. Mehrere Packages können zu einem Stack zusammengefasst werden, wenn diese zusammen eine Funktion bereitstellen. Desweiteren gibt es den *Master*. Der Master dient als Look-Up Tabelle für die Nodes. Jede Node registriert sich beim Master und kann bei diesem Anfragen für Nachrichten und andere Nodes stellen. Der Master leitet nötige Informationen an die einzelnen Nodes, damit diese dann direkt miteinander kommunizieren können. In Abbildung 3.1 ist beispielhaft die Kommunikation von

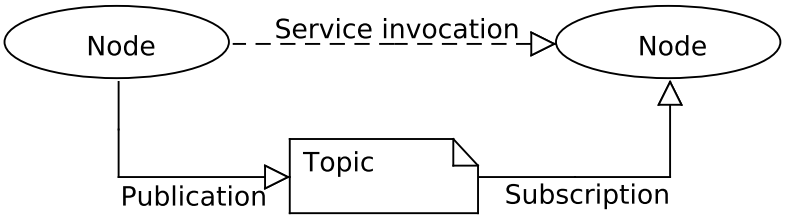


Abbildung 3.1: Basiskonzept von ROS, nach [ros13b]

zwei Nodes dargestellt. Die linke Node veröffentlicht eine Nachricht welche von der rechten Node abonniert wurde. Die Nachricht wurde mit einem speziellen *Topic* versehen. Ein Topic ist ein Identifier, um verschiedene Nachrichten auseinander halten zu können, so veröffentlicht die Leg-Detection-Node eine Nachricht mit dem Topic „leg “, mit Informationen über detektierte Beine. Bei Nachrichten handelt es sich um asynchrone Kommunikation, es ist möglich, dass eine Nachricht auch von mehreren Node abonniert werden kann. Die rechte Node bietet einen Service an, welche von der linken Node angefragt wird (Service invocation). Ein Service ist eine synchrone Kommunikation zwischen zwei Nodes, die Anfrage ist gerichtet und erwartet eine Antwort.

ROS wird stetig weiterentwickelt. Die aktuelleste Version ist „Hydro “vom 4. September 2013, die nächste Version „Indigo “soll im Mai 2014 erscheinen. Zu Beginn dieser Arbeit wurde bei Lisa die Version „Fuerte “verwendet, daher ist das Programm auch für Fuerte erstellt.

3.2 Systemüberblick

Wie oben genannt besteht das System aus zwei Nodes, der Leg-Detection-Node und der People-Tracking-Node. Im folgenden werden die beiden Nodes vorgestellt, sowie die Nachrichten, die von den beiden Nodes veröffentlicht werden.

3.2.1 Messages

Leg-Detection-Message: Die Leg-Detection-Message enthält Informationen zu allen detektierten Beinpaaren. Als Informationen werden die Position in x und y-Richtung in Weltkoordinaten und der Radius des Beinpaares mitgegeben. Die Nachricht wird unter dem Topic „leg“ veröffentlicht.

People-Tracking-Message: Die People-Tracking-Message enthält alle Informationen über aktuell verfolgte Personen. Als Informationen werden die Position in x- und y-Richtung in Weltkoordinaten, die Größe und Orientierung der Posi-

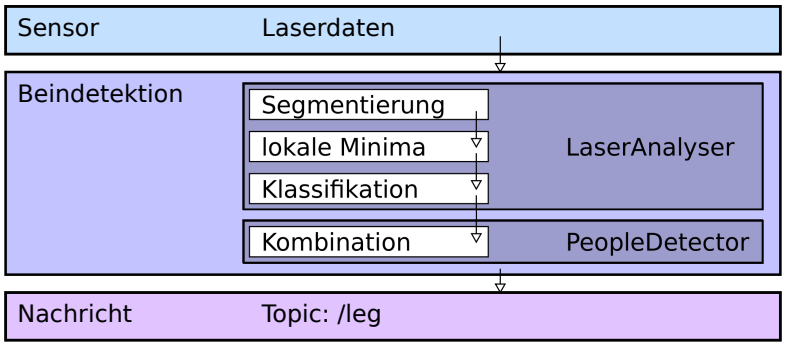


Abbildung 3.2: Ablauf der Beindetektion

tionsgenauigkeit, sowie eine eindeutige ID mitgegeben. Die Nachricht wird unter dem Topic „tracked_people“ veröffentlicht.

3.2.2 Nodes

Leg-Detection-Node:

Die Leg-Detection-Node läuft ab wie in Kapitel 2.1 beschrieben. Abbildung 3.2 zeigt den Ablauf der Beindetektion. Nach Eintreffen von Laserdaten aus dem Sensor gelangen diese zuerst in den „Laser2dAnalyser“. Der Laser2dAnalyser ist eine Bibliothek zum Verarbeiten von 2-dimensionalen Laserdaten. Im ersten Schritt werden die Daten nach der Jump-Distance-Clustering Methode segmentiert, um anschließend auf lokale Minima untersucht zu werden. Die verbleibenden Segmente werden dann noch klassifiziert. Die Klassifikationsbedingung ist, dass nur konvexe Segmente erhalten bleiben. Die übrig gebliebenen Segmente werden nun als Hypothesen einzelner Beine betrachtet. Im letzten Schritt der Beindetektion werden diese einzelnen Beine vom der Bibliothek „PeopleDetector“ kombiniert. Es werden nur Beinpaare akzeptiert. Einzelne Beine werden verworfen.

People-Tracking-Node:

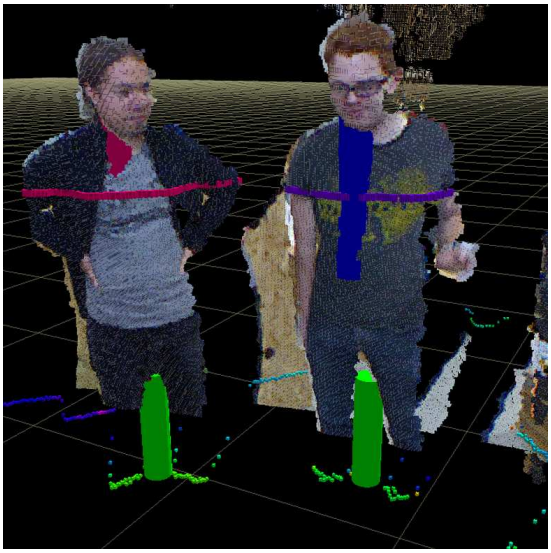
Die People-Tracking-Node beinhaltet das OnlineBoosting aus Kapitel 2.2 und den Kalman-Filter aus Kapitel 2.3. In Abbildung 3.5 ist der Ablauf des People-Tracking-Node dargestellt. Auf den genauen Ablauf wird später noch eingegangen. In der Abbildung sieht man 3 hellgraue Kästen, welche für die 3 wichtigen Libraries der Simple-Person-Detection, dem GMPHDF und dem Classifier stehen. Zuerst werden die einzelnen Libraries erklärt und dann deren Zusammenarbeit.

