Erfahrungsbasierte Lernmethoden zur visuellen Trajektorienvorhersage humanoider Roboter

D I P L O M A R B E I T

Humboldt-Universität zu Berlin
Mathematisch-Naturwissenschaftliche Fakultät II
Institut für Informatik

eingereicht von: Thomas Krause
geb. am 14.08.1984 in Berlin

Erstgutachter/Betreuer: Prof. Dr. sc. nat. Verena V. Hafner
Zweitgutachter: Prof. Dr. Hans-Dieter Burkhard

Berlin, den 16.08.2011

Zusammenfassung

Robotersysteme müssen in der Lage sein, ihre Aufgabe in einer komplexen und dynamischen Umwelt zu erfüllen. Dabei interagieren sie mit anderen intentional handelnden Agenten. In dieser Arbeit wird der humanoide Nao Roboter als Agent verwendet und die Umwelt entspricht einem Fußballspiel mit mehreren Robotern. Ziel ist es, die anderen Fußballroboter zu erkennen und ihre Bewegung auf dem Spielfeld zu modellieren. Dieses Modell dient dann als Grundlage zum erfahrungsbasierten Lernen. Mithilfe des Lernens soll es möglich sein, die Bewegung des anderen Agenten vorherzusehen. Verschiedene Lernalgorithmen, Modelle und Parameter von Modellen werden miteinander verglichen.

Abstract

Robot systems must be able to achieve their task in a complex and dynamic enviroment. While executing their task they have to interact with other intentional acting agents. In this work the humanoid Nao robot will be used as an agent and the enviroment corresponds to a soccer game with several robots. The goal ist to detect the other robots and model their movement on the soccer field. This model serves as base for knowledge based learning. By using learning we shall be able to predict the movement of the other agents. Different learning algorithms, models and parameters of the models will be compared to each other.

Inhaltsverzeichnis

1 Einführung

Menschen bewegen sich selbstverständlich jeden Tag in einer komplexen und dynamischen Umwelt. Die Komplexität dieser Umwelt begründet sich in der schieren Menge und Art an bekannten und unbekannten Objekten und Unterobjekten, die in dieser Welt vorhanden sind, und durch die Größe der Welt selbst. Unsere Umwelt ist dynamisch, da sich ihre Elemente in jedem Augenblick verändern können. Die Veränderungen geschehen durch physikalische Prozesse oder durch die Einwirkung von anderen agierenden Teilnehmern in der Umwelt.

Während Menschen der Umgang mit der Umwelt als alltäglicher Vorgang erscheint, so sind die Vorgänge zum Verstehen und Deuten der Umwelt alles andere als trivial. Möchte man computergesteuerte autonome Robotersysteme in unserer Umwelt einsetzen, so benötigen sie einige der Fähigkeiten, die auch der Mensch besitzt, um sich in dieser Welt zurechtzufinden. Ein Beispiel für solch eine Fähigkeit ist die Vorhersage, in welche Richtung sich ein Objekt bewegen wird. Dazu muss einerseits ein gewisses physikalisches Wissen vorhanden sein, bei einem sich intentional bewegenden Menschen aber auch die Intention des Menschen erahnt werden.

1.1 Intentionen anderer Agenten und lernen von physikalischen Modellen

Aktiv agierende dynamische Objekte der Umwelt werden im Folgenden als Agent bezeichnet. Diese Agenten besitzen einen Mechanismus, der es ihnen erlaubt Entscheidungen zu treffen. Möchte man nun verstehen, welche Aktionen ein anderer Agent ausführen wird, so ist die Fähigkeit Annahmen über das mentale Modell des anderen Agenten zu tätigen von Vorteil. Solch eine Fähigkeit wird in der Psychologie als „Theory of Mind“ bezeichnet. Dieser Begriff wurde zum ersten Mal durch David Premack und Guy Woodruff 1978 in einem Artikel über die mögliche Fähigkeit von Schimpansen zu eben genau dieser „Theory of Mind“ erwähnt. Darin definieren sie den Begriff wie folgt:

> „An individual has a theory of mind if he imputes mentals states to himself and others.“ [33]

Ein Individuum unterstellt also anderen Individuen (oder sich selbst) mentale Zustände. Hypothesen über die Gefühle, Intentionen und beabsichtigte Handlungen eines Gegenübers können als Grundlage zur eigenen Entscheidungsfindung dienen.

Kinder nehmen die Umwelt bis zum 18. Monat nur als sich ständig aktualisierendes Modell ohne Alternativmodelle war, in dem Objekte nur im Augenblick ihrer Betrachtung existieren [15, Seite 37]. Die Fähigkeit zur Umweltwahrnehmung entwickelt sich

immer weiter (zum Beispiel durch die Entwicklung von Gedächtnis und Alternativmodellen) und mit ungefähr 4 Jahren sind Kinder in der Lage, den Unterschied zwischen dem mentalen Modell und der tatsächlich dadurch repräsentierten Welt zu unterscheiden [15, Seite 41]. Säuglinge sind allerdings schon sehr viel früher in der Lage, physikalische Gesetzmäßigkeiten zu lernen und anzuwenden. So wissen sie bereits „in den ersten Lebensmonaten“, dass sich zwei Festkörper bei einem Aufprall nicht durchdringen sondern voneinander abgestoßen werden [24, Seite 109]. Dieses Wissen äußert sich durch längere Betrachtung („wundern“) von Ereignissen, die von der physikalischen Norm abweichen. Auch können sie intuitiv zwischen belebten Subjekten (in unserem Sinne intentional agierende Agenten) und unbelebten Objekten unterscheiden [24, Seite 109].

Agenten steuern ihre Bewegung selbstständig unter Beachtung der physikalischen Gegebenheiten. Die verschiedenen Laufwege, die daraus entstehen, können ebenfalls von außen betrachtet Gesetzmäßigkeiten aufweisen. Das Lernen dieser Gesetzmäßigkeiten ist für Menschen im Alltag sehr hilfreich. So erlaubt sie uns anderen Personen (teilweise unterbewusst) auszuweichen oder die Laufbahn von geworfenen Objekten vorherzusehen. Dazu kann das Hineinversetzen in die Intentionen des anderen von Vorteil sein, ist allerdings keine zwingende Bedingung für eine erfolgreiche Vorhersage.

1.2 Testfelder für Robotik

Möchte man Mechanismen erforschen, die sich mit der Interaktion von Agenten in einer natürlichen Umwelt befassen, so benötigt man ein geeignetes Testfeld. Das Testfeld sollte dabei die Probleme aufweisen, an deren Lösung man arbeitet, ohne die volle Komplexität der allgemeinen natürlichen Umwelt zu besitzen. Der RoboCup wurde 1996 ins Leben gerufen und ist ein solches Testfeld der Künstlichen Intelligenz (KI) für realitätsnahe Szenarien [11]. Inzwischen existieren verschiedene Teildisziplinen im RoboCup, bei denen sich aber ein Großteil mit der ursprünglichen Gründungsidee befasst. Nach dieser Vision sollen Roboter gegeneinander Fußball spielen und später so gut werden, dass sie 2050 gegen den menschlichen Weltmeister im Fußball gewinnen können. Jedes Jahr werden Weltmeisterschaften ausgetragen, bei denen die Roboter von Forschern gegeneinander antreten. Dadurch, dass Fußball im Gegensatz zu reinen wissenschaftlichen Veröffentlichungen ein Wettbewerb ist und es eindeutige Gewinner eines Spiels gibt, können erfolgreiche Konzepte gut lokalisiert und verglichen werden. Neben dem Wettbewerb dienen die Weltmeisterschaften auch dem Austausch von Ideen und Technologien zwischen den Forschern.

Der Grund für die Entscheidung für Fußball ist, dass das Fußballspiel eine sehr dynamische Sportart ist, bei der sich die Situationen in jedem Augenblick ändern. Zudem ist das Fußballfeld zwar eine eingeschränkte Umwelt, aber doch viel reichhaltiger, komplexer und realitätsnäher als zum Beispiel das Schachspiel. Auf dem Fußballfeld müssen sich die Roboter in einer echten physischen Umgebung orientieren, Entscheidungen treffen und sich fortbewegen. Zudem ist Fußball stark interaktiv. Ein Spieler agiert nicht allein, sondern muss mit Mitspielern und gegnerischen Robotern im Rahmen der Regeln kooperieren. Die Kommunikation ist zudem nicht perfekt, sondern durch Umgebungslärm

gestört. Im RoboCup sollen sich die Roboter der Vision schrittweise annähern, daher spielen im Moment noch Roboter gegen Roboter. Das Ziel ist aber, die Roboter immer menschenähnlicher werden zu lassen, damit sie in der Lage sind gegen reale Menschen spielen und interagieren zu können.

Auch im Roboterfußball ist eine Vorhersage über die Bewegung eines gegnerischen Spielers von Vorteil, zum Beispiel um ihm den Laufweg abzuschneiden und einen Ballgewinn zu erreichen. Im Gegensatz zu einem rein physikalisch bewegten Objekt handelt ein Agent im Roboterfußball intentional, kann also auch seine Richtung aktiv aufgrund einer eigenen Entscheidungsfindung basierend auf den ihn vorliegenden Umweltdaten ändern.

1.3 Fragestellung

Die Frage ist nun, ob es möglich ist, die zweidimensionale Bewegung von Fußball spielenden Robotern auf dem Spielfeld zu erkennen, aufzuzeichnen und vorherzusagen. Dabei ähnelt diese Fähigkeit den Fähigkeiten von unter vierjährigen Kindern, nur dass das Modell nicht nur die Physik sondern auch die typischen Entscheidungsmodelle des anderen Roboters mitlernen muss. Diese Vorhersage soll auf Erfahrung beruhen und nicht voraussetzen, dass die konkreten Gedanken des anderen Agenten nachvollzogen werden müssen. Um dieses Ziel zu erreichen, soll ein möglichst allgemeines Modell gefunden werden, das möglichst wenig Annahmen über die Umwelt und das zu lösende Problem enthält. Stattdessen soll das Modell durch die Lernbeispiele, mit denen der Agent konfrontiert wird, konkretisiert und auf die spezifische Problemstellung trainiert werden. In einem konkreten Experiment soll die Qualität der Vorhersage von Bewegung durch verschiedene erfahrungsbasierte Lernverfahren verglichen werden.

1.4 Vorgehensweise und Gliederung

Zum Vorhersagen von Bewegung durch Erfahrungen benötigt man geeignete Lernverfahren. Eine Auswahl solcher Verfahren wird in Kapitel 2 beschrieben. Zu den verwendeten Lernverfahren gehören k-NN (Kapitel 2.2), Künstliche Neuronale Netze (Kapitel 2.3) und Entscheidungsbäume (Kapitel 2.4). Die Lernverfahren arbeiten auf allgemein formulierten Trainings- und Testdaten in Matrixform. Durch geeignete Kodierung muss das vorhandene Wissen über die Bewegungen in diese Form gebracht werden. Dabei können verschiedene Arten von Merkmalen auf den Sensordaten generiert werden. Es muss auch geklärt werden, welche Informationen über die Welt einem Agenten überhaupt zur Verfügung stehen und was die beobachteten Verhaltensweisen und die Aktionen des Roboters verbindet. In Kapitel 3 wird die Diskussion über diese Problematik geführt und ein konkreter Vorschlag gemacht, wie solch eine Kodierung aussehen kann und welche Merkmale benutzt werden können. Zudem wird in diesem Kapitel auch ein weiteres Verfahren zur Vorhersage vorgestellt, das auf dem sogenannten „episodischen Gedächtnis“ aufbaut. Dabei werden Umweltsituationen mit den Positionen von verschiedenen Objekten als episodische Folge von Einzelereignissen gespeichert. Für die Vorhersage werden

nun ähnliche Situationen gesucht und deren weiterer Verlauf wird zur Vorhersage genutzt. Um überhaupt Erfahrungen sammeln zu können, muss ein Roboter die Objekte der Umwelt wahrnehmen können. In dieser Arbeit soll der Fokus auf der visuellen Wahrnehmung anderer Objekte liegen. Die allgemeine visuelle Wahrnehmung von Objekten durch Merkmalsextraktion auf Kamerabildern wird in Kapitel 4 behandelt.

Um die Fragestellung zu untersuchen, werden ein konkreter Anwendungsfall und eine konkrete Roboterplattform benötigt. Kapitel 5 beschreibt den hier verwendeten humanoiden Nao Roboter und weitere technische Hilfsmittel, wie zum Beispiel die OpenCV-Softwarebibliothek, die Algorithmen zur Bildverarbeitung und zum maschinellen Lernen bietet. In Kapitel 6 wird ein konkretes Verfahren beschrieben, um den Nao Roboter mithilfe der Techniken aus Kapitel 4 in einem Kamerabild erkennen zu können. Dieses Verfahren musste auch dahingehend angepasst werden, dass die Rechenkapazität auf dem Roboter im Vergleich zu modernen PCs nicht leistungsfähig ist. Die verschiedenen Erkenntnisse und Techniken werden in Kapitel 7 zusammengeführt. Dort wird das Experiment beschrieben, mit dem die verschiedenen Lernverfahren miteinander verglichen werden sollen. Die Ergebnisse werden vorgestellt und in Kapitel 8 kann dann ein Fazit über die in dieser Arbeit gewonnenen Erkenntnisse gezogen werden.

2 Lernverfahren

Dieses Kapitel befasst sich damit, wie Maschinen lernen können, und beschreibt klassische Verfahren, die später im Experiment verwendet werden sollen. Das Lernen soll die Agenten befähigen Vorhersagen basierend auf Erfahrungen zu tätigen. Es werden drei unterschiedliche Lernverfahren vorgestellt, deren Leistungsfähigkeit zum Lösen des spezifischen Problems im Experiment verglichen werden kann. Zunächst wird aber die Regressionsanalyse als allgemeines Konzept zur Vorhersage beschrieben.

2.1 Statistische Regressionsanalyse

Die Regressionsanalyse ist ein statistisches Verfahren, das sich zum Vorhersagen von Werten für Zufallsvariablen eignet. In unserem Zufallsexperiment haben die Beobachtungen die Form $(Y_i, \boldsymbol{X_i})$ mit $i \in \mathbb{N}, i \leq n$. n beschreibt die Anzahl der Beobachtungen. $\boldsymbol{X_i}$ ist ein Vektor von Zufallsvariablen. $p^X \in \mathbb{N}$ bezeichnet die Dimension dieses Vektors. Die Zufallsvariablen entsprechen den Sensorinformationen des Systems zum Zeitpunkt der Beobachtung. Y_i bezeichnet die dazugehörige Antwortvariable. Diese und folgende Notationen und Definitionen zur Regressionsanalyse orientieren sich an [12].

Es wird angenommen, dass die Zufallsvariablen aus $\boldsymbol{X_i}$ zueinander unabhängig sind. Diese Annahme wird von den meisten der folgenden Verfahren als Grundlage benötigt, ist in der Realität aber kaum gegeben. So könnte zum Beispiel ein Roboter nur eine maximale Laufgeschwindigkeit aufweisen, um nicht instabil zu werden. In diesem Fall wären die Einzelkomponenten des Geschwindigkeitsvektors nicht unabhängig zueinander. Trotzdem ist die Unabhängigkeitsannahme eine wichtige Vereinfachung, die die folgenden Berechnungen erst möglich macht.

Bei der parametrisierten linearen Regressionsanalyse wird angenommen, dass es eine Korrelation zwischen Y_i und $\boldsymbol{X_i}$ gibt und dieser Zusammenhang mit der Gleichung

$$Y_i = \boldsymbol{\beta}^T \boldsymbol{X_i} + \varepsilon_i = \beta_0 + \beta_1 X_{1,i} + \ldots + \beta_p X_{p,i} + \varepsilon_i \tag{2.1}$$

beschrieben werden kann. $\boldsymbol{\beta}$ ist ein Vektor von Parametern und ε_i der spezifische Fehler für die i. Beobachtung. Das Modell besteht also aus einer linearen Grundfunktion und einem Satz von a priori unbekannten Parametern, die mithilfe der Beobachtungen gelernt werden müssen.

Die nicht-parametrisierte Regressionsanalyse verwendet hingegen eine glatte Funktion f, um die Werte für Y_i vorherzusagen. Dadurch erhält man die Gleichung

$$Y_i = f(\boldsymbol{X_i}) + \varepsilon_i \ , \tag{2.2}$$

bei der keine variablen Parameter benötigt werden, dafür aber die Struktur von f nicht fest definiert ist. Anstatt der Parameter bildet in diesem Fall die Funktion selbst das Modell.

2.2 k-Nearest-Neighbor-Verfahren

k-Nearest-Neighbor (k-NN) ist ein Verfahren, bei dem das gelernte Modell durch Beispieldaten aus tatsächlichen Beobachtungen gebildet wird. Es wird daher oft auch als „Learning by Example“ bezeichnet. Ein anderer Begriff ist „Lazy Learning“, da k-NN im Gegensatz zu den meisten anderen Lernverfahren keine Generalisierung vornimmt, sondern nur bekannte Daten wiedererkennen kann [16]. Je nach Anwendung ist k-NN ein Regressions- oder Klassifizierungsverfahren. Im Folgenden soll das Regressionsverfahren näher betrachtet werden.

Angenommen der Vektor $\boldsymbol{x}_i \in W, i \in \mathbb{N}$ beschreibt eine Sensorinformation der i. Beobachtung. W ist ein Vektorraum der Dimension $p^X \in \mathbb{N}$ (zum Beispiel $W = \mathbb{R}^{p^X}$). Während der Lernphase werden diese Sensorinformationen gesammelt und abgespeichert. Dies ergibt den Vektor

$$D^X = \begin{pmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_n^T \end{pmatrix} \tag{2.3}$$

aller Sensorinformationen im Lernzeitraum n. Man kann D^X auch als Matrix darstellen, bei der der Index der Zeile i repräsentiert und jede Zeile einem Eingangsvektor entspricht.

$$D^X = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & \dots & x_{1,p^X} \\ x_{2,1} & x_{2,2} & x_{2,3} & \dots & x_{2,p^X} \\ \dots & \dots & \dots & \ddots & \dots \\ x_{n,1} & x_{n,2} & x_{n,3} & \dots & x_{n,p^X} \end{pmatrix} \tag{2.4}$$

Nach der Lernphase können für einen neuen Eingabevektor $\boldsymbol{x}_{neu}$ nun die k ähnlichsten Vektoren in D^X bestimmt werden. Dafür benötigt man ein geeignetes Abstandsmaß. Da im Folgenden mit geometrischen Positionen gearbeitet werden soll, ist eine Möglichkeit die euklidische Distanz

$$d(\boldsymbol{x}_{neu}, \boldsymbol{x}_i) = \sqrt{\sum_{j=1}^{p^X} \left(x_{neu,j} - x_{i,j}\right)^2} \,. \tag{2.5}$$

Der einfachste Algorithmus zum Finden dieser Nachbarn ist eine lineare Suche über die gesamte Matrix D^X. Dabei wird jedes Element aus D^X mit $\boldsymbol{x}_{neu}$ verglichen und eine Menge $K(\boldsymbol{x}_{neu})$ mit den Indizes i der k ähnlichsten Elementen erstellt. Sollen nicht nur ähnliche Nachbarn gefunden, sondern auch Vorhersagen getroffen werden, benötigt man

einen zweiten Vektor D^Y, der jeder Beobachtung zu einem Zeitpunkt $i \in \mathbb{N}$ $(i \leq n)$ einen Ausgangswert y_i zuordnet:

$$D^Y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_n \end{pmatrix} . \tag{2.6}$$

Es gilt $y_i \in \mathbb{R}$ für ein Regressionsverfahren oder $y_i \in \mathbb{N}$ für eine Klassifizierungsaufgabe. Die Vorhersage $\hat{Y}_{neu}$ für einen neuen Vektor $\boldsymbol{x}_{neu}$ ist dann der arithmetische Mittelwert über alle passenden y_i

$$\hat{Y}_{neu}(\boldsymbol{x}_{neu}) = \frac{1}{k} \sum_{i \in K(\boldsymbol{x}_{neu})} y_i \, . \tag{2.7}$$

k-NN ist ein nicht parametrisiertes Regressionsverfahren wie in Gleichung (2.2) beschrieben. Die Modellfunktion f ist dabei durch $\hat{Y}_{neu}$ gegeben und die Zufallsvariable Y_i und der Vektor von Zufallsvariablen $\boldsymbol{X}_i$ werden durch die konkreten Mengen D^Y bzw. D^X realisiert.

Da k-NN ein einfach zu verstehendes und implementierendes Verfahren ist, wurde es bereits sehr früh ausführlich diskutiert und analysiert. Bereits 1967 wurde durch Cover und Hart festgestellt, dass jedes andere mögliche Klassifizierungsverfahren im Vergleich zur Klassifizierung durch k-NN (welche analog zur Regression definiert ist) die Wahrscheinlichkeit einer fehlerhaften Klassifizierung R maximal halbieren kann [13]. Für den Beweis nahmen sie an, dass die kleinste zu erreichende Wahrscheinlichkeit eines Klassifikationsfehlers R^* bei einem komplett verstandenen Problem durch den Bayes-Klassifikator modelliert werden kann. Bei M verschiedenen Klassen konnten sie zeigen, dass die Wahrscheinlichkeit für eine Fehlklassifikation durch k-NN durch die Formel

$$R^* \leq R \leq R^* \left(2 - \frac{MR^*}{M-1}\right) \tag{2.8}$$

beschränkt wird.

Allerdings ist k-NN sehr anfällig für hohe Dimensionen von $\boldsymbol{X}_i$ [12]. Je höher die Dimension, um so mehr Elemente werden im Distanzmaß zu einem Wert vereinigt. In [12] wurde gezeigt, dass bei unendlichen vielen Dimensionen der Unterschied zwischen beliebigen Beobachtungen $\boldsymbol{x}_i$ gegen 0 geht. Damit verliert das Distanzmaß bei hoher Dimensionalität die Aussagekraft und die Basisannahme für k-NN ist nicht mehr gegeben.

Das Durchsuchen einer großen Datenmenge um die exakten nächsten Nachbarn zu finden besitzt eine lineare Komplexität. Es ist allerdings möglich, Approximationsalgorithmen mit deutlich geringerer Komplexität zu nutzen. Ein Beispiel dafür ist FLANN (Fast approximate nearest neighbors with automatic algorithm configuration) [29]. Dieser verwendet randomisierte kd-Bäume (siehe [21], auch in Indexstrukturen für relationale Datenbanken eingesetzt) und hierarchisches k-Means-Clustering. Bei FLANN kann nicht garantiert werden, das auch wirklich die am nächsten gelegenen Nachbarn gefunden werden, die Ergebnisse reichen im praktischen Einsatz aber völlig aus. Anzumerken ist, dass bei FLANN immer alle Trainingsbeispiele im Index vorhanden sind und abgerufen

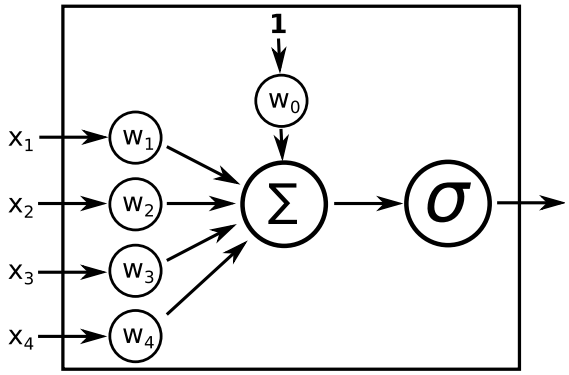


Abbildung 2.1: Schematische Darstellung des Modells eines künstlichen Neurons.

werden können. Dies unterscheidet diesen Ansatz von den generalisierenden Entscheidungsbäumen, die später erläutert werden.

2.3 Neuronale Netze

Das Künstliche Neuronale Netzwerk (ANN) ist ein von biologischen Vorbildern inspiriertes mathematisches Modell zum Lernen von Funktionen. Im Vergleich zu den realen Vorbildern ist es zwar stark vereinfacht und formalisiert, aber trotzdem noch ein sehr mächtiges Verfahren zum maschinellen Lernen. Dieses Kapitel beschreibt den grundsätzlichen Aufbau von Künstlichen Neuronalen Netzen und erläutert den Backpropagation-Algorithmus zum Lernen der Parameter eines ANN.

2.3.1 Grundlagen

Analog zum natürlichen Vorbild besteht ein ANN aus einer Menge von Neuronen, die durch Synapsen miteinander verbunden sind. Ein biologisches Neuron ist in Abbildung 2.2 gezeigt. Während biologische Neuronen über elektrische Impulse über die Synapsen bzw. Axonen und über chemische Botenstoffe an den Synapsenenden interagieren, wird ein künstliches Neuron über eine Formel beschrieben. Jedes Neuron besitzt eine Menge von Eingangswerten $x_1 \dots x_k$ (analog der eingehenden Synapsen). k ist die Anzahl aller Neuronen eines Systems. Damit können grundsätzlich alle Neuronen miteinander verknüpft werden. Zu jedem Eingang existiert zusätzlich ein Gewicht w_i. Sind zwei Neuronen nicht verknüpft, so kann dies durch das Gewicht $w_i = 0$ ausgedrückt werden. Die gewichtete Summe über alle Eingänge eines spezifischen $\boldsymbol{x}$

$$\varphi(\boldsymbol{x}) = \sum_{i=1}^{k} w_i x_i = \boldsymbol{w}\boldsymbol{x} \tag{2.9}$$

wird als Aktivierung $\varphi(\boldsymbol{x})$ bezeichnet. Eine Aktivierungsfunktion $\sigma(\varphi(\boldsymbol{x}))$ bestimmt den Ausgabewert des Neurons abhängig von φ.

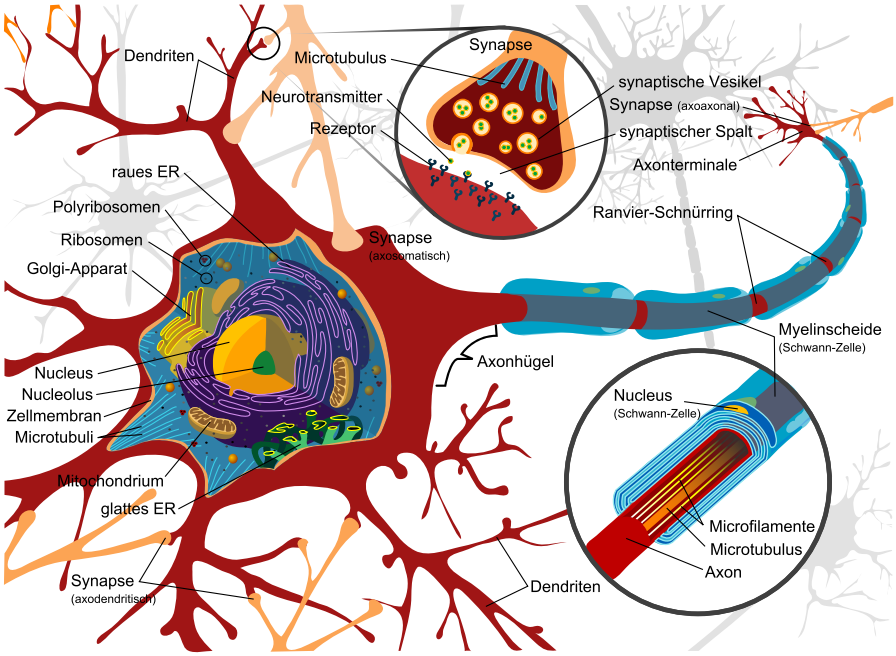


Abbildung 2.2: Zeichnung eines biologischen Neurons (Quelle: Wikipedia Commons, Mariana Ruiz Villarreal [41])

Dabei können verschiedene Funktionen gewählt werden, typisch sind die Schwellenwertfunktion

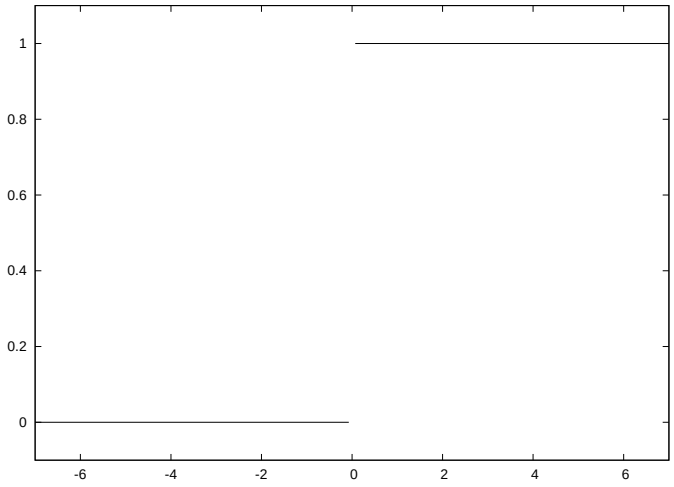
$$\sigma_{fix}(x) = \begin{cases} x > 0 : 1 \\ x \leq 0 : 0 \end{cases} \tag{2.10}$$

oder eine differenzierbare Sigmoidfunktion wie zum Beispiel

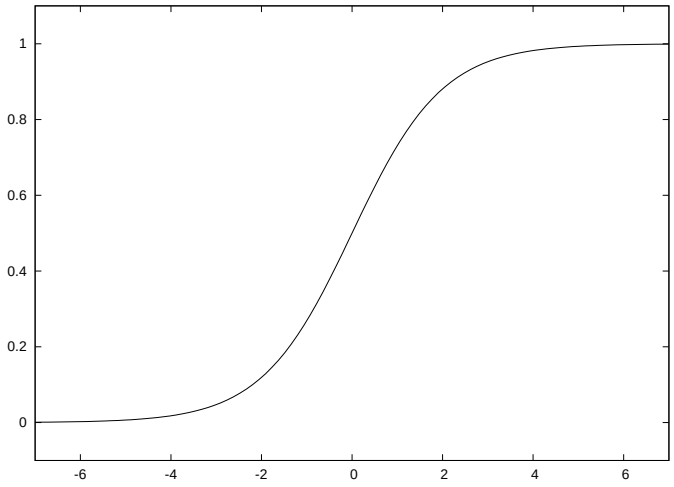
$$\sigma_{sig}(x) = \frac{1}{1 + e^{-x}} \,. \tag{2.11}$$

Zur Veranschaulichung der Charakteristik dieser Funktionen sind diese in Abbildung 2.3 dargestellt.

