Multi-Target Tracking Algorithmen

für die humanoide Roboterplattform Nao

Bachelorarbeit

im Fachgebiet Informatik

vorgelegt von:	Hannah Maria Müller
Fachbereich:	Informatik
Erstprüfer:	Prof. Dr. rer. nat. Johannes Waldmann
Zweitprüfer:	Prof. Dr. rer. nat. habil. Siegfried Schönherr

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst habe, und keine anderen Quellen und Hilfsmittel verwendet habe, also die angegebenen.

Leipzig, den 20.09.2013,

HANNAH MARIA MÜLLER

Zusammenfassung

Um in der Standard Plattform Liga des RoboCup autonom Fußball spielen zu können, ist es nötig alle Roboter auf dem Fußballfeld zuverlässig zu erkennen. Objekterkennung ist jedoch immer mit Fehlerkennungen belastet. Diese können mit Mitteln des Multi-Target-Trackings entdeckt und verbessert werden. In dieser Arbeit werden zwei Ansätze zum Multi-Target Tracking vorgestellt: Der erste ist ein sehr grundlegender Ansatz, der auf dem Greedy-Prinzip beruht. Der zweite, ein Partikelfilter, baut auf aufwändigerer Theorie auf. Er wird hier, in Kombination mit dem K-means-Algorithmus zur Gruppenbildung, eingesetzt. Der Vergleich der beiden Ansätze, anhand des hier eigens für die Anwendung im Roboterfußball entwickelten Gütemaßes, zeigt, dass der Greedy-Ansatz dem Partikelfilter nur knapp unterlegen ist.

Inhaltsverzeichnis

1 Einführung

1.1 Motivation der Arbeit

Der Mensch ist den humanoiden Robotern überlegen: Er beherrscht Multi-Target-Tracking — zum Beispiel die Fähigkeit des Menschen einer Person mit dem Blick zu Folgen, während sie umher läuft, oder Personen wiederzuerkennen, wenn ihr Weg sich mit dem einer zweiten Person gekreuzt hat, ist Multi-Target-Tracking. Diese Fähigkeiten möchte man auch humanoiden Robotern verleihen.

Der konkrete Anreiz dieser Arbeit entstammt dem Nao-Team HTWK. Dies ist eine Gruppe von Studenten die am RoboCup teilnehmen - einer Initiative zur Weiterentwicklung der humanoiden Robotik [Sven Behnke, 2013]. Das Nao-Team HTWK nimmt am RoboCup in einer der Fußball-Ligen teil: der “Standard Plattform Liga” (kurz SPL). In dieser Liga verwenden alle Teams den Roboter “Nao”, der von der französischen Firma Aldebaran gebaut wird.

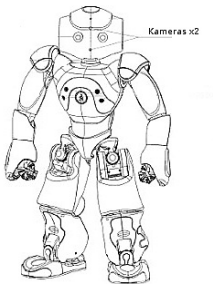


Abbildung 1.1: Der Nao [Aldebaran-Robotics].

Die Roboter werden zum autonomen Fußballspielen programmiert. Um ihre Umwelt wahrzunehmen und sich darin zurechtzufinden, stehen ihnen unter anderem zwei Kameras im Kopf zur Verfügung. Aus den Kamerabildern werden mit verschiedenen Methoden der Bildverarbeitung Informationen gewonnen. Neben den Feldlinien und dem Ball, sind das insbesondere andere Roboter. Diese werden natürlich nicht immer fehlerfrei erkannt - und hier setzt das Multi-Target-Tracking (kurz MTT) an: Es hilft falsche Objekterkennungen zu entdecken und auszugleichen. Das ist in verschiedenen Situationen in einem Fußballspiel nützlich:

1. Schnelle und regelkonforme Fortbewegung auf dem Feld: Das Berühren anderer Roboter ist nur sehr eingeschränkt erlaubt, in den offiziellen Regeln der SPL [Robocup Technical Committee, 2013, S. 22f] befindet sich zu diesem Thema ein eigenes Kapitel. Wenn der Roboter sich auf dem Feld bewegen möchte,

muss er also wissen wo sich andere Roboter befinden, um nicht mit diesen zu kollidieren. Dieses Wissen darf auch nicht "vergessen" werden, wenn er, aufgrund eines Blickes zur Seite, die anderen Roboter vor sich kurzzeitig nicht mehr sieht, und daher auch mit der besten Objekterkennung nicht detektieren kann.

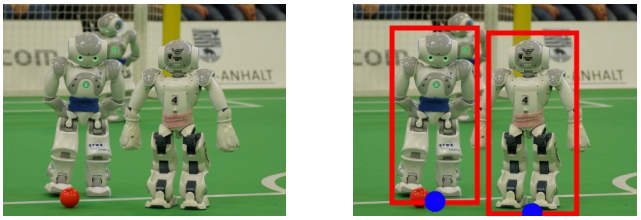
2. Strategische Spielzüge: Im Fußball gibt es zwei Möglichkeiten den Ball zu bewegen: Dribbeln oder Schießen. Beim Dribbeln kann MTT eingesetzt werden, um den Gegnern auszuweichen anstatt zwischen die Beine zu dribbeln. Beim Schießen muss sicher gestellt werden, dass das Schussfeld frei ist. Und wenn der Schuss kein Torschuss sondern ein Pass ist, ist es sinnvoll bei der Wahl des Pass-Partners, die Anzahl der Gegner in seiner Nähe mit einzubeziehen.

1.2 Formale Spezifikation des Multi-Target-Trackings

Die genaue Spezifikation des Multi-Target-Tracking-Problems variiert je nach Publikation (z.B. [Oh u. a., 2004], [Breitenstein u. a., 2011] und [Andriyenko u. a., 2012]). In dieser Arbeit soll das Folgende darunter verstanden werden:

Beim MTT möchte man von Kameras gefilmte, sich möglicherweise bewegende, Objekte $o \in Objekte$ mathematisch beschreiben. Im Nao-Team HTWK sind die Objekte die Roboter. Es können sich mehrere Objekte gleichzeitig im Bild befinden.

Als Eingabedaten bekommen MTT-Algorithmen jedoch nicht die Kamerabilder. Stattdessen ist der Ausgangspunkt das Ergebnis einer Objekterkennung auf diesen Bildern. Das heißt, zu jedem Bild mit bekanntem Aufnahmezeitpunkt, Koordinaten an denen sich vermutlich Objekte befinden. Diese Tupel werden im Folgenden als "Detections" bezeichnet.



(a) (b)

Abbildung 1.2: (a) zeigt das Kamerabild eines Roboters. In (b) sind die Koordinaten der Detections als blaue Punkte markiert.

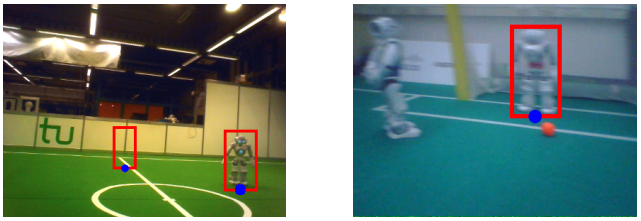
Die Menge der Detections ist:

$D = \{d_j^t = (t, j, x, y) | t \in Frames; j \in \mathbb{N}; x, y \in WK\}$ wobei gilt:

- t ist der Frame (nummeriertes Einzelbild eines Videos) $\in Frames \subseteq \mathbb{N}$ aus dem die Detection stammt
- j ist die laufende Nummer: alle Detections des gleichen Frames sind durchnummeriert
- (x, y) ist die Koordinate der Detection $\in \mathbb{R} \times \mathbb{R}$

In der Bildverarbeitung werden oft die Begriffe "false positive" (ein fälschlicherweise positives Ergebnis) und "false negative" (ein fälschlicherweise negatives Ergebnis) verwendet. Unter den Detections können auch Fehlerkennungen in Form von "False positives" und "False-Negatives" sein. Hierbei bedeutet:

- False-Positive-Detection: Eine Detection $d \in D$, der in der Realität kein Objekt $o \in Objekte$ zu Grunde liegt
- False-Negative-Detection: Ein in der Realität vorhandenes Objekt $o \in Objekte$ wird nicht erkannt



(a) False-Positive (b) False-Negative

Abbildung 1.3: Beispiele für eine False-Positive-Detection und eine False-Negative-Detection

Die Eingabe von MTT-Algorithmen ist eine Menge von Detections. Die Aufgabe eines MTT-Algorithmus besteht dann zum einen darin, die Detections $d \in D$ einem Individuum i zuzuordnen, wobei $i \in I_\perp$ mit $I_\perp = I \cup \{\perp\}$, mit $\perp$ als "False-Positive"-Label (also für Detections die zu keinem Objekt passen) und $I = \{1, \cdots, N\}$ die Individuen. Dieser Teil des MTT wird als "Data Association" bezeichnet.

Nachdem die Data Association beendet ist, schließt sich der zweite Teil des MTTs an, der zum Ziel hat die "False-Negatives" auszugleichen. Dies erfordert:

- eine Identifizierung der Fehlerkennungen ("False-Positives" und "False-Negatives") als solche
- eine Interpolation der tatsächlichen Objekt-Koordinaten zu diesen Zeitpunkten

Ziel ist es, für jedes Individuum $i \in I$ eine Trajektorie T_i anzugeben, die zu jedem Frame Auskunft über die Position des Individuums gibt, also $T_i : Frames \to WK$.

Die Ausgabe des MTTs ist dann eine Menge von Funktionen $T = \{T_i : t \to (x, y) | i \in I, t \in Frames, (x, y) \in WK\}$

Im Roboterfußball werden, sobald für einen Frame t die Koordinaten der Individuen $i \in I$ bekannt sind und die False-Positive-Detections mit $\perp$ markiert wurden, diese Informationen direkt zur Planung der Spielzüge eingesetzt. Die Menge der Trajektorien T wird nicht explizit gespeichert.

Im Nao-Team HTWK soll MTT insbesondere zur Aufdeckung von Fehlerkennungen führen und zur Interpolation der korrekten Koordinaten der betreffenden Objekte eingesetzt werden. Außerdem ist die Eigenschaft das Individuen, auch ohne aktuell durch Detections d detektiert worden zu sein, in der Menge der Individuen eine gewisse Zeit erhalten bleiben, eine sehr wünschenswerte. Dies wird auch in Zukunft, wenn die Objekterkennung deutlich verbessert wurde, noch nützlich sein, denn die Roboter bewegen ihre Köpfe, und brauchen daher immer eine Möglichkeit sich kurzzeitig zu merken wo sich andere Roboter befinden.