Simple-Person-Detection: Mit der Simple-Person-Detection sollen fehlerhafte Beindetektionen verworfen werden und zugleich eine genaue Position der Person

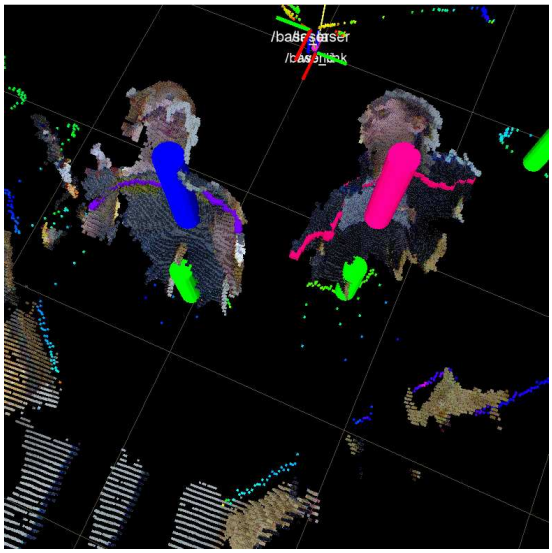
im Bild bestimmt werden, für den Fall dass die Person nicht gerade über den bestimmten Mittelpunkt der Beindetektion steht, sondern den Körperschwerpunkt auf das rechte oder linke Bein verlagert hat. Dies basiert auf der Idee von Carballo et al. [COY09] 2 Laserscanner (LRF) auf unterschiedlicher Höhe anzubringen, um damit Beine und Oberkörper einer Person zu detektieren. Da *Lisa* nur über einen LRF verfügt wird für die Oberkörperdetektion die Kinect genutzt. Häufig werden Tische und Stühle fehlerhaft als Person detektiert. Dazu benötigt die Simple-Person-Detection die detektierten Beine und die Kinect Punktewolke. Im ersten Schritt wird aus der Punktewolke ein horizontales Tiefenbild ausgeschnitten, sodass ein 2-dimensionales Tiefenbild ähnlich dem des 2d-Laserscanners erhalten bleibt. Dieses Tiefenbild befindet sich auf ungefähr 150 Zentimeter Höhe. Das Tiefenbild wird dann mittels Jump-Distance-Clustering segmentiert und anschließend klassifiziert. Als Klassifikationsbedingung gilt hier die Breite eines Segments. Zu kleine Segmente werden verworfen, als Mindestgröße gilt hier 50 Zentimeter. Die übrig bleibenden Segmente werden als Hypothesen für Oberkörper gesehen und nun mit vorhandenen Beindaten kombiniert. Es wird geprüft, ob im Umkreis von 40 Zentimeter um ein Beinpaar ein Oberkörpersegment vorhanden ist. Ist dies der Fall, so wird das Beinpaar als „bestätigt“ markiert. In Abbildung 3.3 (a) und (b) ist die Oberkörperdetektion zu sehen. Der blaue und rosa Zylinder stellen die Mittelpunkte der detektierten Oberkörper da. Die grünen Zylinder die Mittelpunkte von detektierten Beinpaaren. Mit der bestimmten Position des Mittelpunktes des Oberkörpers ist es nun möglich einen Bildbereich (*Patch*) des RGB-Bildes zu bestimmen. In (d) sind die zwei zugehörigen Patches mit blauen Kästen markiert. Ein Patch hat eine Größe von 40x40 Zentimeter und wird auf 1.5 Meter Höhe aus dem Bild ausgeschnitten.

GMPHDF: Ist eine generische Bibliothek für Multi-Target-Tracking auf Basis des Probability-Hypothesis-Density-Filter (PHDF). Die Bibliothek enthält eine Implementierung des Gaussian-Mixture-PHDF (GMPHDF), wie in [VM06] vorgestellt mit Track-Management und Datenassoziation Schema aus [PCV09]. Die in der Arbeit entwickelte erscheinungsbasierte Datenassoziation wurde in eine existierende Software integriert.

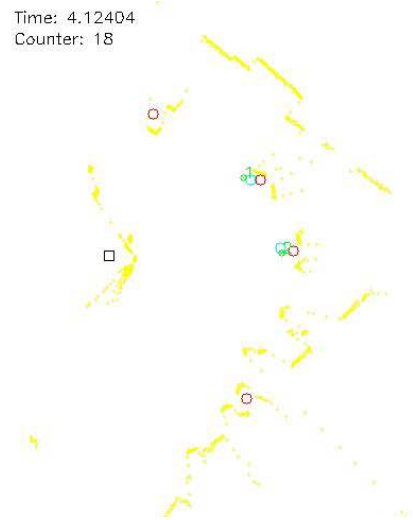
Classifier: Der Classifier basiert auf dem „BoostingTracker“ von [GGB13]. Die benötigten Schnittstellen wurden an das System angepasst und ein Lernverfahren für Patches integriert. Ein ausgeschnittenes Patch ist etwas größer als der eigentliche zu lernende Bildbereich. Ein ausgeschnittes Patch hat die Größe von 54x54 Pixel, der zu lernende Bereich aber nur 48x48 Pixel. Es soll nicht nur der eigentliche (kleinere) Patch gelernt werden, sondern auch die Umgebung. Ziel davon ist es, einen größeren Bereich zu lernen um bei leichten Drehungen oder Verschiebun-



(a) Vorne



(b) Hinten



(c) Karte

(d) Bild der Szene

Abbildung 3.3: Übersicht einer Szene mit Darstellung der Bein- und Oberkörperdaten

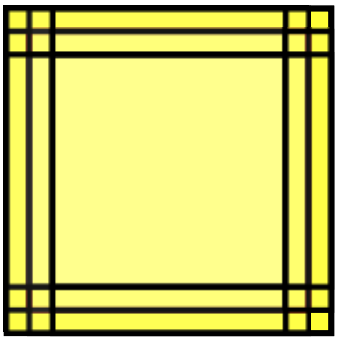


Abbildung 3.4: 9 zu lernende Bildbereiche

gen trotzdem eine gute Erkennung zu ermöglichen. Abbildung 3.4 zeigt ein 3x3 Patch-Gitter. Bei dem Lernprozess werden immer abwechselnd positive und negative Beispiele gelernt. Als negative Beispiele werden die vier Ecken sowie andere sichtbare Personen genutzt. Als positive Beispiele gelten die 9 kleinen Patches.

3.3 Systemablauf

In diesem Abschnitt wird der Systemablauf beschrieben. Zudem soll an einem Beispiel die Idee der Kombination von Kalman-Filter und Online-Boosting verdeutlicht werden.

In Abbildung 3.5 ist ein Aktivitätsdiagramm des People-Tracking-Modules dargestellt. Beim Starten der Node werden die einzelnen Unterprogramme, Simple-Person-Detection, GMPHDF und Classifier gestartet. Ab dann reagiert das Programm auf eingehende Nachrichten. Es gibt zwei Nachrichten, die zu Aktionen führen, zum einen die Leg-Detection-Message und zum anderen die Sensor-Message der Kinect mit der registrierten Punktewolke.