Sollen mehrere Neuronen zu einem Netz zusammengeschaltet werden, ergibt sich das Problem, dass der Wertebereich der Eingabe ($[-1..1]$) ungleich dem der Ausgabe ($[0..1]$) ist. Bei rein positiven Gewichten und dem Schwellwert 0 würde daher ein nachgeschaltetes Neuron unabhängig von den Eingangswerten des vorherigen Neurons immer aktiviert werden. Das Problem kann durch negative Gewichte an einigen Eingangsneuronen ausgeglichen werden. Um auch in Spezialfällen wie Neuronen mit nur einem vorgelagerten Neuron das Problem aufzulösen, wird ein spezielles Bias-Neuron benötigt. Dieses wird wie ein normales Neuron in die Berechnungen als x_0 einbezogen, hat aber immer den



(a) Schwellenwertfunktion (2.10)



(b) Sigmoidfunktion (2.11)

Abbildung 2.3: Verschiedene Aktivierungsfunktionen für ein ANN.

konstanten Wert 1. Durch sein Gewicht w_0 kann der Einfluss des Bias geregelt werden. Alternativ können andere Sigmoidfunktionen wie zum Beispiel

$$\sigma_{signorm}(x) = \frac{2}{1 + e^{-x}} - 1 \tag{2.12}$$

verwendet werden. Allerdings wird in der Literatur bevorzugt σ_{sig} verwendet und auch als Voraussetzung für Lernverfahren wie den BACKPROPAGATION-Algorithmus eingesetzt. Alle Verfahren, die auf σ_{sig} angewendet werden, können natürlich analog auch für $\sigma_{signorm}$ hergeleitet werden.

2.3.2 Feedforward-Netze

Die einzelnen Neuronen eines ANN können auf verschiedenste Weise zu einem Netzwerk kombiniert werden. Im Allgemeinen wird dabei zwischen verschiedenen Arten von Neuronen unterschieden, die in speziellen Schichten organisiert sein können. Eingabe-Neuronen, deren Eingangswerte direkt aus den (normierten) Vektorkomponenten der Eingangsfunktion gebildet werden, dienen als Schnittstelle der zu lernenden Daten in das Netzwerk. Das berechnete Ergebnis der durch das ANN beschriebenen Funktion wird durch den Wert der Aktivierungsfunktion von speziellen Ausgabeneuronen repräsentiert. Alle Eingabe-Neuronen zusammen bilden die Eingabeschicht; analog dazu bilden die Ausgabeneuronen die Ausgabeschicht. Neben den Eingabe- und Ausgabeneuronen existieren in ANN auch so genannte „hidden“ („versteckte“) Neuronen, die nicht direkt mit der Eingabe und Ausgabe des Netzwerkes verknüpft sind.

An sich können die Hidden-Neuronen in beliebiger Anzahl vorhanden und miteinander verknüpft sein. Diese Verknüpfung bildet eine Graphstruktur. Bei Feedforward-Netzwerken dürfen keine Zyklen in dieser Graphstruktur vorhanden sein. Dadurch wird die Berechenbarkeit des ANNs vereinfacht. In Netzen, die Zyklen erlauben, muss der Wert eines Neurons unter Umständen mehrmals berechnet werden, da sich die Eingabewerte durch die eigene Berechnung ändern können. Bei Feedforward-Netzen hingegen erfolgt die Berechnung des Neurons garantiert nur einmalig sobald die Berechnung aller verknüpften (also mit einem Gewicht ungleich null) Eingangs-Neuronen beendet ist.

Die Flexibilität der Anordnung der Neuronen ist einerseits eine große Stärke von ANN, andererseits verursacht sie auch eine hohe Komplexität. Um eine bestimmte Funktion auf Trainingsdaten zu lernen, müssen einerseits die Gewichte der Neuronen gelernt, andererseits aber auch eine grundsätzliche Anordnung des ANNs durch den Experimentator festgelegt werden. Daher bietet es sich bei Feedforward-Netzen an, die Hidden-Neuronen immer in einer oder mehreren Schichten zu organisieren. Hinzu kommen die Schichten aus Eingabe- bzw. Ausgabeneuronen. Die Ausgabewerte der Neuronen einer Schicht sind die ausschließlichen Eingabewerte der Neuronen der nächsten Schicht. Es existieren keine zusätzlichen Querverknüpfungen zu Neuronen anderer Schichten oder der gleichen Schicht.

In Abbildung 2.4 ist ein beispielhaftes Feedforward-Netzwerk angegeben. Es besteht aus einer Eingabeschicht mit drei Neuronen, das ANN berechnet also eine Funktion mit einem dreidimensionalen Eingabevektor. Die erste versteckte Schicht besitzt zwei

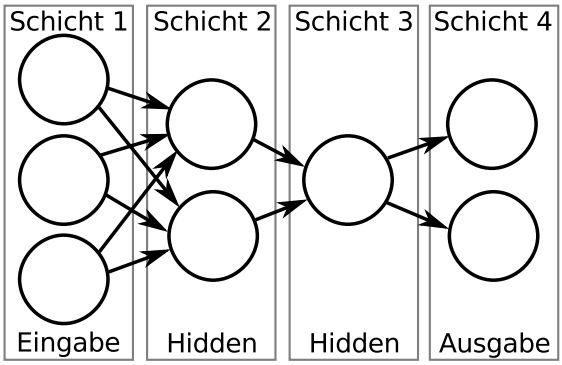


Abbildung 2.4: Ein beispielhaftes Feedforward-Netzwerk mit zwei verstecken Schichten und einer Eingabe- und Ausgabeschicht.

Neuronen, die zweite versteckte Schicht nur eines und das ANN berechnet einen zweidimensionalen Vektor als Ergebnis der Funktion.

Feedforward-Netze mit einer Eingabeschicht, einer Ausgabeschicht und zwei Hidden Layer können theoretisch jede beliebige Funktion approximieren [28, Seite 105]. Allerdings ist die Anzahl Neuronen in den jeweiligen Hidden Layern dafür entscheidend und es müssen die passenden Gewichte gefunden werden.

2.3.3 Lernen der Gewichte durch Backpropagation

Wenn die Struktur des ANNs festgelegt ist, bleibt die Frage wie die Gewichte bestimmt werden sollen. Bei einer bekannten Funktion könnten die Gewichte direkt berechnet und angegeben werden. Da beim Lernen die Funktion ja unbekannt ist, benötigt man einen Algorithmus zum Bestimmen der Gewichte abhängig von den Lerndaten. Für Feedforward-Netzwerke mit in Schichten angeordneten Neuronen wird im Normallfall ein Backpropagation-Algorithmus verwendet.

Die Grundidee dabei ist, dass der Fehler zwischen der tatsächlichen Ausgabe der zu lernenden Funktion und der Ausgabe des ANNs mit einer bestimmten Gewichtskonfiguration in einen Korrekturwert für die einzelnen Neuronen umgewandelt wird. Der Fehler wird dabei schichtweise ausgehend von der Ausgangschicht immer weiter an die Neuronen der nachfolgenden Schichten propagiert. Während bei der normalen Berechnung des Ausgabewertes des ANNs die Berechnung der Werte „nach vorne“ hin erfolgt, wird bei der Rückwärtspropagierung der inverse Weg gewählt.

Es existieren viele Varianten des Backpropagation-Algorithmus, die diese Grundidee auf verschiedene Art und Weise umsetzen. Eine konkrete mögliche Umsetzung ist in Algorithmus 1 beschrieben. Diese Variante ist aus dem Buch von Tom Mitchell [28, Seite 95 ff.] entnommen und wird dort im Detail erläutert.

Im Algorithmus wird zuerst eine zufällige Startbelegung für die Gewichte $\boldsymbol{w}$ des gesamten ANNs gewählt. Danach werden iterativ die Gewichte des Netzes angepasst, bis der Fehler ein fest definiertes Maß unterschreitet oder die maximale Anzahl der Iterationen erreicht ist. In jedem Iterationsschritt wird ein anderes Beispielpaar $\langle \boldsymbol{x}, \boldsymbol{t} \rangle$ aus den

Eingabe : *trainingsdaten*, η
Daten : N_l: Neuronen der Schicht l
Ausgabe : $w_{i,j}$: Gewicht für das j. Neuron im i. Layer
Ergebnis : optimale Gewichte für bestimmte Trainingsdaten

initialisiere Gewichte mit kleinen zufälligen Werten ;
solange *maximale Anzahl an Iterationen erreicht oder Gesamtfehler kleiner als Schwellwert* **tue**
  für alle $\langle \boldsymbol{x}, \boldsymbol{t} \rangle \in$ *trainingsdaten* **tue**
    `/* 1. Forward-Propagierung */`
    berechne Ausgabe o_u für jedes Neuron u des Neuronales Netzes bei Eingabe $\boldsymbol{x}$;
    `/* 2. Backward-Propagierung */`
    für $l \leftarrow n_{layer}$ **bis** 1 **tue**
      für jedes *Neuron* k *der Schicht* l **tue**
        wenn $l = n_{layer}$ **dann**
          `/* Berechne Fehlerkomponente` δ_k `für Ausgabeschicht */`
          $\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k)$;
        sonst
          `/* Berechne Fehlerkomponente` δ_k `für Hidden Layer oder Eingabeschicht */`
          $\delta_k \leftarrow o_k(1 - o_k) \sum_{i \in N_{l+1}} w_{ik}\delta_i$;
        Ende
      Ende
    Ende
    `/* 3. Aktualisierung der Gewichte */`
    für jedes *Gewicht* w_{ji} **tue**
      $d \leftarrow \Delta w_{ji} = \eta \delta_j x_{ji}$;
      $w_{ji} \leftarrow w_{ji} + d$;
    Ende
  Ende
Ende

Algorithmus 1 : Stochastischer Backpropagation-Algorithmus für ein in Schichten organisiertes Feedforward ANN (nach [28, Seite 98], erweitert auf beliebige Anzahl von versteckten Schichten).

Trainingsdaten verwendet. $\boldsymbol{x}$ bezeichnet den Eingabevektor und $\boldsymbol{t}$ den Ausgabevektor („target“) des Beispiels. Sobald die Trainingsdaten aufgebraucht sind, aber die Abbruchbedingung noch nicht erfüllt ist, werden die gleichen Trainingsdaten wiederverwendet. Nachdem für ein bestimmtes Trainingsbeispiel die Ausgabe $\boldsymbol{o}$ berechnet wurde, muss der Korrekturwert bestimmt werden. Dazu benötigt man ein Fehlermaß E, das den Abstand zwischen zwei Ausgaben von gleich aufgebauten, aber mit unterschiedlichen Gewichten konfigurierten, neuronalen Netzen definiert.

$$E_d(\boldsymbol{w}) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2 \tag{2.13}$$

E_d summiert die Differenz der erwarteten Ausgabe t_k und der berechneten Ausgabe o_k über jedes Neuron der Menge aller Ausgabeneuronen (*output*) eines speziellen Trainingsbeispiels d.

Beim Backpropagation-Algorithmus wird zur Bestimmung des Korrekturwertes ein komponentenweises Gradientenabstiegsverfahren auf dem Fehlermaß E_d eingesetzt. Die Gewichte w_{ji} der Ausgangsneuronen werden jeweils mit dem Term

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} \tag{2.14}$$

$$= \eta \, (t_j - o_j) \, o_j (1 - o_j) \, x_{ji} \tag{2.15}$$

addiert. η bezeichnet die Lernrate, j den Index des Neurons und i ist der Index des Ausgangs. Der Term ist negativ, da der Gradient minimiert werden soll. An dieser Stelle wird auch klar, warum die Differenzierbarkeit der Sigmoidfunktion so wichtig ist und die nicht differenzierbare Schwellwertfunktion (siehe Gleichung 2.10) nicht für Backpropagation geeignet ist. Mit

$$\delta_j = (t_j - o_j) \, o_j (1 - o_j) \tag{2.16}$$

lässt sich die Gleichung für die Neuronen der Ausgangsschicht auch als

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \tag{2.17}$$

verallgemeinern.

Für die Gewichte der versteckten Schichten und der Eingangsschicht ist die Formel komplexer, da der Fehler in der Ausgabe nicht direkt übertragen werden kann, sondern über die jeweils nachfolgende Schicht auf die vorherige Schicht verteilt wird. Sei die gewichtete Summe aller Eingaben des j. Neurons mit

$$net_j = \sum_i w_{ji} \, x_j i \tag{2.18}$$

und der Term

$$\delta_j = -\frac{\partial E_d}{\partial net_j} \tag{2.19}$$

$$= o_j(1 - o_j) \sum_{k \in Nachfolger(j)} \delta_k \, w_{ki} \tag{2.20}$$

sowie

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Nachfolger(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \tag{2.21}$$

gegeben, so ergibt sich für die Hidden- und Eingangschichten wieder der zum Ursprungsgewicht zu addierende Term

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial net_j} \, x_{ji} \tag{2.22}$$

$$= \eta \, \delta_j \, x_{ji} \, . \tag{2.23}$$

Die hier gezeigte Variante des Backpropagation-Algorithmus berechnet nur den Gradienten eines einzelnen Trainingsbeispiels, aber nicht den der gesamten Trainingsmenge. Er wird deswegen als inkrementeller oder stochastischer Backpropagation-Algoritmus bezeichnet. Wenn erst die Fehler aller Trainingsdaten berechnet werden und auf dieser Summe der tatsächliche Gradient berechnet wird, bezeichnet man dies als „Batch-Update“ (im Gegensatz zum „inkrementellen Update"). Der stochastische Ansatz führt im Allgemeinen zu besseren Trainingsergebnissen, da jedes Trainingsbeispiel eine unterschiedliche Fehleroberfläche besitzt und der Wechsel zwischen diesen verschiedenen Oberflächen beim Gradientenabstieg hilft, lokale Minima zu vermeiden [28, Seite 104 ff.].

Zusätzlich zu dem hier vorgestellten Parameter kann man beim Backpropagation-Algoritmus auch noch ein sogenanntes Moment einführen [28, Seite 100]. Dabei wird jedem Gewicht zusätzlich noch ein proportionaler Anteil der Änderung dieses Gewichts in der vorherigen Iteration hinzugefügt und es ergibt sich eine gewisse Trägheit bei den Änderungen. Diese Trägheit kann dabei helfen das Gradientenabstiegsverfahren durch lokale Minima zu führen [28, Seite 104].

2.4 Entscheidungsbäume

Entscheidungsbäume sind ein maschinelles Lernverfahren zur Klassifikation oder Regression. Dazu wird eine Baumstruktur gelernt, die die globale Entscheidung der globalen Zugehörigkeit zu einer Klasse oder den berechneten Wert durch die iterative Abfrage nach einzelnen Attributwerten erlaubt. Neben der Klassifikation bzw. Regression erlauben Entscheidungsbäume auch eine Analyse von Daten, da die erzeugte Baumstruktur im Gegensatz zu einem Blackbox-Verfahren wie z.B. ANN eine verständliche Repräsentation der Einflüsse einzelner Attribute auf die Gesamtklassifikation darstellt („Whitebox“).

2.4.1 Aufbau und Arbeitsweise

Ein Entscheidungsbaum besitzt eine Menge von Knoten, die mit einem Bezeichner versehen sind. Dieser Bezeichner entspricht jeweils genau einem Attribut eines Samples der Lernmenge. Wenn die Elemente der Lernmenge als Vektor organisiert sind, so sind die möglichen Attribute die Indexe des Vektors. Da Entscheidungsbäume allerdings auch zur Datenanalyse eingesetzt werden, wird im Normalfall nicht der Index als Bezeichner verwendet, sondern ein aussagekräftiger Name.

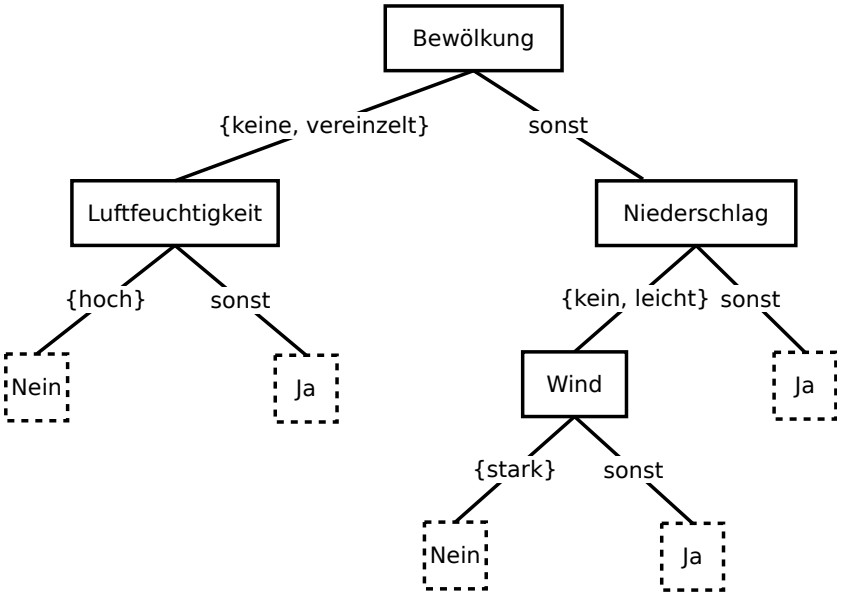
Nicht-Terminalknoten besitzen genau zwei ausgehende Kanten, ein Entscheidungsbaum nach dieser Definition ist daher eine Unterart der Binärbäume. Andere Definitionen von Entscheidungsbäumen erlauben beliebig viele Kanten, in diesem Fall werden aber darauf angepasste Lernalgorithmen benötigt. Diese Kanten beschreiben den sogenannten „Split“, also die Bedingung, nach der entschieden wird, ob der linke oder rechte Pfad beim Herabsteigen des Baumes gewählt wird. Diese Bedingung ist entweder eine Menge von Werten (bei kategorialen Eingangsdaten) oder ein Grenzwert (geordnete Eingangswerte). In Abbildung 2.5 sind zwei beispielhafte Entscheidungsbäume gegeben.

Zum Klassifizieren bzw. zur Berechnung eines Wertes für eine konkrete Instanz beginnt man am Wurzelknoten des Entscheidungsbaumes. Der Bezeichner des Knotens legt das Attribut fest auf, das geprüft werden soll. Die Werte der linken Kante werden mit denen der Instanz für dieses Attribut verglichen. Bei kategorialen Werten muss der Wert der Instanz in der Menge enthalten sein, bei geordneten Werten muss er kleiner als der Split-Wert sein. Aus der Anwendung der Bedingung ergibt sich ein neuer Teilbaum: bei Erfüllung der Bedingung der linke, bei Nichterfüllung der rechte Teilbaum. Dieses Verfahren wird rekursiv auf dem Teilbaum ausgeführt, bis ein Terminalknoten ohne Kanten erreicht ist. Der in diesem Blatt bzw. Terminalknoten stehende Wert wird als Ergebnis des Verfahrens ausgegeben.

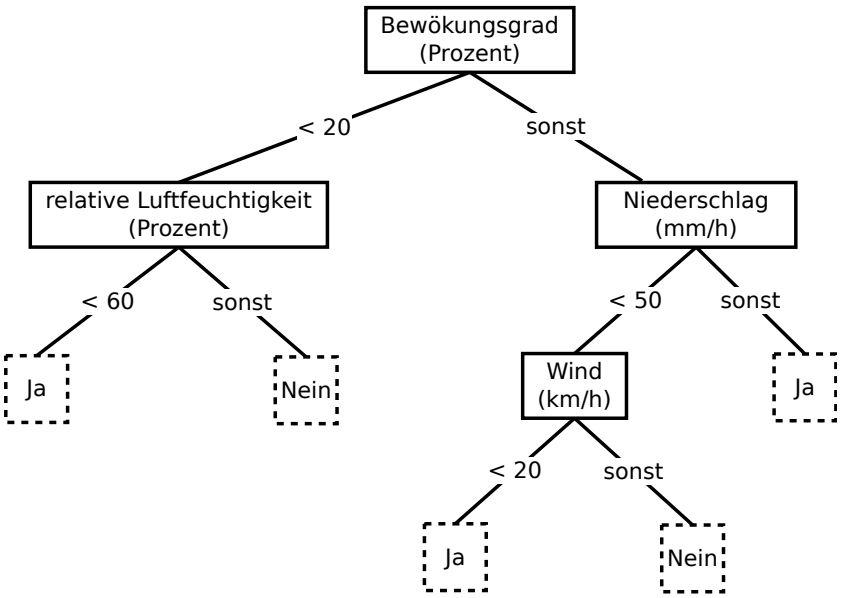
2.4.2 Lernen des Baumes durch CART

In einer der ersten Beschreibungen von Entscheidungsbäumen durch Leo Breiman wurde der CART-Algorithmus („Classification and Regressions Trees“) vorgestellt [8]. Der CART-Algorithmus unterstützt nur Binärbäume, es existiert aber eine weitere Gruppe von Algorithmen zum Lernen von Entscheidungsbäumen, die von J.R. Quinlan entwickelt wurde und Entscheidungsbäume mit beliebiger Anzahl von Kanten unterstützt. Dazu zählt der ID3-Algorithmus [34] und die verbesserten Varianten C4.5 sowie C5.0. Im praktischen Teil der Arbeit wird Implementierung der OpenCV-Softwarebibliothek zum Lernen von Entscheidungsbäumen benutzt. Diese baut wiederum auf dem originalen CART-Algorithmus und nicht auf ID3 auf.

Der Algorithmus beginnt mit einem leeren Baum und der vollständigen Trainingsmenge t. Es soll derjenige Split bestimmt werden, der t am besten in die beiden Teilmengen t_L und t_R trennt. Ein Split besteht aus dem Attribut und der Bedingung an der Kante. Bei kategorialen Attributwerten muss also die Menge der Werte für die linke Kante und bei geordneten Attributwerten der Grenzwert für den Split gefunden werden. Dafür wertet der Algorithmus alle möglichen Attribute und Splits aus und berechnet jeweils einen



(a) kategorielle Attributwerte



(b) geordnete Attributwerte

Abbildung 2.5: Beispiel für einen Entscheidungsbaum mit *kategorialen* (a) bzw. *geordneten* (b) Attributwerten. Die Bäume entscheiden, ob man am Wochenende Tennis spielen sollte oder ob das Wetter dazu nicht geeignet ist. Die verwendeten Attribute sind die Bewölkung, der Niederschlag, die Luftfeuchtigkeit und der Wind. (Angelehnt an Abbildung 3.1 in [28, Seite 53])

sogenannten „Verunreinigungs-Grad“ $I(t)$ (im Original „impurity“). Dieser Wert soll beschreiben, wie ähnlich sich die Elemente einer Teilmenge t sind. Für kategoriale Werte nutzt der CART-Algorithmus den Gini-Koeffizienten. Sei $P(i \mid t)$ die bedingte Wahrscheinlichkeit für das Auftreten der Klasse i in der Trainingsmenge t, dann berechnet sich die „impurity“ für kategorielle Werte $I_{cat}(t)$ wie folgt:

$$I_{cat}(t) = \sum_{i \neq j} P(i \mid t) P(j \mid t) \qquad \text{nach Breimann [8, Seite 38]} \tag{2.24}$$

$$= \sum_{i} P(i \mid t) \left(1 - P(i \mid t)\right) \tag{2.25}$$

$$= \sum_{i} \frac{i}{t} \left(1 - \frac{i}{t}\right) \tag{2.26}$$

$$= \sum_{i} \frac{i}{t} - \left(\frac{i}{t}\right)^2 \tag{2.27}$$

$$= \sum_{i} \frac{i}{t} - \sum_{i} \left(\frac{i}{t}\right)^2 \tag{2.28}$$

$$= 1 - \sum_{i} \left(\frac{i}{t}\right)^2 \qquad \text{Gini-Koeffizient .} \tag{2.29}$$

Sind die Attributwerte hingegen geordnet, wird die Summe der quadrierten Fehler

$$I_{ord}(t) = \sum_{i} \left(y_i - \bar{y}(t)\right)^2 \tag{2.30}$$

genutzt. $\bar{y}(t)$ beschreibt dabei den Mittelwert aller Samples in t (genauer den ihrer Antwortvariable) und y_i ist der einzelne Antwortwert des i. Samples. Der Split, bei dem die Untermengen am einheitlichsten sind, wird gewählt und zwei neue Knoten erstellt. Diese Knoten beinhalten nur die jeweiligen Untermengen t_L bzw. t_R der Samples. Für jeden dieser Knoten wird die Prozedur noch einmal ausgeführt als wäre es der Wurzelknoten. Dies wird solange wiederholt, bis ein Terminierungskriterium wie z.B. die maximale Höhe des Baumes oder eine ausreichende Ähnlichkeit erreicht ist.

Um Overfitting zu vermeiden, werden verschiedene Techniken wie zum Beispiel „Pruning“ eingesetzt. Dabei werden die abhängig von einer Testdatenmenge und eines zu wählenden Qualitätsmerkmals untere Knoten des Baumes wieder entfernt, um eine größere Generalisierung zu erreichen.

3 Kodierung von Verhalten und Situationen

In diesem Kapitel geht es einerseits um die Steuerung von Robotern und andererseits um die Wahrnehmung und Kodierung des gezeigten Verhaltens. Es soll dargestellt werden, wie entschieden wird, welches Verhalten ein Roboter zeigt. Mit diesem Wissen wird untersucht, wie Roboter die Aktionen eines anderen Roboters durch ihre Sensoren wahrnehmen können. Die wahrgenommenen Informationen müssen auf eine geeignete Art und Weise kodiert werden, sodass diese Informationen als Erfahrungswissen den Lernalgorithmen zur Verfügung stehen.

3.1 Steuerung und Verhalten von Robotern

Ein Roboter ist ein komplexes technisches System, dass in der Lage ist seine Umwelt mit Hilfe von Effektoren zu manipulieren [35, Seite 1094]. Dazu benötigt er Sensoren und die Möglichkeit, diese Sensorinformationen zu verarbeiten.