Nach dieser Einführung in das Thema MTT werden im nächsten Kapitel die in der Arbeit verwendeten Koordinatensysteme, und die Berechnung der Abbildungsmatrix der projektiven Transformation, auch Homographie genannt, erklärt. Diese kommen in Kapitel 4 zum Einsatz. In Kapitel 3 werden bekannte Publikationen zum Thema MTT vorgestellt. Da die Entwicklung von neuen Algorithmen nur zu guten Ergebnissen führen kann, wenn man deren Güte auch beurteilen kann, wird in Kapitel 4 ein Gütemaß entwickelt. Es wird speziell für die Anwendung im Roboterfußball entwickelt. Es vergleicht die Ausgabe der Algorithmen mit der “Wahrheit”, welche durch die so genanten Ground-Truth-Daten repräsentiert wird. Deren Ermittlung wird ebenfalls geklärt. Es folgt die Erläuterung der beiden, in dieser Arbeit umgesetzten Ansätze: Ein Greedy-Ansatz und ein Partikelfilter. Abschließend wird die Güte der beiden Ansätze mit dem entwickelten Gütemaß bewertet.

2 Mathematische Grundlagen

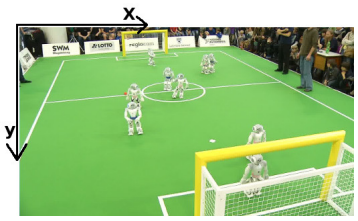
2.1 Koordinatensysteme

Im Laufe der Arbeit sind zwei Koordinatensysteme von wesentlicher Bedeutung:

2.1.1 Pixel-Koordinaten

Dieses 2-dimensionale kartesische Koordinatensystem wird zur Angabe von Koordinaten in Kamerabildern verwendet. Es wird in Kapitel 4.1 benötigt, wenn Ground-Truth Daten zur Gütemessung zu ermittelt werden, und dafür Koordinaten von Pixel- nach Welt-Koordinaten transformiert werden müssen.

Der Ursprung des Koordinatensystem befindet sich in der linken oberen Ecke des Kamerabildes, die positive x-Richtung ist nach rechts, die positive y-Richtung nach unten. Eine Längeneinheit entspricht dabei einem Pixel.



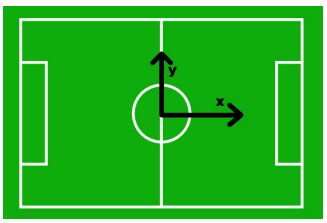
2.1.2 Welt-Koordinaten

Dieses 2-dimensionale kartesische Koordinatensystem entspricht einem Blick von oben auf das Spielfeld. Hier gespeicherte Informationen sind unabhängig von der Position des, die Daten erfassenden, Roboters. Koordinaten von erkannten Robotern, Bällen, oder anderen Informationen, die relativ zur eigenen Position ermittelt wurden, werden nach Welt-Koordinaten transformiert. So können sie mit anderen Robotern geteilt werden, und sind auch nach eigener Positionsänderung noch gültig.

Zum Beispiel sind die Koordinaten der Detections $d \in D$ Weltkoordinaten, ebenso wie die Ground-Truth Daten, die in Kapitel 4 eingeführt werden.

Der Ursprung des Welt-Koordinatensystems liegt im Mittelpunkt des Spielfeldes, die positive x-Richtung zeigt in Richtung des gegnerischen Tores, die positive y-Richtung

gegen den Uhrzeigersinn 90 Grad dazu. Im Weltkoordinatensystem entspricht eine Längeneinheit einem Meter.



2.2 Projektive Transformation

Die Schwierigkeit eines Problems hängt oft davon ab, in welchem Koordinatensystem man es formuliert. Deshalb werden auch im Nao-Team HTWK verschiedene Koordinatensysteme verwendet. Wenn man dann Koordinaten von einem System in ein anderes überführen möchte ist dies mit einer projektiven Transformation möglich. Diese wird durch eine Abbildungsmatrix, Homographie genannt, dargestellt.

Im Folgenden wird erklärt wie die Homographie berechnet werden kann.

2.2.1 Spezifikation

Das Ziel ist es, eine Matrix, die Homographie H, zu erhalten, mithilfe derer Koordinaten von einem Koordinatensystem in ein anderes überführt werden können. Das heißt H stellt die bijektive Abbildung von dem einem System in das andere dar.

Um die Homographie berechnen zu können müssen Punkte aus beiden Koordinatensystemen bekannt sein, die aufeinander abgebildet werden sollen. [Hartley u. Zisserman, 2004, S. 87ff]

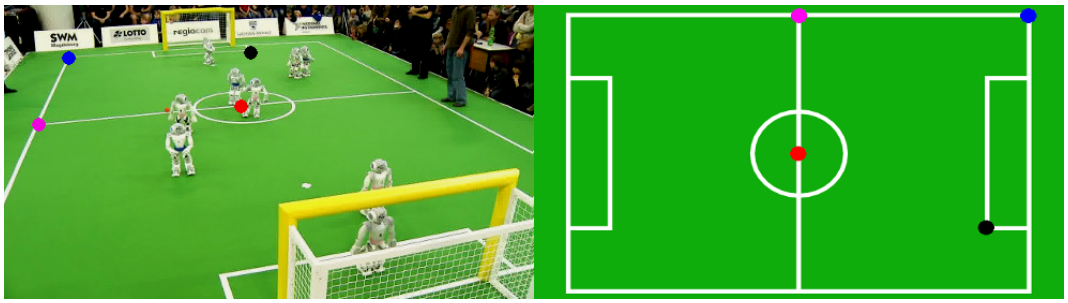


Abbildung 2.1: Beispielhafte Auswahl von vier Punkten im Kamerabild (links), deren Koordinaten in Weltkoordinaten (rechts) bekannt sind, so dass sie als Eingabe-Punktepaare für die Homographieberechnung geeignet wären.

Als Eingabedaten stehen Punktepaare $(p, p') = ((p.x, p.y), (p'x, p'.y))$ aus den beiden Koordinatensystemen zur Verfügung. Für die Abbildung gilt: $Hp = p'$. Dabei gilt auch: Je drei verschiedene Punkte, die auf einer Geraden liegen, werden stets auf drei verschiedene Punkte abgebildet, die wieder kollinear sind. [Dubrofsky, 2009, S. 4f] Die Punkte sind 2-dimensionale Koordinaten, die Homographie-Matrix H hat die Dimension 3x3.

2.2.2 Berechnung der Homographie H

Die Berechnung der Homographie erfolgt nach dem "Direct Linear Transformation Algorithm" (DLT-Algorithmus), beschrieben in [Hartley u. Zisserman, 2004, S. 88ff]. Gegeben sind Punktepaare der Form $(p, p') = ((p.x, p.y), (p'.x, p'.y))$. Hierbei soll p nach der Transformation p' entsprechen: $Hp = p'$.

Die gegebenen Punkte werden für die weiteren Berechnungen als homogene Koordinaten dargestellt. Die Umrechnung erfolgt, indem die kartesischen Koordinaten $(p.x, p.y)$ als $(X/Z, Y/Z)$ aufgefasst werden, und als homogene Koordinaten in der Form (X, Y, Z) notiert. Aus der kartesischen Koordinate $(p.x/1, p.y/1)$ wird also die homogenen Koordinate $(p.x, p.y, 1)$.

Die Homographie H, die berechnet werden soll, hat die Form:

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \tag{2.1}$$

Das Ergebnis der Multiplikation eines Punktes mit der Homographie ist wieder eine (homogene) 3-dimensionale Koordinate:

$$H \begin{bmatrix} p.x \\ p.y \\ 1 \end{bmatrix} = \begin{bmatrix} a.x \\ a.y \\ a_3 \end{bmatrix} = a$$

Dies lässt sich auch als Gleichungssystem notieren:

$$h_{11}p.x + h_{12}p.y + h_{13} = a.x \tag{2.2}$$

$$h_{21}p.x + h_{22}p.y + h_{23} = a.y \tag{2.3}$$

$$h_{31}p.x + h_{32}p.y + h_{33} = a_3 \tag{2.4}$$

Hier lässt sich die Zahl der Unbekannten reduzieren indem p', welcher am Anfang gegeben ist, verwendet wird. Denn p' entspricht a, bis auf Skalierung. Gleichheit gilt, wenn a so skaliert wird, dass die 3.Komponente = 1 ist, also:

$$\begin{bmatrix} \frac{a.x}{a_3} \\ \frac{a.y}{a_3} \\ \frac{a_3}{a_3} \end{bmatrix} = \begin{bmatrix} p'.x \\ p'.y \\ 1 \end{bmatrix} \tag{2.5}$$

Multiplizieren von (2.2) und (2.3) mit $1/a_3$:

$$(h_{11}p.x + h_{12}p.y + h_{13})\frac{1}{a_3} = \frac{a.x}{a_3} = p'.x \tag{2.6}$$

$$(h_{21}p.x + h_{22}p.y + h_{23})\frac{1}{a_3} = \frac{a.y}{a_3} = p'.y \tag{2.7}$$

Einsetzen von Gleichung (2.4) für a_3:

$$\frac{h_{11}p.x + h_{12}p.y + h_{13}}{h_{31}p.x + h_{32}p.y + h_{33}} = p'.x \tag{2.8}$$

$$\frac{h_{21}p.x + h_{22}p.y + h_{23}}{h_{31}p.x + h_{32}p.y + h_{33}} = p'.y \tag{2.9}$$

$\Leftrightarrow$

$$h_{11}p.x + h_{12}p.y + h_{13} = (h_{31}p.x + h_{32}p.y + h_{33})p'.x \tag{2.10}$$

$$h_{21}p.x + h_{22}p.y + h_{23} = (h_{31}p.x + h_{32}p.y + h_{33})p'.y \tag{2.11}$$

$\Leftrightarrow$

$$h_{11}p.x + h_{12}p.y + h_{13} - h_{31}p.x \cdot p'.x - h_{32}p.y \cdot p'.x - h_{33}p'.x = 0 \tag{2.12}$$

$$h_{21}p.x + h_{22}p.y + h_{23} - h_{31}p.x \cdot p'.y - h_{32}p.y \cdot p'.y - h_{33}p'.y = 0 \tag{2.13}$$

Dies sind zwei Gleichungen für die acht Unbekannten h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}. Die Gesamtzahl der Unbekannten ist nicht neun, da $h_{33} = 1$ gesetzt werden kann, da H mit einer beliebigen Konstante $\neq 0$ multipliziert, immer noch die gleiche projektive Transformation darstellt [Dubrofsky, 2009]. Da wir die zwei Gleichungen (2.12) und (2.13) aus nur einem Punktepaar (p, p') gewonnen haben, können wir H mit vier Punktepaaren vollständig bestimmen. Dies wird im "Acht-Punkte-Algorithmus" umgesetzt, der in Kapitel 4.1 ausführlich erklärt wird.