Ablauf nach Erhalt einer Leg-Detection-Message: Nach dem Erhalt einer Leg-Detection-Message werden die in der Nachricht enthaltenen Positionen direkt genutzt um den Kalman-Filter zu aktualisieren. Es erfolgt eine Prädiktion mit δt, wobei δt die Zeitdifferenz zwischen der letzten und der jetzigen Aktualisierung des Kalman-Filters ist. Dann werden die Positionen als Messung genutzt um eine Korrektur der Schätzung durchzuführen. Anschließend werden die neuen bestimmten Positionen mit der People-Tracking-Message verschickt.

Ablauf nach Erhalt einer Sensor-Message: Damit dieser Schritt durchgeführt werden kann, ist es nötig, dass unmittelbar vorher ($<\frac{1}{2}$s) eine Leg-Detection-Message empfangen wurde. Sollte die letzte Leg-Detection-Message zu „alt “sein,

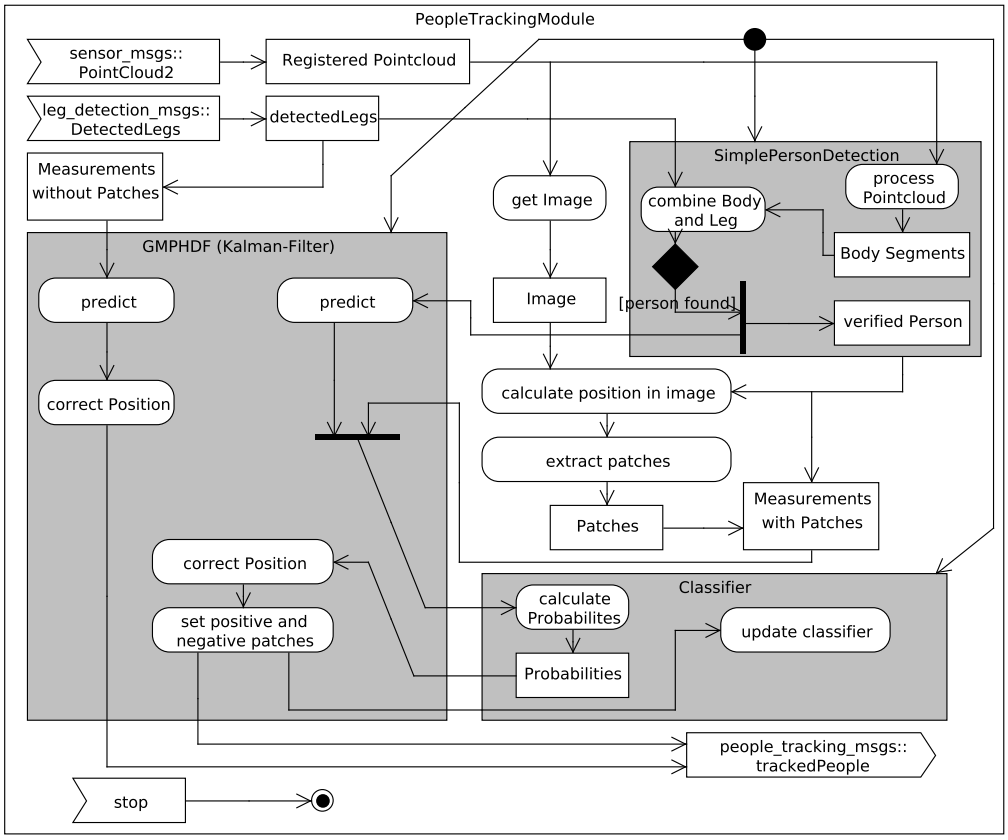


Abbildung 3.5: Übersicht der PeopleTrackingModules

kann nicht mehr sichergestellt werden, dass sich Personen noch in einer akzeptablen Distanz zur letzten Beinmessung befinden. Wenn die Bedingung erfüllt ist, wird die Punktwolke - wie im vorherigen Abschnitt beschrieben - von der Simple-Person-Detection verarbeitet. Für bestätigte Beinpaare wird im folgenden Schritt ein Patch im RGB-Bild ausgeschnitten und zusammen mit der Position in Weltkoordinaten an den Kalman-Filter übergeben. Es erfolgt eine Prädiktion. Im Korrekturschritt wird nun mit der bildbasierten Datenassoziation gearbeitet. Es wird abgefragt wie ähnlich ein gefundener Patch zu vorhandenen Online-Boosting Klassifikatoren passt. Mit den Ähnlichkeitswerten wird dann die Korrektur durchgeführt. Danach werden die Patches für jeden Online-Boosting Klassifikator in positiv und negativ Beispiele eingeteilt, um dann die Klassifikatoren zu aktualisieren. Zum Schluss werden die aktualisierten Positionen mit der People-Tracking-Message verschickt.

Beispiel

In diesem Abschnitt soll beispielhaft das Zusammenwirken von Kalman-Filter und Online-Boosting für die Personenverfolgung mit der People-Tracking-Node in der Theorie gezeigt werden. Das Beispiel ist in Abbildung 3.6 schematisch dargestellt und stark vereinfacht. In dem Beispiel werden zwei Durchläufe des Kalman-Filters gezeigt. Im ersten Durchlauf findet eine Datenassoziation auf Grundlage der Beindaten statt, im zweiten Durchlauf auf Grundlage von RGBD-Daten und dem Online-Boosting.

(a) **Ausgangssituation - 1. Prädiktion:** In diesem Beispiel wird angenommen, dass die beiden Personen schon längere Zeit zu sehen sind und für beide Personen schon ein Online-Boosting vorhanden ist. Die beiden Personen stellen in diesem Schritt den geschätzten Mittelwert und die Kreise die Kovarianz dar. Farblich sind die Kreise der „T-Shirt-Farbe“ zugeordnet. In diesem Schritt ist eine Prädiktion eines nicht dargestellten vorherigen Zustandes zu sehen.

(b) **1. Messung:** Nun ist eine Messung mit der Beindetektion erfolgt und die tatsächliche Position der Personen ist eingezeichnet. Es wurden die Beine der blauen Person detektiert, dargestellt durch den roten Strich. Die grüne Person ist für die Beindetektion nicht zu sehen.

(c) **1. Korrektur:** Auf Grundlage der Prädiktion und der Messung ist nun eine Korrektur durchgeführt worden. Die Beine der blauen Person wurden in diesem Schritt fälschlicherweise der grünen Person zugeordnet. Die Kovarianz der blauen Person ist unverändert, da angenommen wurde, dass diese Person nicht zu sehen sei und deshalb keine Messung assoziiert wurde.

(d) **2. Prädiktion:** Er ist eine erneute Schätzung durchgeführt worden. Für die blaue Person (blauer Kreis) ist die Kovarianz stark angewachsen, da für diese im vorherigen Schritt keine Korrektur durchgeführt wurde. Die Mittelpunkte der beiden Kreise entsprechen der wahrscheinlichsten Position.

(e) **2. Messung:** Bei dieser Messung handelt es sich um eine Messung mit RGBD-Daten, es konnten Bildpatches für beide Personen erstellt werden (rote Quadrate). Die beiden Personen sind wieder an ihrem tatsächlichen Positionen eingezeichnet.