3.1.1 Sense-Think-Act-Zyklus

Ein mögliches Prinzip zur Verarbeitung von Umweltinformationen ist der „Sense-Think-Act“-Zyklus (STA). In Abbildung 3.1 ist dieser Zyklus graphisch abgebildet. Beim STA-Verfahren werden erst alle verfügbaren Sensorinformationen über die Umwelt gesammelt. Ausgehend von diesen Sensorinformationen werden neue Modelle über die Umwelt erstellt oder geändert. Diese Modelle beinhalten auch interne Zustände des Roboters, die zum Beispiel langfristige Planungen oder Motivationen abbilden. Ausgehend von diesen Modellen muss der Roboter nun entscheiden, welche Effektoren er wie einsetzt. Durch die Auswirkungen der Effektoren und durch andere Umwelteinflüsse verändert sich die Umwelt und der Roboter beginnt den Zyklus wieder von vorne. Da sich die Umwelt sehr

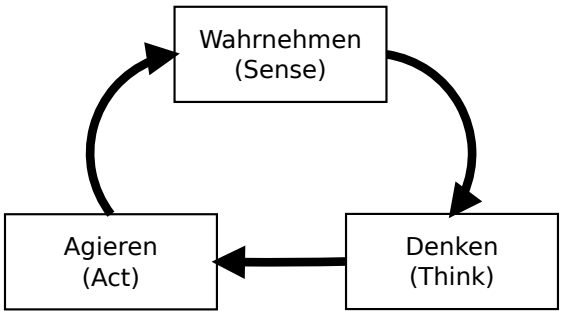


Abbildung 3.1: Sense-Think-Act-Zyklus

schnell ändern kann, sollte der Zyklus mit ausreichender Frequenz wiederholt werden. Solche Frequenzen zur kognitiven Verarbeitung bewegen sich in der Größenordnung von 30 Hertz, also 30 Mal pro Sekunde. Für die Steuerung von Bewegungsabläufen muss der Zyklus unter Umständen deutlich häufiger ausgeführt werden. Diese hohen Frequenzen unterscheiden den STA-Zyklus vom Sense-Plan-Act-Zyklus (SPA), bei dem eine langfristige Planung „stur“ und ohne Berücksichtigung der Umweltänderungen abgearbeitet wird.

3.1.2 Entscheidungsfindung

Für die eigentliche Kontrolle und Entscheidung des Roboters, welche Aktionen er aufgrund seiner Modelle ausführt, gibt es unzählige mögliche Implementierungen. Dazu gehören unter anderem Zustandsautomaten, neuronale Netze, Entscheidungsbäume, regelbasierte Systeme und ad hoc bzw. domänenspezifisch programmierte Algorithmen. Der Roboter nimmt die Rolle eines rationalen Agenten ein, wie er in [35, Seite 58] beschrieben ist. Man kann diese verschiedenen Formen der Algorithmen nach verschiedenen Kriterien unterscheiden und kategorisieren.

Speicherung von Wissen

Das weiter oben beschriebene interne Weltmodell kann rein auf dem aktuellen Zustand der Welt beruhen und gedächtnislos sein. Wenn in dem Modell aber Beobachtungen aus vorherigen Zeitpunkten einfließen und die Entscheidung beeinflussen, existiert eine Art Gedächtnis. Einige Algorithmen sind in der Lage durch Lernen ihre Aktionen auch während des Betriebs nach einem bestimmten Ziel zu optimieren und die Verhaltensweisen anzupassen, andere sind statisch und ändern ihre Verhaltensweisen während eines durchgängigen Einsatzes nicht. Ein Mittelweg sind Entscheidungsfindungsprozesse, die zwischen Trainingsphase und Anwendungsphase unterscheiden und nur in der erstgenannten Phase in der Lage sind Wissen hinzuzulernen.

Determinismus

Deterministische Algorithmen verhalten sich bei gleicher Eingabe (also gleichen Informationen der Sensoren und gleicher Abfolge von Ereignissen) immer exakt gleich. Dagegen können nicht-deterministische Algorithmen andere Aktionen wählen. Für einen externen Beobachter entspricht dieses Verhalten dem eines Zufallsprozesses. Der zufällige Prozess folgt dabei im Normalfall bestimmten Häufigkeiten. Zufällige Prozesse können dem Agenten unter anderem bei Lernverfahren helfen, lokale Optima zu umgehen oder neue und unbekannte Situationen für das Wissensmodell zu erzeugen.

3.1.3 Bewegung

Heutigen Robotern stehen verschiedenste Effektoren zur Verfügung. Dazu gehören Drehgelenke oder prismatische Gelenke, die meist von elektrischen Motoren angesteuerter werden. Drehgelenke erlauben dem Roboter die Rotation von Körperteilen um bestimmte

Achsen, den Freiheitsgraden. Prismatische Gelenke dienen zur Erzeugung einer Gleitbewegung. Die komplette Gelenkkonfiguration, also die tatsächlich eingestellten Winkel an jedem Gelenk, eines Roboters wird als Pose bezeichnet.

Durch sinnvolle Anordnung und Konstruktion der Körperteile um die Gelenke und einer geeigneten Steuerung können Roboter sich selbst fortbewegen. Dabei wurden verschiedenste teils durch biologische Vorbilder inspirierte Ansätze in der Robotikforschung umgesetzt. So gibt es neben Robotern auf Rädern auch kriechende, laufende (auf typischerweise zwei, vier oder mehr Beinen), kletternde, schwimmende und auch fliegende Roboter [27]. Dabei entspricht die Anzahl der Freiheitsgrade der Gelenke nicht zwangsläufig den steuerbaren Freiheitsgraden eines Roboters. Bei holonomen Robotern ist dies jedoch der Fall, nichtholonome Roboter haben dagegen weniger steuerbare Freiheitsgrade als Gelenke [35, Seite 1098]. Beispiele für nichtholonome Antriebe sind mit Rädern umgesetzte Synchronantriebe, wie sie auch bei Autos zu finden sind. Aber auch ein gehender (also nicht springender) Roboter auf zwei Beinen benötigt deutlich mehr Gelenke, als er sich effektiv auf einer planaren Ebene fortbewegen kann. So besitzt beispielsweise der Nao-Roboter 11 Gelenke verteilt auf zwei Beine und die Hüfte [17]. Damit kann bei einem omnidirektionalen Laufen die X- und Y-Translation sowie die Rotation des Roboters geändert werden. Für drei steuerbare Freiheitsgrade benötigt man also 11 effektive Freiheitsgrade.

3.2 Beobachtung von Verhalten durch Sensoren

Sensoren dienen der Kodierung von Umweltinformationen in für Computer verständliche Formate. So ergibt ein Temperatursensor einen einzigen Wert: die Temperatur in einer bestimmten Skala an einem bestimmten Punkt im Raum. Diese als Zahl kodierte Information kann vom (unter Umständen mechanischen) Sensor an das auf dem Prozessor laufende Programm weitergereicht werden. Dabei tritt eine Latenz, also eine Verzögerung, zwischen dem eigentlichen Messen und der Verarbeitung durch das System auf.

Der Nao Roboter besitzt eine Kamera als seinen Hauptsensor. In Kapitel 6 ist beschrieben, wie durch Bildverarbeitungsalgorithmen aus dem Kamerabild die relative Position eines anderen Roboters berechnet werden kann. Bei diesem Algorithmus werden die einzelnen Füße durch Featurepunkte erkannt und beschrieben. Es ist daher möglich, nicht nur die Translation sondern auch die Rotation zu erkennen. Trotzdem gibt diese Sensorinformation nicht den kompletten Zustand des anderen Roboters wieder. So kann die verwendete Bildverarbeitung keine Informationen zu den einzelnen Gelenkstellungen erkennen. Auch gibt es keinerlei Möglichkeiten, die internen Modelle des Gegenübers zu beobachten. Eine Möglichkeit wäre es nun, komplexere und aufwendigere Sensoren zu bauen. So existieren Exoskelette um Gelenkstellungen von Menschen oder humanoiden Robotern direkt detektieren zu können [4]. Will man aber wirklich den vollständigen Zustand des Roboters erfassen, so benötigt man nicht nur die Zustände der Effektoren sondern auch dessen mentales Modell. Dieses wiederum wird aus dem Umweltwissen des Agenten gebildet.

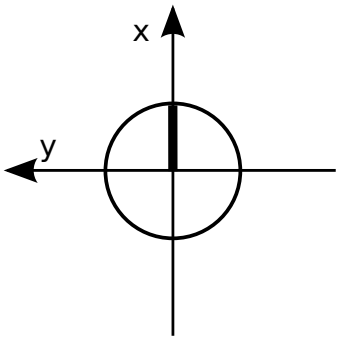


Abbildung 3.2: Relatives Koordinatensystem des Roboters. Eine positive x-Koordinate bedeutet „geradeaus“, eine positive y-Koordinate „links“.

Agenten sind für einen außen stehenden Beobachter (der ja selbst auch ein Agent sein kann) also eine Art Blackbox. Selbst mit dem Wissen nach welchem Algorithmus der zu beobachtende Agent seine Entscheidung trifft bleiben Unsicherheitsfaktoren. So gibt es auch bei einem deterministischen Agenten Rauschen in den Sensordaten, das zufällig auftritt. Wenn ein Agent also das Verhalten des anderen Agenten beobachtet, kann er zwar versuchen die Umweltinformationen des anderen Agenten zu rekonstruieren, seine eigenen Sensoren werden aber ein anderes Rauschen produzieren als die des anderen Agenten. Zudem ist die Sichtweise auf die Umwelt des beobachtenden Agenten im Normalfall eingeschränkt und erschwert eine Rekonstruktion des Umweltwissens des zu beobachteten Agenten. So können zwei Agenten in einem u-förmigen Gang, an dessen Biegungen die beiden Agenten stehen, sich zwar gegenseitig erkennen, aufgrund der Wand sehen beide Agenten aber auch Dinge, die der jeweils andere nicht erkennen kann. Daher können im Normallfall nur die Aktionen eines Agenten und die eigene Sicht auf die Umwelt beobachtet werden, aber das Umweltwissen des anderen Agenten und seine inneren Zustände sind im Allgemeinen nicht rekonstruierbar.

3.3 Statische Modellierung des zu beobachtenden Roboters

Es soll ein sich in einer zweidimensionalen Welt bewegender Roboter beobachtet und modelliert werden. Die Reduktion auf 2 Dimensionen vereinfacht viele Berechnungen und ist zudem kaum verlustbehaftet, da der verwendete Nao Roboter zu schwer und unbeweglich zum Springen ist und daher immer mindestens einen Fuß Kontakt mit dem Boden hat. Diese Annahme setzt allerdings eine planare Welt voraus, bei der es keine größeren Erhebungen gibt. Im Roboterfußball, in Büroräumen, Hallen oder auf Wiesen ist diese planare Welt meistens gegeben. Das Koordinatensystem des Roboters ist in Abbildung 3.2 gezeigt. Bei einer Veränderung der Position des Roboters kann sich einerseits die x- und y-Koordinate ändern, anderseits aber auch die Ausrichtung des Roboters (Rotation).

Positionen von Objekten im Raum sind für jeden Roboter immer relativ zu sich selbst. In einigen Szenarien (zum Beispiel dem Spielfeld im Roboterfußball mit dem Mittelkreis)

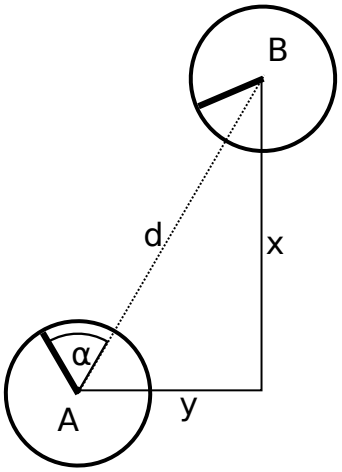


Abbildung 3.3: Relative Position eines beobachteten Roboters B durch den Roboter A in kartesisches Koordinaten (x, y) und Polarkoordinaten (d, α)

existieren zwar mögliche globale Koordinatensysteme, die für alle Roboter gleich sind, diese sollen aber hier für die Modellierung von Roboterpositionen nicht benutzt werden. Ein Grund dafür ist, dass eine Position in solch einem globalen Koordinatensystem erst durch womöglich aufwendige und fehlerbehaftete Selbstlokalisierungsalgorithmen bestimmt werden muss.[1] Nicht in jedem Szenario, dass abgedeckt werden soll, ist ein immer gut erkennbarer und eindeutiger globaler Koordinatenursprung bestimmbar.

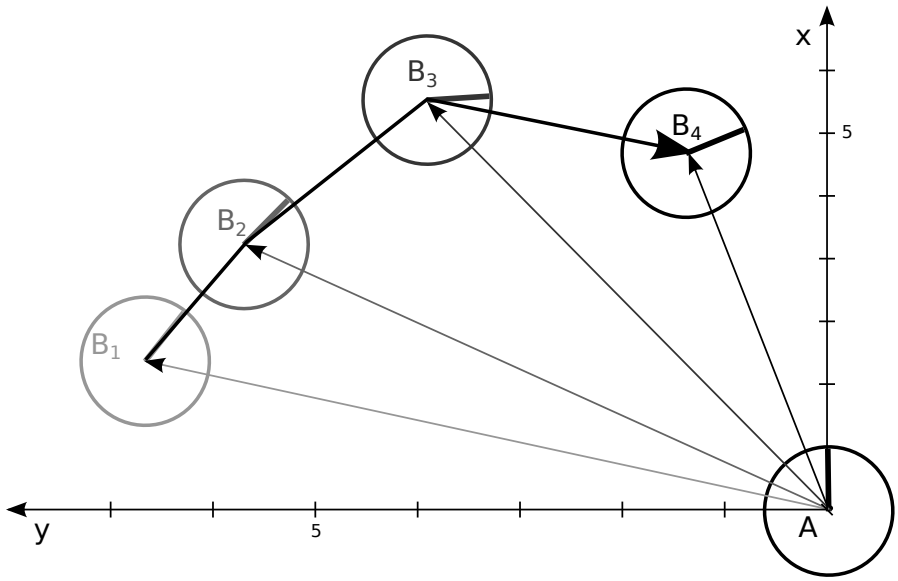
Ein beobachtender Roboter A kann einen beobachteten Roboter B wie in Abbildung 3.3 gezeigt entweder über die x- und y-Koordinaten oder die äquivalenten Polarkoordinaten (Winkel und Distanz) referenzieren.

3.4 Beschreibung von Laufwegen durch Trajektorien

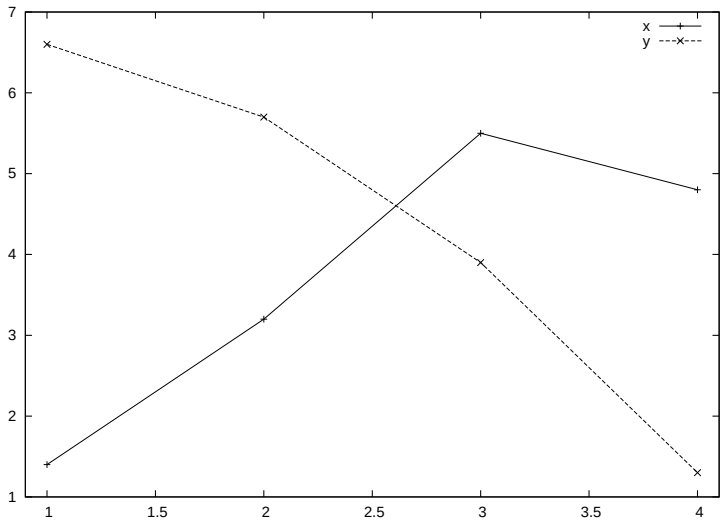
Die Modellierung der Position des Roboters in Abschnitt 3.3 ist statisch, es beschreibt also mehr die aktuelle Konfiguration und weniger das Verhalten. Abbildung 3.4 zeigt den zeitlichen Verlauf einer Trajektorie. Der Vektor $\boldsymbol{AB_t} = \boldsymbol{B_t} = (x_t, y_t)^T \in F \subseteq \mathbb{R}^2$ beschreibt die relative Position B_t ausgehend von der Beobachtungsposition $\boldsymbol{A}$. $\boldsymbol{A} = (0, 0)^T$ ist per Definition der Koordinatenursprung.

Da Trajektorien die Position abhängig von der Zeit beschreiben, lassen sie sich als diskrete Zeitreihen beschreiben. Diskrete Zeitreihen sind Mengen von Beobachtungen der Form $\{o_t\}$ zu diskreten (spezifischen) Zeitpunkten $t \in \mathbb{N}$. Nehmen wir einen Sense-Think-Act-Zyklus beim Beobachter A an, so wird t durch die Frequenz des Zyklus bestimmt. Im Falle der Position benötigen wir zwei Zeitreihen $\{x_t\}$ und $\{y_t\}$, da es unabhängig von der Darstellung als kartesische Koordinate oder Polarkoordinate zwei Komponenten

[1] Die Lokalisierung im Roboterfußball erfolgt normalerweise durch die Tore und Linien. In der Standard-Plattform-Liga (SPL) werden verschiedenfarbige Tore verwendet, um eine eindeutige Referenz zu erlauben



(a) 2D-Darstellung



(b) Zeitreihe von x und y

Abbildung 3.4: Trajektorie (B_1, B_2, B_3, B_4) mit den beobachteten Vektoren $\boldsymbol{AB_1} \ldots \boldsymbol{AB_4}$ als 2D-Darstellung (a) und als Zeitreihe der einzelnen Komponenten (b)

gibt. Die Rotation wird nicht weiter betrachtet. Das Model einer Zeitreihe wird in [9, Seite 7] folgendermaßen beschrieben

> „A time series model for the observed data $\{o_t\}$ is a specification of the joint distributions (or *possible only the means and covariances*) of a sequence of random variables $\{O_t\}$ of which $\{o_t\}$ is postulated to be a realization“

Zeitreihenmodelle sind also multivariate Wahrscheinlichkeitsverteilungen der Form

$$P[O_1 \leq o_1, \ldots, O_n \leq o_n], \forall\, 1 \leq i \leq n : \; o_i \in \mathbb{R}\,,\; n \in [1, \infty[\,. \tag{3.1}$$

Dieses Modell betrachtet die Zeitreihen zusammenhängend, die Wahrscheinlichkeit für einen Wert zu einem Zeitpunkt t ist also immer abhängig von allen vorherigen Werten in der Zeitreihe. Die Länge der Zeitreihe ist durch n gegeben, welches die aktuelle Anzahl der Messungen beschreibt aber potentiell unbegrenzt ist.

Wichtig bei der Definition ist der Einschub „possible only the means and covariances“. Das Lernen von multivariaten Wahrscheinlichkeitsverteilungen aus Beispieldaten durch Verfahren, wie sie in Kapitel 2 beschrieben sind, ist kaum möglich, da zu viele Parameter gelernt werden müssten [9, Seite 7]. Daher werden nur die Erwartungswerte $E(O_t)$ der einzelnen Zufallsvariablen O_t zu den Zeitpunkten t und die Erwartungswerte der Produkte $E(O_{t+h}O_t)$ (mit $h \in [0 : n]$) weiter betrachtet. Mithilfe des Produktes der Erwartungswerte kann die Kovarianz

$$Cov(O_{t+h}, O_t) = E(O_{t+h}O_t) - E(O_{t+h})E(O_t) \tag{3.2}$$

zweier Variablen der Zeitreihe gebildet werden. Diese Momente erster und zweiter Ordnung bilden zusammen die Eigenschaften zweiter Ordnung.

Wenn die beiden Komponenten der Position als getrennte Zeitreihen betrachtet werden, ergibt sich das Problem der Korrelation zwischen den zwei zeitgleichen Zufallsvariablen X_t und Y_t. Intuitiv sind die beiden Zufallsvariablen nicht unabhängig zueinander, da der Roboter als physisches System nur bestimmte maximale Geschwindigkeiten erreichen kann. Seien mit $(x_t, y_t), (x_{t+1}, y_{t+1}) \in F$ zwei aufeinander folgende Beobachtungen zum Zeitpunkt t und $t+1$ gegeben. Die maximale Geschwindigkeit ist durch ein festes $c \in \mathbb{R}_+$ vorgegeben. Dann gilt

$$\|(x_{t+1}, y_{t+1})^T - (x_t, y_t)^T\|_2 \leq c\,. \tag{3.3}$$

Um dennoch eine Vorhersage zu ermöglichen, werden die beiden Vektorkomponenten im Folgenden als unabhängig angenommen.

3.5 Partitionierung von Trajektorien

Beobachtet ein Agent A durchgängig die Positionen einer Menge von anderen Agenten $\{B^i\}$ über einen Zeitraum von n Zyklen, so kann er die Trajektorien

$$T^i = \left\{ \left(x_t^i, y_t^i \right) \right\},\; t \in [1, n] \tag{3.4}$$

als sein Wissen $W = \{T^i\}$ sammeln. Während dieses Zeitraumes können die anderen Agenten ihre internen Zustände und Absichten ändern. Dies führt zu einer Verhaltensänderung, die Auswirkungen auf die Trajektorie hat. Über die Trajektorien wird also auf das Verhalten geschlossen und sie werden als Abbild des Verhaltens betrachtet. Wenn auf diesem Abbild später gelernt werden soll und dieses Lernen eine Klassifizierung der Trajektorien voraussetzt, müssen die Trajektorien partioniert werden, sobald eine Verhaltensänderung des Agenten wahrgenommen bzw. angenommen wird.

In [22] wird vorgeschlagen, die Trajektorien an den Punkten aufzutrennen, an denen sich die Bewegungsrichtung besonders stark ändert. Um diese Punkte zu finden, wird ein Minimum-Description-Length-Verfahren (MDL) vorgeschlagen. MDL basiert auf der Idee einen Code für die Daten zu finden, der möglichst kurz ist. Durch einen Approximation-Algorithmus mit einer linearen Komplexität hinsichtlich der Länge der Trajektorie wird die Trajektorie so partioniert, dass die Summe aus der Länge der Partition sowie der Summe der Differenz zwischen Partition und Trajektorie minimiert wird. Dieser Ansatz wurde auf Bewegungsdaten von Schiffen und Tierherden angewendet.

Eine dem entgegengesetzte Heuristik zum Partionieren ist das Auftrennen an Punkten, an denen sich der Agent für einen bestimmten Zeitraum nicht oder nur kaum bewegt. Die Annahme ist hier, dass der Agent zur Erfüllung eines Ziels eine bestimmte Position bzw. Bewegung erreichen muss. Hat er das Ziel erreicht oder führt der Agent keine Bewegung mehr aus, so hat er das Ziel des Verhaltens erreicht. Nachteil ist hier, dass nur aktive Bewegung als Verhalten erkannt wird und das Verhalten „stehen“ nicht vorhergesagt werden kann.

Nimmt man hingegen an, dass der Agent sein Verhalten immer dann ändert, wenn sich seine Umweltinformationen drastisch ändern, so muss man wieder eine andere Heuristik entwickeln. Für diese Heuristik benötigt man allerdings selbst Informationen über die Umwelt des Agenten, die zum Beispiel bei einer reinen Betrachtung der Zeitreihe nicht gegeben ist.

3.6 Features über Trajektorien

Zur Charakterisierung von Trajektorien benötigt man Merkmale („Features“), die diese Funktion numerisch beschreiben. Diese Merkmale müssen nicht eindeutig, sollten aber deterministisch sein. Zudem sollen Features Ähnlichkeiten ausdrücken, ähnliche Trajektorien führen also zu ähnlichen Werten des Merkmals. Dabei sollten Features invariant zu Translokationen und Stauchungen der Trajektorien sein, da diese im Allgemeinen die Ähnlichkeit oder den „Charakter“ der Trajektorie nicht beeinflussen. Um mehrere Merkmale zusammenzufassen, werden diese in Featurevektoren oder -matrizen gebündelt. Während eine Trajektorie eine beliebige Anzahl von Punkten haben kann, so ist die Anzahl der Merkmale die auf die Trajektorie angewendet wird und damit auch die Dimension der Featurematrix konstant. Die konstante Dimension der Featurematrix ist für die weitere Verarbeitung und für das Lernen von Eigenschaften essentiell. Ähnlich zu den Merkmalen aus der Bildverarbeitung (siehe Kapitel 4.3) können die Features von Trajektorien in lokale und globale Features unterschieden werden.

3.6.1 Existierende Forschung

Es existiert viel Forschung zu Features von Trajektorien, die sich mit der Klassifizierung der Trajektorien beschäftigt. Ein häufig genanntes Anwendungsgebiet ist hierbei die Videoüberwachung. Die Klassifizierung soll dabei helfen, typische von untypischen Laufwegen in einer Umgebung zu unterscheiden und „unerwünschtes“ Verhalten zu detektieren (zum Beispiel der Aufenthalt in einem gesperrten Gebiet). In [22] werden die durch Radar gewonnenen Trajektorien von Schiffen ausgewertet um sie in Kategorien wie „Tanker“, „Containerschiff“ oder „Fischerboot“ einzuordnen. Dazu werden zwei Arten von Features generiert: Die einen beziehen sich auf die globale Region in denen sich das Schiff aufhält und sind nicht abhängig von der Bewegung, die anderen entstehen durch das Auftrennen der Trajektorie in Mengen von linearen Partitionen. Ein anderer Ansatz wird in [19] gewählt, mit dessen Hilfe Anomalien in den Trajektorien entdeckt werden sollen. Dazu werden normale Trajektorien als Modell gesammelt. Soll eine neue Trajektorie klassifiziert werden, so wird zu erst überprüft, ob sie von ihren absoluten räumlichen Koordinaten innerhalb einer gewissen maximalen Distanz vom Modell bleibt. Ist diese Überprüfung positiv, wird die Geschwindigkeit auf der Gesamttrajektorie berechnet und auch diese mit dem Modell verglichen. Als letztes Feature wird die Krümmung berechnet und mit den vorhandenen Trajektorien verglichen.

Will man hingegen die Trajektorien nicht klassifizieren sondern die zukünftige Entwicklung vorhersagen, benötigt man eine andere Art von Features. In [23] werden die Trajektorien als Pfad in einem Graphen beschrieben. Die Kanten des Graphen beschreiben Straßen und die Knoten sind die Wegkreuzungen, an denen sich die Richtung der Trajektorie ändern kann. An jedem Entscheidungspunkt kann die Trajektorie durch folgende Features beschrieben werden:

- Distanz zwischen dem Startpunkt und dem Entscheidungspunkt,
- Bewegungsrichtung als Differenzwinkel vom Vektor zwischen dem Start- und Entscheidungspunkt sowie dem Vektor zwischen dem vorletzten Punkt der Trajektorie und dem Entscheidungspunkt
- sowie die Tageszeit [23].

Mithilfe dieser Features wird die Entscheidung gelernt, wie Autofahrer an einer Kreuzung wahrscheinlich abbiegen werden. Dadurch erklärt sich auch die Verwendung der Tageszeit, denn die Entscheidung kann dadurch in diesem konkreten Fall natürlich beeinflusst werden. Auch in [38] werden die elementaren Schritte der Bewegung durch Position, Ausrichtung, Richtung und Distanz kodiert. Zusätzlich kommt hier allerdings auch noch die Dauer der Elementarbewegung hinzu und es wird zwischen quantitativen Einheiten wie Meter und Sekunde und qualitativen Beschreibungen („lang“, „kurz“) unterschieden. Das Ziel hierbei war es, Touristen in einem Skigebiet abhängig von ihren Bewegungsmustern und vermuteten Intentionen Informationen über ihre Umgebung anzubieten.

Diese Beispiele zeigen, dass obwohl die grundsätzliche Kodierung von Trajektorien sich in den verschiedenen Anwendungen stark ähnelt, die verwendeten Features auf der

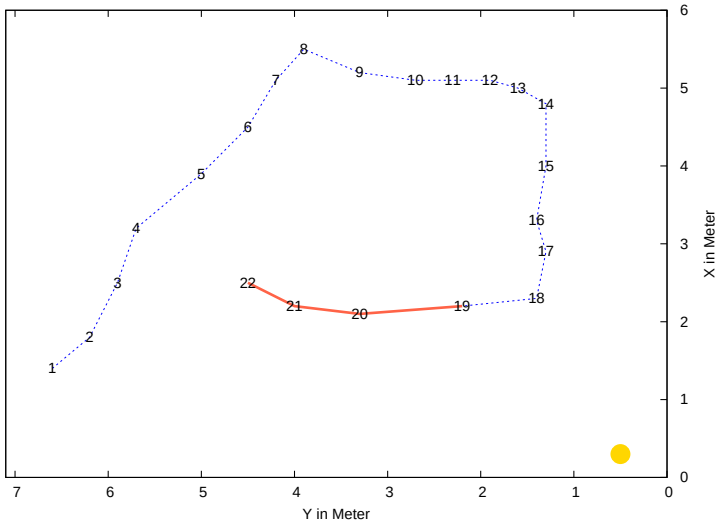


Abbildung 3.5: Beispiel-Trajektorie mit lokalen Features für $t = 22$ und $m = 3$ (rot markiert) sowie der der Unterteilung für die globalen Features (gelb markiert)

Trajektorie davon abhängig sind, aus welcher Domäne die Daten stammen und was das Ziel der Vorhersage oder Klassifizierung ist.