3 Bekannte Multi-Target-Tracking-Ansätze

3.1 Klassifizierung der bisher bekannten Ansätze

MTT-Ansätze können auf verschiedene Arten klassifiziert werden. Im Folgenden werden einige davon vorgestellt.

3.1.1 Offline-Algorithmen - Online-Algorithmen

Die Unterscheidung von Algorithmen in Offline- und Online-Algorithmen ist beim MTT von wesentlicher Bedeutung.

Online-Algorithmen bekommen Eingaben zu dem Zeitpunkt, an dem sie zur Verfügung stehen. Über zukünftige Eingaben gibt es keine Informationen. Ziel der Algorithmen ist die Verarbeitung der Eingaben in "Echtzeit", das heißt, dass immer wenn neue Eingaben geliefert werden, die vorherigen Eingaben fertig verarbeitet sind. [Black, 1999a]

Offline-Algorithmen stehen sämtliche Eingaben gleichzeitig zur Verfügung [Black, 1999b]. Das heißt, dass über die gesamte Zeit, zu der es Eingabedaten gibt, optimiert werden kann. Meist nimmt die Laufzeit hier einen geringeren Stellenwert ein.

Dementsprechend soll im Nao-Team HTWK ein Online-MTT-Verfahren eingesetzt werden.

3.1.2 Unterteilung nach der Minimierungsfunktion

Beim MTT möchte man stets einen Fehler minimieren. Dieser ist im Allgemeinen abhängig von der Distanz zwischen Koordinate der Messung (Detection d) und der, durch den MTT-Algorithmus zugewiesenen, Koordinate. Wenn man MTT-Ansätze nach der Funktion, die sie zu minimieren versuchen, unterteilt, so führt dies zu zwei Klassen von Ansätzen: [Oh u. a., 2009]

- Heuristische Ansätze: Heuristische Ansätze haben meist keine explizite Funktion die sie minimieren. Ein typisches Beispiel ist der "Nearest Neighbourhood Filter" NNF. Hier wird eine neue Messung stets derjenigen Vorhergesagten zugeordnet die am nähesten ist (nach euklidischer Distanz)[Song u. Shin, 2003]. Das heißt, entsprechend dem Greedy-Prinzip wird der nächste kleinste Fehler gewählt, weshalb das Ergebnis abhängig ist von der Reihenfolge in der die Punkte verarbeitet werden.

- Bayessche Ansätze: Bei den Bayesschen Ansätzen wird eine Gesamtfehlerfunktion minimiert. Beispiele sind der "Maximum A Posteriori" MAP und der "Bayes Estimator Minimum Mean Square Error" MMSE, welcher den mittleren quadratischen Fehler minimiert.

3.1.3 Unterteilung nach der Aktualisierungsfunktion

Wenn man MTT-Ansätze danach unterteilt wie sie neue Messungen verarbeiten, so führt dies zu zwei [Oh u. a., 2009] Klassen von Ansätzen:

- Single-scan-Ansätze verwenden alte Zustände und die aktuelle Messung zur Berechnung der neuen Zustände.
- Multi-scan-Ansätze verwenden alte Zustände, die aktuelle Messung und alte Messungen zur Berechnung der neuen Zustände.

[Benfold u. Reid, 2011] und [Andriyenko u. a., 2012] unterteilen in:

- Feed-forward-Systeme: Nutzen aktuelle und alte Messungen zur rekursiven Berechnung des aktuellen Zustands.
- Data Association basierte Systeme: Erlauben eine gewisse Latenz bei der Zustandsberechnung und nutzen somit auch "zukünftige" Informationen zur Schätzung des aktuellen Zustands.

3.1.4 Unterteilung nach konkreten Ansätzen

- Tracking-by-Detection Ansätze: Basis für das Tracken der Objekte ist ein Objekt-Modell, welches in jedem Frame das Objekt repräsentiert [Andriyenko u. a., 2012]. In manchen Fällen wird dieses Objekt-Modell zusätzlich mit einem Online-Modell kombiniert, welches wechselnde Lichtverhältnisse und andere Änderungen im Erscheinungsbild online ausgleichen kann.
- Sliding-Window-Verfahren: Hier wird die Data Association offline, über große temporäre Zeitfenster, optimiert (siehe [Breitenstein u. a., 2011]). Das hat zur Folge, dass das Ergebnis zeitlich verzögert vorliegt, weshalb dieses Verfahren nicht für die Anwendung im Nao-Team HTWK geeignet ist.
- Verfahren die auf dem Bayes-Filter basieren:
  - Sequentielle Monte-Carlo-Methoden (Abk. SMC), auch: Partikelfilter: sind für zeit-kritische online Applikationen geeignet [Breitenstein u. a., 2011].

  - Markov Chain Monte Carlo Data Association (Abk. MCMCDA): Variante des Bayes-Filters mit Single-Scan-Version für fixe Anzahl an Zielen und Multi-Scan-Version für unbekannte Zielanzahl.

- Joint-Probabilistic-Data-Association-Filter (Abk. JPDAF): Hier wird versucht in jedem Zeitschritt die beste Data Association zu erreichen [Breitenstein u. a., 2011] durch Kombination von jedem Individuum mit jeder Detection. Dies hat exponentiellen Aufwand zur Folge, und schließt das Verfahren daher für Online-Anwendungen aus.
- Ungarische Methode: Hier wird eine stabile Data Association in kubischer Laufzeit erreicht, abhängig von der Anzahl der zu trackenden Objekte.[Breitenstein u. a., 2011]
- Multi-Hypotheses-Tracker (Abk. MHT): Der MHT betrachtet über mehrere Zeitschritte verschiedene Data Associations, und zählt damit auch zu den Sliding-Window-Verfahren. [Breitenstein u. a., 2011]

3.2 Darstellung einiger Ansätze

3.2.1 Die Basis: Der Bayes-Filter

Der Bayes-Filter ist ein Algorithmus zur Schätzung des Zustands eines Systems. Da viele MTT-Verfahren auf ihm basieren, soll er hier erklärt werden.

Der Bayes-Filter berechnet bedingte Wahrscheinlichkeiten $p(x_t|z_{1:t}, u_{1:t})$ [Thrun u. a., 2005, S. 26]. $z_{1:t} = z_1, z_2, \cdots, z_t$ bezeichnet Zustandsmessungen (bei uns die Detections $d \in D$) und $u_{1:t} = u_1, u_2, \cdots, u_t$ bezeichnet Zustandsänderungsschätzungen. Eine Zustandsänderungsschätzung wäre im MTT zum Beispiel eine Geschwindigkeitsschätzung. p gibt für jeden, theoretisch möglichen, Zustand x (nach unserer Definition $(x, y) \in WK$), zum Zeitpunkt t an, wie wahrscheinlich es ist, dass sich das System in diesem Zustand befindet. “In dem Zustand befindet” bedeutet zum Beispiel, dass sich ein Objekt an der Position x befindet. Die Menge der theoretisch möglichen Zustände x wird als Zustandsraum bezeichnet.

Der besseren Lesbarkeit wegen wird $p(x_t|z_{1:t}, u_{1:t})$ auch $bel(x_t)$ genannt. Dies rührt vom englischen Begriff “Believe” her. Es soll ausdrücken dass, je höher der Wert von $bel(x_t)$ ist, desto mehr glauben wir, dass sich zum Zeitpunkt t das System im Zustand x befindet.

Der Bayes-Filter berechnet den Wert $bel(x_t)$ rekursiv, $bel(x_t)$ ergibt sich aus dem alten Wert $bel(x_{t-1})$, der Zustandsänderungsschätzung u_t und dem aktuellsten Messwert z_t.

Die Rekursion beruht auf der sogenannten “Markov-Annahme”: Diese besagt, dass ein Zustand nur vom direkten Vorgängerzustand abhängt, und nicht von älteren Zuständen. Somit kann, wenn man diesen Vorgängerzustand und die aktuelle Messung vollständig kennt, der Folgezustand berechnet werden. Diese Annahme ist natürlich fehleranfällig. Das zeigt sich zum Beispiel wenn man annimmt man kennt den Vorgängerzustand, jedoch beruht er auf Daten die mit Messfehlern behaftet sind, und entspricht deshalb gar nicht dem tatsächlichen Vorgängerzustand.

Der Hauptteil des Bayes-Filters hat als Pseudo-Code notiert, folgende Form: [Thrun u. a., 2005, S. 27]:

1: **function** BayesFilter($bel(x_{t-1}), u_t, z_t$)
2: **for all** x_t ∈Zustandsraum **do**
3: $p(x_t|z_{1:t-1}, u_{1:t}) = \int p(x_t|u_t, x_{t-1}) \cdot bel(x_{t-1}) dx_{t-1}$ ▷ prediction
4: $bel(x_t) = \eta \cdot p(z_t|x_t) \cdot p(x_t|z_{1:t-1}, u_{1:t})$ ▷ measurement update
5: **return** $bel(x_t)$

In Zeile 3 wird die Wahrscheinlichkeit für den Zustand x_t berechnet, unter der Bedingung aller bisherigen Zustandsmessungen z, ausgenommen der aktuellsten, und unter der Bedingung aller bisherigen Zustandsänderungsschätzungen u. Hierfür wird der Satz der totalen Wahrscheinlichkeit verwendet. Dieser Schritt wird auch “prediction” genannt, da der Zustand für den $p(x_t|z_{1:t-1}, u_{1:t})$ maximal ist, derjenige ist, den der Filter “vorhersagen” würde. Die “prediction” wird dann in Zeile 4 mit der Wahrscheinlichkeit dafür, dass die aktuelle Messung z_t eintritt wenn wir uns im “vorhergesagten” Zustand x_t befinden, multipliziert. Erst hier wird die aktuelle Zustandsmessung z_t verwendet, daher heißt dieser Schritt “measurement update”. Die Konstante η steht für $\frac{1}{p(z_t|z_{1:t-1})}$. Sie normalisiert die Produkte $p(z_t|x_t) \cdot p(x_t|z_{1:t-1})$, damit sie, über alle x_t aufsummiert 1 ergeben [Thrun u. a., 2005, S. 17 und 31].

Auf den Beweis der Korrektheit wird hier verzichtet, der interessierte Leser sei auf [Thrun u. a., 2005, S. 31ff] verwiesen.

Im Folgenden werden einige wissenschaftliche Publikationen vorgestellt, die das MTT-Problem lösen. Die Erste, Önline Multi-Person Tracking-by-Detection from a Single, Uncalibrated Camera"basiert auf dem Partikelfilter, ein Algorithmus der den Bayes-Filter approximiert.

3.2.2 Online Multi-Person Tracking-by-Detection from a Single, Uncalibrated Camera

Die Publikation “Online Multi-Person Tracking-by-Detection from a Single, Uncalibrated Camera” [Breitenstein u. a., 2011] von den Autoren Breitenstein, Rechlin,

Leibe, Meier und Gool, beschreibt einen MTT Ansatz der auf einem Partikel-Filter basiert.