(f) **2. Korrektur:** Im letzten Schritt des Beispiels ist erneut eine Korrektur durchgeführt worden. Die fehlerhafte Assoziation der Beindaten konnte durch die Bilddatenassoziation korrigiert werden und die Vertauschung der Personen liegt nun nicht mehr vor.

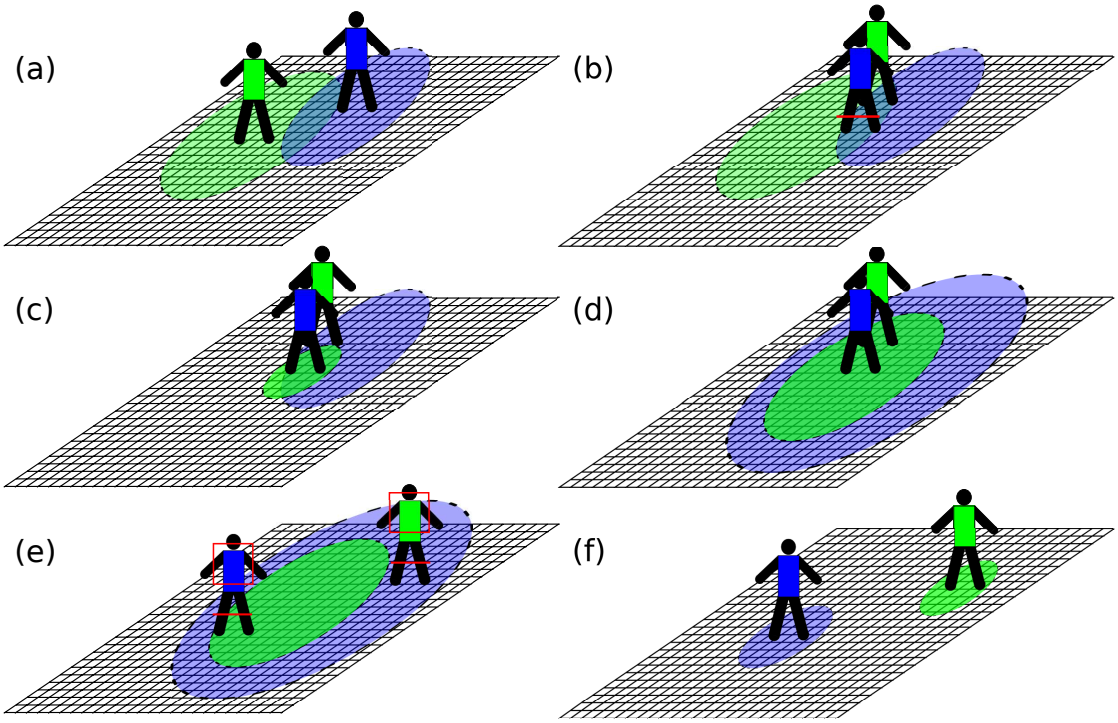


Abbildung 3.6: Schematischer Ablauf einer Korrektur mit Online-Boosting - (a) Ausgangssituation - 2 Personen werden verfolgt (Schätzung), (b) die Personen kreuzen sich wobei nur die vordere (blaue) durch Beindetektion gesehen werden kann, (c) fehlerhafte Zuordnung der Messung (Korrektur), (d) Schätzung der Personenbewegung, (e) Messung mit Boosting und Beindetektion, (f) Korrektur der vorherigen fehlerhaften Zuordnung.

3.4 Zusammenfassung

In diesem Kapitel wurde die Implementierung des Kalman-Filters und des Online-Boosting in der People-Tracking-Node vorgestellt, sowie die Beindetektion in der Leg-Detection-Node. Es wurden die Abläufe der der Leg-Detection-Node und People-Tracking-Node erklärt und in einem Beispiel gezeigt wie der Kalman-Filter und das Online-Boosting zusammen funktionieren sollen. Desweiteren wurden die erstellten Nachrichten zur Kommunikation mit anderen Nodes von Lisa vorgestellt.

Kapitel 4

Evaluation

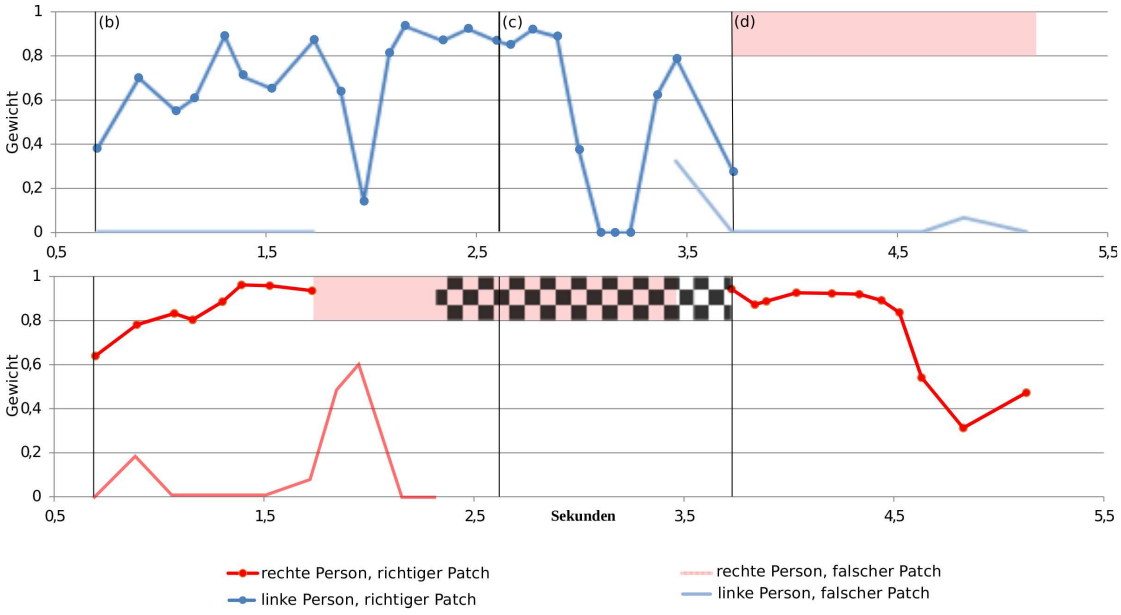
In diesem Kapitel wird die Arbeit evaluiert. In Abschnitt 4.2.1 wird die Implementation auf das Ziel der Personenerkennung nach einer zeitweisen Verdeckung untersucht. In Abschnitt 4.2.2 wird ein Vergleich zwischen dem Tracking mit und ohne Online-Boosting angestellt. Zudem wird die Geschwindigkeit der Implementation in Abschnitt 4.2.3 betrachtet. Zum Schluss werden in Abschnitt 4.3 die Ergebnisse zusammengefasst und eine Schlussfolgerung gezogen.

4.1 Testumgebung und Testszenario

Testumgebung Zum Testen wurde ein Notebook mit 2 GHz Intel Core 2 Quad-Prozessor und 8192 MB-RAM genutzt. Die aufgezeichneten Bilddaten des ROS-Bag-Files haben eine Auflösung von 640x480 Pixeln mit 30Hz. Der Sick Laserscanner hat einen Aufnahmebereich von 270° mit 25Hz, wobei aufgrund des Aufbaus von Lisa nur 180° genutzt werden.