3.6.2 Lokale Features

Lokale Features von Trajektorien enthalten Informationen über den Verlauf der Trajektorie um einen bestimmten Zeitpunkt t herum. Ein mögliches lokales Feature ist die stückweise Differenzierung zu einem Zeitpunkt t. Dazu definieren wir die Funktion $featloc$, die aus einer Trajektorie T, dem Zeitpunkt t und der erwünschten Anzahl an Zeilen die Featurematrix berechnet.

$$featloc : (T, t, m) \rightarrow \begin{pmatrix} dx_1 & dy_1 \\ \dots & \dots \\ dx_m & dy_m \end{pmatrix}, \tag{3.5}$$
$$dx_i = x_{t-i+1} - x_{t-i},$$
$$dy_i = y_{t-i+1} - y_{t-i}$$

Diese Features sind unabhängig von den Anfangswerten und daher invariant gegenüber einer Translokation der Trajektorie. Angewandt auf einen beliebigen anderen Startpunkt, wird die so erstellte Trajektorie trotzdem die gleichen lokalen Features besitzen. In Abbildung 3.5 ist eine beispielhafte Trajektorie mit ihrem Bereich für die lokalen Features

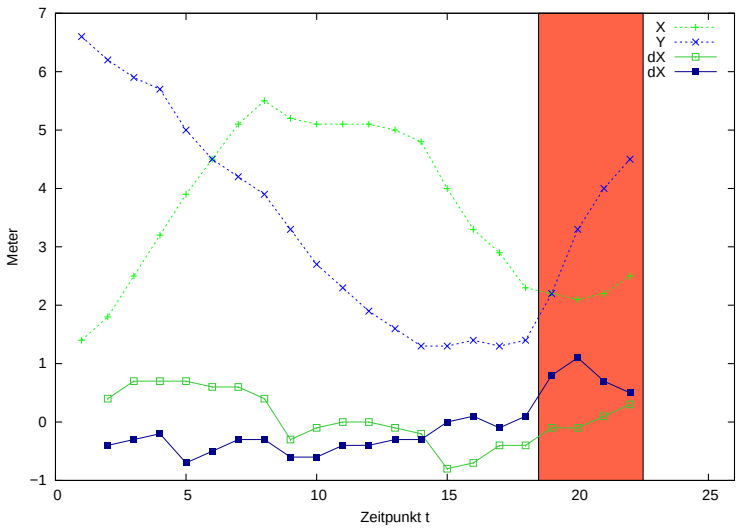


Abbildung 3.6: Differenzen (dX und dY) abhängig von t für die Trajektorie aus Abbildung 3.5.

zu sehen. Dazu passend ist in Abbildung 3.6 der Verlauf der Trajektorie für die einzelnen Komponenten und die Differenzen abgebildet.

3.6.3 Globale Features

Bei der Klassifikation von Trajektorien benötigt man nicht nur lokale sondern globale Features, welche die Charakteristik der gesamten Trajektorie widerspiegeln. Ein Problem ist hierbei, dass die Trajektorien unterschiedliche Längen haben können, eine Stauchung oder Streckung der Trajektorie aber keine wirkliche Änderung der Charakteristik hervorruft. Zudem muss die Dimension einer Featurematrix für eine Trajektorie konstant über alle möglichen Eingabetrajektorien unabhängig von deren Länge sein.

In [20] wird das PAA-Verfahren (Piecewise Aggregate Approximation) als Lösung für dieses Problem vorgeschlagen. Dabei wird die Zeitreihe in eine konstante Anzahl n von in etwa gleich großen Segmenten geteilt. Über diese Segmente wird jeweils der Mittelwert gebildet. Die PAA ist nun der Vektor der Mittelwerte. Im Algorithmus 2 wird das Verfahren und insbesondere die gleichmäßige Segmentierung konkret beschrieben.

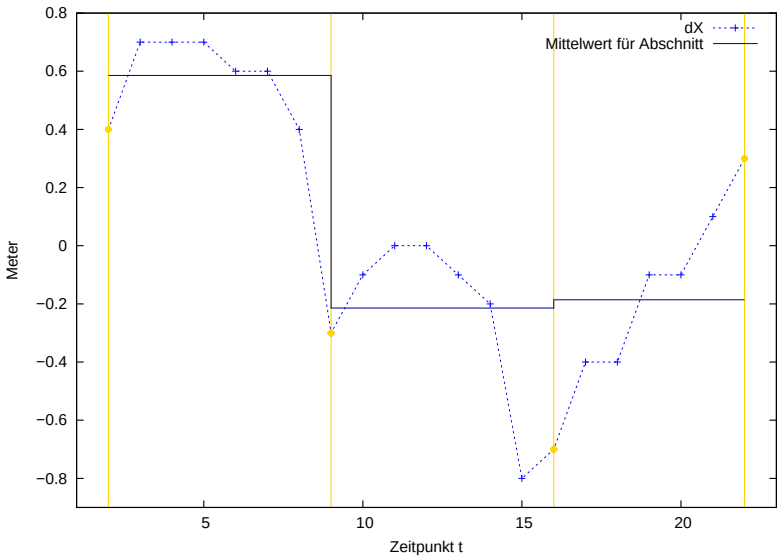
Um von einer Trajektorie nun die globalen Features zu berechnen, werden erst die lokalen Features (also die Differenzen) auf der gesamten Trajektorie mit

$$featloc(T, laenge(T), laenge(T) - 1) \tag{3.6}$$

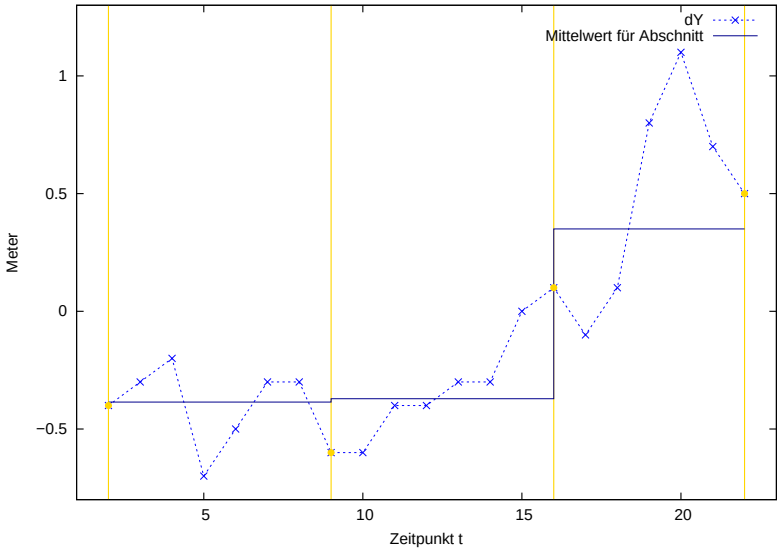
berechnet. Auf diese neuen Zeitreihen wird nun für jede Komponente eine PAA mit Hilfe des FEATGLOB-Algorithmus durchgeführt. In Abbildung 3.7 sind die globalen

Eingabe : $\boldsymbol{d}$ (Datenpunkte), n (Anzahl der Features)
Daten : s, $standardSchritt$, $\boldsymbol{step}$, p
Ausgabe : $\boldsymbol{f}$
`/* Initialisierung */`
$s = laenge(\boldsymbol{d})$;
$standardSchritt = floor(s/n)$;
`/* Schrittweite für alle gleich */`
für $i \leftarrow 1$ **bis** n **tue**
  $step_i = standardSchritt$;
Ende
`/* Verteilung des Restes auf die ersten Schritte */`
für $i \leftarrow 1$ **bis** $s \mod n$ **tue**
  $step_i = step_i + 1$;
Ende
`/* Berechnung der Mittelwerte auf jedem Abschnitt mithilfe der Schrittweite */`
$p \leftarrow 1$;
für $i \leftarrow 1$ **bis** n **tue**
  $f_i = mittelwert\,(subvektor(\boldsymbol{d}, p, step_i))$;
  $p = p + step_i$;
Ende

Algorithmus 2 : FEATGLOB: Berechnung der globalen Features einer Trajektorie mithilfe des stückweisen Mittelwertes



(a) Globale Features für die x-Komponente



(b) Globale Features für y-Komponente

Abbildung 3.7: Globale Features für die Trajektorie aus Abbildung 3.5. Die Abschnitte für die Berechnung der Mittelwerte sind durch die gelben Markierungen begrenzt.

Features für die Beispieltrajektorie (Abbildung 3.5) angegeben.

3.7 Episodisches Gedächtnis

Das Konzept des episodischen Gedächtnisses entstammt der Psychologie. Der Name „Episodisches Gedächtnis“ soll diesen Teilbereich der menschlichen Gedächtnisleistung vom semantischen Gedächtnis abgrenzen. Diese Unterscheidung wurde 1972 unter diesem Namen durch Endel Tulving postuliert, basierend auf ähnlichen Ideen vor ihm unter verschiedenen Namen und einer teilweisen nicht so strikten Trennung zum semantischen Gedächtnis [39, Seite 9]. Tulving hat diese Idee immer weiter ausgebaut und die Definition verbessert. Folgende Definition wurde 2005 von ihm aufgestellt:

> „Episodisches Gedächtnis ist ein [evolutionär] spät entstandenes, sich [ontogenetisch] spät entwickelndes und früh abbauendes neurokognitives Gedächtnissystem. Es ist vergangenheitsorientiert [...] und wahrscheinlich nur dem Menschen eigen. Es erlaubt mentales Zeitreisen durch die subjektive Zeit – Vergangenheit, Gegenwart, Zukunft. Diese mentalen Zeitreisen erlaubt dem ‚Besitzer‘ von episodischem Gedächtnis [...] seine eigenen vorangegangenen ‚gedachten‘ Erfahrungen zu erinnern, wie auch über mögliche zukünftige Erfahrungen zu denken. [...] Die Essenz des episodischen Gedächtnisses liegt in der Verbindung (‚Konjunktion‘) dreier Konzepte – des Selbst, des autonoetischen Bewußtseins und der subjektiven Zeit“ (zitiert nach [25, Seite 232], Original in [40]).

Danach benötigt man ein Konzept von sich selbst und zudem das Bewusstsein, dass es sich um persönlich erlebte Erfahrungen handelt (autonoetisch). Diese Erinnerungen sind relativ zu einem persönlichen Zeitbezug, also nicht zwingend globalen Zeitentitäten wie Jahreszahlen unterworfen.

Tulving selbst hat 1983 in seinem Buch „Elements of Episodic Memory“ künstlichen Intelligenzen abgesprochen in der Lage zu sein, mit einem episodischen Gedächtnis ausgestattet werden zu können [39, Seite 53ff.]. So könnten sie externes Wissen nicht selbst erlebt haben und nur über Wörter, nicht aber Erfahrungen reden. Ein Mensch würde ihnen niemals das von ihnen kommunizierte Erfahrungswissen glauben oder verifizieren können. Diese Ansicht ist offenbar auf KI-Systeme ohne eigenen Körper bezogen. Zudem sind Erfahrungen relativ von der Umwelt die man erlebt. Während eine Simulation natürlich komplett in ihrer eigenen Umgebung „lebt“, so sind Roboter zumindest physisch in der gleichen Welt wie unserer verankert. Allerdings unterscheiden sich ihre Sensoren von den unseren, die erlebte und wahrgenommene Welt ist also nicht zwingend vergleichbar.

Lernende Maschinen, die aus Traingsdaten heraus generalisierende Modelle erstellen, entsprechen eher der Vorstellung von semantischem Wissen als das reine Sammeln von subjektiven Erfahrungen in einer Implementierung eines episodischen Gedächtnisses. Aber auch aus diesem reinen Erfahrungswissen kann neues Wissen hervorgehen. So ist in der Definition von Tulving explizit die Zukunft als mögliches Ziel der mentalen „Zeitreisen“ genannt. Da das episodische Gedächtnis nicht statisch ist, sondern ständigen

Veränderungen unterliegt können auch bestehende Situationen noch einmal durchlebt und die Auswirkung anderer Entscheidungen auf diese Situationen durchdacht werden.

Es gibt zudem Erlebnisse, die immer wieder in nur leicht veränderter Form wiederkehren. Hier könnte man die Annahme machen, dass ähnliche Situationen auch zu ähnlichen Ergebnissen führen. Wenn es gelingt, ähnliche Ereignisse aus dem episodischen Gedächtnis zu extrahieren und sich auch an die damaligen Konsequenzen zu erinnern und diese auf die aktuelle Situation anzupassen, so wäre eine Vorhersage möglich.

3.8 Situationskonzept

Um Situationen abspeichern und wiederfinden zu können, benötigt man ein abstraktes Konzept dieser Situationen. Dieses Konzept muss es erlauben, ein Ähnlichkeitsmaß zu definieren. Trotzdem muss es möglichst dynamisch und allgemein gefasst sein, um auch unvorhersehbare Ereignisse und Situationen zu erfassen.

Im Folgenden soll solch ein System zur Beschreibung von Situationen definiert werden.

- Situationen sind zusammenhängende Einzelereignisse.
- Jede Situation besitzt eine bestimmte Anzahl von Objekten.
- Ein Objekt gehört genau einer Klasse an und besitzt eine zweidimensionale Position.
- Jede Klasse besitzt einerseits einen numerischen Index, andererseits einen eindeutigen Text als Bezeichner.
- Eine Menge an Situationen wird Wissensbasis genannt.
- Für jede Wissensbasis gibt es eine maximale Anzahl von Klassen, die in allen Situationen vorkommen können.
- Kommt trotzdem eine neue Klasse zu der Wissensbasis hinzu, so wird ihr der nächst-höhere Index vergeben und alle bisherigen Situationen müssen aktualisiert werden.

Sei jede Situation ein Paar aus einem Schlüssel $\boldsymbol{key}$ und einer Matrix E.

$$S = \{\boldsymbol{key}, E\} \tag{3.7}$$

$$\boldsymbol{key} = (c_1, c_2, \ldots, c_n) \tag{3.8}$$

key ist ein Vektor, bei dem jedes Element c_i die Anzahl der Objekte einer Klasse i in der Situation angibt. Jede Zeile der Matrix E beschreibt ein einzelnes Ereignis der Situation S zum Zeitpunkt t.

$$E = \begin{pmatrix} \boldsymbol{e}_{11} & \dots & \dots & \dots & \dots & \dots & \dots & \boldsymbol{e}_{1\sum_{i=1}^{n-1} c_i + c_n} \\ \vdots & \ddots & & & & & & \vdots \\ \boldsymbol{e}_{t1} & \dots & \boldsymbol{e}_{tc_1} & \dots & \boldsymbol{e}_{tc_1+1} & \dots & \boldsymbol{e}_{tc_1+c_2} & \dots \boldsymbol{e}_{t\sum_{i=1}^{n-1} c_i + c_n} \\ \vdots & & & & & & & \vdots \\ \boldsymbol{e}_{t_{max}1} & \dots & \dots & \dots & \dots & \dots & \dots & \boldsymbol{e}_{t_{max}\sum_{i=1}^{n-1} c_i + c_n} \end{pmatrix} \tag{3.9}$$

Da die Gesamtzahl der Objekte über alle Klassen der Summe der Anzahl der Objekte pro Klasse $\sum_{i=1}^{n} c_i$ entspricht, muss E dementsprechend viele Spalten c_E haben. Eine einzelne Zelle $\boldsymbol{e}_{tj}$ in E entspricht der Position des Objektes:

$$\boldsymbol{e}_{tj} = (x_{tj}, y_{tj}) \, . \tag{3.10}$$

4 Merkmalsextraktion aus Bildern

Dieses Kapitel dient dazu Verfahren zu erläutern, mit denen Computer bzw. Roboter ihre Umwelt visuell wahrnehmen. Es dient als Grundlage zum Verständnis dafür, wie Roboter andere Roboter erkennen können.

4.1 Farbmodelle und Kodierung

Optische Sensoren wie Kameras benötigen eine Kodierung, um die Intensität und die Zusammensetzung der Wellenlänge des auf sie auftreffenden Lichtes zu beschreiben. Dabei kann das Licht in die einzelnen Intensitäten von einer bestimmten Wellenlänge zerlegt werden. Menschen können nicht beliebige Farbspektren wahrnehmen und die Kombination aus drei Farbkomponenten („Chroma“) reicht aus um alle wahrnehmbaren Farben zu beschreiben [35, Seite 1053]. Danach werden in der Regel die Farbkomponenten Rot mit einer Wellenlänge von 700nm, Grün mit 546nm und Blau mit 436nm verwendet [35, Seite 1053]. Roboter nutzen Videokameras zur visuellen Wahrnehmung, die ebenfalls nicht alle Farben erfassen können. Die Farbkodierung von Kameras und der abgedeckte Farbbereich können sich vom menschlichen Sehen unterscheiden.

4.1.1 Y'CbCr-Farbmodell

Verwendet man eine digitale Videokamera, so wird im Normalfall Y'CbCr als natives Farbmodell eingesetzt. Y' bezeichnet das Helligkeitssignal der Videokamera. Der Strich gibt an, dass es sich nicht um die Helligkeit oder Luminanz Y nach der Definition der CIE (Commission internationale de l'éclairage) handelt, sondern um eine nicht-linearisierte Ableitung davon [32, Seite 88]. Die Definition von Luminanz versucht der menschlichen Farbwahrnehmung zu entsprechen, während Y' technischen Zwängen unterworfen ist. Y'CbCr kann aus den roten, grünen und blauen Farbkanälen R'G'B' berechnet werden. Während Y' als Kombination der Helligkeit über alle Kanäle berechnet wird, werden die beiden Chroma-Komponenten als Differenz zwischen B' und Y' bzw. R' und Y' aufgefasst. Die Konstanten, mit denen die Komponenten multipliziert werden, sind abhängig vom verwendeten Wertebereich und dem gewählten Farbstandard. Für die Umwandlung aus dem in Computern benutzten RGB-Standard und bei einem Wertebereich von 0 bis 255 (8-Bit-Ganzzahlen) kann folgende Formel benutzt werden (entnommen aus [32, Seite 319]:

$$\begin{pmatrix} Y' \\ Cb \\ Cr \end{pmatrix} = \begin{pmatrix} 16 \\ 128 \\ 128 \end{pmatrix} + \frac{1}{256} \begin{pmatrix} 65,738 & 129,057 & 25,064 \\ -37,945 & -74,494 & 112,439 \\ 112,439 & -94,154 & -18,285 \end{pmatrix} \begin{pmatrix} R' \\ G' \\ B' \end{pmatrix} . \tag{4.1}$$

(a) Original

(b) Y’

(c) Cb

(d) Cr

Abbildung 4.1: Ein Bild zerlegt in seine einzelnen Farbkanäle Y’, Cb und Cr

In Abbildung 4.1 ist ein Bild zu sehen, das in seine einzelnen Luma- bzw. Chroma-Kanäle zerlegt wurde.

Y’CbCr versucht, die volle Helligkeitsinformation zu erhalten, während die Farbinformation reduziert wird. Dabei wird ausgenutzt, dass das menschliche Sehen Farben im Vergleich zu Helligkeit unschärfer wahrnimmt [32, Seite 89] und eine solche Reduktion nicht so stark wahrgenommen wird. Für Bildverarbeitungsalgorithmen, die nur auf der Helligkeitsinformation basieren, ist also von Vorteil die Helligkeit möglichst genau und differenziert wahrzunehmen.

4.1.2 Komprimierung von Farbinformationen mit Y’CbCr-4:2:2

Eine weitere Möglichkeit, Helligkeits- und Farbinformationen effizient gemeinsam abzuspeichern, ist das Verschränken von Informationen zwischen zwei Pixeln durch die Y’CbCr-4:2:2-Kodierung. Bei der RGB- oder Y’CbCr-Kodierung und einem ganzzahligen Wertebereich von 0 bis 255 pro Komponente benötigt man $3 \cdot 8 = 24$ Bit um ein Pixel zu repräsentieren. Bei der Y’CbCr-4:2:2-Kodierung teilen sich zwei horizontal benachbarte Pixel die Cb- und Cr-Komponente und unterscheiden sich nur in Y’. Jedes Pixel benötigt damit nur noch $2 \cdot 8 = 16$ Bit an Information. In Abbildung 4.2 ist gra-

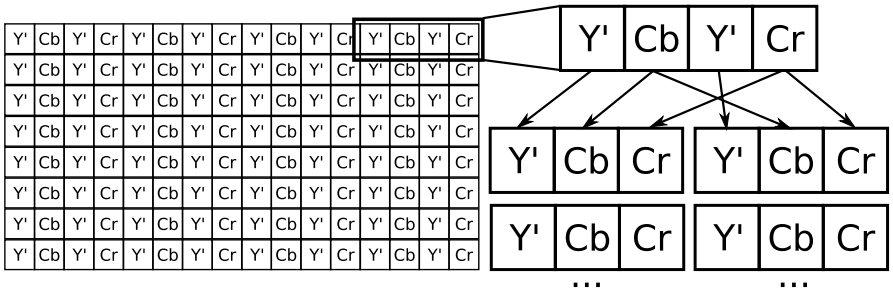


Abbildung 4.2: Bild mit 8 mal 8 Pixeln in Y'CbCr-4:2:2 kodiert und die Zuordnung der Pixel zu einem Y'CbCr-Pixel mit voller Farbinformation.

phisch dargestellt, wie aus zwei 4:2:2-kodierten Pixeln wieder zwei Pixel mit vollständiger Kodierung gewonnen werden können und wie die Pixel als Matrix abgespeichert werden.

4.1.3 Interpretation als quadratische Matrix

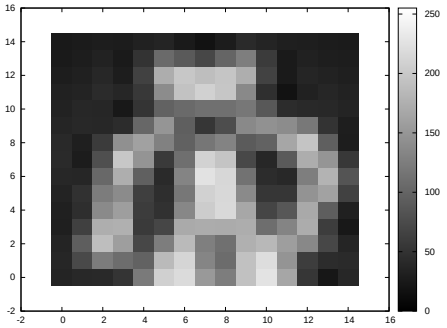
Soll ein Bildverarbeitungsalgorithmus nur auf dem Helligkeitskanal, also dem Graustufenbild, arbeiten, bietet sich die Darstellung als Matrix an. Wie in Abbildung 4.3 zu sehen, kann ein Bild mit einer horizontalen Auflösung n und einer vertikalen Auflösung m als eine Matrix I der Dimension (n, m) dargestellt werden. Jedes Matrixelement i_{xy} (mit $x, y \in \mathbb{N}$, $0 < x \leq n$ und $0 < y \leq m$) entspricht dabei dem Helligkeitswert an der Koordinate (x, y) des Bildes. Der Wertebereich ist von 0 bis 255, wobei 0 „schwarz“ und 255 „weiß“ entspricht. Um besser mit den Matrizen arbeiten zu können, wird oft angenommen, dass die Bilder eine quadratische Matrix mit $n = m$ bilden.

4.2 Lokale Features

Lokale Features oder Merkmale beschreiben für ein spezifisches Pixel in einem Bild eine zu wählende Eigenschaft. Das Besondere dabei ist, dass nur die nähere Umgebung des Pixels zur Berechnung herangezogen wird und keine ganzheitliche Betrachtung des Bildes erfolgt. Zudem werden lokale Features direkt auf den Pixelinformationen berechnet. Man benötigt daher keine Vorverarbeitung des Bildes durch z.B. Segmentierung oder andere Umwandlungen in abstrakte Konzepte.

4.2.1 Operatoren und Templates

Lokale Merkmale werden durch die Umwandlung jedes Pixels eines Bildes I in das neue Bild N generiert. N hat nach der Umwandlung die gleiche Auflösung bzw. Dimension wie I. Eine solche pixelweise Umwandlung erfolgt durch die sogenannten *Operatoren*. Da lokale Features auch ein Graustufenbild ergeben, können ganze Verarbeitungsketten aufgebaut werden. Zudem ist es damit möglich, die lokalen Features eines Bildes optisch zu visualisieren.



$$I = \begin{bmatrix}
38 & 73 & 124 & 109 & 98 & 186 & 213 & 133 & 108 & 193 & 221 & 148 & 61 & 44 & 42 \\
36 & 94 & 190 & 158 & 71 & 125 & 186 & 127 & 112 & 176 & 184 & 158 & 136 & 71 & 37 \\
31 & 64 & 174 & 176 & 58 & 70 & 170 & 172 & 171 & 173 & 110 & 132 & 172 & 61 & 24 \\
32 & 46 & 138 & 158 & 60 & 49 & 135 & 204 & 217 & 167 & 69 & 87 & 169 & 94 & 32 \\
35 & 49 & 124 & 139 & 63 & 46 & 119 & 209 & 216 & 138 & 54 & 54 & 147 & 162 & 65 \\
39 & 38 & 104 & 174 & 96 & 42 & 132 & 226 & 215 & 114 & 44 & 40 & 126 & 179 & 84 \\
42 & 28 & 79 & 199 & 148 & 58 & 111 & 210 & 196 & 72 & 39 & 90 & 173 & 149 & 56 \\
41 & 31 & 59 & 144 & 161 & 129 & 97 & 118 & 131 & 91 & 98 & 166 & 196 & 95 & 29 \\
37 & 40 & 39 & 44 & 103 & 149 & 95 & 54 & 77 & 137 & 145 & 140 & 120 & 51 & 30 \\
34 & 39 & 38 & 24 & 52 & 98 & 106 & 112 & 112 & 119 & 88 & 44 & 41 & 40 & 36 \\
33 & 35 & 43 & 38 & 56 & 140 & 194 & 209 & 198 & 137 & 47 & 18 & 33 & 38 & 34 \\
29 & 32 & 39 & 29 & 65 & 173 & 199 & 190 & 197 & 174 & 66 & 26 & 37 & 34 & 31 \\
26 & 29 & 34 & 26 & 51 & 106 & 90 & 71 & 101 & 126 & 59 & 26 & 33 & 30 & 29 \\
25 & 27 & 29 & 28 & 33 & 36 & 26 & 18 & 28 & 42 & 36 & 31 & 30 & 28 & 27
\end{bmatrix}$$

Abbildung 4.3: Graustufenbild mit 15x15 Pixeln als Plot (oben) und als Matrix I (unten).

Die sogenannten *Templates* sind eine allgemeine Art Operatoren zu beschreiben. Die 3x3 *Umgebung* eines Pixels an der Position (x, y) des Bildes I besteht aus seinen angrenzenden Pixeln

$$U_{x,y} = \begin{bmatrix} I_{x-1,y-1} & I_{x,y-1} & I_{x+1,y-1} \\ I_{x-1,y} & I_{x,y} & I_{x+1,y} \\ I_{x-1,y+1} & I_{x,y+1} & I_{x+1,y+1} \end{bmatrix} . \tag{4.2}$$

Umgebungen können größere Bereiche umfassen, sind im Normalfall aber immer quadratisch und besitzen eine ungerade Seitenlänge bzw. -breite [30, Seite 81]. Weitere übliche Umgebungsgrößen sind 5x5, 7x7 und 9x9. Ein Template weist jedem dieser Pixel der Umgebung ein Gewicht zu. Möchte man nun das Merkmal eines Pixels ermitteln, muss man die Summe der Umgebungspixel multipliziert mit ihrem jeweiligen Gewicht bilden. Sei ein 3x3 Template

$$T = \begin{bmatrix} w_{-1,-1} & w_{0,-1} & w_{1,-1} \\ w_{-1,-1} & w_{0,0} & w_{1,0} \\ w_{-1,1} & w_{0,1} & w_{1,1} \end{bmatrix} \tag{4.3}$$

gegeben. Dann ergibt sich das neue Bild N durch die Anwendung des Templates an jeder Stelle (x, y) eines Bildes I:

$$N_{x,y} = \sum_{i=-1}^{1} \sum_{j=1}^{1} w_{i,j} \cdot I_{x+i,y+j} \quad . \tag{4.4}$$

An den Randzonen der Bilder entsteht dabei das Problem, dass die Umgebung nicht vollständig gegeben ist. Man behilft sich damit, den Merkmalswert an den Rändern immer auf 0 zu setzen und das Template dort nicht anzuwenden.