Ihr Verfahren löst das MTT-Problem durch Zusammenspiel von Objekterkennungs-, Klassifizierungs- und Tracking-Komponenten.

Das Verfahren lässt sich in einen bottom-up und einen top-down Prozess unterteilen:

- Der Bottom-up-Prozess kümmert sich um die Lokalisierung und Repräsentation der Objekte und gleicht Veränderungen im optischen Erscheinungsbild der Objekte aus (wechselnde Lichtverhältnisse o.Ä.).
- Der Top-down-Prozess kümmert sich um die Data Association und Filterung, und gleicht die Dynamik der Objekte aus.

Bestandteile des Programms:

1. Klassen-spezifischer Objekt-Detektor, Beispiele für Klassen sind Menschen oder NAOs (nicht Teil des MTTs)
2. Objekt-Klassifikator der in Echtzeit trainiert wird, zur Unterscheidung der Ziele voneinander und zur Erkennung von “false positives”
3. Partikelfilter, mit eingebautem Bewegungsmodell, zur Vorhersage der Koordinaten der Objekte
4. Greedy Data Association, jedoch nicht nur wie üblich auf Basis der euklidischen Distanz, sondern zusätzlich mit den Daten aus dem Objekt-Klassifikator von Punkt 2.

Es handelt sich bei dem Programm um ein Markov-Modell 1.Ordnung, was heißt, dass nur Informationen der Gegenwart und des vorhergehenden Frames nötig sind, um den Folgezustand zu bestimmen. Für diejenige Detection, die im jeweiligen Frame die höchste Wahrscheinlichkeit hat, wird ein eigener Partikel-Filter initialisiert. Dann wird Data Association ausgeführt. Hierbei werden die euklidischen Abstände zwischen Detections und Individuen zu Rate gezogen werden, zusätzlich zu einer Schwellwert-Funktion, die wiederum die Größe des Objektes im Kamerabild, die Bewegungsrichtung und dessen Geschwindigkeit mit einbezieht. False positives werden über die Sicherheiten der Objekterkennung und der Personen-Klassifikatoren entdeckt.

3.2.3 Stable Multi-Target Tracking in Real-Time Surveillance Video

Das Ziel der Publikation "Stable Multi-Target Tracking in Real-Time Surveillance Video" [Benfold u. Reid, 2011] der Autoren Benfold und Reid, ist es Kopf-Positionen zu tracken.

Es wird ein Sliding-Window Verfahren zur Data Association verwendet, aufgrund seiner Fehler- und false negative ausgleichenden Eigenschaften.

Aufbau des Programms:

1. Asynchrone HOG-Detections
2. Simultanes KLT-tracking ("lokal Feature basiertes" tracking)
3. MCMCDA 3.1.4 (mit Sliding-Window Verfahren) für Echtzeit-Tracking in HD
4. MDL für Data Association über Ähnlichkeitsmodellierung.

Das Tracking basiert bei Benfold und Reid auf MCMCDA kombiniert mit Bewegungsschätzung, welche wiederum auf einem eigens dafür entwickelten Modell beruht. False positives werden mit SLAM behandelt: Es wird ein separates Modell für false-positives erstellt, und dann die false-positive Erkennungen mit den Ergebnissen der Data Association abgeglichen. Echtzeit-Fähigkeit wird durch Multi-Threading erreicht.

3.2.4 Discrete-Continuous Optimization for Multi-Target-Tracking

In dem Paper "Discrete-Continuous Optimization for Multi-Target-Tracking" der Autoren Andriyenko, Schindler und Roth [Andriyenko u. a., 2012] wird das Problem des MTT in zwei Teilprobleme zerlegt:

1. Schritt: Data Association
2. Schritt: Trajectory estimation (Trajektorienschätzung)

Das Problem der Data Association wird über diskrete Optimierung mit Label-Kosten gelöst. Trajektorienschätzung wird als kontinuierliches Fitting-Problem formuliert: Die Trajektorien werden durch Polynomstücke dargestellt, welche auf eine Menge von möglichen Positionen der Objekte angepasst werden können. Die Lösung der Trajektorienschätzung wird verwendet um die Label-Kosten mit Hilfe der "α-Expansion" (mehr dazu in einschlägiger Literatur, z.B. [Schmidt u. Alahari, 2011]) zu aktualisieren. Schließlich wird MTT als eine diskret-kontinuierliche Energiefunktion formuliert, welche durch alternierende Ausführung der beiden Schritte 1 und 2 minimiert wird.

4 Verfahren zur Gütemessung

Das folgende Kapitel befasst sich mit der Gütemessung von MTT-Algorithmen. Es wird ein, speziell auf die Bedürfnisse des MTTs im Roboterfußball angepasstes, Gütemaß entwickelt. Hierbei soll die Güte eines Algorithmus daran beurteilt werden wie klein sein Fehlermaß F ist. Die Basis für die Berechnung von F bildet der schrittweise Vergleich zwischen den Ausgaben der MTT-Algorithmen $T_i(t), t \in Frames, i \in I$ und den Ground-Truth Daten $GT_o(t), t \in Frames, o \in Objekte$, mit $Objekte$ Menge der Objekte. GT, die Menge der Ground-Truth-Daten, umfasst die korrekten Welt-Koordinaten der $o \in Objekte$ zu jedem Zeitpunkt (=Frame):

$$GT = \{GT_o : t \to (x, y) | o \in Objekte, t \in Frames, (x, y) \in WK\}$$

Korrekt bedeutet hier von Menschen, von Hand erhobene Koordinaten. Dafür werden in Kamerabildern die Koordinaten der Roboter angeklickt. Folglich liegen die Koordinaten zunächst im Pixel-Koordinatensystem der Kamera. Die Transformation in Weltkoordinaten WK erfolgt dann mit der Homographie-Matrix (Kapitel 2.2), eingebettet in den "Acht-Punkte-Algorithmus", der nun erläutert wird. Danach folgt die Spezifikation und Implementierung des Fehlermaßes F.

4.1 Acht-Punkte-Algorithmus zur Beschaffung der Ground-Truth-Daten

In Kapitel 2.2 wurde beschrieben, wie man aus vier Punktepaaren (also acht Punkten) die Homographie berechnet. Nun soll sie eingesetzt werden um die Ground-Truth-Daten zu bestimmen: Mit der Homographie werden die, von Hand in Videos markierten, Roboterpositionen von Pixel- in Weltkoordinaten umgerechnet. Würde man die Berechnung wie in Kapitel 2.2 beschrieben implementieren, müssten die vier Punktepaare frei von Mess-Ungenauigkeiten sein, andernfalls würde die Homographie-Matrix zu ungenau [Hartley, 1997]. Die Berechnung lässt sich jedoch durch eine Vorverarbeitung der Punkte nach Hartley [Hartley, 1997] und Gupta [Gupta u. a., 2011] numerisch deutlich stabilisieren. Auch können, bei der von Ihnen vorgeschlagenen Erweiterung, mehr als vier Punktepaare gegeben werden. Die zusätzlichen Punkte werden dann verwendet um eventuelle Mess-Ungenauigkeiten auszugleichen.

Zuerst müssen die folgenden Schritte, einmal für die gegebenen Punkte des Eingabe-Koordinatensystems, und dann noch einmal für die Punkte des Ausgabe-Koordinatensystems, ausgeführt werden:

1. Schwerpunkt $s = (s.x, s.y)$ berechnen: Aufsummieren der Punkte und Teilen durch die Anzahl
2. Mittleren Abstand d zum Schwerpunkt s berechnen
3. Punkte als homogene Koordinaten notieren
4. Punkte verschieben, so dass der Schwerpunkt im Ursprung $(0, 0)$ liegt, und der mittlere Abstand zum Schwerpunkt $\sqrt{2}$ ist. Dies geschieht durch Multiplikation von links mit der Matrix M, deren Komponenten eben berechnet wurden:

$$M = \begin{pmatrix} 1 & 0 & -s.x \\ 0 & 1 & -s.y \\ 0 & 0 & \frac{d}{\sqrt{2}} \end{pmatrix}$$

5. Rücktransformation, von homogenen Koordinaten in 2D-Punkte, durch Teilen durch die letzte Koordinate.

Danach werden, für alle (nun skalierten und verschobenen) Punkte gemeinschaftlich, folgende Schritte durchgeführt:

1. Für jedes gegebene Punktepaar $(p, p') = ((p.x, p.y), (p'.x, p'.y))$, welches wir eben skaliert und verschoben haben, notieren wir jene zwei Zeilen die wir am Ende von Kapitel 2.2 erhielten. Dabei lassen wir die gesuchten Variablen h_{11}, .., h_{32} weg, und notieren nur die Koeffizienten. Das heißt wir stellen eine Matrix auf, nennen wir sie L, welche neun Spalten hat, und bei vier Punktepaaren acht Zeilen. Bei mehr Punktepaaren je Paar zwei Zeilen mehr.

$$L = \begin{pmatrix} p_1.x & p_1.y & 1 & 0 & 0 & 0 & -p_1.x \cdot p'_1.x & -p_1.y \cdot p'_1.x & -p'_1.x \\ 0 & 0 & 0 & p_1.x & p_1.y & 1 & -p_1.x \cdot p'_1.y & -p_1.y \cdot p'_1.y & -p'_1.y \\ p_2.x & p_2.y & 1 & 0 & 0 & 0 & -p_2.x \cdot p'_2.x & -p_2.y \cdot p'_2.x & -p'_2.x \\ 0 & 0 & 0 & p_2.x & p_2.y & 1 & -p_2.x \cdot p'_2.y & -p_2.y \cdot p'_2.y & -p'_2.y \\ & & & & & \vdots & & & \\ p_8.x & p_8.y & 1 & 0 & 0 & 0 & -p_8.x \cdot p'_8.x & -p_8.y \cdot p'_8.x & -p'_8.x \\ 0 & 0 & 0 & p_8.x & p_8.y & 1 & -p_8.x \cdot p'_8.y & -p_8.y \cdot p'_8.y & -p'_8.y \\ & & & & & \vdots & & & \end{pmatrix}$$

2. Das eigentlich dahinter stehende lineare Gleichunssystem wird nun durch Singulärwertzerlegung von L in $U\Sigma V^T$ gelöst: In der Diagonalmatrix Σ sind die Singulärwerte steigend geordnet[Björck, 1996, S. 145]. V besteht aus den "Rechts-Singulärvektoren"der Matrix L. Die Lösung des linearen Gleichungssytems, also die Homographie, ist der Rechts-Singulärvektor in V der zum kleinsten Singulärwert gehört, also der rechteste [Björck, 1996, S. 145].