Testszenarion In dem Testszenario gibt es zwei Personen (Abb. 4.1(b)). Die linke Person (grauer Pollover) geht in dem Szenario im Bild von links nach rechts. Dabei wird sie eine Zeit lang die rechte Person (blauer Pullover) verdecken, sodass diese weder von der Kinect noch vom Laserscanner wahrgenommen werden kann (Abb. 4.1(c)). Danach bewegt sich die linke Person nach rechts aus dem Bild und die rechte nach links (Abb. 4.1(c)). Ziel ist es, dass die rechte Person nach der Verdeckung wiedererkannt wird.

Konfiguration des Systems Die Patches zur Erkennung wurden auf 1.25m Höhe ausgeschnitten. Sie haben eine reale Größe von 0.4mx0.4m und werden auf 44x44 Pixeln skaliert. Dieses Patch wird nun in neun sich überlappende kleinere Patches mit der Größe 40x40 Pixeln aufgeteilt, welche auf zum Erkennen genutzt



(a)

(b) (c) (d)

Abbildung 4.1: Grafiken zum Testszenario

werden. Es werden 100 Selektoren mit einem Feature-Pool von 1000 Features für jeden Online-Booster erstellt. Für die Simple-Person-Detection wird eine Mindestgröße vom 0.6m für ein Segment erwartet.

4.2 Ergebnisse der neuen Implementierung

4.2.1 Erkennung

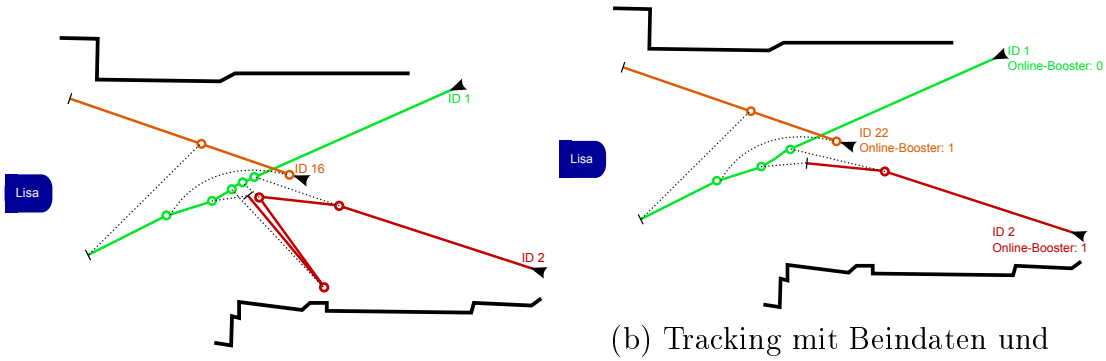
Abbildung 4.1 zeigt das Testszenario. In (a) sind die Gewichte der zwei Online-Booster über die Zeit des Tests für die linke und rechte Person zu sehen (Abb. 4.1(b)). Zu Beginn bewegen sich beide Personen nebeneinander und es wird für beide ein Online-Booster erstellt. Im Zeitraum von 0.7s bis 1.7s sind beide Perso-

nen zu sehen. In diesem Zeitraum haben beide Personen, neben der Initiierung, acht weitere Zeitpunkte in denen gelernt werden kann. In dieser Lernphase kann man gut erkennen, dass die beiden Online-Booster in ihrer Gewichtung steigen. Die rote Kurve mit Punkten steigt fast kontinuierlich an, die blaue Kurve mit Punkten schwankt etwas. Diese beiden Kurven zeigen die Gewichtung bezogen auf den der Person zugehörigen Patch. Beide Kurven zeigen in dieser Phase eine große Gewichtung. Interessant ist auch die Betrachtung der Gewichte der falschen Patches, also denen der jeweils anderen Person. Hier ist eindeutig zu Erkennen, dass ein falscher Patch auch als ein solcher identifiziert wird. Die hellblaue Kurve ist in dieser Phase dauerhaft bei null. Auch die hellrote Kurve der rechten Person ist größtenteils bei null. Nur zu Beginn und Ende dieser Phase steigt die Kurve ein wenig an.

In der zweiten Phase ist die rechte Person nun nicht mehr zu sehen. Der rote Balken im Bereich von 0,8 bis 1,0 auf der vertikalen Achse und 1,8s bis 3,4s auf der horizontalen Achse zeigt den Zeitraum an, in welchem die rechte Person nicht zu sehen ist. Zu Beginn dieser Phase steigt die hellrote Kurve für den falschen Patch stark an, es kommt zu einer fehlerhaften Zuordnung des Patches. Dieser Fehler ist aber nur kurz. Die Kurve fällt schnell wieder ab und der Patch wird richtigerweise als falsch identifiziert. Zu diesem Zeitpunkt wird dann auch der Online-Booster von der rechten Person auf inaktiv gesetzt und ihr Track im Kalman-Filter wird verworfen. Dies ist angezeigt durch das Schachbrettmuster. Betrachten wir nun den Verlauf der Kurve der linken Person: Zu Beginn fällt das Gewicht der Erkennung stark ab und es fällt die Parallelität zur hellroten Kurve sofort auf. Die Patches zu den Zeitpunkten 1,8s und 2,0s scheinen schwer zu klassifizieren zu sein. Wie auch schon bei der hollroten Kurve ist dieser Fehler nur kurz. Es folgt ein Zeitraum in welchem die Patches gut erkannt werden. Im Zeitraum von 3,1s bis 3,3s werden die richtigen Patches der linken Person nicht erkannt. Bevor die Person den Sichtbereich verlässt und die rechte Person wieder auftaucht (Abb. 4.1(d)) wird sie aber nochmals erkannt.

Ab Zeitpunkt 3,7s ist die linke Person nicht mehr zu sehen. Zu diesem Zeitpunkt wird der Online-Booster der rechten Person wieder aktiviert, da diese in den Bilddaten erkannt wurde. Die Aktivierung dauert einen kurzen Zeitraum (von 3,5s bis 3,7s), da erst vom Kalman-Filter eine Verfolgung initialisiert werden muss, bevor eine optische Wiedererkennung oder das Erstellen eines neuen Online-Boosters durchgeführt wird.

In der letzten Phase von 3,7s bis 5,2s ist nur noch die rechte Person zu sehen. Die rechte Person wird in diesem Zeitraum gut erkannt. Zum Schluss fällt das Gewicht der Erkennung jedoch ab. Es fällt auf, dass der Online-Booster der linken Person nicht auf inaktiv gesetzt wird. Dies liegt daran, das die Person den Sichtbereich des Roboters verlassen hat, aber noch immer über die Beine wahrgenommen wird,



(a) Tracking nur auf Beindaten

(b) Tracking mit Beindaten und Online-Booster

Abbildung 4.2: Vergleich des Tracking mit und ohne Online-Boosting

also nicht wie im vorherigen Fall der Verdeckung komplett verschwunden ist. Die hellblaue Kurve der linken Person zeigt, das der Patch der rechten Person vom Online-Booster der linken Person als falscher Patch richtig klassifiziert wird.