4.2.2 Kanten als lokale Features und Sobel-Operator

Kanten in einem Bild sind ein mögliches lokales Feature. Sie entstehen immer dann, wenn sich die Bildinformation rapide ändert. Betrachtet man das Bild zeilenweise (bzw. spaltenweise), so entspricht jede Zeile einer Folge von Zahlen. Interpretiert man diese Folge als Funktion, können von ihr Ableitungen erster Ordnung gebildet werden. An einer Stelle x einer Zeile bildet sich die Ableitung also durch $f'(x) = f(x) - f(x-1)$. Kanten können nun als zeilen- oder spaltenweise lokale Maxima von f' definiert werden.

Der Sobel-Operator versucht, die Differenz der Helligkeitsinformation in eine bestimmte Richtung zu erkennen. Dabei werden zwei verschiedene Templates für die x- und y-Richtung benutzt (aus [30, Seite 124]):

$$T_{Sx} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & \mathbf{0} & -2 \\ 1 & 0 & -1 \end{bmatrix} , \tag{4.5}$$

$$T_{Sy} = \begin{bmatrix} 1 & 2 & 1 \\ 0 & \mathbf{0} & 0 \\ -1 & -2 & -1 \end{bmatrix} . \tag{4.6}$$

Abbildung 4.4: Anwendung des Sobel-Operators zur Kantenerkennung auf die Abbildung 4.1b

Mithilfe dieser beiden Templates werden die neuen Bilder N_x und N_y gebildet, die mit der Formel

$$N(x,y) = \sqrt{N_x(x,y)^2 + N_y(x,y)^2} \tag{4.7}$$

wieder zu einem neuen Bild zusammengesetzt werden. Der Sobel-Operator bevorzugt die Reihe bzw. Spalte in der der Pixel steht bei der Differenzbildung, beachtet aber auch die beiden daneben liegenden Reihen oder Spalten. In Abbildung 4.4 ist das Resultat der Anwendung des Sobel-Operators sichtbar. Der Sobel-Operator liefert keine binäre Entscheidung, ob ein Pixel zu einer Kante gehört oder nicht. Durch die Wahl eines geeigneten Schwellwertes kann dieses Bild in ein Binärbild überführt werden. Damit ist dann die Unterscheidung zwischen Kantenbereich und nicht Kantenbereich im Bild möglich.

Der Sobel-Operator betrachtet nur zwei Richtungen, kann also bei einer Rotation des Bildes andere Ergebnisse liefern. Dagegen ist er tolerant gegenüber Änderungen in der Skalierung und Positionierung. Diese drei Eigenschaften werden immer wieder bei der Beurteilung von lokalen Features herangezogen.

4.3 Feature-Erkennung mit SURF

SURF (Speeded Up Robust Features) wurde 2006 von Herbert Bay als Verbesserung der bestehenden Verfahren wie SIFT (Scale-invariant feature transform), insbesondere mit Hinsicht auf die Ausführungsgeschwindigkeit, vorgestellt [3] und 2008 noch einmal umfassend erläutert [2]. Soweit nicht anders angegeben, sind die folgenden Ideen und Formeln in diesem Abschnitt aus diesen Publikationen entnommen. Bei SURF wird die Beschreibung des Bildes in zwei Schritte aufgeteilt. Im ersten Schritt werden sogenannte Keypoints aus der Pixelmenge gefunden. Dies sind besonders markante Punkte eines Bildes. Die Idee ist, dass in verschiedenen Bildern vom gleichen Objekt die Keypoints auch bei Translokation, Skalierung und Rotation des Objektes trotzdem noch an der gleichen Stelle relativ zum Objekt vorhanden sind (nur eben an anderer Stelle im Bild). Im zweiten Schritt wird für jeden dieser Keypoints ein Vektor mit konstanter Länge, der Deskriptor, berechnet. Dieser Deskriptor kann zur Analyse, ob zwei Featurepunkte sich ähnlich sind, eingesetzt werden.

4.3.1 Integralbild

Während SURF auf den Ideen von SIFT aufbaut, so enthält er doch ein wichtiges Konzept, dass seine Ausführungsgeschwindigkeit deutlich verbessert. So wird vom Bild I das sogenannte Integralbild

$$I_\Sigma(x, y) = \sum_{i=1}^{x} \sum_{j=1}^{y} I(i, j) \tag{4.8}$$

berechnet. Jedes Pixel des Integralbildes ist die Summe aller Pixel vor diesem Pixel. Damit ist es möglich, die Summe von Pixelwerten eines rechteckigen Bereiches sehr effizient durch eine konstante Anzahl von Summen bzw. Differenzen zu bilden. Sei dieser rechteeckige Bereich durch die Punkte $\boldsymbol{a}$ (rechts unten), $\boldsymbol{b}$ (rechts oben), $\boldsymbol{c}$ (links unten) und $\boldsymbol{d}$ (links oben) definiert. Dann bildet sich die Summe der Pixel in diesem Rechteck durch

$$sum(I, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, \boldsymbol{d}) = I_\Sigma(a_1, a_2) - I_\Sigma(b_1, b_2) - I_\Sigma(c_1, c_2) + I_\Sigma(d_1, d_2)\ . \tag{4.9}$$

Mit der Summe über die rechteckigen Bereiche kann zum Beispiel die Auflösung eines Bildes verringert oder ein entsprechendes Template berechnet werden.

4.3.2 Erkennen der Keypoints

Der Ansatz von SURF zur Erkennung der Keypoints ähnelt dem Finden von sogenannten Blobs, also Flächen im Bild. Dazu wird die Hessische Matrix der zweidimensionalen Gaußschen Funktion gebildet. Die Hessische Matrix besteht aus den partiellen Ableitungen der Gaußschen Funktion gefaltet mit dem Bild I (entspricht L_{xx}, L_{yy} und L_{xy}):

$$\mathcal{H}(x, y, \sigma) = \begin{pmatrix} L_{xx}(x, y, \sigma) & L_{xy}(x, y, \sigma) \\ L_{xy}(x, y, \sigma) & L_{yy}(x, y, \sigma) \end{pmatrix}\ . \tag{4.10}$$

σ ist ein Skalierungsfaktor. Die hieraus entstehenden diskreten Templates (deren Größe von σ abhängig ist) werden durch einen Box-Filter approximiert. Bei diesen Box-Filter-Templates hat jedes Matrixelement nur ganzzahlige Werte, die in rechteckigen Bereichen angeordnet sind. Zusammen mit dem Integralbild können die einzelnen Bereiche des Templates daher sehr effizient für jeden Teilbereich ausgewertet werden. Für $\sigma = 1.2$ ergeben sich die kleinst-möglichen Templates mit einer Größe von 9x9:

$$L_{yy}(x, y, 1.2) \approx D_{yy} = \begin{bmatrix} 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & 0 & 0 \\ 0 & 0 & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & 0 & 0 \\ 0 & 0 & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & \mathbf{-2} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & 0 \end{bmatrix}, \tag{4.11}$$

$$L_{xy}(x, y, 1.2) \approx D_{xy} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & \mathbf{-1} & \mathbf{-1} & \mathbf{-1} & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & \mathbf{-1} & \mathbf{-1} & \mathbf{-1} & 0 \\ 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 & \mathbf{-1} & \mathbf{-1} & \mathbf{-1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{-1} & \mathbf{-1} & \mathbf{-1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ 0 & \mathbf{-1} & \mathbf{-1} & \mathbf{-1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ 0 & \mathbf{-1} & \mathbf{-1} & \mathbf{-1} & 0 & \mathbf{1} & \mathbf{1} & \mathbf{1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{4.12}$$

L_{xx} wird analog zu L_{yy} gebildet. Nun kann die approximierte Determinante

$$det(\mathcal{H}_{approx}) = D_{xx}D_{yy} - (w \cdot D_{xy})^2 \tag{4.13}$$

gebildet werden. w ist ein (als konstant angenommener) Gewichtungsfaktor um die Auswirkungen der Approximation auszugleichen.

x und y sind konkrete Punkte im Raum, da sie den Pixeln entsprechen. Aber auch σ muss in eine diskrete Form gebracht werden. Dazu werden die sogenannten Oktaven benutzt. Jede Oktave ist eine Folge von diskreten Werten für σ, aus denen sich dann die Größe der Templates ergibt. Die Oktaven sind nach oben und unten begrenzt. Valide Werte für σ werden durch die Topologie der Templates und deren mögliche Kantenlänge begrenzt. So müssen sie immer ein zentrales Pixel besitzen. Daher müssen Templates immer eine ungerade Kantenlänge haben. Dieses zentrale Pixel soll später mit allen Nachbarn in x und y-Ausrichtung, aber auch mit seinen Nachbarn im vorherigen und nächsten Skalierungsschritt verglichen werden. Wir benötigen also nach oben und unten je einen Schritt mehr in der Oktave, als wir mögliche Werte für die Position des Keypoints in der Skalierungsebene annehmen. $\sigma = 1.6$ ergibt die Oktave mit der kleinstmöglichen Skalierung. Daraus leitet sich eine Folge von Templates ab, deren Ränder die Länge 9, 15,

21 und 27 haben. In diesem Fall kann σ also genau zwei verschiedene Werte einnehmen. Weitere in [2] beschriebene Folgen von Kantenlängen sind 15, 27, 39, 51 sowie 27, 51, 75, 99.

Um nun Punkte zu finden, die ein Keypoint darstellen, wird für jeden Punkt im Bild für den Punkt (x, y, σ) selbst und seine Umgebung

$$\begin{aligned} U = \{(i,j,l)\} \text{ mit } & i \in \{x-1, x, x+1\}, \\ & j \in \{y-1, y, y+1\}, \\ & l \in \{\text{Schicht davor}, \text{gleiche Schicht}, \text{Schicht danach}\} \end{aligned} \tag{4.14}$$

die Determinante der approximierten Hessischen Matrix $det(\mathcal{H}_{approx})$ an der Stelle (i, j, l) im Raum berechnet. Nur wenn der zentrale Punkt das Maximum oder Minimum in der Umgebung darstellt, wird er als Keypoint vermerkt. Dies entspricht einem „non-maximum suppression“ Filter. Für jeden Keypoint ist nach diesem Verfahren die x- und y-Koordinate im Bild, sowie die Skalierung σ bekannt. Das Verfahren kann mit verschiedenen Oktaven wiederholt werden.

Wenn das SURF-Verfahren rotationsabhängig sein soll, wird in SURF für jeden Keypoint ein Rotationswert berechnet. Dazu wird in der kreisförmigen Nachbarschaft des Keypoints für jeden Punkt eine Antwort („response“) für je ein Haar-Wavelet für die x- und für die y-Richtung berechnet. Die x- und y-Antworten werden abhängig von ihrer Distanz zum Zentrum mit der Gauß-Funktion gewichtet. Diese gewichteten Werte spannen einen Raum auf, auf dem ein gleitendes Fenster in Form eines Kreissektors mit einem bestimmten Öffnungswinkel gelegt wird. An der Stelle, an der die Summe der durch das Fenster abgedeckten Antworten am höchsten ist, ist auch der Winkel des Keypoints. Die Bestimmung des Rotationswinkels eines Keypoints ist optional. Wird darauf verzichtet, so wird das daraus resultierende Verfahren „u-SURF“ (für „upright SURF“) genannt. In Abbildung 4.5 ist das Ergebnis der Anwendung des SURF-Verfahrens zur Erkennung der Keypoints auf die Abbildung 4.1b zu sehen. Jeder farbige Punkt entspricht einem Keypoint und die ausgehenden Linien entsprechen der Rotation und Größe des Keypoints.

4.3.3 Beschreibung der Keypoints

Für jeden Keypoint soll ein Vektor mit 64 Elementen als „Deskriptor“ berechnet werden. Dazu wird eine quadratische Region um den Keypoint herum angelegt. Die Größe der Region ist abhängig von der Skalierung des Keypoints. Falls die Rotation des Keypoints vorhanden ist, so wird auch die Region um den Keypoint herum mit diesem Wert rotiert. Diese Region wird nun in 4 x 4 Unterregionen aufgeteilt. Innerhalb der Unterregionen wird auf 5 x 5 gleichmäßig geteilten Flächen die Haar-Wavelet-Antwort berechnet, jeweils in x- und y-Richtung. Die Summe der Antworten einer Richtung in einer Unterregion geht als Wert in den Vektor ein. Damit sind bereits $(4 \cdot 4) \cdot 2 = 32$ Vektorwerte bestimmt. Zusätzlich wird aber auch die Summe der Beträge der Haar-Wavelet-Antworten in jede Richtung berechnet. Damit ergibt sich die Gesamtzahl von 64 Vektorelementen. Der

Abbildung 4.5: SURF-Keypoints mit Rotation für die Abbildung 4.1b

Vektor beschreibt im Groben die Gradienteninformation in den Unterregionen um den Keypoint herum.

4.4 CenSurE- und Star-Detektor

SURF ist zwar mit Hinblick auf eine deutliche Laufzeitverbesserung im Vergleich zu SIFT entwickelt worden, auf dem Nao Roboter benötigt die Ausführung auf dem kompletten Bild aber immer noch deutlich zu viel Zeit. Eine weitere Verbesserung in der Geschwindigkeit verspricht das CenSurE-Verfahren (Center Surround Extrema), das in [1] beschrieben wird. CenSurE nutzt ebenfalls Integralbilder, der interne Filter entspricht aber nicht der Hessischen Matrix, sondern ist eine Approximation des Laplace-Operators. Der Filter zur Approximation kann entweder die Form eines Oktagons (besonders genau), eines Hexagons (weniger genau, schneller) oder einer Box (sehr schnell, ungenauer) haben. Eine Besonderheit von CenSurE ist, dass es nicht nur innerhalb einer Oktave nach Extrema sucht, sondern im gesamten Skalenbereich. Allerdings kann im Gegensatz zu SURF keine Rotation der Keypoints erkannt werden. Es ist trotzdem möglich CenSurE zum Erkennen der Keypoints und SURF als Deskriptor zu kombinieren. Die Implementierung von CenSurE in OpenCV wird Star-Detektor genannt.

5 Technische Hilfsmittel

In diesem Kapitel wird der verwendete Roboter und die SimSpark-Simulationsumgebung für diesen Roboter beschrieben. Zudem wird ein Überblick über die verwendete Programmierumgebung inklusive der verwendeten OpenCV Softwarebibliothek gegeben. Die Rechenkapazität auf dem Roboter ist stark eingeschränkt und daher muss die Implementierung unter Umständen speziell für den Roboter angepasst werden. Dies wird am Beispiel der Konvertierung des Bildes erläutert.

5.1 Humanoide Roboterplattform Nao

Der Nao ist ein humanoider Roboter, der kommerziell von der Firma Aldebaran Robotics produziert wird. In Abbildung 5.1 ist ein Nao Roboter der dritten Generation zu sehen. Der Roboter besitzt 25 Freiheitsgrade, zwei Kameras als Hauptsensor, ein Gyrometer, Beschleunigungssensoren, Drucksensoren an den Füßen und Ultraschallsensoren zur Erkennung von Hindernissen [17]. Er misst eine Höhe von 57cm und wiegt etwas über 4 Kilogramm. Da der Nao Roboter autonom agieren soll, besitzt er einen eigenen Prozessor und wird per Batterie mit Strom versorgt. Der verwendete Prozessor ist ein AMD Geode LX mit 500Mhz Taktfrequenz und es stehen 256MB Arbeitsspeicher zur Verfügung.

5.2 SimSpark-Simulator

Der SimSpark-Simulator [5] wurde für den Einsatz in der RoboCup 3D-Simulationsliga entwickelt. Mit ihm ist es möglich, Teams von bis zu neun simulierten Robotern gegeneinander Fußball spielen zu lassen. Als Robotermodell wird der Nao verwendet. SimSpark simuliert eine komplette physikalische Welt mit einem Fußballfeld, den Robotern und dem Ball. In Abbildung 5.2 ist ein Ausschnitt aus einem Spiel in SimSpark zu sehen.

Die einzelnen Agenten in SimSpark agieren autonom. Sie kommunizieren über ein Netzwerkprotokoll mit dem Simulationsserver, der für jeden einzelnen Roboter individuelle Sensorinformationen berechnet. Als Reaktion auf die Sensorinformationen können die simulierten Roboter die verschiedenen Motoren ihres Körpers ansteuern und somit die physikalische Welt des Simulators manipulieren. Untereinander dürfen die Agenten nur über den Simulationsserver als Makler kommunizieren. Die dabei erlaubte Informationsmenge ist künstlich begrenzt.

Jeder Agent besitzt eine Kamera als Hauptsensor. Diese Kamera hat einen bestimmten Öffnungswinkel und zu jedem Zeitpunkt eine bestimmte Position im dreidimensionalen Raum, die sich aus der Konfiguration der Gelenke und der Lage des Roboters ergibt.

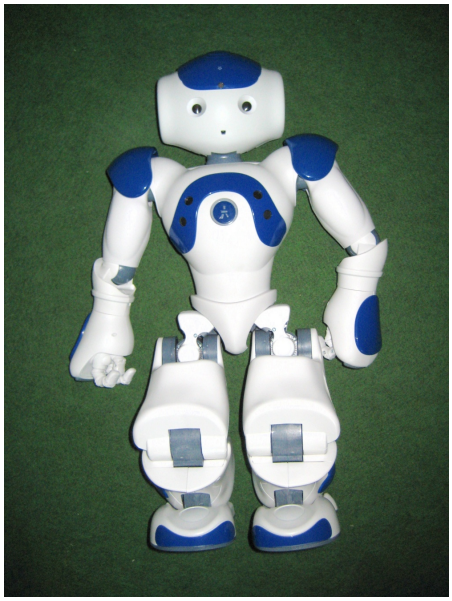


Abbildung 5.1: Nao Roboter

Abbildung 5.2: Screenshot eines simulierten Fußballspiels im SimSpark-Simulator.

Es wird allerdings nicht ein generiertes Kamerabild an den Agenten geschickt, sondern verrauschte Perzeptdaten von den Objekten, die der Roboter von der aktuellen Kamerapose aus sehen kann. Damit ist eine Bildverarbeitung auf der Seite des Agenten nicht notwendig. Zudem ist der simulierte Öffnungswinkel recht breit, sodass ein Roboter eine gute Übersicht über das Spielfeld hat. Die Daten sind zwar verrauscht (je stärker je weiter das Objekt entfernt ist) aber immer noch deutlich besser als die Perzepte, die man in Echtzeit durch Bildverarbeitungsmethoden auf den Nao generieren könnte.

SimSpark bildet eine Art idealisierte Welt ab. Einige der Probleme, die beim Einsatz realer Roboter in einer realen physischen Umgebung auftreten, können durch den Einsatz von SimSpark vermieden werden. So gibt es keinen Verschleiß der Gelenke und die Roboter können durch Kollisionen oder Stürze nicht beschädigt werden. Außerdem können Spiele mit vielen Spielern automatisiert und ohne Eingriff eines menschlichen Experimentators durchgeführt werden. Bei einem Spiel von neun gegen neun Robotern stehen 18 Roboter auf dem Spielfeld. Solche Spiele sind taktisch sehr interessant, aber die dafür notwendige Hardware ist äußerst teuer. In der Standard-Plattform-Liga (SPL) spielen daher nur vier gegen vier reale Roboter gegeneinander. Die Simulation in SimSpark eignet sich dagegen für die Generierung von vielen Trainingsdaten, da es möglich ist, viele Spiele unter kontrollierten und konstanten Umweltbedingungen durchzuführen. Die so gewonnenen Trainingsdaten sind aber nicht automatisch auf die reale Welt übertragbar, da die jeweilige Physik zwangsweise Unterschiede haben muss. Es eignet sich aber trotzdem als Testfeld, um die Eignung von Algorithmen zu überprüfen und ihre Leistungsfähigkeit zu vergleichen.

5.3 Programmierumgebung

Für die Implementierung der Experimente wurde die Programmiersprache C++ verwendet. Der Roboter wird durch ein modulares Framework gesteuert, dass am Lehrstuhl für Künstliche Intelligenz der Humboldt Universität entwickelt wurde und ursprünglich für den Einsatz im RoboCup gedacht war. Dieses Framework erlaubt es, eigene Module zur Erweiterung der Fähigkeiten des Roboters zu implementieren. Zudem können die Module unverändert auf dem echten Roboter und in der Simulation eingesetzt werden [26].

5.4 OpenCV

OpenCV [6] ist eine allgemeine Softwarebibliothek zur Bildverarbeitung. Sie wurde ursprünglich durch Intel entwickelt, die aktuelle Entwicklung wird allerdings durch die Robotikfirma Willow Garage vorangetrieben [7, Seite 512]. Es existieren Schnittstellen für die Programmiersprachen C, C++ und Python. Durch die Nutzung von OpenCV können zahlreiche Bildverarbeitungsalgorithmen vergleichsweise einfach in die eigene Anwendung integriert werden und eine eigenständige zeitaufwendige und fehleranfällige Implementierung kann vermieden werden.

Alle Algorithmen der OpenCV-Bibliothek arbeiten auf einer einheitlichen Datenstruktur für Bilder. Diese Datenstruktur entspricht einer allgemeinen Matrix, wobei die einzelnen Matrixelemente wiederum Vektoren mit unterschiedlicher Länge und Datentyp (im Sinne der verwendeten Programmiersprache) darstellen [7]. Jedes Matrixelement entspricht dem Pixel im Bild, während die einzelnen Komponenten der Vektoren die verschiedenen Farbkanäle repräsentieren. Je nach Implementation des Algorithmus ist das verwendete Farbmodell fest vorgeschrieben oder variabel. Da die meisten Feature-Erkennungsalgorithmen nur auf dem Graustufenbild arbeiten, ist auch nur das Vorhandensein dieses Farbkanals notwendig.

Während OpenCV für jedes Pixel die vollen Farbinformationen benötigt, verwendet die Kamera des Nao Roboters nativ ein Interlacing-Verfahren [31]. Bei dieser YCbCr-4:2:2 Kodierung wird nur die Graustufeninformation voll aufgelöst, während sich zwei Pixel die gleiche Farbinformation teilen müssen (siehe Kapitel 4.1.2 für eine genauere Beschreibung).

Um das Kamerabild in OpenCV nutzbar zu machen, ist daher eine Konvertierung nötig. OpenCV ermöglicht es, bestehende C++-Arrays als OpenCV-Matrix zu betrachten. Zudem gibt es die allgemeine Funktion „cv::mixChannels“, die es erlaubt die Kanäle einer Matrix neu angeordnet in eine andere Matrix zu kopieren. Diese Methode ist konzeptionell sehr klar, die Laufzeit auf dem Roboter ist dagegen sehr langsam.

Das Kopieren und Konvertieren des Bildes musste daher im Rahmen dieser Arbeit möglichst effizient implementiert werden. Da die Größe der zu kopierenden Daten die Laufzeit stark beeinflusst, wurde nur einer der drei Helligkeitskanäle kopiert. Zudem wird die Auflösung des Bildes beim Konvertieren verkleinert. Während die Ursprungsgröße 320 mal 240 Pixel ist, ist das OpenCV-Bild nur noch 160 mal 120 Pixel groß. Diese Konvertierung beeinflusst die spätere Bildverarbeitung nur in geringem Maße. Der SURF-Algorithmus benötigt nur ein Graustufenbild und kann auch auf der kleinerer Auflösung Objekte, wie zum Beispiel die Füße des Naos, bis zu einigen Metern Entfernung erkennen. Zudem benötigt SURF auf einem kleineren Bild auch weniger Rechenzeit, was einen weiteren Vorteil darstellt. Auf dem Roboter benötigt die selbst implementierte Konvertierung nur 2 bis 3 Millisekunden Rechenzeit.

Leider weist der auf dem Nao eingesetzte Geode LX Prozessor einige architekturbedingte Schwächen auf. So besitzt er zwar einen Cache für den Zugriff auf den Hauptspeicher, dieser ist aber nicht als „read-ahead“ Cache implementiert. Es kann daher beim Kopieren vorkommen, dass das lineare Lesen der Daten aus dem Hauptspeicher langsamer vonstattengeht als das Schreiben. Um dieses Problem zu beseitigen, muss man die Bilddaten blockweise in einen Zwischenspeicher kopieren und erst dort die Konvertierung vornehmen. Im konkreten Fall hat es sich als günstig erwiesen, immer eine Zeile als Ganzes zu lesen.

6 Erkennung des Roboters

Um die Laufwege eines Roboters vorhersagen zu können, muss die relative Position des Roboters erst einmal erkannt werden. Ein autonomer Roboter, der als Hauptsensor das Kamerabild benutzt, muss in der Lage sein, selbstständig Roboter in einem Bild zu erkennen, die Position in relative Koordinaten auf dem Spielfeld umzurechnen und diese Position zu verfolgen.

6.1 Existierende Forschung

In den Bereichen der humanoiden Robotik, die sich mit der Interaktion mit anderen Robotern beschäftigen, ist die Erkennung der Roboter eine elementare Fähigkeit. Im RoboCup ist diese Interaktion besonders ausgeprägt (siehe Kapitel 1.2). Daher gibt es hier immer wieder Ansätze, um dieses Ziel zu erreichen. Da in der SPL des RoboCups ebenfalls der Nao Roboter eingesetzt wird, lohnt sich also ein Blick auf die dort verwendeten Strategien. Dies ist insbesondere sinnvoll, da der Nao nur eine begrenzte Rechenkapazität besitzt und im RoboCup hohe Bildverarbeitungsraten erreicht werden müssen, um erfolgreich sein zu können.

6.1.1 Farbbasierte Ansätze

Wenn die Objekte der Umwelt farbkodiert sind, kann die Nutzung dieser Information zur Beschleunigung der Erkennung benutzt werden. Dabei wird der Farbraum meist in Farbklassen eingeteilt. Diese Einteilung kann zum Beispiel manuell durch sogenannte Farbtabellen erfolgen, wie es in [10] beschrieben wird. Dabei wird für jeden Punkt im Farbraum die Farbklasse explizit per Hand annotiert. Diese Annotation kann allerdings auch automatisiert erfolgen [18]. Trotzdem haben auf Farbtabellen basierte Ansätze große Nachteile. So muss die Farbtabelle bei Änderungen der Beleuchtung angepasst werden und es gehen Informationen über die Nachbarschaft der Pixel, die ebenfalls die subjektive Farbinformation beeinflusst, verloren [36].

In [37] wird grob ein Ansatz beschrieben, wie mithilfe des farbsegmentierten Bildes Nao Roboter im RoboCup erkannt werden könnten. Dabei werden pinke und blaue Regionen gesucht, die den farbigen Markierungen, die im RoboCup eingesetzt werden, entsprechen. Anschließend müssen diese Fundstellen durch eine Plausibilitätsprüfung, die den Grad an weißen Pixeln in der Umgebung der gefundenen Regionen misst, bestätigt werden. Dies ist notwendig, da z.B. auch eines der beiden Tore eine blaue Färbung hat und blaue beziehungsweise pinke Farbregionen auch in der näheren Umgebung des Spielfeldes nicht ausgeschlossen werden können.

Solche farbbasierten Ansätze haben sich immer wieder als sehr fehleranfällig erwiesen. Zudem ist eine sehr gute Kalibrierung der Farben notwendig, um den Fehler minimieren zu können. Ein großer Teil der Arbeit muss bei solchen Ansätzen also in die genaue Kalibration auf die konkrete Umgebung angewendet werden.

6.1.2 Haartraining

Neben der manuellen Implementierung eines Erkennungsalgorithmus gibt es Versuche, generellere Bilderkennungsmethoden einzusetzen. So wird in [14] ein Verfahren beschrieben, dass „Haar like features“ und einen darauf abgestimmten Lernprozess, das Haartraining, beschreibt.

Für diesen Prozess müssen bestimmte am Roboter wiederzuerkennende Teile ausgewählt werden. Danach erstellt man eine Datenbank aus Testbildern. Dafür dürfen die Bilder der Positivbeispiele nur den später zu erkennenden Bereich enthalten, müssen also per Hand beschnitten werden. Das zitierte Paper hat für die Experimente den Roboter als Ganzes, das Gesicht und den markanten Brustbereich ausgewählt. Bereits in der auf Simulationen basierenden Explorationsphase wurde die Erkennung des Roboters als Ganzes verworfen, da die Erkennungsrate deutlich zu niedrig und die Fehlerrate astronomisch hoch war. So gab es nur 20 Hits, aber 180 nicht erkannte Roboter und 1200 fehlerhaft als Roboter erkannte Bereiche des Bildes. Auch die Erkennung des Gesichts war problematisch. So gab es hier zwar eine Hit-Ratio von 88 Prozent, allerdings lag die Rate der False-Positives zwischen 400 (beim Lernen auf verrauschten Bildern) und 200 Prozent (nicht verrauschte Bilder). Nur die Erkennung des Musters des Brustbereiches war mit einer Hit-Ratio von 79 Prozent und einer False-Positive-Rate von 2,5 Prozent ausreichend genau.