3. Dieser neun-elementige Vektor wird nun nach jeweils drei Elementen umgebrochen, so dass er die Form einer 3x3 Matrix erhält. Diese Homographie wurde aus einer Punktemenge berechnet die mit M verschoben und skaliert wurde. Damit nicht alle Punkte, die mit der Homographie transformiert werden sollen, vorher von Hand mit M verschoben und skaliert werden müssen, wird M mit der Homographie verkettet. Die Homographie wird von rechts mit der Matrix M multipliziert, erstellt aus den Punkten des Eingabe-Koordinatensystems, und danach von links mit der Matrix M^{-1}, erstellt aus den Punkten des Ausgabe-Koordinatensystems.

Nun da Ground-Truth-Daten zur Verfügung stehen, kann ein Fehlermaß entwickelt werden, welches auf einem Vergleich der MTT-Ausgaben mit "der Wahrheit" basiert.

4.2 Spezifikation des Fehlermaßes

Folgende Anforderungen werden an das Fehlermaß F gestellt:

1. Gleiche Eingaben müssen gleiche Ergebnisse liefern, folglich darf die Fehlerberechnung keine randomisierten Bestandteile enthalten.
2. Das Fehlermaß soll einfache Vergleiche von Verfahren ermöglichen, daher sollte es aus einer oder wenigen reellen Zahlen bestehen.
3. Wenn ein MTT-Algorithmus die Ground-Truth-Daten als Ausgabe liefert, so muss $F = 0$ sein
4. F soll erhöht werden, wenn in der Ausgabe des MTT-Algorithmus' Koordinaten von vorhandenen Objekten fehlen (False-Negatives).
5. Es soll den Fehler erhöhen, wenn in der Ausgabe des MTT-Algorithmus' Koordinaten sind, die keinem vorhandenen Objekt entsprechen (False-Positives).
6. Die Güte der Schätzung der eigenen Position des Roboters ("Selbstlokalisierung") soll nicht bewertet werden, da der Roboter die Koordinaten der Detections relativ zu seinem eigenen Standort berechnet, das heißt wenn dieser falsch ist, sind auch die Detections falsch.

4.3 Implementation des Fehlermaßes

In der Spezifikation des Fehlermaßes wurde eine Erhöhung des Fehlers in zwei Situationen gefordert (siehe Kapitel 4.2, Punkt 4 und Punkt 5): Bei False-Negatives

und bei False-Positives. Dies wird an drei Stellen in der Implementierung des Fehlermaßes umgesetzt (Zeilen 5, 14, 19). Diese sind, wie auch der Rest der Implementierung, durch reelle Konstanten parametrisiert, mithilfe derer die Berechnung an die Wünsche und Anforderungen des Anwenders angepasst werden kann. Die Konstanten sind: $noPosPenalty$, $maxAllowedSelflocErr$, $maxDistPosGT$, $lonelyPosPen$, $minLonelyGTerr$ und $maxLonelyGTerr$. Die genaue Bedeutung der Konstanten wird nach der Implementierung aufgeschlüsselt. Zunächst noch einige Vorbemerkungen zur Implementierung:

- $|GT|$ bezeichnet die Mächtigkeit der Menge GT
- $\setminus$ bezeichnet die Mengendifferenz
- $distance(a, b)$ ist eine Funktion die die euklidische Distanz zwischen a und b zurückgibt
- $error \in \mathbb{R}$ wird nach Aufruf der Methode durch cnt geteilt, um einen vergleichbaren Zahlenwert zu erhalten
- $findLonely(A, B, c)$ ist eine Funktion, die alle Elemente der Menge A zurückgibt, die keinem Element der Menge B näher als $c \in \mathbb{R}$ (euklidische Distanz) sind
- $findDistToNearest(A, b)$ ist eine Funktion, die die kleinste euklidische Distanz von b zu allen $a \in A$ findet und zurückgibt

Außerdem sind folgende Anfangsbedingungen zu erfüllen:

1. $GT(t) \neq \emptyset$
2. $error = 0$
3. $cnt = 0$

Nun zur Implementierung des Fehlermaßes F: F ergibt sich nach jedem Durchlauf der Funktion Update Error als Quotient:

$F := \frac{error}{cnt}$, ausgenommen $cnt = 0$, dann gilt $F = 0$.

Die Übergabeparameter der Funktion Update Error sind die folgenden Daten:

1. $ownCoord_t$: Die Koordinate an der der Roboter gerade (zum Zeitpunkt t) denkt zu sein.
2. $T(t)$: Die Menge der Positionen der Individuen zum Zeitpunkt t.
3. $GT(t)$: Die Menge der Ground-Truth-Daten, also die Menge der 2-dimensionalen Koordinaten, an denen sich tatsächlich Roboter befinden zum Zeitpunkt t.
4. $error$: Der bisherige Fehler.

5. *cnt*: Die Anzahl der bisher durchgeführten Fehlerberechnungen.

1: **function** UPDATE ERROR($ownCoord_t, T(t), GT(t), ownGT_t, error, cnt$)
2: $remainingPos := \emptyset$
3: $remainingGT := \emptyset$
4: **if** $T(t) = \emptyset$ **then**
5: $error := error + noPosPenalty$
6: $cnt := cnt + 1$
7: **else**
8: **if** $distance(ownCoord_t, ownGT_t) \leq maxAllowedSelflocErr$ **then**
9: $cnt := cnt + 1$
10: $lonelyPos := findLonely(T(t), GT(t), maxDistPosGT)$
11: $remainingPos := T(t) \backslash lonelyPos$
12: **for all** $lPos \in lonelyPos$ **do**
13: $error := error +$
14: $findDistToNearest(GT(t), lPos) \cdot lonelyPosPen$
15: $lonelyGT := findLonely(GT(t), remainingPos, maxDistPosGT)$
16: $remainingGT := GT(t) \backslash lonelyGT$
17: **for all** $lGT \in lonelyGT$ **do**
18: $error := error +$
19: $min(max(\frac{1}{distance(lGT, ownGT_t)}, minLonelyGTerr), maxLonelyGTerr)$
20: **return** $error, cnt$

Nun zur Bedeutung und Wahl der Konstanten, mit denen der Algorithmus parametrisiert wurde:

- *noPosPenalty* aus Zeile 5 ist die Konstante, die zu *error* addiert wird, falls das MTT keine Koordinaten in $T(t)$ zurückgibt, obwohl Objekte vorhanden sind. Für die Anwendung im Roboterfußball wurde $noPosPenalty = 50$ gewählt, da 50 ungefähr $9 \cdot \sqrt{4^2 + 6^2}$ ist, mit 9 = Anzahl maximal möglicherweise sichtbarer Roboter, da in der SPL gegenwärtig zehn Roboter auf dem Feld erlaubt sind, 4 = Breite des Feldes, 6 = Länge des Feldes. Dieser verhältnismäßig große Wert wurde gewählt, da es hier sehr wichtig ist, dass, wann immer Objekte zu sehen sind, diese auch erkannt werden.
- *maxAllowedSelflocErr* in Zeile 8 ist die euklidische Distanz in Metern, die maximal zwischen dem Standort des Roboters und dem Punkt an dem er denkt zu stehen (also sein Selbstlokalisierungsfehler), liegen darf, damit noch eine Fehlerberechnung durchgeführt wird. Hier wurde 0.7 gewählt, da im Moment

im Nao-Team HTWK der Fehler der Selbstlokalisierung oft in diesem Bereich liegt, und das MTT auch in diesen Fällen auf seine Güte getestet werden soll.

- $maxDistPosGT$ ist in Zeile 10 und 15 die maximale euklidische Distanz in Metern die zwischen einer Koordinate eines Individuums $T_i(t)$ und einer Ground-Truth-Koordinate $GT_o(t)$ liegen darf, damit $T_i(t)$ noch das Objekt o repräsentieren kann. Hier wurde $maxDistPosGT = 0.6$ gewählt, was infolge der, mit der Distanz zur eigenen Position zunehmenden, schlechter werdenden Robotererkennung, ein moderater Wert ist.
- Nun werden in Zeile 14 diejenigen $T_i(t)$ bestraft, die wegen $maxDistPosGT$ keiner $GT_o(t)$ zugeordnet wurden. Ihr Fehleranteil am Gesamtfehler ergibt sich aus der Distanz zur räumlich nächsten Ground-Truth-Koordinate, multipliziert mit dem Faktor $lonelyPosPen$. Hier hängt die Wahl des Zahlenwertes davon ab, wie man das Fortbestehen von False-Positives (also "einsamen' $T_i(t)$) nach dem MTT bestrafen möchte. Hier wurde 0.2 gewählt.
- In Zeile 19 schließlich wird die Strafe für diejenigen Ground-Truth-Koordinaten $lGT \in lonelyGT$ addiert, die von keiner Koordinate $T_i(t)$ repräsentiert werden. Der Strafsummand ist $\frac{1}{distance(lGT, ownGT_t)}$. Dieser wird jedoch begrenzt auf das Intervall $[minLonelyGTerr, maxLonelyGTerr]$ um extreme Werte zu vermeiden. Hier wurde $minLonelyGTerr = 0.1$ und $maxLonelyGTerr = 4$ gewählt.

4.4 Testen der Fehlerspezifikation

Es stehen Videoaufnahmen zur Verfügung, in denen Fußballspiele der SPL von einer externen Kamera aufgenommen wurden. Mithilfe der Videos wurden die Ground-Truth-Daten ermittelt. Hierfür wurden von Menschen die Pixelkoordinaten der Roboter ermittelt, und diese danach mithilfe des Acht-Punkte-Algorithmus in Welt-Koordinaten umgerechnet.

Nun können diese Daten verwendet werden um zu überprüfen ob die Implementierung des Fehlermaßes bestimmte Aspekte der Spezifikation umsetzt.

Ein wichtiger Aspekt der Spezifikation ist, dass das Fehlermaß $F = 0$ sein muss, wenn ein MTT-Algorithmus die Ground-Truth-Daten als Ausgabe liefert. Dies bedeutet, dass die Funktion zur Fehlerberechnung, Update Error,

- anstatt $ownCoord_t$ ebenfalls $ownGT_t$
- und anstatt $T(t)$ ebenfalls $GT(t)$

übergeben bekommt.

Die Tests zeigten das dies erfüllt wird und *error* und damit auch $F = 0$ bleibt. Somit sind alle sechs Punkte der Spezifikation aus Kapitel 4.2 erfüllt.

Im Folgenden Kapitel werden die beiden, im Rahmen dieser Arbeit umgesetzten MTT-Verfahren erklärt, und daraufhin ausführlich anhand von verschiedenen Parametrisierungen von F untersucht.

5 Eigene Ansätze

Im Rahmen dieser Bachelorarbeit wurden zwei MTT-Ansätze umgesetzt und auf ihre Eignung zur Verwendung im Nao-Team HTWK untersucht. Dies wird im Folgenden vorgestellt.