4.2.2 Vergleich Tracking ohne Online-Boosting

Das Testszenario aus Abschnitt 4.2.1 wurde zudem auch ohne das Online-Boosting getestet. In den meisten Fällen sah das Ergebnis aus wie in Abbildung 4.2(a). Eingezeichnet sind die Pfade der zwei Personen. Die Pfade der IDs 2 und 16 gehören zu einer Person. Die eingezeichneten gepunkteten Linien verbinden immer zwei zeitgleiche Ereignisse. Die Person mit der ID 2 ist meistens in der Zeit der Verdeckung verloren gegangen. Falsch detektierte Beine, oder Beine anderer Personen wurden falsch zugeordnet. Dadurch entstehen Sprünge wie zwischen den Punkten 2 und 3, 3 und 4, 4 und 5. Vergleicht man den im selben Zeitraum zurückgelegten Weg der Person 1, so wird die Dimension der Sprünge deutlich. Nachdem die Person wieder sichtbar ist bekommt sie eine neue ID zugewiesen und wird somit als neue Person betrachtet. In Abbildung 4.2(b) sind die Pfade aus dem Test aus Abschnitt 4.2.1 eingezeichnet. Es ist zu erkennen, dass es nicht zu solch starken und vielen Sprüngen kommt. Auch in diesem Test geht der Pfad der ID 2 verloren und nach der Wiedererkennung hat die Person die ID 22, allerdings wurde ihr auch wieder eine Online-Boosting ID zugewiesen, diese entspricht der von ID 2. Somit ist eine Weiterverfolgung dieser Person möglich. Dieses Beispiel zeigt den Vorteil der Kombination der beiden Verfahren für die Personenverfolgung.

4.2.3 Geschwindigkeit

In diesem Abschnitt wird die Geschwindigkeit der Implementation analysiert. Es wird betrachtet wie viel Zeit verschiedene Bereiche des Verfahrens unter Berück-

sichtigung der Anzahl der Selektoren und der Anzahl der aktiven Online-Booster benötigen. Zudem wird die Geschwindigkeit des kompletten Ablaufs nach Erhalt einer Kinect Punktewolke bis zum versenden der People-Tracking-Message dargelegt. Zur Analyse wurden 50, 100, 150 und 200 Selektoren für einen Online-Booster gewählt. Der zugehörige Menge an Feature im Feature-Pool eines Online-Boosters entspricht der zehnfachen Anzahl an Selektoren, also 500, 1000, 1500 und 2000 Merkmalen.

Anzahl Selektoren	Ø positiv Lernen	Ø negativ Lernen	Ø pos. und neg. Lernen
50	0,00231 s	0,00268 s	0,00240 s
100	0,00688 s	0,00702 s	0,00691 s
150	0,01403 s	0,01473 s	0,01465 s
200	0,02497 s	0,02489 s	0,02495 s

Tabelle 4.1: Zeit zum Lernen eines Patches

Zuerst betrachten wir die Zeit die benötigt wird, um einen einzelnen Patch zu lernen. In Tabelle 4.1 wird dies dargestellt. Es ist zu erkennen, dass das Lernen von negativen und positiven Patches ähnlich viel Zeit erfordert. Da es keinen wesentlichen Unterschied gibt, ist es somit für die Geschwindigkeit des Algorithmus nicht relevant. Vielmehr fällt der Unterschied in Betrachtung der Dauer bei der Anzahl der Selektoren auf. Bei 50 Selektoren wird eine Zeit von durchschnittlich 0,0024s benötigt, bei 100 Selektoren, also dem Zweifachen, wird eine fast schon dreifache Zeit benötigt. Bei 150 die sechsfache und bei 200 die zehnfache Zeit. Es ist zu erkennen, dass es in diesem Wertebereich kein linearer Zusammenhang zwischen Zeit und Anzahl Klassifikatoren besteht.

Anzahl Selektoren	1 Patch Øs	2 Patches Øs
50	0,00368 s	0,00955 s
100	0,00856 s	0,02342 s
150	0,01705 s	0,04799 s
200	0,03138 s	0,08030 s

Tabelle 4.2: Zeit zum aktualisieren eines Online-Boosters

In Tabelle 4.1 wurde gezeigt wie viel Zeit benötigt wird um einen einzelnen Patch zu lernen. Tabelle 4.2 zeigt unter Berücksichtigung der Anzahl der Selektoren, wie lange die Aktualisierung eines Online-Boosters bei einem und bei zwei Patches dauert. Beim Lernen von einem Patch besteht die Möglichkeit, dass der Patch zu der entsprechenden Person gehört, dann wird nur einmal positiv gelernt.

Gehört der Patch nicht zu der Person, so wird einmal negativ mit dem Patch und einmal positiv mit dem letzten positiven Patch der Person gelernt. Es wird also ein- oder zweimal bei einem Patch gelernt. Bei zwei Patches wird zwischen drei- und viermal gelernt. Neben der vorherigen Feststellung, dass mit steigender Anzahl der Selektoren die Zeit stark anwächst, fällt nun noch auf, dass das Lernen von zwei Patches 2,5 bis 2,8-mal soviel Zeit benötigt wie das Lernen eines Patches.

Anzahl	1 Online-Booster		2 Online-Booster	
Selektoren	1 Patch Øs	2 Patches Øs	1 Patch Øs	2 Patches Øs
50	0,07515 s	0,07905 s	0,08054 s	0,09699 s
100	0,07173 s	0,08938 s	0,08834 s	0,11278 s
150	0,07883 s	0,11082 s	0,10422 s	0,16947 s
200	0,08981 s	0,14444 s	0,15060 s	0,23133 s

Tabelle 4.3: Zeit für einen Ablauf

Nachdem betrachtet wurde wie lange es dauert einen Patch zu lernen und einen Online-Booster zu aktualisieren, wird nun die komplette Zeit für einen Ablauf des Verfahrens betrachtet, von dem Erhalt einer Kinect Punktewolke bis zum Versenden der People-Tracking-Message. In Tabelle 4.3 ist eine Übersicht über die verschieden betrachteten Fälle. Wenn es nur einen Online-Booster und einen Patch gibt, dann spielt die Anzahl der Selektoren nur eine geringe Rolle. Vielmehr fällt hier die Geschwindigkeit des restlichen Algorithmus (z.B. Simple-Person-Detection oder das bestimmen der Bildpatches) in das Gewicht. Bei zwei Patches und einem Online-Booster sieht es schon anders aus. Die benötigte Zeit bei 150 und 200 Selektoren ist stark angewachsen, bei 50 und 100 Selektoren ist der Anstieg nicht so stark. Ähnliche Werte gibt es auch bei einem Patch und zwei Online-Booster. Dies kommt daher, da es ähnlich viele Aktualiesierungsvorgänge bei beiden gibt. Im letzten Fall, bei zwei Patches und zwei Online-Booster, steigt die benötigte Zeit bei 150 und 200 Selektoren wieder stark an. Bei 100 Selektoren entspricht die benötigte Zeit nun der bei 150 Selektoren mit zwei Patches und einem Online-Booster. Auffällig ist vor allem die benötigte Zeit bei 200 Selektoren von 0,23133s oder ungefähr 4 Hz. Im Vergleich zu 150 Selektoren mit mit 6 Hz, 100 Selektoren mit 9 Hz und 50 Selektoren mit 10 Hz.