Die Verarbeitungszeit für ein einzelnes Bild geben die Autoren mit etwa 10 ms auf einem „konventionellen Computer“ an. Sie erwähnen auch eine Laufzeit von etwa 40 ms auf dem Nao Roboter. Leider wird nicht deutlich gemacht, ob diese 40 ms eine tatsächlich gemessene Zeit oder nur eine Schätzung aufgrund der Ausführungszeit auf dem Desktop-PC und von Erfahrungswerten sind. So hat sich in den hier ausgeführten Experimenten herausgestellt, dass die Laufzeiten nicht linear auf den Geode-Prozessor übertragbar sind und er einige „Eigenheiten“ besitzt, die problematisch werden können.

6.2 Robotererkennung durch Features

Da die bisherigen Methoden zur Erkennung eines Naos entweder einen zu hohen Kalibrierungsaufwand hatten oder die Laufzeit auf dem Roboter zu hoch ist, wurde die Erkennung von Robotern durch CenSurE und SURF umgesetzt. Dabei wurde die Implementierung von OpenCV genutzt. Dabei mussten die Parameter der Erkennungsalgorithmen so angepasst werden, dass sie nicht mehr als etwa 20ms Laufzeit auf dem realen Roboter erreichen. Ein Nachteil der gewählten Parameter war, dass nur eine maximale Anzahl von Featurepunkten gefunden werden kann. Existieren bereits „interessante“ Bereiche im Zentrum des Bildes, so werden am Rand keine weiteren Features mehr entdeckt. Dieses Problem lässt sich teilweise durch eine aktive Ausrichtung des Blickfeldes

auf einen bereits erkannten Roboter umgehen. Durch Tracking lässt sich aber immer nur ein Roboter gleichzeitig verfolgen.

6.2.1 Klassifikation der Features

Featurepunkte müssen klassifiziert und benannt werden, um ihnen sinnvolle Bedeutungen zu geben. Dazu wurde eine graphische Oberfläche entwickelt, die einem menschlichen Annotator erlaubt einzelne beispielhafte Features auszuwählen und zu benennen. Diese Beispielfeatures werden dann in einer Featuretabelle (ähnlich den Farbtabellen aus 6.1.1) abgespeichert. Da Featuredeskriptoren numerische Vektoren sind, können Ähnlichkeitsmaße wie die L2-Distanz darauf angewendet werden. Die Idee ist, dass ähnliche Featuredeskriptoren auch eine ähnliche Featureklasse implizieren. Sind viele Beispielfeatures in einer Featuretabelle bekannt, kann mithilfe des k-NN-Verfahrens der nächste Nachbar bestimmt werden. Die Klasse des nächsten Nachbars ist dann auch gleich die Klasse des unbekannten Featurepunktes.

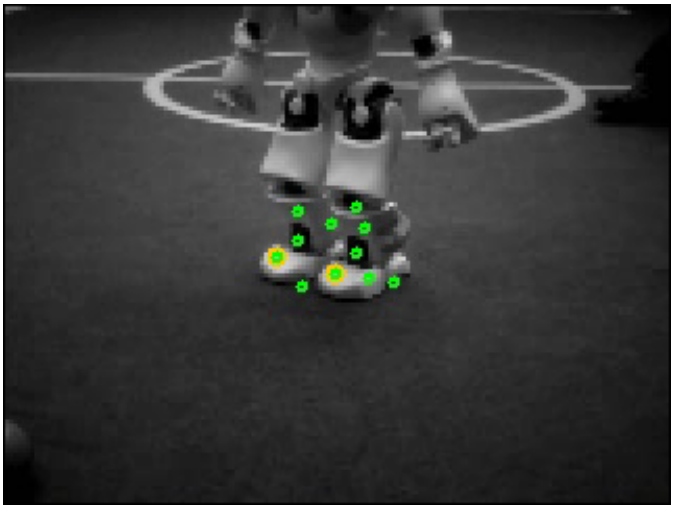
In der graphischen Oberfläche ist es auch möglich, die typische Höhe der Objekte einer Klasse abzuspeichern. Diese Information wird benötigt, um später eine Peilung des Objektes zu erlauben. Nur wenn Klassen eine typische Höhe haben, von denen die Objekte nur kaum abweichen, macht diese Zuordnung einen Sinn. Beispiele für diese Art von Klassen sind Objekte, die immer direkt auf den Boden aufliegen wie der Ball oder die Fußpunkte von Robotern.

6.2.2 Auswahl der geeigneten Features zur Robotererkennung

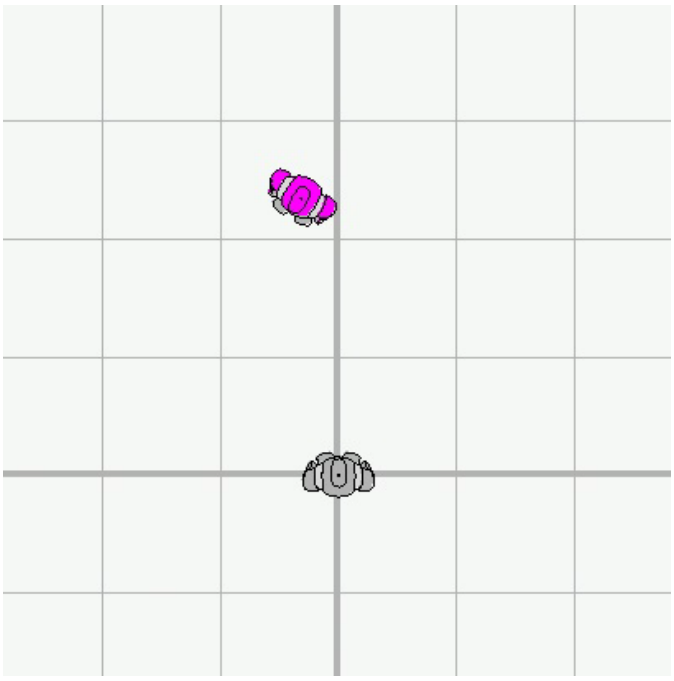
Die Auswahl der Features zur Erkennung eines Objektes ist stark domänenabhängig. So werden beim Nao Roboter nur bestimmte Körperteile als Feature stabil und wiederkehrend erkannt. Diese Stellen sind typischerweise kontrastreiche Übergänge wie das Logo auf der Brust des Roboters oder die Gelenke, die deutlich dunkler als die helle Verschalung um sie herum sind. In Abbildung 6.1a sind einige solcher Stellen markiert. Wichtig ist dabei, dass die Höhe der Features über Grund möglichst konstant bleibt. Bei den Füßen ist diese im begrenzten Rahmen gegeben, da die normale Schritthöhe nicht höher als 1 cm ist. Die runden Einkerbungen am Fuß (in Abbildung 6.1a orange markiert) sind ein gut zu erkennendes Feature, da sie von verschiedenen Winkeln (nicht nur von vorne) und aus verschiedenen Entfernungen sichtbar sind. In den Experimenten konnten Features (allerdings mit verschiedenen Deskriptoren) an diesen Stellen von wenigen Zentimetern Abstand bis zu 1,5 Meter um Bild erkannt werden.

6.2.3 Roboterposition auf dem Feld

Wurde ein Feature eines Fußes als solches erkannt, muss die Bildkoordinate in Feldkoordinaten umgerechnet werden. Da die Position der Kamera im Raum sowie deren Parameter wie Öffnungswinkel bekannt sind, kann dies durch Peilung zum Objekt (das eine bekannte Höhe besitzt) erfolgen. Die Peilung wird für jeden Fuß einzeln durchgeführt. Werden genau zwei Füße erkannt, so kann die Rotation des Roboters berechnet werden. Ein Problem ist es, wenn mehr als zwei Spieler im Bild sind. Hier müssen die



(a) Erkennung der Features am Fuß



(b) Bestimmung der Position

Abbildung 6.1: Erkennung der Features am Fuß (orange umrandet) und die daraus resultierende relative Position auf dem Feld nach der Peilung.

einzelnen Füße zu Robotern zusammengefasst werden. Dies kann zum Beispiel durch die Nähe der Füße auf dem Feld zueinander geschehen (es werden immer die Zwei am nächsten zusammenliegenden Paare von Füßen gruppiert), ist aber im Rahmen dieser Diplomarbeit nicht umgesetzt worden. Gute Erfolge wurden hingegen mit dem Verfolgen („Tracking“) eines gegnerischen Roboters erzielt. Dabei wird gezielt nach den Füßen eines Roboters gesucht, und sobald diese gefunden sind, wird die Kamera so ausgerichtet, dass ihr Zentrum auf die Füße zeigt. Damit ist es möglich, die Bewegung eines einzelnen Gegenspielers über einen längeren Zeitraum zu verfolgen.

7 Experiment zum Lernen von Lauftrajektorien

Die Erkenntnisse aus den vorherigen Kapiteln sollen nun in ein Experiment zur Vorhersage von Lauftrajektorien von Fußballrobotern eingehen. Mithilfe der vorgestellten Techniken sollen verschiedene konkrete Verfahren implementiert und ihre Vorhersagequalität miteinander verglichen werden.

7.1 Ziel des Experiments

Das Ziel dieses Experimentes ist es, zu zeigen ob und wenn ja mit welchen Verfahren und Kodierungen es grundsätzlich möglich ist, die Trajektorien von intentional gesteuerten Agenten vorherzusagen. Dazu soll eine Beispielanwendung gefunden werden, die folgende Eigenschaften aufweist:

- große Mengen an Lern- und Trainingsdaten zuverlässig generierbar
- Wiederholbarkeit der Experimente um verschiedene Algorithmen und Parameter fair vergleichen zu können
- Umwelt mit mehreren passiven und aktiven Objekten

Die Experimente wurden nicht in einem realen Fußballspiel mit echten Robotern durchgeführt, sondern im SimSpark-Simulator (siehe Kapitel 5.2). Durch den Einsatz eines Simulators konnten möglichst viele Trainings- und Testdaten in einer kontrollierten und zuverlässigen Umgebung gewonnen werden.

7.2 Verhaltenssteuerung der Agenten

Um das Bewegungsverhalten von Fußball spielenden Robotern zu lernen, benötigt man Agenten, die sinnvolle Aktionen ausführen und nicht zufällig Entscheidungen treffen. Die Steuerung des Roboters ist dabei ein komplexes Unterfangen. In jedem Zeitschritt müssen die verschiedenen Sensordaten zu einem Modell des eigenen Zustands und den der äußeren Welt aggregiert werden. So ist es für den Spieler essentiell zu wissen, wo er selbst in absoluten Koordinaten auf dem Spielfeld steht und wie die relative Position des Balls zu sich selbst und dessen Geschwindigkeit ist. Diese Informationen können nicht direkt aus der simulierten Welt abgelesen werden, sondern müssen durch die Modellierung und Fusion von Sensordaten über einen bestimmten Zeitraum generiert werden. Zudem muss

der Agent die eigenen Aktionen mit denen der Mitspieler koordinieren und langfristige Verhaltensplanungen treffen, adaptieren und ausführen.

Um ein sinnvolles Verhalten zu gewährleisten, wurden die bestehende Agentensoftware des NaoTeams-Humboldt eingesetzt. Dieses ist 2010 Vizeweltmeister in der 3D-Simulationsliga geworden und entspricht daher dem Stand der Technik. Zudem wird die gleiche Software neben der Simulation auch auf den realen Robotern und in der SPL eingesetzt [42]. Dabei unterscheiden sich zwar die Parameter (zum Beispiel des Laufens) und das Verhalten wird an die Anzahl der Spieler angepasst, grundsätzlich handelt es sich aber um das gleiche Programm. Das NaoTH-Framework ist in Module aufgeteilt [26]. Man kann daher Erweiterungsmodule schreiben, die Beobachtungen protokollieren und abspeichern. Diese Module sind unverändert auf dem echten und auf dem simulierten Roboter in SimSpark einsetzbar.

7.3 Gewinnung der Trainings- und Testdaten

Die Trainingsdaten wurden in simulierten Spielen in SimSpark zwischen zwei Mannschaften mit der gleichen Steuerungssoftware erhoben. Insgesamt wurden drei Halbzeiten aufgezeichnet, zwei zum Training und eine als Grundlage für die Testdaten. Jede Halbzeit simuliert ein Spiel von fünf Minuten. Auf jedem Roboter läuft ein Modul, das in jedem Zeitschritt die Situation im Sinne von Kapitel 3.8 aufzeichnet und einer Datei abspeichert. Der SimSpark-Simulator simuliert in einem Zeitschritt in etwa 20ms reale Zeit. Es werden die eigenen Mitspieler, die Gegenspieler und der Ball als jeweils eigene Objektklasse betrachtet und gespeichert. Für jedes Objekt wird die relative Position in Millimetern angegeben. Die Daten aller Spieler eines Spiels bilden die Wissensbasis und können von einem separaten Programm zum Lernen und Evaluieren der Lernqualität eingelesen werden.

Das Speichern der einzelnen Positionen der Objekte ergibt noch nicht automatisch eine Trajektorie des jeweiligen Objektes. Zuerst muss bestimmt werden, welches Objekt in einem Zeitschritt zu welchem Objekt im nächsten Zeitschritt gehört. Im Algorithmus 3 ist beschrieben, wie diese Zuordnung erfolgt. Es handelt sich bei BESTMAPPING um einen Greedy-Algorithmus, der vermerkt welche Objekte bereits zugeordnet sind und für jedes nicht zugeordnete Objekt die beste lokale Zuordnung findet. Die Komplexität ist $O(n^2)$, wobei n die Länge der Vektoren beschreibt. Zusätzlich zu der eigentlichen Zuordnung berechnet BESTMAPPING auch die durchschnittliche Distanz aller Punkte zweier Ereignisse zueinander.

Wird die Wissensbasis geladen, so werden zuerst die Ereignisse zu Situationen zusammengefasst. Immer wenn sich die Anzahl der Objekte einer Klasse ändert, wird ein neues Ereignis angelegt. Die Situationen besitzen immer die gleiche Konfiguration an Objekten, deshalb ist es möglich für jede Situation eine Matrix wie sie in Gleichung 3.9 definiert wurde aufzustellen. Der BESTMAPPING-Algorithmus wird benutzt, um sicherzustellen, dass die Spalten auch durchgängig zum selben Objekt gehören. Überschreitet die Bewegung einer der Objekte einen gewissen Schwellwert, so wird angenommen, dass sich die Situation grundsätzlich geändert hat und eine neue Situation wird erstellt.

Eingabe : $\boldsymbol{a}$, $\boldsymbol{b}$: Vektoren deren Elemente zugeordnet werden sollen
Daten : $T = \varnothing$: Menge von noch zu mappenden Indexen
Ausgabe : $\boldsymbol{m}$ (Zuordnung der Indizes von $\boldsymbol{a}$ auf $\boldsymbol{a}$, $a_i \leftrightarrow b_{m_i}$), d (Distanz aller Punkte)

```
/* Füge alle Indizes der Menge T hinzu.                                    */
für i ← 1 bis |m| tue
    T = T ∪ {i};
Ende
d = 0;
/* Greedy-Suche nach dem besten passenden Objekt, das noch nicht
   zugeordnet wurde.                                                       */
für i ← 1 bis |m| tue
    b = −1;
    d̂ = ∞;
    für alle t ∈ T tue
        d̃ = ‖a_i, a_t‖_2; wenn d̃ < d̂ dann
            b = t;
            d̂ = d̃;
        Ende
    Ende
    wenn b >= 0 dann
        d = d + d̂;
        T = T\{b};
        m_i = b;
    Ende
Ende
d = d/|m|;
```

Algorithmus 3 : BESTMAPPING zum Finden der besten Zuordnung zweier Objektmengen von zwei Zeitschritten.

Für die Lernalgorithmen, die auf der reinen Trajektorie und nicht auf der Gesamtsituation arbeiten, muss aus dieser vorverarbeiteten Wissensbasis vor dem Lernen eine Menge von Trajektorien in Matrixform berechnet werden (siehe Gleichung 3.4 in Kapitel 3.5). Dabei wird für jeden beobachteten Gegenspieler eine eigene Trajektorie erstellt. Da die Objektklassen und ihre Indexe innerhalb einer Ereignismatrix E bekannt sind, ist diese Herauslösung trivial. Allerdings müssen die Trajektorien noch weiter verarbeitet werden indem die lokalen sowie globalen Features wie sie in Kapitel 3.6.2 und 3.6.3 beschrieben sind für jede Trajektorie berechnet werden. An Bereichen, an denen sich der Roboter nicht bewegt, werden die Trajektorien aufgetrennt. Nur Trajektorien mit einer gewissen Mindestlänge werden in die Wissensbasis aufgenommen.

Für die Evaluierung wird jeweils eine komplette Trajektorie herangezogen. Ihr Ende ist der Wert, der vorhergesagt werden soll und die Features der Teiltrajektorie ohne das letzte Element sind die Eingabe. Damit die Vorhersage zu jedem Zeitpunkt der Trajektorie in die Evaluierung eingehen kann, wird bei der Generierung der Lern- und Testdaten jede Teiltrajektorie einer Gesamttrajektorie kopiert und als eigenständige Trajektorie in die Wissensbasis aufgenommen. Aus einer Trajektorie mit n Zeitschritten und einer Mindestlänge von 20 würden sich daher $n - 20$ Teiltrajektorien ergeben.

7.4 Zu vergleichende Vorhersagealgorithmen und Parameter

In diesem Experiment sollen verschiedene Verfahren miteinander verglichen werden. Folgend eine Auflistung der Verfahren und die Parameter, die sie benötigen. Die Kurznahmen für die Vorhersagealgorithmen wie sie auch in den Graphen verwendet werden stehen jeweils in Klammern.

7.4.1 Alte Position als Vorhersage (STAY)

Bei diesem Verfahren handelt es sich strenggenommen um keine wirkliche Vorhersage. Als „Vorhersage“ wird immer nur die aktuelle Position ohne Veränderung propagiert. Dieses Verfahren dient als Vergleichswert zu den anderen Algorithmen. Zudem ist es als Absicherung vor fehlerhaften Verhalten der anderen Algorithmen gedacht: ist die Genauigkeit eines Algorithmus im Experiment der von STAY zu ähnlich, muss ein Fehler im Lernen vorliegen (zum Beispiel wenn ein neuronales Netz größtenteils nur 0 als Differenz ausgibt).

7.4.2 Geschwindigkeit (SPEED)

Der einfachste hier benutzte Algorithmus ist die Geschwindigkeitsberechnung. Dabei wird die Differenz zwischen der letzen bekannten Position (x_t, y_t) und der vorletzten Position (x_{t-1}, y_{t-1})

$$dx = x_t - x_{t-1} \ , \tag{7.1}$$

$$dy = y_t - y_{t-1} \tag{7.2}$$

berechnet und mithilfe dieser Differenz die Vorhersage

$$x_{t+1} = x_t + dx \,, \tag{7.3}$$
$$y_{t+1} = y_t + dy \tag{7.4}$$

ermöglicht. SPEED benötigt keinerlei Parameter und seine Vorhersagen werden als Vergleichswert benötigt. Die Grundidee ist, dass alle hier verwendeten Vorhersageverfahren besser sein müssen als die trivialen Algorithmen SPEED und STAY.

7.4.3 k-Nearest-Neighbor (K-NN)

Das Prinzip vom k-NN-Verfahren als Regressionsanalyse wird in Kapitel 2.2 erläutert. Hier wird k-NN auf die lokalen und globalen Features der einzelnen Trajektorien und *nicht* auf die Gesamtsituation angewendet. Da die Trainingsmengen sehr groß werden können, wird das FLANN-Approximierungsverfahren eingesetzt. Als Parameter benötigt K-NN die Anzahl der lokalen und globalen Features *low* und *high* sowie die Anzahl der gesuchten Nachbarn k. Die Anzahl der lokalen Features in die Vergangenheit bestimmt, wie weit K-NN in die „Vergangenheit" der Trajektorie gehen kann, um sie mit den anderen Trajektorien zu vergleichen. Je höher *low* ist, desto ähnlicher müssen die Trajektorien auf einem längeren Abschnitt sein. *high* gibt hingegen an, wie grob oder fein die Struktur der vollständigen Trajektorie verglichen werden soll. Der Parameter k ist schwer zu wählen. Ein höherer Wert führt zur Glättung der Ergebnisse und kann Außreißer vermeiden helfen, bringt aber auch unter Umständen Beispiele in die Vorhersage ein die kaum ähnlich sind.

Die beiden Komponenten der Position (also entweder die kartesischen Koordinaten x und y oder die Polarkoordinaten) werden durch zwei voneinander unabhängige Modelle gelernt. Die Länge der zu lernenden Eingangsvektoren beträgt damit jeweils $low + high$ und jedes Modell berechnet als Ausgabe nur die Differenz einer einzelnen Positionskomponente. Die beiden Differenzen addiert mit der alten Position ergibt die Vorhersage für die neue Position. Ob Polarkoordinaten eingesetzt werden oder nicht, wird durch den Parameter *pol* ausgedrückt.

7.4.4 Künstliche neuronale Netze (NEURONAL)

Auch dieses Verfahren benutzt die lokalen und globalen Features wie K-NN, besitzt also auch die Parameter *low* und *high*. Zudem kann auch hier durch *pol* gewählt werden, ob Polarkoordinaten oder kartesische Koordinaten eingesetzt werden. Zum Lernen wird ein Feedforward-Netz, wie es in Kapitel 2.3.2 beschrieben wird, benutzt. Es gibt nur einen Hidden-Layer und die Anzahl der Neuronen in diesen Layer wird durch den Parameter *hidden* beschrieben. Jedes Neuron der Eingabeschicht ist mit je einem der Parameterwerte verbunden und die Differenz der beiden Positionskomponenten wird durch zwei Ausgabeneuronen berechnet. Im Gegensatz zu K-NN verwendet NEURONAL ein gemeinsames Modell für beide Positionskomponenten. Die Anzahl der Neuronen der Eingabeschicht beträgt daher $2 \cdot (low + high)$ und die der Ausgabeschicht besitzt zwei Neuronen. Das

Lernen der Gewichte erfolgt durch den stochastischen BACKPROPAGATION-Algorithmus ohne Momentum (siehe Kapitel 2.3.3).

7.4.5 Entscheidungsbäume (DTREE)

DTREE ist eine Umsetzung des Regressionsverfahrens, wie es in Kapitel 2.4 beschrieben wird. Analog zu den vorherigen Verfahren existieren die Parameter *low*, *high* und *pol*. Für jede Positionskomponente gibt es ein unabhängiges Modell und die Ausgabe jedes Modells ist wieder die Differenz, die auf die alte Position addiert werden muss. DTREE benötigt sonst keine weiteren Parameter, die die Qualität oder Art und Weise des Lernens beeinflussen würden.

7.4.6 Episodisches Gedächtnis (EPISODIC)

Dieses Verfahren nutzt den Grundgedanken des episodischen Gedächtnisses (Kapitel 3.7) und setzt das Situationskonzept, wie es in Kapitel 3.8 beschrieben ist, praktisch um. Es benutzt also nicht die Features auf den Trajektorien wie die weiter oben beschriebenen Verfahren, sondern arbeitet direkt auf den eingelesen und verarbeiteten Situationen.

Jede Situation $S = \{\boldsymbol{key}, E\}$ besitzt einen Schlüssel $\boldsymbol{key}$, der für jede mögliche Objektklasse die Anzahl von Objekten dieser Klasse in der Situation beschreibt. EPISODIC erstellt nun für jeden Schlüssel, der in den Trainingsdaten vorkommt, ein auf FLANN basierendes Modell mit den Zeilen aller passenden Ereignismatrizen E als Trainingsdaten. Zur Vorhersage wird das Modell mit dem gleichen Schlüssel wie die zu vorhersagende Situation $\tilde{S}$ abgerufen. Dieses Modell entspricht der Matrix E, dessen einzelne Zeilen den einzelnen Ereignissen in chronologischer Reihenfolge entsprechen. In E werden nun alle Zeilen $\{E_{t1}, E_{t2}, \ldots, E_{tk}\}$ gesucht, die in einem bestimmten Umkreis zum Ereignis der aktuellen Situation $\tilde{E}_t = (\tilde{e}_{t1}, \ldots, \tilde{e}_{tn})$ (mit $n = cols(\tilde{E})$) liegen. Wie groß dieser Radius ist, bestimmt sich aus dem Parameter *maxdist*, der mit der Anzahl der Elemente einer Situation multipliziert wird. Innerhalb dieser gefundenen Zeilen wird nun diejenige bestimmt, deren vorherige Zeile am ähnlichsten zu der vorherigen Zeile des aktuellen Ereignisses ist. Diese Ähnlichkeit wird über die Distanz d des BESTMAPPING-Algorithmus definiert. Ist die ähnlichste Zeile E_i gefunden, so wird die Zeile E_{i+1} die dieser ähnlichsten Zeile folgt bestimmt. Für das zu untersuchende Objekt wird nun der Spaltenindex der Positionskomponenten bestimmt, die Differenz von E_{i+1} zu E_i an diesen Spalten berechnet und die Differenz auf die aktuelle Position addiert. Daraus ergibt sich die zu vorhersagende Position.

7.5 Evaluierungsprozess

Die verschiedenen Vorhersagealgorithmen wurden auf den gleichen Trainings- und Testdaten ausgeführt. Für die Verfahren K-NN, NEURONAL und DTREE mussten sie in Trajektorienform überführt werden. Jeder Algorithmus wurde mit verschiedenen Parametern und verschiedenen Größen der Trainingsdaten ausgeführt. In der Trajektorienform wurden die Zeilen der Eingangsmatrix dafür zufällig sortiert und erst dann die ersten l

Spalten als Trainingsdaten extrahiert. Damit ist sichergestellt, dass auch bei einer größeren Trainingsmenge die Elemente der kleineren Trainingsmengen enthalten sind. Für EPISODIC konnte keine Sortierung vorgenommen werden, da hier sonst die Chronologie verloren gegangen wäre. Anstatt dessen wurde nach Erreichen von l bei der Gesamtanzahl von Ereignissen über alle Arten von Situationen das Lesen abgebrochen. Jedes Verfahren wurde auf verschieden großen Trainingsdaten trainiert. Dagegen blieben die Testdaten immer gleich. Das auf den Trainingsdaten trainierte konkrete Modell musste für alle Testdaten eine Vorhersage treffen. Die individuelle Abweichung für jedes Element der Testmenge wird durch die L2-Distanz zwischen der vorhergesagten und der tatsächlichen Position errechnet. Daraus bildet sich der Gesamtfehler für das jeweilige Modell, indem die durchschnittliche L2-Distanz über alle Elemente der Testmenge berechnet wird.

Das Lernen und die Evaluierung erfolgten in einem separaten Programm und nicht auf dem simulierten Roboter. Während die Trainingsmenge auf maximal 400000 Trajektorien reduziert wurde, waren 129090 Trajektorien in der Testmenge vorhanden. EPISODIC arbeitet nicht auf den Trajektorien, sondern auf den Ereignissen direkt. Die Anzahl der Ereignisse in der Trainingsmenge für EPISODIC wurde auf maximal 160000 begrenzt und die Testmenge enthielt immer 29076 Ereignisse. Dabei entsprechen die 29076 Ereignisse der Testmenge für EPISODIC exakt den 129090 Trajektorien aus der Testmenge für die auf Einzeltrajektorien basierenden Lernverfahren. Trainings- und Testdaten wurden in separaten Spielen aufgenommen, die Software der Agenten war aber in jedem Durchgang gleich. Jedes Verfahren wurde mit mehreren verschiedenen, manuell ausgewählten, Parametern ausgeführt.

7.6 Ergebnisse

7.6.1 Vergleichswerte von STAY und SPEED

Bevor die eigentlichen Vorhersageverfahren miteinander verglichen werden, benötigen wir die Eckdaten der Vergleichsverfahren. Dadurch kann die relative Qualität besser eingeschätzt werden. Die durchschnittliche Bewegung aller Punkte der Trainingsmengen betrug 14,214545. Dementsprechend schnitt auch die STAY-Vorhersage mit einer durchschnittlichen L2-Distanz von 14,214427 ab. Interessant ist, dass dieser Wert, der durch pure Vorhersage der alten Position als die neue entstanden ist, immer noch besser ist als das Ergebnis der Geschwindigkeitsvorhersage mit SPEED. Dieser erreichte nur eine L2-Distanz von 20,552095 pro Zeitschritt. Während in den Daten also eine gewisse Bewegung vorhanden war (die Roboter standen nicht nur auf ihren Positionen), war die Vorhersage mithilfe der Geschwindigkeit trotzdem außerordentlich schlecht.