5.1 Greedy-Lösung

Eine Möglichkeit das MTT-Problem zu lösen ist ein Greedy-Ansatz.

Ein Greedy-Ansatz zeichnet sich dadurch aus, dass er in jedem Berechungsschritt, auf dem Weg zur Lösung, die momentan günstigste Entscheidung trifft. Genauere Erläuterungen zum Greedy-Prinzip können einschlägiger Literatur, zum Beispiel "Algorithmen: eine Einführung" von Cormen u.a. [Cormen u. a., 2010, S. 417] entnommen werden.

Im Folgenden Abschnitt wird der in dieser Arbeit umgesetzte Greedy-Ansatz erläutert.

5.1.1 Umsetzung

Beim MTT stehen in jedem Schritt, zum Zeitpunkt t, Detections d_j^t von Objekten zur Verfügung. Zusätzlich ist in d_j^t, im Nao-Team HTWK, die Teamfarbe *teamcolor* des erkannten Objekts enthalten, da diese hier im Rahmen der Bildverarbeitung erkannt wird. Diese Detections d_j^t müssen verarbeitet werden.

Es muss entschieden werden welche Informationen der MTT-Algorithmus speichert. Bei diesem Greedy-MTT-Ansatz wird eine Menge M von Detections gespeichert und zur Verfügung gestellt, die Informationen über die aktuell bekannten Individuen enthält. Die $m \in M$ sind ohne Frame t gespeichert, da M stets für den aktuellen Frame gültig ist. Weiterhin ist die Teamfarbe des erkannten Objekts und ein Counter, auf dessen Zweck noch eingegangen wird, gespeichert. Die Menge der vom Algorithmus gespeicherten Detections ist also $M = \{(x, y, teamcolor, counter) | x, y \in \mathbb{R}, teamcolor \in \{rot, blau\}, counter \in \mathbb{N}$.

Der erste Schritt des Greedy-MTT's ist die Data Association. Dies ist auch der Teil, in dem der Greedy-Aspekt zum Tragen kommt.

Folgendes Prinzip wurde umgesetzt: Jede Detection d wird derjenigen, bisher gespeicherten Detection $m_k \in M$ zugeordnet für die gilt:

- Zu m_k ist die euklidische Distanz kleiner als zu allen anderen $m \in M$
- Zu m_k ist die euklidische Distanz kleiner des Schwellwerts $s \in \mathbb{R}^+$
- Die Teamfarbe *teamcolor* stimmt überein

Wenn es keine solche Detection m_k gibt, wird die Detection d als zu einem bisher noch unbekannten Individuum $i \in I$ gehörig angenommen. Zum Beispiel könnte die Detection zu einem Roboter gehören, der vorher immer hinter dem Beobachter lief. Dem Rechnung tragend wird der Menge M eine neue Detection m, mit den Koordinaten der betreffenden Detection d, hinzugefügt. Die Schwierigkeit besteht hierbei darin, den richtigen Schwellwert s festzulegen, ab dem dies geschehen soll. Bei der Wahl von s sind verschiedene Aspekte zu beachten:

- s sollte nicht kleiner sein als die Distanz, die die beobachteten Objekte von einem Berechnungsschritt zum nächsten maximal zurücklegen können. Andernfalls könnte der Fall auftreten, das derselbe Roboter nicht derselben Detection m zugeordnet wird, nur aus dem Grund das er sich zu schnell bewegt hat.
- s sollte Berechnungsungenauigkeiten berücksichtigen. Denn zum Beispiel Ungenauigkeiten in der Positionsberechnung der Detections können zum (scheinbaren) überschreiten der Maximalgeschwindigkeit durch die Roboter führen. Deshalb muss im Schwellwert ein Summand für Ungenauigkeiten enthalten sein.

Nun zur Bedeutung des anfangs erwähnten Counters. Der Counter *counter* repräsentiert die Aktualität einer gespeicherten Detection m. Je größer der Counter, desto veralteter ist die Detection. In jedem Schritt in dem neue Detections d verarbeitet werden, werden zunächst die Counter der gespeicherten Detections erhöht: $\forall m \in M : counter := counter + 1$.
Danach werden die Counter, in Abhängigkeit des Ergebnis' der Data Association, gegebenenfalls wieder $= 0$ gesetzt. Dies kann in zwei Situationen auftreten:

1. Falls eine Detection $d$keiner bereits gespeicherten Detection m zugeordnet werden kann, wird sie neu zu M hinzugefügt, mit $counter = 0$.
2. Wenn eine Detection d einer bereits gespeicherten Detection m zugeordnet werden kann, so werden die beiden verrechnet und der Counter *counter* wieder $= 0$ gesetzt.

Somit entspricht *counter* der Anzahl der Verarbeitungszyklen, seit denen die Detection m kein Update mehr erhielt. Dieses Wissen wird verwendet um "alte" Detections aus M zu entfernen. Wobei hier wieder ein Schwellwert, sei dies $a \in \mathbb{N}$, definiert werden muss, ab dem das geschehen soll. Dieser ist abhängig davon, wie weit sich

die beobachteten Objekte je Verarbeitungsschritt bewegen. Je weiter sie kommen, desto kleiner muss a sein.

Dieses Greedy-MTT-Verfahren lässt sich so implementieren wie hier beschrieben. Von Interesse ist natürlich die Güte des Ansatzes, gemessen an dem Gütemaß aus Kapitel 4. Diese wird in Kapitel 6 untersucht. Zunächst soll noch das zweite hier umgesetzte MTT-Verfahren erläutert werden.

5.2 Partikelfilter-Lösung

In den letzten Jahren erfreuen sich die sogenannten Partikelfilter, auch bekannt unter dem Namen sequentiellen Monte-Carlo-Methoden, im Bereich der Tracking-Probleme wachsender Beliebtheit. Zunächst soll die grundlegende Funktionsweise erklärt werden. Danach wird auf die Anpassungen eingegangen, die zur Verwendung zum MTT nötig sind.

5.2.1 Funktionsweise des Partikelfilters

Der Partikelfilter beruht auf dem Bayes-Filter (siehe Kapitel 3.2.1). Die gesuchten Wahrscheinlichkeiten $bel(x_t)$ werden beim Partikelfilter durch eine Menge $Particles_t = \{particle_t^1, particle_t^2, \cdots, particle_t^M\}$, von sogenannten Partikeln approximiert. Diese stellen Stichproben der Funktion $bel(x_t)$ dar. Partikel sind Tupel $particle_t^m = (x, t, m, weight)$, bestehend aus einem Punkt x im Zustandsraum, Zeitpunkt t, laufende Nummer m, mit $1 \leq m \leq M$, M Anzahl der Partikel in $Particles_t$, und einem Gewicht $weight$. Ziel des Partikelfilters ist, dass die Wahrscheinlichkeit für einen Partikel, in der Menge $Particles_t$ enthalten zu sein, gleich $bel(x_t)$ ist.

Der Basis-Algorithmus des Partikelfilters hat folgenden Aufbau, wobei u_t wieder die Zustandsänderungsschätzung ist, und z_t die Zustandsmessung:

1: **function** ParticleFilter($Particles_{t-1}, u_t, z_t$)
2: $\quad \overline{Particles_t} = Particles_t = \emptyset$
3: $\quad$ **for** $m = 1$ to M **do**
4: $\quad\quad$ ziehe $particle_t^m \sim p(x_t | u_t, particle_{t-1}^m)$ ▷ Ziehen
5: $\quad\quad$ $weight_t^m = p(z_t | particle_t^m)$ ▷ Berechnung der Gewichte
6: $\quad\quad$ $\overline{Particles_t} = \overline{Particles_t} \cup particle_t^m$
7: $\quad$ **for** $m = 1$ to M **do**
8: $\quad\quad$ ziehe i with probability α w_t^i ▷ resampling
9: $\quad\quad$ add $particle_t^i$ to $Particles_t$
10: $\quad$ **return** $Particles_t$

Eingabe des Algorithmus' ist die Menge der Partikel des vorhergehenden Schrittes $Particles_{t-1}$, die aktuellste Zustandsänderungsschätzung u_t, und die aktuelle Zustandsmessung z_t.

Zunächst wird in Zeile 4 aus der "vermuteten" neuen Verteilung gezogen [Thrun u. a., 2005, S.99]. Diese entspricht der Wahrscheinlichkeit eines Zustands x_t unter der Voraussetzung des Partikels $x_{t-1}^m, m \in M$ des vorherigen Schritts, und der aktuellen Zustandsübergangsschätzung u_t. Es sei $|M|$ die Mächtigkeit der Menge M, dann wird $|M|$-mal gezogen. Dies stellt den "prediction" Schritt des Bayes-Filters dar.

Dann wird in Zeile 5 das "Gewicht" *weight* des Partikels berechnet. Es entspricht der Wahrscheinlichkeit der eingetretenen Zustandsmessung z_t unter der Bedingung, das der vorherige Zustand der eben gezogene $particle_t^m$ war.

Der entscheidende Schritt des Partikelfilters wird in Zeile 8 ausgeführt, beim "resampling". Aus der Menge der Partikel $\overline{Particles_t}$ werden *mit Zurücklegen* so viele Partikel gezogen wie enthalten sind, nämlich $|M|$ Stück. Dabei wird ein Partikel mit der Wahrscheinlichkeit seines Gewichts gezogen. Dadurch ändert sich die "Verteilung" der Wahrscheinlichkeit über dem Zustandsraum. Sie entspricht nach dem Ziehen näherungsweise $bel(x_t)$. Somit gilt, je dichter ein Teilgebiet des Zustandsraums mit Partikeln besiedelt ist, desto höher ist die Wahrscheinlichkeit das dort tatsächlich ein zu trackendes Objekt vorhanden ist [Thrun u. a., 2005, S. 98], vorausgesetzt die Anzahl der Partikel ist groß genug. Die Anzahl orientiert sich an der Größe der zu trackenden Objekte und der räumlichen Ausdehnung des Zustandsraums. Ein grober Richtwert für die Anzahl der Partikel ist die Anzahl an Objekten die nötig wäre um den Zustandsraum auszufüllen. Das "resampling" wird auch als Implementierung der Darwinschen Idee der natürlichen Selektion bezeichnet.

5.2.2 Erweiterung des Partikelfilters zum Multi-Target-Tracking-Ansatz

Partikelfilter wurden zum Tracken eines einzelnen Objektes entworfen. Um sie zum MTT einsetzen zu können, bedarf es einiger Abwandlungen. Der umgesetzte Partikelfilter orientiert sich an den Publikationen von Vermaag [Vermaak u. a., 2003] und von Breitenstein [Breitenstein u. a., 2011].