Zum Schluss wird noch die Zeit zum Erstellen eines Online-Boosters untersucht. In Tabelle 4.4 sind die durchschnittlichen Zeiten zum initiieren eines Online-Boosters aufgeführt. Am schnellsten erfolgt dies bei 50 Selektoren mit 0,03190s. Bei 100 Selektoren werden 0,06793s, bei 150 Selektoren 0,16329s und bei 200 Selektoren 0,23064s benötigt. Hier zeichnet sich ein ähnliches Bild wie schon in Tabelle 4.1 ab. Die Steigerung der Zeit erfolgt nicht linear.

Anzahl Selektoren	Ø Zeit
50	0,03190 s
100	0,06793 s
150	0,16329 s
200	0,23064 s

Tabelle 4.4: Zeit zum initialisieren eines Online-Boosters

4.3 Zusammenfassung

In den ersten beiden Abschnitten wurde die Erkennung mit dem Online-Boosting an einem Beispiel erklärt und der Vorteil der Kombination der beiden Verfahren für die Personenverfolgung dargestellt. Danach wurde die Geschwindigkeit des Online-Boosting analysiert.

Im Rahmen der Evaluation und vieler Tests mit verschiedenen Parametern wurde herausgefunden, dass das Online-Boosting mit 100 Selektoren stabil funktioniert. Es ist damit möglich fehlerhaft positive Erkennungen zu vermeiden, wodurch eine Wiedererkennung schon bei niedrigem Online-Boosting Gewicht möglich ist. Mit 100 Selektoren erreicht das Online-Boosting zudem eine gute Geschwindigkeit von 9 Hz bei zwei sichtbaren Personen. Vergleicht man die benötigte Zeit für einen Ablauf mit 50 Selektoren, so fällt auf, dass es nur unwesentlich länger braucht. Hingegen ist es im Vergleich zu 150 oder 200 Selektoren wesentlich schneller. Die Wahl auf 100 Selektoren beruht darauf, da dieser Wert einen guten Kompromiss zwischen benötigter Zeit und Erkennung bietet. Mehr Selektoren würden zwar die Erkennung verbessern, und damit auch die Unterscheidung von Personen mit sehr ähnlichen Merkmalen verstärken, jedoch geht dies stark zu Lasten der Geschwindigkeit.

Kapitel 5

Zusammenfassung

5.1 Fazit

In dieser Arbeit wurde eine neue Personenverfolgung für den Serviceroboter Lisa entwickelt. Die vorherige Personenverfolgung durch die Auswertung von Beindaten kombiniert mit einem Partikelfilter wurde durch ein neues System ersetzt: Dieses System verwendet neben den Beindaten auch Bilddaten und kombiniert diese in einer Implementation des Gaussian-Mixture-Probability-Hypothesis-Density-Filter mit Track-Management und Datenassoziation Schema. Das System ist modular aufgebaut und besteht aus einzelnen Nodes und Bibliotheken, welche auch für anderen Aufgaben genutzt werden können.

Die Beindetektion wurde aus dem Robbie-Framework nach ROS portiert und entsprechend den neuen Richtlinien für modularen Aufbau umgestaltet. Robbie-Worker für die Beindetektion wurden in Bibliotheken umgewandelt. Die neue Node zur Personenverfolgung ist ebenfalls modular aufgebaut und besteht aus einzelnen Bibliotheken und Nodes. Neu hinzugekommen sind eine Bibliothek für Track-Management und Datenassoziationschema, den GMPHDF, eine Bibliothek für eine einfache Personendetektion, dem Simple-Person-Detection und eine Bibliothek für Online-Boosting.

Die Schwierigkeit dieser Arbeit bestand darin ein Lernverfahren zu entwickeln, welches schnell, aber auch gut genug ist, um eine Erkennung zu ermöglichen. Da in den RoboCup-Challenges nur begrenzt Zeit zur Verfügung steht und speziell bei *Follow Me* die Schnellsten die meisten Punkte holen, ist eine lange Lernphase des Online-Boosting hinderlich. In einem Test konnte gezeigt werden, dass eine kurze Lernphase mit wenigen positiven und negativen Beispielen ausreicht, um eine sichere Erkennung zu ermöglichen und Verwechslungen von Personen zu verhindern.

Des Weiteren ist die Kombination von Bilddaten und Beindaten wichtig. Durch die häufigen Messungen von Beinen mit 25Hz ist es möglich den Kalman-Filter in

kurzen Intervallen zu aktualisieren, sodass eine stabile Verfolgung mit großer Positionssicherheit gewährleistet ist. Zudem kann diese Positionssicherheit mittels niedriger frequentierter Bilddaten bestätigt oder korrigiert werden. Außerdem wurde in einem Test zur Personenverfolgung, der ausschließlich auf Beindaten basierend, gezeigt, dass die Gefahr des Verlustes einer verfolgten Person durch Verdeckung besteht. Mit dem Online-Boosting konnte diese Gefahr verringert werden, da eine Wiedererkennung der Person möglich ist.

Eine weitere Schwierigkeit bestand in der Bestimmung von Patches. Es konnte nicht einfach im gesamten Bild nach einer Übereinstimmung gesucht werden, da dies zu langsam ist. Ebenfalls war es nicht möglich einen Bildbereich über detektierte Beine zu bestimmen, da es beim Gehen zu Gewichtsverlagerungen von einem auf das andere Bein kommt und somit der Körpermittelpunkt nicht über der Mitte der Beine ist. So wurde mittels der Simple-Person-Detection ein Verfahren entwickelt, welches in einer Punktewolke nach Oberkörpersegmenten sucht. Diese Segmente mussten jedoch groß genug sein, um als Oberkörper akzeptiert zu werden. Zudem wurden nur solche Segmente akzeptiert, in deren unmittelbarer Nähe zum Segmentmittelpunkt mindestens ein Bein kurz vorher gesehen wurde. Gefundene und bestätigte Segmente werden dann genutzt um den richtigen Bereich im Bild zu bestimmen. Der Segmentmittelpunkt wird in das Kamerabild transformiert und dann ein Bildbereich in einer festen Größe ausgeschnitten. Da es aber auch hier noch zu Problemen bei der Erkennung kam, wurde der Bildbereich ein wenig vergrößert, um Translationen ausgleichen zu können, und in einem Patch gespeichert.

Die dadurch bestimmte Position des Segmentmittelpunktes und der Patch werden an den GMPHDF übergeben. Dieser ermittelt dann die wahrscheinlichste neue Position einer Person auf Grundlage der Messungen. Bei Beindaten kann nur ein Gewicht über die Position bestimmt werden, bei der Messung mit Bilddaten hingegen stehen neben der Position auch das Gewicht der Messung des Online-Boosters zur Verfügung, um den GMPHDF zu aktualisieren. Bei dieser Aktualisierung wird den Personen der wahrscheinlichste Bildpatch zugeordnet.