7.6.2 K-NN

In den verschiedenen Konfigurationen mit verschiedenen Parametern für K-NN ist eine deutlich unterschiedliche Entwicklung bei der Benutzung von kartesischen Koordinaten im Vergleich zu den Polarkoordinaten zu erkennen. In Abbildung 7.1 sind die Lernkurven

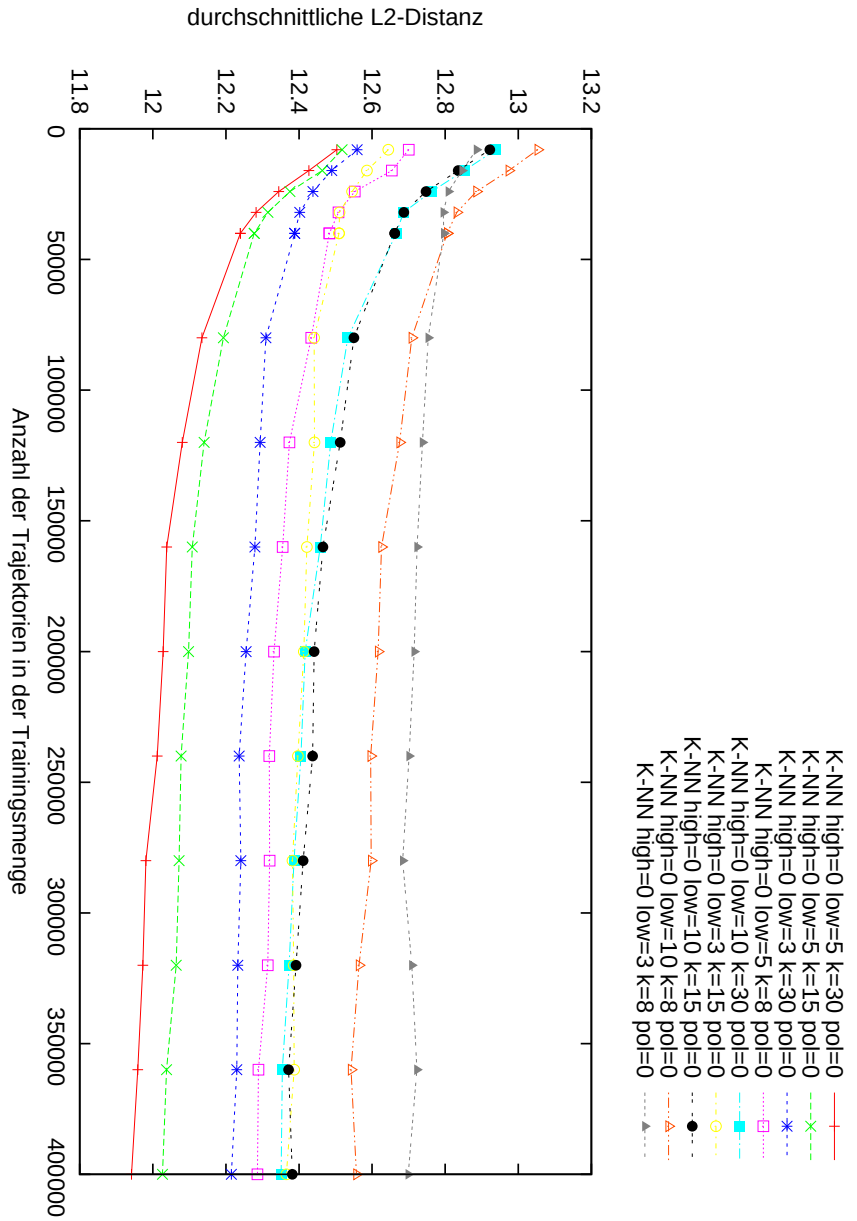


Abbildung 7.1: Qualität der Vorhersage für K-NN mit kartesischen Koordinaten und ohne Highlevel-Features.

für verschiedene Parameter und der Angabe der Position in kartesischen Koordinaten angegeben. Die beste Konfiguration an Parametern erreicht bei maximaler Trainingsmenge eine durchschnittliche L2-Distanz von 11,941597, die schlechteste 12,698690. Bei Benutzung von kartesischen Koordinaten ist der Verlauf der Lernkurven typisch, mit einem stetig abnehmenden Fehler.

Abbildung 7.2 zeigt hingegen den Verlauf der Lernkurve bei Verwendung von Polarkoordinaten. Alle Konfigurationen sind deutlich schlechter als ihr Pendant mit kartesischen Koordinaten. Zudem wird deutlich, dass bei zunehmender Anzahl an lokalen Features die Lernerfolge schlechter werden. Bei $low = 10$ ist sogar ein deutliches Overfitting zu erkennen, bei dem mit zunehmender Trainingsmenge die Fehlerrate ansteigt. Dieser Effekt erklärt sich dadurch, dass die Wertebereiche der Einzelkomponenten bei kartesischen Koordinaten gleich sind, bei Polarkoordinaten die Winkelangabe und Länge des Vektors aber zu unterschiedlichen Wertebereichen gehören. K-NN nutzt die L2-Distanz, um ähnliche Vektoren in der Wissensbasis zu finden. Die L2-Distanz ist aber nicht gewichtetet, die viel kleineren Winkelangaben tragen weniger zur Ähnlichkeit bei als die Längenangaben. Ohne Normalisierung oder Gewichtung wird K-NN also immer die Längenangabe zur Unterscheidung bevorzugen und verwirft damit die wichtige Information des Winkels. Wie bereits in Kapitel 2.2 beschrieben ist, ist k-NN zudem anfällig für höhere Dimensionen. Eine höhere Anzahl an lokalen Features führt zu einer größeren Länge des Featurevektors. Dies erklärt, warum es bei $low = 10$ zu Overfitting kommt, bei $low = 3$ aber nicht.

Die Ergebnisse unterscheiden sich also stark nach der verwendeten Positionskodierung. Für die anderen Parameter von K-NN ist es schwierig, einen dominanten Faktor für die Qualität zu bestimmen. Tendenziell führen kleinere Werte für k und höhere Werte für low zu schlechteren Ergebnissen. Die hier gefundenen optimalen Werte sind aber natürlich nur für diese Trainingsmenge optimal.

Die bisher gezeigten Ergebnisse verwenden nur lokale Features, nicht aber globale. Wie in Abbildung 7.3 zu sehen ist, führen diese bei K-NN zu keiner messbaren Verbesserung der Qualität für große Trainingsmengen.

7.6.3 NEURONAL

Bei maximaler Trainingsmenge erreicht die beste Parameterkonfiguration von NEURONAL einen durchschnittlichen Fehler von 13,562687. In dieser Konfiguration sind die Positionen als kartesische Koordinaten kodiert, es gibt keine globalen Features, 15 lokale Features und der Hidden-Layer des ANN enthält 10 Neuronen. Abbildung 7.4 zeigt die Lernkurven für weitere Parameterkonfigurationen, bei denen keine globalen Features vorkommen und kartesische Koordinaten genutzt wurden. Hier ist die Tendenz eher so, dass zu wenige lokale Features und versteckte Neuronen zu schlechteren Ergebnissen führen, die zudem nicht stetig sinken, sondern sich immer wieder zwischendurch verschlechtern.

Auch NEURONAL zeigt eine schlechtere Lernkurve, sobald Polarkoordinaten verwendet werden (Abbildung 7.5). Die Verwendung von globalen Features bringt eine leichte Verbesserung, wie in Abbildung 7.6 zu sehen ist.

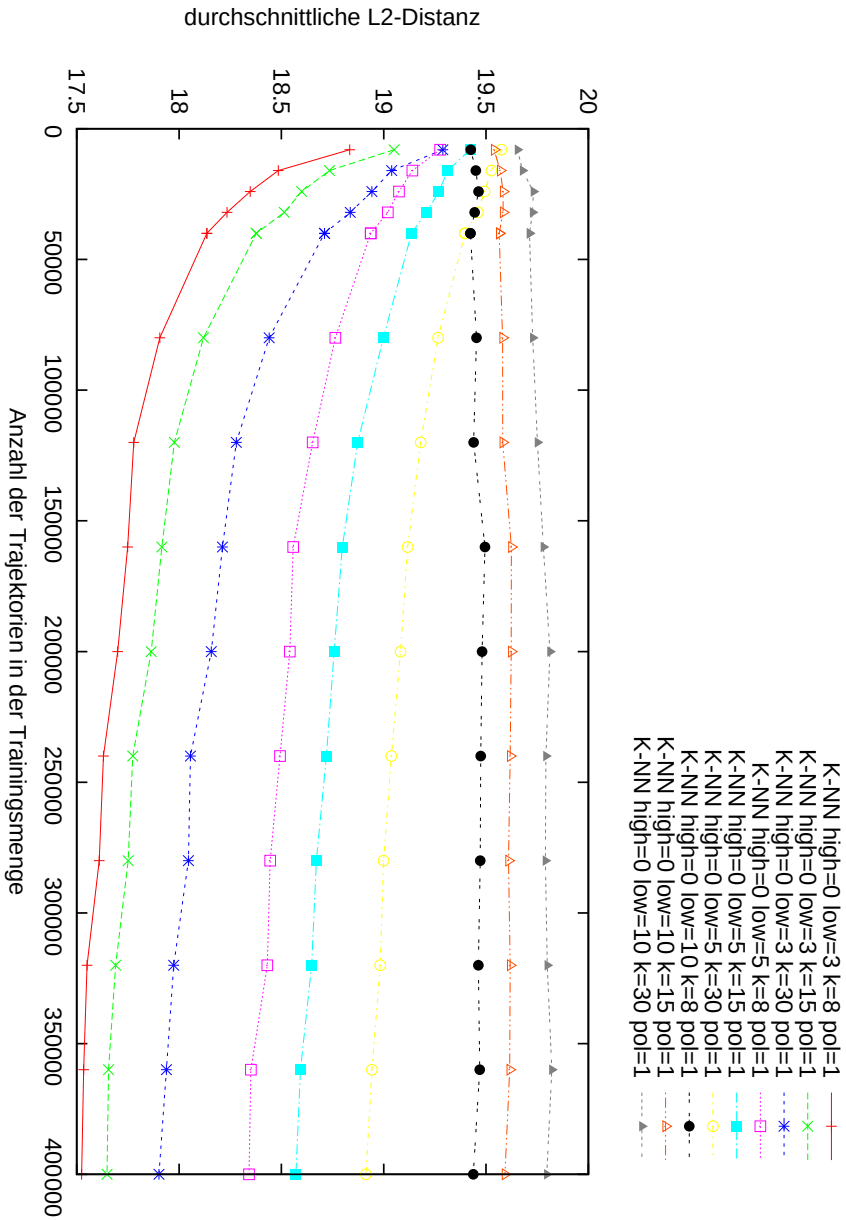


Abbildung 7.2: Qualität der Vorhersage für K-NN mit Polarkoordinaten und ohne Highlevel-Features.

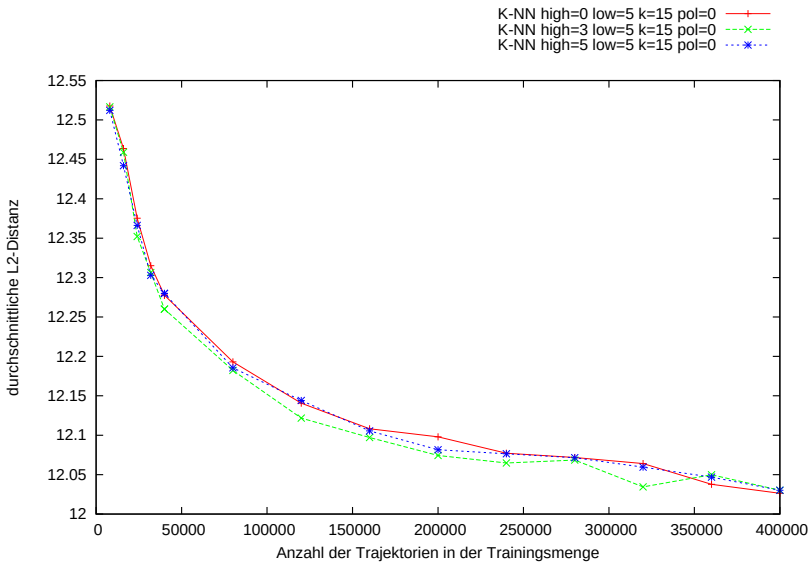


Abbildung 7.3: Vergleich der Qualität der Vorhersage für K-NN mit und ohne Highlevel-Features.

7.6.4 DTREE

In Abbildung 7.7 sind die Ergebnisse für verschiedene Werte von *low* bei Verwendung von kartesischen Koordinaten und ohne globale Features abgebildet. Der beste erreichte Wert bei maximaler Trainingsmenge ist 15,409943 und der schlechteste 15,465684. Bei DTREE ist die maximale Trainingsmenge in Vergleich zu den anderen Verfahren reduziert, da das Training von DTREE für größere Trainingsmengen zu langsam wird. Konkret wurde das Training abgebrochen, als das Lernen auf 160000 Trainingsbeispielen mehr als eine Stunde Laufzeit auf einem Server mit einem Quad-Core AMD Opteron 2,7 GHz Prozessor benötigt hat.

Die verkürzten Lernkurven für DTREE verlaufen erwartungsgemäß und in typischer Form. Auch bei DTREE bewirkt die Verwendung von globalen Features eine leichte Verbesserung der Qualität (siehe Abbildung 7.8). Dabei steigt der Qualitätsunterschied an, je größer die Trainingsdatenbasis ist.

Die Verschlechterung bei Nutzung von Polarkoordinaten zeigt sich ebenfalls bei der DTREE-Vorhersage. Auch DTREE nutzt die L2-Distanz, um beim Lernen die Beispiele in gleichartige Mengen zu unterteilen. Daher wird auch DTREE wie K-NN die Winkelwerte der Vektoren weniger einbeziehen als die Länge.

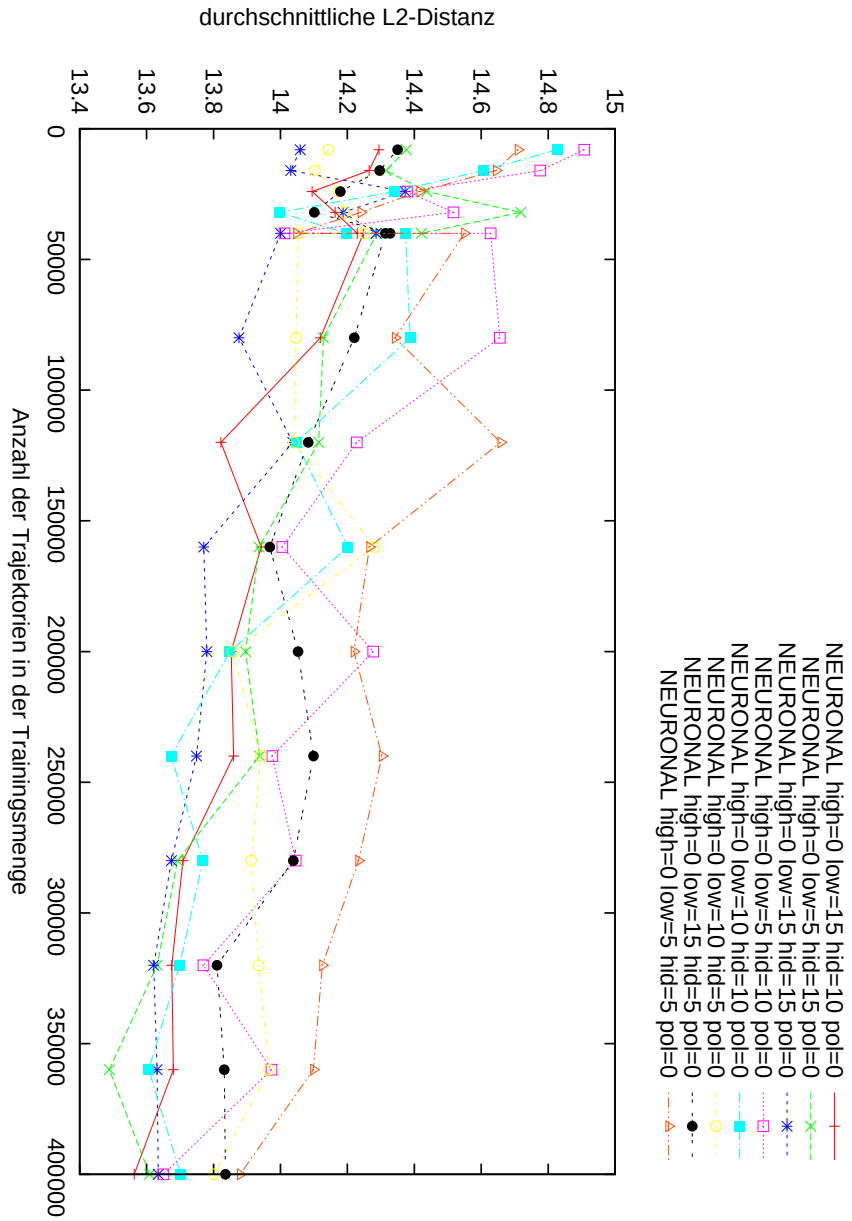


Abbildung 7.4: Qualität der Vorhersage für NEURONAL mit kartesischen Koordinaten und ohne Highlevel-Features.

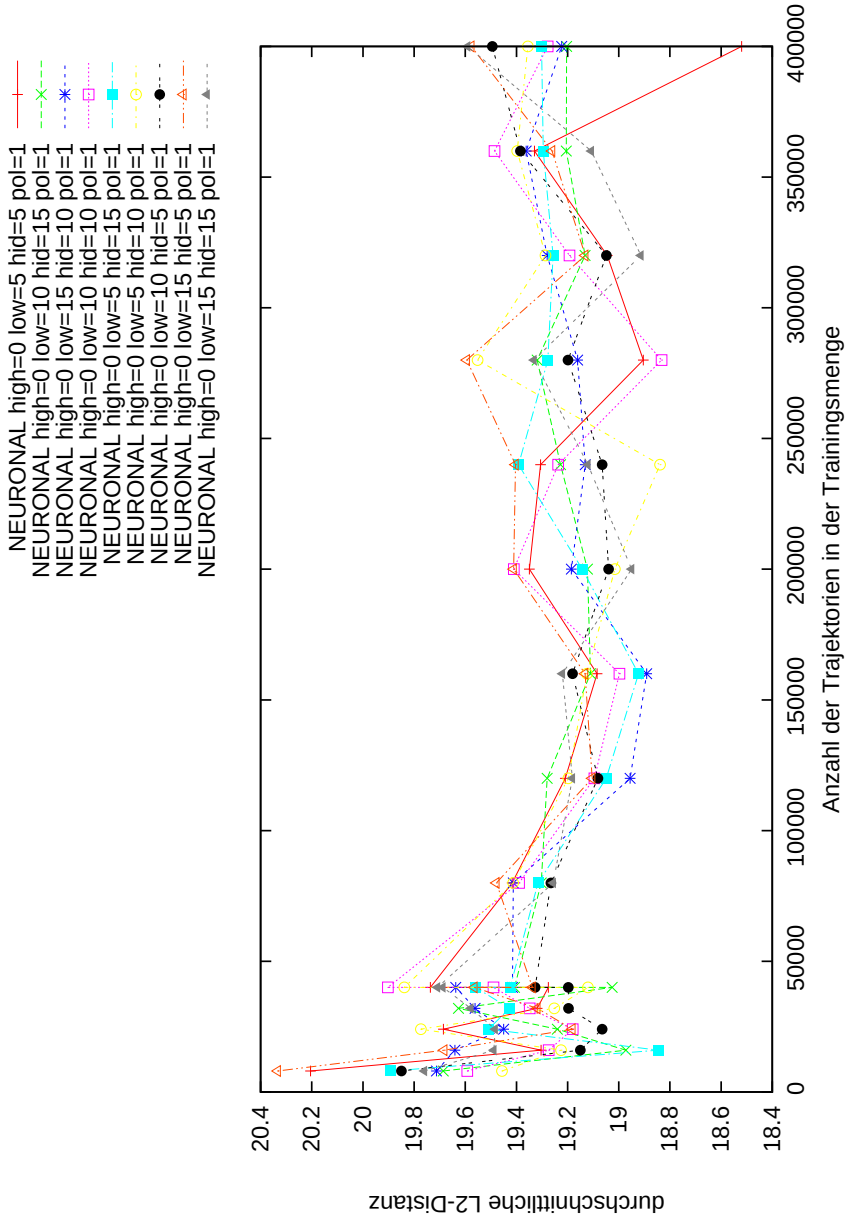


Abbildung 7.5: Qualität der Vorhersage für NEURONAL mit Polarkoordinaten und ohne Highlevel-Features.

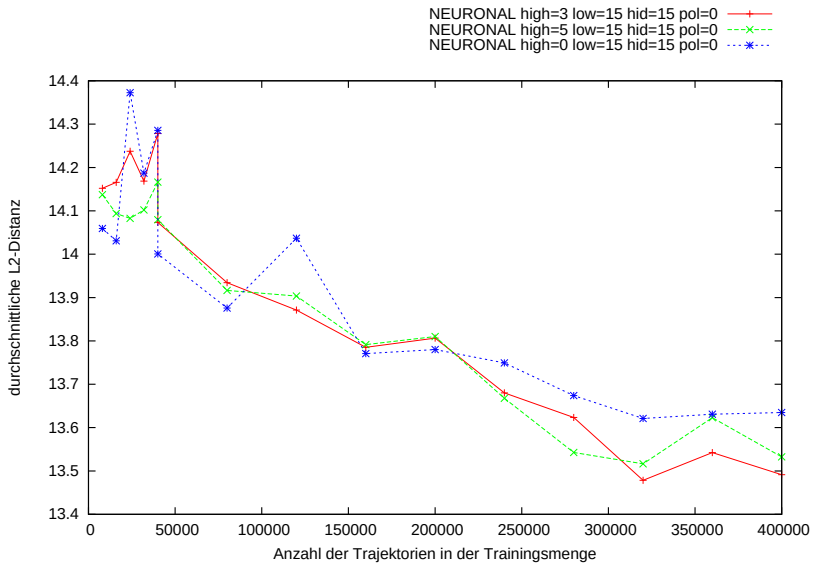


Abbildung 7.6: Vergleich der Qualität der Vorhersage für NEURONAL mit und ohne Highlevel-Features.

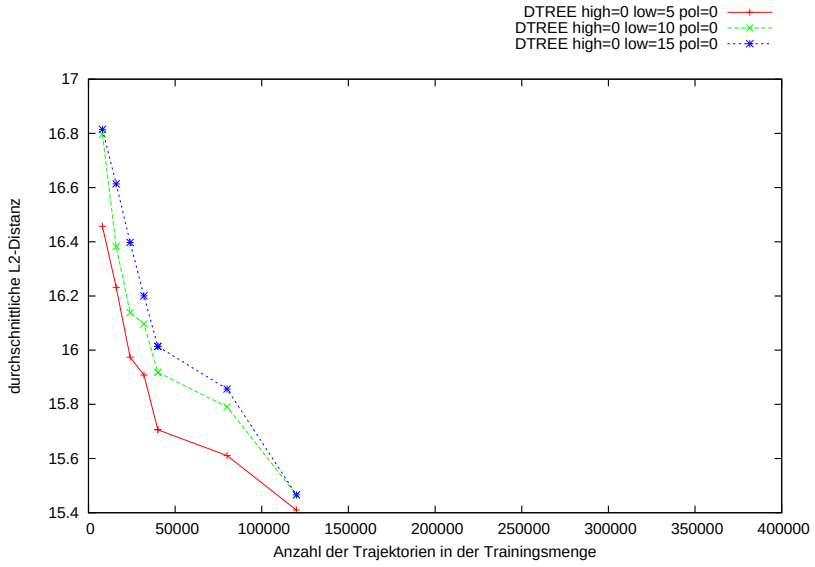


Abbildung 7.7: Qualität der Vorhersage für DTREE mit kartesischen Koordinaten und ohne Highlevel-Features.

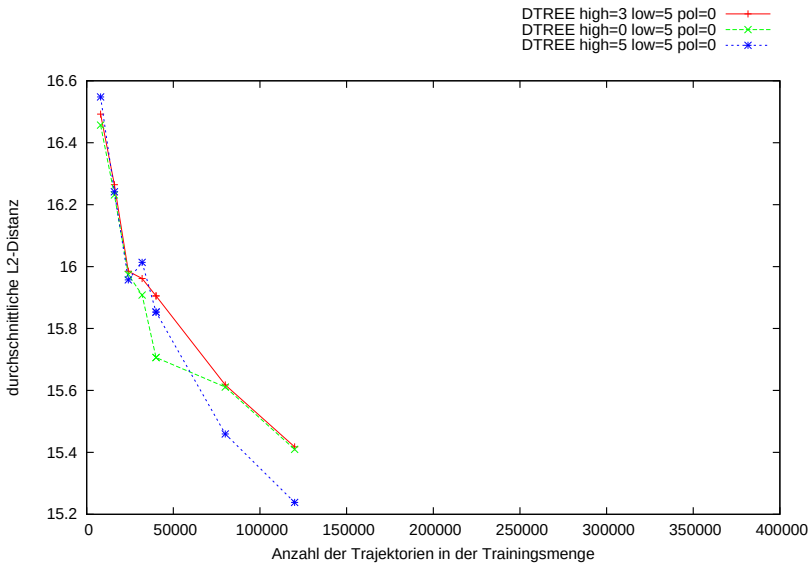


Abbildung 7.8: Vergleich der Qualität der Vorhersage für DTREE mit und ohne Highlevel-Features.

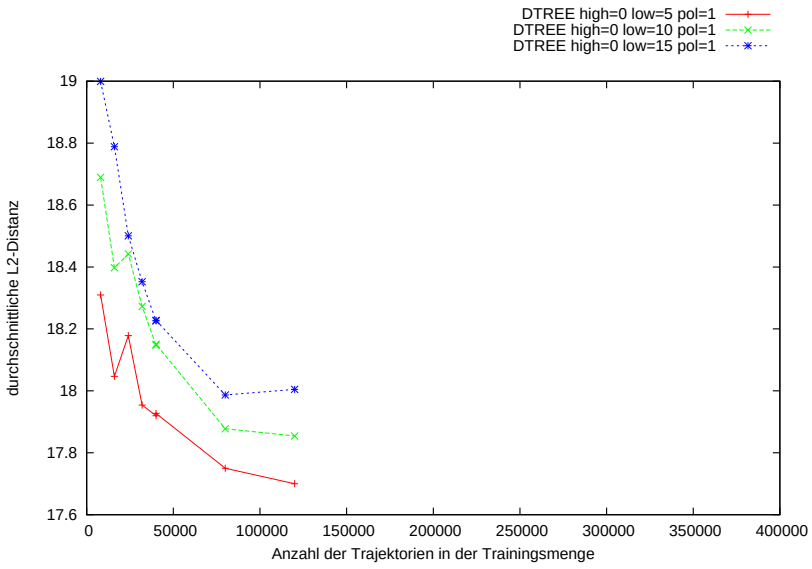


Abbildung 7.9: Qualität der Vorhersage für DTREE mit Polarkoordinaten und ohne Highlevel-Features.

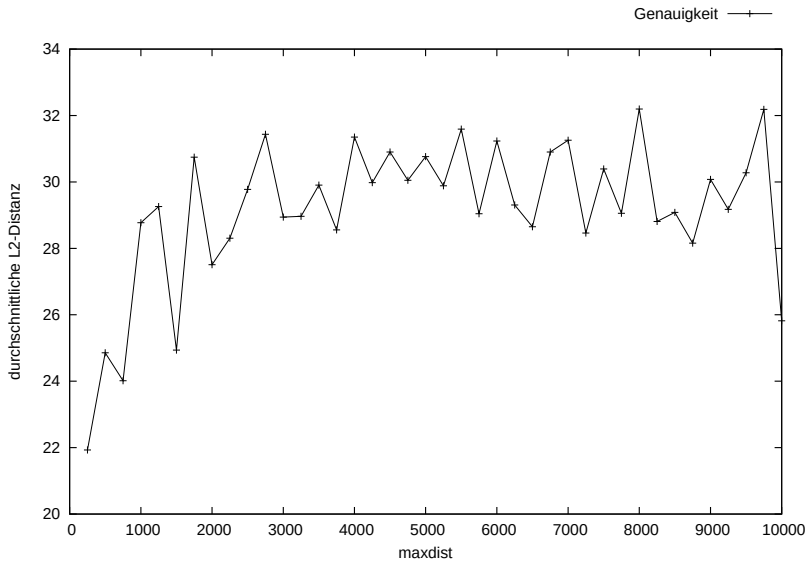


Abbildung 7.10: Qualität der Vorhersage für EPISODIC bei maximaler Trainingsmenge abhängig vom Parameter *maxdist*.