Um mehrere Ziele zu tracken, werden Partikel in Cluster C zusammengefasst. Ein Cluster ist eine Menge von Partikeln, und hat ein eigenes Gewicht, das *clusterweight*. Die Anzahl an Partikeln in einem Cluster bleibt stets konstant gleich $particleAmount \in \mathbb{N}$. Derjenige Partikel *particle* in einem Cluster, der das größte Gewicht *particle.weight* hat, wird als "Mean" bezeichnet. Der Zweck eines Clusters ist es, ein Objekt tracken. Ein Cluster kann höchstens ein Objekt darstellen. Die Menge aller Cluster wird als *Clusters* bezeichnet.

Bei diesem MTT-Partikelfilter-Ansatz hat ein Partikel hat die Form: $particle = (x, y, weight, clusterweight, age)$ mit $(x, y) \in WK$ die Weltkoordinate, $weight \in \mathbb{R}+$ das Gewicht, $clusterweight \in \mathbb{R}+$ das Gewicht des Clusters zu dem der Partikel gehört, und $age \in \mathbb{N}$ das Alter des Partikels. Das Speichern des Zeitpunkts t ist nicht nötig, da die Partikel stets den aktuellsten Systemzustand approximieren. Das *clusterweight* ist für alle Partikel in einem Cluster gleich. Wenn jedoch ein Partikel einem anderen Cluster zugeordnet wird, so bleibt zunächst das bisherige *clusterweight* erhalten, und kann für Berechnungen genutzt werden, bevor es neu gesetzt wird.

Der MTT-Partikelfilter hängt von verschiedenen Parametern ab:

- $newClusterIfDetectionThisFar \in \mathbb{R}^+$
- $particleAmount \in \mathbb{N}$
- $maxAge \in \mathbb{N}$
- $clusterThrowAwayWeight \in \mathbb{R}^+$

Ihre Bedeutung wird in der folgenden Beschreibung des Algorithmus erklärt. Passende Zahlenwerte für die Parameter müssen experimentell ermittelt werden.

In jedem Frame, in dem neue Detections D^t zur Verfügung stehen, wird die Hauptfunktion des Partikelfilters aufgerufen. Diese durchläuft folgende Schritte:

1. Altern: $\forall particle \in Particles : particle.age := particle.age + 1$
2. Verarbeiten der aktuellen Detections D^t:
   $\forall d \in \{d^t | d^t \in D \wedge t =$aktueller Frame$\}$:
   - Suche Cluster C mit minimaler euklidischer Distanz zum Mean des Clusters.
   - Falls die Distanz $\geq newClusterIfDetectionThisFar$ ist, wird ein neues Cluster C mit zufällig initialisierten Partikeln erstellt. Dabei ist immer $age = 0$.
   - $\forall particle \in C$: Update der Partikelgewichte $weight$.
3. Normalisieren: Nachdem nun in einem der Cluster Gewichte verändert wurden, muss $\forall C \in Clusters$ wieder normalisiert werden: Teilen aller Partikelgewichte durch die Summe der Partikelgewichte im Cluster C, Teilen aller *clusterweight* durch die Summe aller $clusterweight \in Clusters$
4. Aktuelle Cluster $AC \subseteq Clusters$ suchen: Suche Cluster deren Summe der Partikelalter kleiner ist als $maxAge \cdot particleAmount$ und deren $clusterweight > clusterThrowAwayWeight$ ist.

5. K-Means-/Lloyd-Algorithmus: Funktionsaufruf des K-Means-Algorithmus mit Übergabe *aller* Cluster $C \in Clusters$ und der Means der $C \in AC$. Der K-Means-Algorithmus berechnet eine Neuaufteilung der Partikel auf Cluster C'. Die konkrete Implementierung kann [Ahmad u. Dey, 2007] entnommen werden. Die Menge der neuen Cluster sei $Clusters'$, die Gesamtmenge der Partikel bleibt dabei zunächst gleich.

6. Anpassen der Gewichte an die neue Aufteilung: $\forall C' \in Clusters'$

   - Errechnen des neuen Clustergewichts: Das *clusterweight* des neuen Clusters C', ergibt sich aus der Summe aller Gewichte der enthaltenen Partikel, multipliziert mit deren altem Clustergewicht:
     $C'.clusterweight := \sum_{\forall particle \in C'} particle.weight \cdot particle.clusterweight$
     Die Summe über alle so berechneten Clustergewichte ergibt eins, weshalb keine Normierung notwendig ist, siehe [Vermaak u. a., 2003, S. 4]. Zunächst wird neue Clustergewicht $C'.clusterweight$ noch keinem Partikel zugewiesen.

   - Dann werden die Partikelgewichte der $particle \in C'$ neu berechnet:
     $particle.weight := \frac{particle.weight \cdot particle.clusterweight}{C'.clusterweight}$

   - Erst jetzt wird das neue Clustergewicht $C'.clusterweight$ allen Partikeln in C' zugewiesen: $\forall particle \in C' : particle.clusterweight = C'.clusterweight$

7. Ziehen von Partikeln: Das Ziehen von Partikeln wird mit einem "Low-variance-Sampler"-Algorithmus umgesetzt. Dieser wurde von S.Thrun [Thrun u. a., 2005, S. 110] für den einfachen Partikelfilter vorgeschlagen, und wird hier für jedes Cluster $C' \in Clusters'$ einmal ausgeführt. Dabei wird Index-Zugriff auf die Partikel im Cluster vorausgesetzt, in dem Sinne das $C'[i]$ der $i-$te Partikel aus C' ist.

```
1:  function Low-variance-Sampler(C')
2:      newParticles := ∅
3:      r := zufällige Zahl aus [0; 1/particleAmount]
4:      thresholdWeight := C'[0].weight
5:      index := 1
6:      for m ← 1 to particleAmount do
7:          U := r + (m − 1) · 1/particleAmount
8:          while U > thresholdWeight do
9:              index := index + 1
10:             thresholdWeight := thresholdWeight + C'[index − 1].weight
11:         füge Partikel C'[index − 1] in newParticles hinzu
```

12: **return** *newParticles*

Dieser MTT-fähige Partikelfilter-Algorithmus wird nun im folgenden Kapitel auf seine Güte untersucht.

6 Güte der eigenen Ansätze

Nachdem nun die beiden, im Rahmen dieser Arbeit umgesetzten, MTT-Verfahren erläutert wurden, soll ihre Güte verglichen werden. Dies soll anhand der in Kapitel 4 konzipierten Fehlerberechnung geschehen. Im Zuge dessen werden die Parameter der Verfahren optimiert. Die Parameter der Fehlerberechnung selbst wurden in Kapitel 4 festgelegt. Alle Fehler werden gerundet auf drei Nachkommastellen.

6.1 Greedy-Ansatz

Beim Greedy-Ansatz können zwei Paramter eingestellt werden (siehe 5.1.1):

1. $a \in \mathbb{N}$: Wenn $m.counter = a$ wird m aus M, der Menge der gespeicherten Detections, entfernt.
2. $s \in \mathbb{R}^+$: Der Schwellwert (in Metern), den die Distanz einer neuen Detection zu einer gespeicherten erreichen oder unterschreiten muss, als das sie einander zugeordnet werden.

Es wurden 1470 Tests durchgeführt. Dabei war $a \in [0; 20]$ und $s \in [0; 100]$, wobei keine äquidistanten Werte gewählt wurden. Ein Auszug aus den Testergebnissen zeigt Folgendes:

- $a \leq 1$ führt zur direkten Löschung der Detection, und dementsprechend, unabhängig vom Wert von s zu einem sehr hohen Fehler $F = 49.978$
- $s = 0$ lässt nur Zuordnungen zu deren Koordinaten exakt aufeinander liegen. Entsprechend ist F, außer für $a = 2$ und $a = 3$, ebenfalls sehr hoch: $F = 49.978$
- für $s = 0.1$ liegen die Werte von $F \in [33.775; 2.194]$ wobei für größere a auch größere F auftreten, jedoch ohne direkt proportionalen Zusammenhang.
- $s = 0.2$ $F \in [15.725; 2.163]$
- $s = 0.3$ $F \in [7.549; 2.111]$
- Dieser Trend setzt sich fort, bis $s = 0.7$, $F \in [2.409; 2.000]$
- Das Beste Ergebnis wird erzielt für $a = 5$ und $s = 0.7 : F = 2.000$.
- Danach steigen die Werte von F wieder, $s = 0.8$, $F \in [2.387; 2.011]$
- $s = 0.9$ $F \in [2.370; 2.0007]$

- $s = 1$ $F \in [2.362; 2.038]$

Somit sollte der Greedy-MTT-Algorithmus im Roboterfußball mit den Parametern $a = 5$ und $s = 0.7$ verwendet werden.

6.2 Partikelfilter-Ansatz

Der MTT-Partikelfilter-Algorithmus hat vier Paramter die eingestellt werden müssen (Kapitel 5.2.2):

- $particleAmount \in \mathbb{N}$, getestete Werte: 10; 50; 500; 1000;
- $clusterThrowAwayWeight \in \mathbb{R}^+$, getestete Werte: 0.05; 0.1; 0.3;
- $newClusterIfDetectionThisFar \in \mathbb{R}^+$ getestete Werte: 0.05; 0.1; 0.5
- $maxAge \in \mathbb{N}$, getestete Werte: 1; 10; 50;

Insgesamt wurden 430 Tests durchgeführt. Im Folgenden eine Auswahl der Ergebnisse:

F	$particleAmount$	$clusterThrowAw..$	$newClusterIf..$	$maxAge$
2.084 bis 2.415	alle Werte	alle Werte	alle Werte	1
1.899 bis 2.597	alle Werte	alle Werte	alle Werte	10
1.885 bis 2.723	alle Werte	alle Werte	alle Werte	50

Eine große Konstante $maxAge$ hat zur Folge, dass in Schritt 4 des MTT-Partikelfilter-Algorithmus' mehr Cluster in die Menge der aktuellen Cluster AC übernommen werden, was eine Verringerung von F zur Folge hat.

F	$particleAmount$	$clusterThrowAw..$	$newClusterIf..$	$maxAge$
2.259 bis 2.723	10	alle Werte	alle Werte	alle Werte
2.061 bis 2.351	50	alle Werte	alle Werte	alle Werte
1.927 bis 2.173	500	alle Werte	alle Werte	alle Werte
1.885 bis 2.171	1000	alle Werte	alle Werte	alle Werte

Dass mehr Partikel je Cluster die Qualität des Ergebnisses verbessern, entspricht den Erwartungen an einen Partikelfilter. Je mehr Partikel, desto besser kann $bel(x_t)$ die Wahrscheinlichkeitsverteilung annähern. Jedoch steigt mit der Partikelzahl auch der Rechenaufwand, der bei Online-Verfahren von wesentlicher Bedeutung ist.