Nach dem Aktualisieren wird der neu zugeteilte Patch gelernt. Da der Patch etwas größer ist als der eigentliche Lernpatch, wird in dem Patch nach einem der Größe des Lernpatch entsprechenden Bereichs gesucht. Der Bereich mit der besten Übereinstimmung wird zum positiven Lernen genutzt. Die zugeteilten Patches von anderen Personen werden zum negativ Lernen genutzt. Es konnte gezeigt werden, dass dieser Lernvorgang schnell und ausreichend genug ist, um eine stabile Erkennung zu ermöglichen.

Nach jedem Aktualisierungsvorgang werden die neu bestimmten wahrscheinlichsten Positionen der Personen mittels der People-Tracking-Message versendet. Die Nachricht enthält neben der Position auch die dazugehörige Gewichtung, ei-

ne eindeutige ID durch den GMPHDF und falls verfügbar eine eindeutige ID des zugehörigen Online-Boosting.

5.2 Ausblick

Das System ist für die Personenverfolgung entwickelt worden, wie es zum Beispiel in den RoboCup@Home-Challenges *Follow Me* und *Restaurant* benötigt wird. Es besteht aber auch die Möglichkeit dieses System für weitere Challenges anzupassen, z.B. für Cocktailparty, wo es das Ziel ist bestimmten Personen ein Getränk zu liefern. Mit dem Online-Boosting können die Personen gelernt und anschließend schnell identifiziert werden. Da Online-Boosting auf AdaBoost basiert und dies erfolgreich in der Gesichtserkennung getestet wurde ist eine Erweiterung dahingehend schnell und einfach umzusetzen.

Literaturverzeichnis

[Bar98] Bartlett, Peter L.: The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. In: *IEEE Transactions on Information Theory* 44 (1998), Nr. 2, S. 525–536

[BBCT05] Bennewitz, Maren ; Burgard, Wolfram ; Cielniak, G. ; Thrun, S.: Learning Motion Patterns of People for Compliant Robot Motion. In: *International Journal of Robotics Research* 24 (2005), Nr. 1

[BH89] Baum, Eric B. ; Haussler, David: What size net gives valid generalization? In: *Neural Computation* 1 (1989), Nr. 1, S. 151–160

[Bre98] Breiman, Leo: Arcing classifiers. In: *The Annals of Statistics* 26 (1998), Nr. 3, S. 801–849

[COY09] Carballo, Alexander ; Ohya, Akihisa ; Yuta, Shiníchi: Multiple People Detection from a Mobile Robot using Double Layered Laser Range Finders. In: *International Conference on Robotics and Automation*, 2009

[DC96] Drucker, Harris ; Cortes, Corinna: Boosting decision trees. In: *Advances in Neural Information Processing Systems* 8 (1996), S. 479–485

[DSS93] Drucker, Harris ; Schapire, Robert ; Simard, Patrice: Boosting performance in neural net works. In: *International Journal of Pattern Recognition and Artificial Intelligence* 7 (1993), Nr. 4, S. 705–719

[FS97] Freund, Y. ; Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. In: *Journal of Computer and System Sciences* 55 (1997), Nr. 1, S. 119–139

[FSA99] Freund, Y. ; Schapire, R. ; Abe, N.: A short introduction to boosting. In: *Journal-Japanese Society For Artificial Intelligence* 14 (1999), Nr. 771-780, S. 1612

[GB06] Grabner, Helmut ; Bischof, Horst: On-line Boosting and Vision. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* Bd. 1, 2006, S. 260–267

[GGB06] Grabner, Helmut ; Grabner, Michael ; Bischof, Horst: Real-Time Tracking via On-line Boosting. In: *Proceedings British Machine Vision Conference (BMVC)*, 2006, S. 47–56

[GGB13] Grabner, Helmut ; Grabner, Michael ; Bischof, Horst: *On-Line boosting for tracking.* `http://www.vision.ee.ethz.ch/boostingTrackers/onlineBoosting.htm`. Version: Dezember 2013

[KV94] Kearns, Michael ; Valiant, Leslie G.: Learning Boolean formulae or finite automata is as hard as factoring. In: *Journal of the Association for Computing Machinery* 41 (1994), Nr. 1, S. 67–95

[OR01a] Oza, Nikunj C. ; Russell, Stuart: Experimental Comparisons of Online and Batch Versions of Bagging and Boosting. In: *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* New York, NY, USA : ACM, 2001 (KDD '01). – ISBN 1–58113– 391–X, S. 359–364

[OR01b] Oza, Nikunj C. ; Russell, Stuart: Online Bagging and Boosting. In: *In Artificial Intelligence and Statistics 2001*, Morgan Kaufmann, 2001, S. 105–112

[PCV09] Panta, Kusha ; Clark, Daniel E. ; Vo, Ba-Ngu: Data Association and Track Management for the Gaussian Mixture Probability Hypothesis Density Filter. In: *IEEE Transactions on Aerospace and Electronic Systems* 45 (2009), Nr. 3, S. 1003–1016

[PN05] Premebida, Cristiano ; Nunes, Urbano: Segmentation and geometric primitives extraction from 2D raser range data for mobile robot applications. In: *Robótica 2005 - Actas do Encontro Científico*, 2005

[Qui93] Quinlan, J. R.: *C4.5: Programs for Machine Learning.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 1993. – ISBN 1–55860–238–0

[Qui96] Quinlan, J. R.: Bagging, boosting, and C4.5. In: *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, 1996, S. 725–730

[ros13a] ROS.ORG: *Robot Operatin System.* `http://wiki.ros.org/`. Version: Dezember 2013

[ros13b] ROS.ORG: *ROS Basic Concept.* `http://wiki.ros.org/ROS/Concepts?action=AttachFile&do=get&target=ROS_basic_concepts.png`. Version: Dezember 2013

[SFBL98] SCHAPIRE, R. ; FREUND, Y. ; BARTLETT, Peter L. ; LEE, Wee S.: Boosting the mar- gin: A new explanation for the effectiveness of voting methods. In: *The Annals of Statistics* 26 (1998), Nr. 5, S. 1651–1686

[SS08] SPINELLO, L. ; SIEGWART, R.: Human Detection using Multimodal and Multidimensional Features. In: *Proc. of The International Conference in Robotics and Automation (ICRA)*, 2008

[TBF05] THRUN, Sebastian ; BURGARD, Wolfram ; FOX, Dieter: *Probabilistic Robotics.* MIT Press, 2005

[TV00] TIEU, Kinh ; VIOLA, P.: Boosting Image Retrieval. In: *Proceedings IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 2000, S. 228–235

[Val84] VALIANT, L.G.: A theory of the learnable. In: *Communications of the ACM* 27 (1984), Nr. 11, S. 1134–1142

[VM06] VO, Ba-Ngu ; MA, Wing-Kin: The Gaussian Mixture Probability Hypothesis Density Filter. In: *IEEE Transactions on Signal Processing* 54 (2006), Nr. 11

[XPCR05] XAVIER, Joâo ; PACHECO, Marco ; CASTRO, Daniel ; RUANO, António: Fast line, arc/circle and leg detection from laser scan data in a player driver. In: *in Proc. of the IEEE Int. Conference on Robotics Automation (ICRA05*, 2005