F	$particleAmount$	$clusterThrowAw..$	$newClusterIf..$	$maxAge$
1.910 bis 2.723	alle Werte	0.05	alle Werte	alle Werte
1.885 bis 2.444	alle Werte	0.1	alle Werte	alle Werte
2.017 bis 2.414	alle Werte	0.3	alle Werte	alle Werte

Das $clusterThrowAwayWeight$ welches in Schritt 4 des MTT-Partikelfilters vom Cluster überschritten werden muss, um zur Menge der aktuellen Cluster AC zu gehören, hat hier seinen optimalen Wert bei 0.1. Dieser hängt jedoch von der Gesamtzahl der Cluster ab, da die Summe der Clustergewichte stets eins ist. Die Gesamtzahl der Cluster wiederum hängt von $newClusterIfDetectionThisFar$, und $maxAge$, siehe S. 26.

F	$particleAmount$	$clusterThrowAw..$	$newClusterIf..$	$maxAge$
1.885 bis 2.723	alle Werte	alle Werte	0.05	alle Werte
1.925 bis 2.690	alle Werte	alle Werte	0.1	alle Werte
2.011 bis 2.535	alle Werte	alle Werte	0.5	alle Werte

Auch hier ist das Ergebnis eindeutig: $newClusterIfDetectionThisFar$ sollte klein gewählt werden. Dies hat zur Folge, dass in Schritt 2 des MTT-Partikelfilters oft neue Cluster erzeugt werden.

Das Minimum von F, 1.885 wird für die Parameter $particleAmount = 1000$, $clusterThrowAwayWeight = 0.1$, $newClusterIfDetectionThisFar = 0.05$ und $maxAge = 50$ erreicht. Dies entspricht jeweils den Parametern, die bei den isolierten Betrachtungen der einzelnen Werte ebenfalls das Optimum darstellten.

Vergleich

Damit ist der MTT-Partikelfilter-Ansatz mit $F = 1.885$ dem Greedy-Ansatz mit $F = 2.000$ knapp überlegen. Es ist zu erwarten, dass sich der MTT-Partikelfilter mit größerem $maxAge$, kleinerem $newClusterIfDetectionThisFar$, und insbesondere größerer Partikelzahl $particleAmount$ noch verbessert.

7 Zusammenfassung, Diskussion und Ausblick

Nach Ergründung des aktuellen Stands der Technik auf dem Gebiet der MTT-Verfahren, wurde in dieser Arbeit ein Greedy-Ansatz und ein Partikelfilter-Ansatz zum MTT umgesetzt. Die Güte der Ansätze wurde am, eigens für die Beurteilung der Eignung von MTT-Verfahren zum Einsatz im Roboterfußball entwickelten, Gütemaß gemessen. Es zeigte sich das der Greedy-Ansatz dem Partikelfilter-Ansatz nur knapp unterliegt. Zwar ließe sich das Ergebnis des Partikelfilters noch verbessern, jedoch nur auf Kosten der Laufzeit. Schon jetzt ist der Greedy-Ansatz um ein vielfaches schneller, was auf der mobilen Roboterplattform Nao, mit begrenzten Ressourcen, einen nicht zu verachtenden Vorteil darstellt.

Die Ergebnisse beider Ansätze wären jedoch besser, wenn die vorgeschaltete Objekterkennung besser funktionieren würde. Diese liefert nicht nur viele False-Positives und False-Negatives, sondern auch quantitativ wenig Ergebnisse. Was sich zum Einen darin begründet, dass der Roboter mehr als die Hälfte der Spielzeit die Kamera in seinem Kinn nutzt, mit der er nur den Boden vor sich sieht, und selten andere Spieler. Zum Anderen bewegt er, bei der Suche nach dem Ball, den Kopf schnell hin und her, was ausreicht um Bälle zu erkennen, jedoch nicht um Roboter zu erkennen.

Zusammenfassend lässt sich sagen, dass beide hier umgesetzten MTT-Ansätze auf dem Nao ausgeführt werden können, die Ergebnisse jedoch ihren Nutzen deutlich vergrößern werden, wenn quantitativ mehr Detections zur Verfügung stehen.

Glossar

$clusterThrowAwayWeight$ .. Konstante $\in \mathbb{R}^+$ – S. 26

$maxAge$ ..... Konstante $\in \mathbb{N}$ – S. 26

$newClusterIfDetectionThisFar$ Konstante $\in \mathbb{R}^+$ – S. 26

$particleAmount$ ..... Konstante $\in \mathbb{N}$ – S. 26

WK ..... Menge der Welt-Koordinaten – S. 5

Cluster ..... S. 25

Data Association ..... Zuordnung von Detections $d \in D$ zu einem Individuum $i \in I_\perp$ – S. 3

False-Negative ..... Fälschlicherweise negatives Ergebnis – S. 3

False-Positive ..... Fälschlicherweise positives Ergebnis – S. 3

Frame ..... Nummeriertes Einzelbild t eines Videos, $t \in Frames \subseteq \mathbb{N}$ – S. 3

Grond-Truth-Daten ..... S. 11

MCMCDA ..... Markov Chain Monte Carlo Data association – S. 11

Mean ..... S. 25

Menge der Detections ..... $D = \{d_j^t = (t, j, x, y, c) | t \in Frames, j \in \mathbb{N}; (x, y) \in WK\}$ – S. 2

Menge der Individuen ..... $I_\perp = \{0, \cdots, N\} \cup \{\perp\}$ – S. 3

Menge der Trajektorien ..... $T = \{T_i : t \to (x, y)\}$ mit $i \in I$ und $t \in Frames$ – S. 4

MTT ..... Multi-Target Tracking – S. 1

Objekte ..... In der Realität vorhandene Objekte die getrackt werden sollen – S. 2

Sliding-Window-Verfahren ... S. 10

SPL ..... Standard Platform Liga – S. 1

Zustandsraum ..... Menge der Zustände die ein System annehmen kann – S. 11

Literaturverzeichnis

[Ahmad u. Dey 2007] Ahmad, Amir ; Dey, Lipika: A k-mean clustering algorithm for mixed numeric and categorical data. In: *Data Knowl. Eng.* 63 (2007), Nr. 2, S. 503–527 5

[Aldebaran-Robotics] 1.1

[Andriyenko u. a. 2012] Andriyenko, Anton ; Schindler, Konrad ; Roth, Stefan: Discrete-Continuous Optimization for Multi-Target Tracking. In: *Conference on Computer Vision and Pattern Recognition CVPR*, 2012 1.2, 3.1.3, 3.1.4, 3.2.4

[Benfold u. Reid 2011] Benfold, Ben ; Reid, Ian: Stable Multi-Target Tracking in Real-Time Surveillance Video. In: *Conference on Computer Vision and Pattern Recognition CVPR*, 2011, S. 3457–3464 3.1.3, 3.2.3

[Björck 1996] Björck, Å.: *Numerical Methods for Least Squares Problems*. Philadelphia : SIAM, 1996 2

[Black 1999a] Black, Paul E.: *Algorithms and Theory of Computation Handbook*. http://www.nist.gov/dads/HTML/offline.html, 1999. – [Online; accessed 18.09.2013] 3.1.1

[Black 1999b] Black, Paul E.: *Algorithms and Theory of Computation Handbook*. http://www.nist.gov/dads/HTML/offline.html, 1999. – [Online; accessed 18.09.2013] 3.1.1

[Breitenstein u. a. 2011] Breitenstein, Michael D. ; Reichlin, Fabian ; Leibe, Bastian ; Koller-Meier, Esther ; Gool, Luc J. V.: Online Multi-Person Tracking-by-Detection from a Single, Uncalibrated Camera. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33 (2011), Nr. 9, S. 1820–1833 1.2, 3.1.4, 3.2.2, 5.2.2

[Cormen u. a. 2010] Cormen, T H. ; Leiserson, Charles E. ; Rivest, Ronald ; Stein, Clifford: *Algorithmen: eine Einführung*. Oldenbourg, 2010. – ISBN 978–3486590029 5.1

[Dubrofsky 2009] Dubrofsky, Elan: *Homography Estimation*, University of British Columbia, Masterarbeit, March 2009. – https://www.cs.ubc.ca/grads/resources/thesis/May09/Dubrofsky_Elan.pdf [Online; accessed 15.07.2013] 2.2.1, 2.2.2

[Gupta u. a. 2011] Gupta, Ankur ; Little, James J. ; Woodham, Robert J.: Using Line and Ellipse Features for Rectification of Broadcast Hockey Video. In: *CRV*, IEEE, 2011. – ISBN 978–1–61284–430–5, 32-39 4.1

[Hartley 1997] Hartley, Richard I.: In Defense of the Eight-Point Algorithm. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19 (1997), June, Nr. 6, S. 580–593 4.1

[Hartley u. Zisserman 2004] Hartley, Richard I. ; Zisserman, Andrew: *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, 2004 2.2.1, 2.2.2

[Oh u. a. 2004] Oh, Songhwai ; Russell, Stuart ; Sastry, Shankar: *Markov Chain Monte Carlo Data Association for General Multiple-Target Tracking Problems*. 2004 1.2

[Oh u. a. 2009] Oh, Songhwai ; Russell, Stuart J. ; Sastry, Shankar: Markov Chain Monte Carlo Data Association for Multi-Target Tracking. In: *IEEE Transactions on Automatic Control* 54 (2009), March, Nr. 3, S. 481–497 3.1.2, 3.1.3

[Robocup Technical Committee 2013] Robocup Technical Committee: *RoboCup Standard Platform League (Nao) Rule Book*. http://www.tzi.de/spl/bin/view/Website/Downloads, Mai 2013. – [Online; accessed 19.08.2013] 1

[Schmidt u. Alahari 2011] Schmidt, Mark W. ; Alahari, Karteek: Generalized Fast Approximate Energy Minimization via Graph Cuts: a-Expansion b-Shrink Moves. In: *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, 2011, S. 653–660 3.2.4

[Song u. Shin 2003] Song, Taek L. ; Shin, Sang A.: A probabilistic nearest neighbor filter for m validated measurements. In: *International Conference of Information Fusion* 6 (2003), July, Nr. 2, S. 1274 – 1281 3.1.2

[Sven Behnke 2013] Sven Behnke: *RoboCup in Deutschland*. http://www.ais.uni-bonn.de/robocup.de, September 2013. – [Online; accessed 07.09.2013] 1.1

[Thrun u. a. 2005] Thrun, Sebastian ; Burgard, Wolfram ; Fox, Dieter: *Probabilistic Robotics*. The MIT Press, 2005. – ISBN 0262201623 3.2.1, 5.2.1, 7

[Vermaak u. a. 2003] Vermaak, Jaco ; Doucet, Arnaud ; Pérez, Patrick: Maintaining Multi-Modality through Mixture Tracking. In: *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, 2003 (ICCV '03). – ISBN 0–7695–1950–4, S. 1110– 5.2.2, 6