7.6.5 EPISODIC

EPISODIC unterscheidet sich von den vorherigen Verfahren dadurch, dass es in der Lage ist, Vorhersagen abzulehnen, wenn seine Datenbasis keine genaue Aussage erlaubt. Durch die verschiedenen Parameter ist es daher möglich die Genauigkeit und die Akzeptanzrate (also den Anteil der Testdaten, für die tatsächlich eine Vorhersage getroffen wurde) zu justieren. Dabei führt eine Verbesserung der Akzeptanzrate unter Umständen allerdings zu einer Verschlechterung der Genauigkeit.

In Abbildung 7.10 wird gezeigt, wie sich die Qualität der Vorhersage abhängig vom Parameter *maxdist* bei maximaler Größe der Trainingsmenge ändert. Die Fehler nimmt erst rapide zu, stagniert dann aber auf dem Niveau von etwa 30. Es besteht also kein linearer Zusammenhang zwischen maximal erlaubter Distanz der Radiussuche und der Qualität der Vorhersage. Ähnlich verhält es sich, wenn *maxdist* auf einen festen Wert gesetzt wird und die Größe der Trainingsmenge variiert wird. Dies ist in Abbildung 7.11 gezeigt. Auch hier ist der Fehler bis zu einer Größe von 80000 Samples stabil auf einem Wert von etwas über 65, sinkt dann aber rapide und stagniert bei einem Wert von in etwa 28. Verglichen mit SPEED oder STAY ist dieser Wert für die Qualität der Vorhersage schlecht. Es konnte mit EPISODIC also keine hohe Qualität in der Vorhersage erreicht werden. Zudem ist die Akzeptanzrate extrem niedrig, wie die Abbildungen 7.12 und 7.13 zeigen. Selbst bei maximaler Trainingsmenge und einer hohen maximalen Distanz wird bei höchstens 0,021% aller Testbeispiele versucht eine Vorhersage zu treffen.

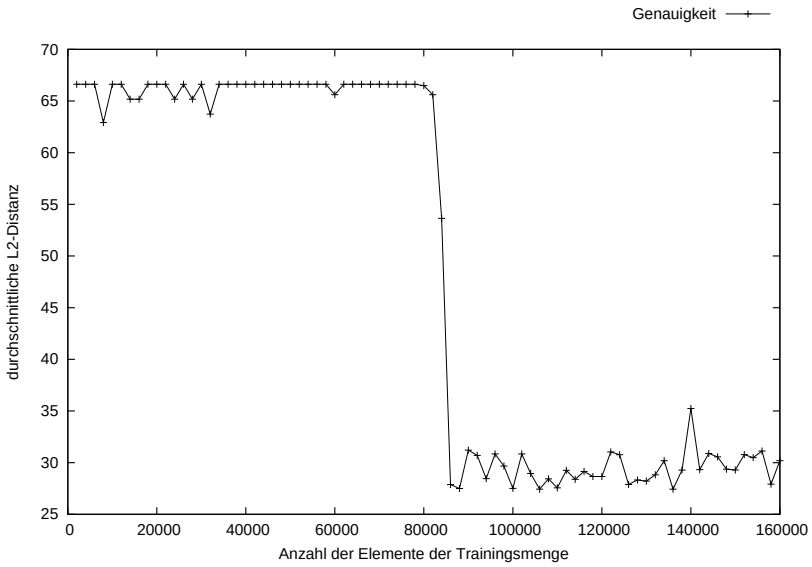


Abbildung 7.11: Qualität der Vorhersage für EPISODIC bei festem *maxdist* abhängig von der Trainingsmenge.

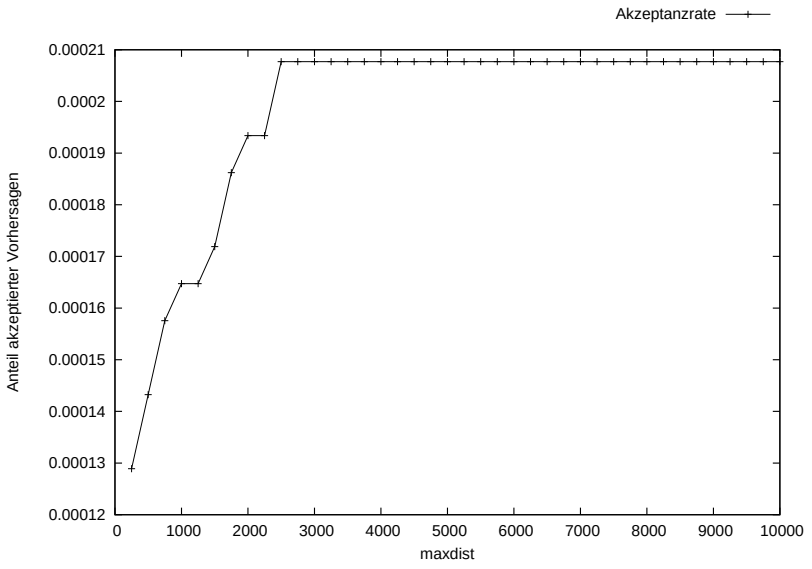


Abbildung 7.12: Akzeptanzrate der Vorhersage für EPISODIC bei maximaler Trainingsmenge abhängig vom Parameter *maxdist*.

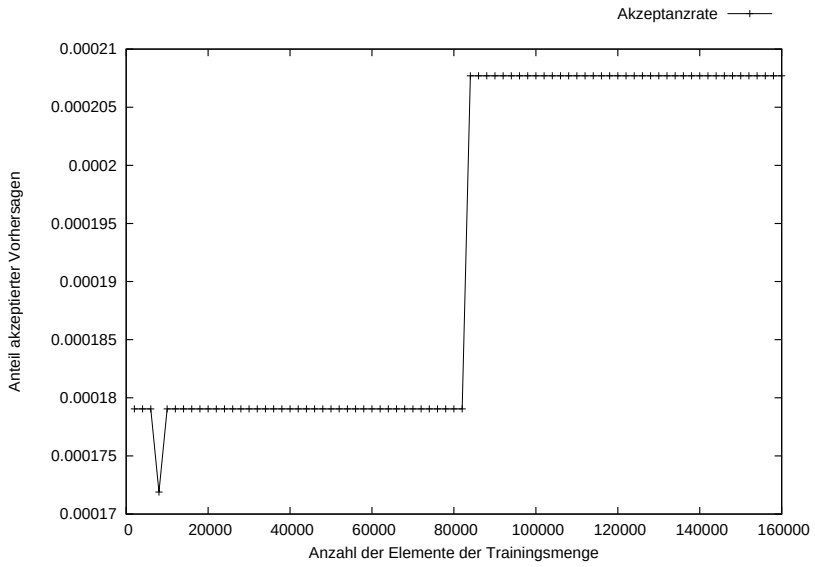
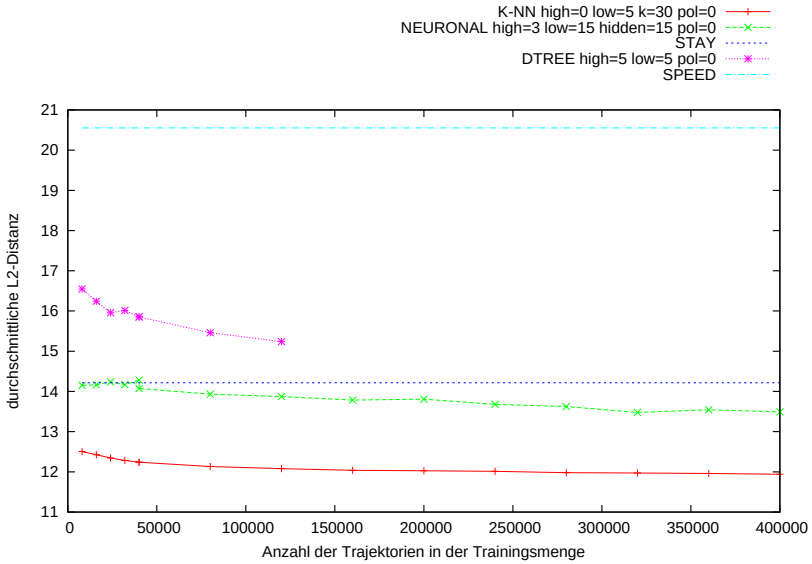


Abbildung 7.13: Akzeptanzrate der Vorhersage für EPISODIC bei festem *maxdist* abhängig von der Trainingsmenge.

7.6.6 Vergleich

Um die verschiedenen Verfahren miteinander vergleichen zu können, wurde die jeweils beste Parameterkonfiguration pro Verfahren ausgewählt und deren Lernkurve in Abbildung 7.14 zusammengestellt. Da EPISODIC eine sehr hohe Fehlerrate hat, wurde dieses Verfahren nicht in den Gesamtvergleich mit einbezogen. NEURONAL und K-NN funktionieren dagegen bei allen Testdaten nach einer ausreichend großen Trainingsmenge (in etwa 50000) sehr gut. Von allen Verfahren erziehlt K-NN bei maximaler Trainingsmenge die mit Abstand besten Ergebnisse. Die Fehler in den Vorhersagen von DTREE sind mehr als doppelt hoch wie die von NEURONAL, aber immer noch deutlich besser als die Geschwindigkeitsvorhersage SPEED. Alle Verfahren (bis auf EPISODIC) sind besser als SPEED. Da auch das STAY-Verfahren, das gar keine Entscheidung für die Vorhersage trifft, besser als die Geschwindigkeitsvorhersage ist, trifft SPEED offenbar oft falsche Entscheidungen über die Richtung.



(a) Plot der besten Verfahren

Verfahren	durchschnittliche L2-Distanz	Trainingsmenge
K-NN high=0 low=5 k=30 pol=0	11,941597	400000
NEURONAL high=3 low=15 hidden=15 pol=0	13,491418	400000
STAY	14,214427	-
DTREE high=5 low=5 pol=0	15,238244	120000
SPEED	20,552095	-

(b) Tabelle mit den durchschnittlichen L2-Distanzen für die besten Verfahren

Abbildung 7.14: Qualität der Vorhersage für die jeweils besten Parameter der Einzelverfahren.

8 Fazit und Ausblick

In dieser Arbeit konnte gezeigt werden, dass es grundsätzlich möglich ist, die kurzfristigen Laufwege von humanoiden Fußballrobotern vorherzusagen. Dabei gliedert sich das Problem in zwei Unterprobleme: der Erkennung und Verfolgung des Roboters und die Vorhersage selbst.

Das Verfahren basierend auf CenSurE zum Detektieren der Featurepunkte und den SURF-Deskriptoren zur Beschreibung der Punkte war in der Lage einzelne Roboter zu im Bild zu erkennen, die relative Position zu berechnen und die Trajektorie eines anderen Roboters zu verfolgen. Durch den allgemeinen Ansatz dieses Bilderkennungsverfahrens eignet es sich auch dazu, weitere Objekte eines Spielfeldes wie das Tor oder Linien zu erkennen. Ein Vorteil der Herangehensweise ist, dass Objekte nur einmalig annotiert werden müssen und dann auch bei wechselnden Lichtverhältnissen wiedererkannt werden können. Diese Flexibilität kommt aber zu dem Preis, dass die Laufzeit des Verfahrens vergleichsweise hoch ist und nicht mit den spezialisierten farbbasierten Methoden konkurrieren kann. Um eine ausreichende Laufzeit zu garantieren, mussten Einschränkungen in der Anzahl der erkannten Featurepunkte und deren erwarteter Position im Bild vorgenommen werden. Dies hat dazu geführt, dass es nicht möglich war, mehr als einen Roboter stabil über einen längeren Zeitraum verfolgen zu können. Die Hoffnung ist jedoch, dass auch auf den durch Batterien betriebenen autonomen Robotersystemen wie dem Nao in Zukunft deutlich schnellere Prozessoren zur Verfügung stehen werden, als es zurzeit der Fall ist. Auch der Einsatz von FPGAs (Field Programmable Gate Array) und Vektorprozessoren (also Grafikkarten) kann hier neue Impulse bieten.

Für das Testen der Vorhersage wurde eine Simulation genutzt, bei der auch die Erkennung der Roboter mit simuliert wird. Die Simulation liefert verrauschte Sensordaten, wie sie auch eine leistungsfähige Bildverarbeitung generieren würde. Obwohl die eigentliche Erkennung mehrerer Roboter auf der konkreten Platform mit dem konkreten Prozessor so noch nicht stabil möglich ist, erlaubt dieser Ansatz es, das Problem der Vorhersage eingebettet in einen größeren Kontext und unter realitätsnahen Bedingungen zu testen.

Zudem war das Verhalten der Roboter, deren Bewegung vorausgesagt werden sollte, ein reales Verhalten eines realen RoboCup-Teams. Die Agenten haben also sehr komplexe Entscheidungen basierend auf der ebenfalls komplexen Umwelt getroffen. Trotzdem war es möglich, eine gute Vorhersage über die nächste Position zu erreichen. Die dabei verwendeten erfolgreichen Algorithmen haben dabei keine Informationen über die eigentliche Umwelt, sondern nur über die bisherige Trajektorie des Roboters erhalten. Intuitiv könnte man annehmen, dass man ohne die Zusatzinformationen über die Umwelt und den internen Zustand des anderen Agenten niemals besser werden kann als die reine physikalische Modellierung der Bewegung des Agenten. Daher wurde zum Vergleich auch ein einfaches Geschwindigkeitsmodell über die Bewegung implementiert. Selbst ein Lernen

der Gesetzmäßigkeit, dass im Normalfall ein sich bewegender Agent die Richtung und Geschwindigkeit beibehält, wäre eine wichtige Lernleistung. Denn diese Annahme oder andere physikalische Konzepte wurden nicht in den Lernalgorithmen kodiert. Trotzdem waren sie in der Lage, sogar um ein Vielfaches bessere Ergebnisse als die Geschwindigkeitsvorhersage zu erzielen.

Wichtig war auch die Kodierung der Trajektorien in den Lerndaten. Hier hat sich gezeigt, dass lokale Features über den konkreten Verlauf einer Trajektorie über einen bestimmten und vergleichsweise kleinen Zeitraum ausreichend für gute Vorhersagen sind. Die lokalen Features ermöglichten durch die Verwendung der Differenz eine gute Generalisierung. Es hat sich aber auch gezeigt, dass unterschiedliche Kodierung der gleichen Information zu unterschiedlichen Ergebnissen führen kann. So ist die Darstellung der Differenzvektoren in Polarkoordinaten im Vergleich zu den kartesischen Koordinaten schlechter. Die Lösung, um diesen Unterschied aufzuheben, wäre entweder eine Normierung, die die Winkelinformation und die Distanz in einen gleichen Wertebereich bringt, oder aber die Verwendung anderer Distanzmaße als L2 (bzw. eine Gewichtung). Folgt man dem Prinzip der Sparsamkeit („Occam's razor"), so zeigt der Erfolg bei der Benutzung von kartesischen Koordinaten, dass die Verwendung dieser zusätzlichen Normierungen und Distanzmaße nicht zwingend notwendig ist.

Der Versuch, die Vorhersage durch globale Features auf der Gesamttrajektorie zu verbessern, brachte keine wesentliche Verbesserung der Vorhersagequalität. Durch Verwendung der PAA sollte eigentlich die Charakteristik der Trajektorie über einen längeren Zeitraum ausgedrückt werden. Die dadurch erreichten Verbesserungen waren aber entweder nicht messbar oder sehr klein. Hier könnten weitere Versuche mit anderen Arten von Features auf der Gesamttrajektorie durchgeführt werden. Ein Misserfolg war die Benutzung des Lernverfahrens, das am episodischen Gedächtnis angelehnt war. Die Nutzung eines Situationskonzeptes, das den Zustand der Umwelt als Ganzes kodiert, verspricht zwar mehr Informationen, die Generalisierung hat darunter aber gelitten. Um dieses Konzept zum Erfolg zu führen, müssten mehr Lerndaten gesammelt und das Finden von ähnlichen Situationen überdacht und überarbeitet werden. Die Notwendigkeit, mehr ähnliche Beispiele im episodischen Gedächtnis zu finden wird durch die extrem niedrigen Akzeptanzraten des hier genutzten Verfahrens deutlich.

Interessant ist der Vergleich zwischen den verschiedenen Lernmethoden. Die besten Vorhersagen wurden durch ein k-NN-Verfahren erreicht, gefolgt von künstlichen neuronalen Netzen. Während ANN prinzipbedingt eine sehr gute Laufzeit hat, so ist die Laufzeit von k-NN traditionell bei großen Datenmengen ein Problem. Hier wurde allerdings das Approximationsverfahren FLANN eingesetzt, sodass auch k-NN trotz großer Datenmengen selbst auf leistungsschwächeren Systemen ausreichend wenig Zeit für die Vorhersage benötigt. Die Qualität hat unter der Benutzung von FLANN offensichtlich nicht gelitten. Bei k-NN sind die originalen Lerndaten selbst das Modell, während ANN ein eigenes internes Modell aufbaut. Letzteres hat Vorteile bei der Größe des Modells, Ersteres erlaubt eine größere Kontrolle über das, was gelernt werden soll. So könnten Lerndaten, die sich sehr ähneln, häufig in den Trainingsdaten auftauchen und damit kaum einen Informationsgewinn darstellen, aus der Wissensbasis entfernt werden. Zudem kann die Trainingsmenge bei k-NN zur Laufzeit, also zum Beispiel auch während

eines Spiels, mit neuem Wissen angereichert werden, ohne dafür neu auf den Trainingsdaten lernen zu müssen. Die Tatsache, dass k-NN auf dieser Art von Problemen gut funktioniert, zeigt aber auch, dass sich offenbar bestimmte Spielsituationen oft wiederholen und auch die gleiche Konsequenz im Verhalten zeigen.

Die hier gemachten Vorhersagen waren zwar erfolgreich, sagen aber immer nur die Position für den nächsten Zeitschritt voraus. Eine Erweiterung des Verfahrens wäre, iterativ ein probabilistisches Modell aufbauen, das beliebig weit in die Zukunft schauen kann. Verschiedene Alternativen mit ihren Wahrscheinlichkeiten könnten durch das k-NN-Verfahren gefunden werden und in ein Bayessches Netzwerk einfließen. Die Blätter dieses Netzes beschreiben dann die Wahrscheinlichkeiten, dass diese Position am Ende der vorhergesagten Zeitspanne erreicht wird und die einzelnen Knoten entsprechen den Zwischenschritten zu dieser Position innerhalb dieser Zeitspanne (jeweils unter der Annahme, dass dieser Teilweg bereits genommen wurde).

Durch die Experimente konnte gezeigt werden, dass erfahrungsbasiertes Lernen dazu geeignet ist auch Trajektorien vorherzusagen, die durch intententional agierende humanoide Roboter erzeugt wurden. Dies ist möglich, selbst wenn die zu beobachtenden Agenten eine Art „black box“ bilden und die Informationen über ihr Verhalten auf die Trajektorie selbst beschränkt ist. Es hat sich auch gezeigt, dass es von Vorteil ist, die Erfahrungen selbst direkt als Modell zu nutzen und dass die Überführung in ein abstrakteres Modell nicht zwingend notwendig ist.

Abkürzungen

ANN Künstliches („Artificial“) Neuronales Netzwerk

CenSurE Center Surround Extrema

CIE Commission internationale de l’éclairage (Internationale Beleuchtungskomission)

FLANN Fast approximate nearest neighbors with automatic algorithm configuration

FPGA Field Programmable Gate Array

KI Künstliche Intelligenz

k-NN k-Nearest-Neighbor, k nächste Nachbarn

MDL Minimum-Description-Length

PAA Piecewise Aggregate Approximation

SIFT Scale-invariant feature transform

SPA Sense-Plan-Act

SPL Standard-Plattform-Liga

STA Sense-Think-Act

SURF Speeded Up Robust Features

Literaturverzeichnis

[1] Agrawal, Motilal ; Konolige, Kurt ; Blas, Morten: CenSurE: Center Surround Extremas for Realtime Feature Detection and Matching. In: Forsyth, David (Hrsg.) ; Torr, Philip (Hrsg.) ; Zisserman, Andrew (Hrsg.): *Computer Vision – ECCV 2008* Bd. 5305. Springer Berlin / Heidelberg, 2008, S. 102–115

[2] Bay, H. ; Ess, A. ; Tuytelaars, T. ; Van Gool, L.: Speeded-up robust features (SURF). In: *Computer Vision and Image Understanding* 110 (2008), Nr. 3, S. 346–359

[3] Bay, H. ; Tuytelaars, T. ; Van Gool, L.: Surf: Speeded up robust features. In: *Computer Vision–ECCV 2006* (2006), S. 404–417

[4] Billard, A. ; Epars, Y. ; Cheng, G. ; Schaal, S.: Discovering imitation strategies through categorization of multi-dimensional data. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003 (IROS 2003)* Bd. 3 IEEE, 2003. – ISBN 0780378601, S. 2398–2403

[5] Boedecker, J. ; Asada, M.: SimSpark–Concepts and Application in the RoboCup 3D Soccer Simulation League. In: *SIMPAR-2008 Workshop on The Universe of RoboCup Simulators, Venice, Italy*, 2008

[6] Bradski, G.: The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000)

[7] Bradski, G. ; Kaehler, A.: *Learning OpenCV: Computer vision with the OpenCV library*. O'Reilly Media, 2008

[8] Breiman, L.: *Classification and regression trees*. Wadsworth International Group, 1984 (Wadsworth statistics/probability series)

[9] Brockwell, P.J. ; Davis, R.A.: *Introduction to time series and forecasting*. 2nd edition. Springer Verlag, 2002

[10] Bruce, J. ; Balch, T. ; Veloso, M.: Fast and Inexpensive Color Image Segmentation for Interactive Robots. In: *2000 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2000.(IROS 2000). Proceedings* Bd. 3, 2000

[11] Burkhard, Hans-Dieter ; Asada, Minoru ; Bonarini, Andrea ; Jacoff, Adam ; Nardi, Daniele ; Riedmiller, Martin ; Sammut, Claude ; Sklar, Elizabeth ; Veloso, Manuela: RoboCup: Yesterday, Today, and Tomorrow Workshop of the

Executive Committee in Blaubeuren, October 2003. In: Polani, Daniel (Hrsg.) ; Browning, Brett (Hrsg.) ; Bonarini, Andrea (Hrsg.) ; Yoshida, Kazuo (Hrsg.): *RoboCup 2003: Robot Soccer World Cup VII* Bd. 3020. Springer Berlin / Heidelberg, 2004, S. 15–34

[12] Clarke, B. ; Fokoué, E. ; Zhang, H.H.: *Principles and theory for data mining and machine learning*. Springer Verlag, 2009. – ISBN 0387981349

[13] Cover, T. ; Hart, P.: Nearest neighbor pattern classification. In: *IEEE Transactions on Information Theory* 13 (1967), Nr. 1, S. 21–27. – ISSN 0018–9448

[14] Danis, F. S. ; Mericli, Tekin ; Mericli, Cetin ; Akin, H. L.: Robot Detection with a Cascade of Boosted Classifiers Based on Haar-like Features. In: Chown, Eric (Hrsg.) ; Matsumoto, Akihiro (Hrsg.) ; Ploeger, Paul (Hrsg.) ; Solar, Javier R. (Hrsg.) ; RoboCup Federation (Veranst.): *RoboCup 2010: Robot Soccer World Cup XIV Preproceedings, Singapore* RoboCup Federation, 2010

[15] Doherty, Martin J.: *Theory of mind: How children understand others' thoughts and feelings*. Psychology Press, 2009

[16] Ertel, Wolfgang: Maschinelles Lernen und Data Mining. In: *Grundkurs Künstliche Intelligenz*. Vieweg+Teubner, 2009. – ISBN 978–3–8348–9989–7, S. 179–242

[17] Gouaillie, David ; Hugel, Vincent ; Blazevic, Pierre ; Kilner, Chris ; Monceaux, Jérôme ; Lafourcade, Pascal ; Marnier, Brice ; Serre, Julien ; Maisonnier, Bruno: Mechatronic design of NAO humanoid. In: *2009 IEEE International Conference on Robotics and Automation*, 2009

[18] Heinemann, P. ; Sehnke, F. ; Streichert, F. ; Zell, A.: Towards a calibration-free robot: The act algorithm for automatic online color training. In: *RoboCup 2006: Robot Soccer World Cup X* (2009), S. 363–370

[19] Junejo, I.N. ; Javed, O. ; Shah, M.: Multi feature path modeling for video surveillance. In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on* Bd. 2 IEEE, S. 716–719

[20] Keogh, E.J. ; Pazzani, M.J.: Scaling up dynamic time warping for datamining applications. In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining* ACM, 2000, S. 285–289

[21] *Kapitel* Der k-d-Baum. In: Klein, Rolf: *Algorithmische Geometrie: Grundlagen, Methoden, Anwendungen*. Springer, 2005, S. 126–134

[22] Lee, J.G. ; Han, J. ; Li, X. ; Gonzalez, H.: TraClass: trajectory classification using hierarchical region-based and trajectory-based clustering. In: *Proceedings of the VLDB Endowment* 1 (2008), Nr. 1, S. 1081–1094

[23] Liu, X. ; Karimi, H.A.: Location awareness through trajectory prediction. In: *Computers, Environment and Urban Systems* 30 (2006), Nr. 6, S. 741–756

[24] Lohaus, Arnold ; Vierhaus, Marc ; Maass, Asja ; Lohaus, Arnold ; Vierhaus, Marc ; Maass, Asja: Kognition. In: *Entwicklungspsychologie des Kindes- und Jugendalters*. Springer Berlin Heidelberg, 2010. – ISBN 978–3–642–03936–2, S. 104–118

[25] Markowitsch, H.J. ; Welzer, H.: *Das autobiographische Gedächtnis: hirnorganische Grundlagen und biosoziale Entwicklung*. 2. Auflage. Stuttgart : Klett-Cotta, 2006

[26] Mellmann, Heinrich ; Xu, Yuan ; Krause, Thomas ; Holzhauer, Florian: NaoTH Software Architecture for an Autonomous Agent. In: *Proceedings of the International Workshop on Standards and Common Platforms for Robotics (SCPR 2010)*. Darmstadt, November 2010

[27] *Kapitel* Biologically Inspired Robots. In: Meyer, Jean-Arcady ; Guillot, Agnès: *Springer Handbook of Robotics*. Berlin Heidelberg : Springer-Verlag, 2008, S. 1395–1422

[28] Mitchell, Tom M. ; Liu, C.L. (Hrsg.) ; Tucker, Allen B. (Hrsg.): *Machine Learning*. McGraw-Hill, 1997 (McGraw-Hill series in computer science : Artificial intelligence). – 414 S.

[29] Muja, Marius ; Lowe, David G.: Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration. In: *International Conference on Computer Vision Theory and Application VISSAPP'09)*, INSTICC Press, 2009, S. 331–340

[30] Nixon, Mark ; Aguado, Alberto: *Feature extraction and image processing*. Academic Press, 2008

[31] OmniVision: *Advanced Information Preliminary Datasheet OV7670 CMOS VGA Camera Chip with OmniPixel Technology*. Internes Datenblatt, April 2006. – Version 1.3

[32] Poynton, Charles: *Digital video and HDTV: algorithms and interfaces*. Morgan Kaufmann, 2003

[33] Premack, David ; Woodruff, Guy: Does the chimpanzee have a theory of mind? In: *The Behavioral and Brain sciences* 1 (1978), S. 515–526

[34] Quinlan, J.R.: Induction of decision trees. In: *Machine learning* 1 (1986), Nr. 1, S. 81–106

[35] Russell, S. ; Norvig, P.: *Künstliche Intelligenz*. 2. Auflage. Pearson Studium, 2004. – ISBN 3–8273–7089–2

[36] Röfer, T. ; Jüngel, M.: Fast and robust edge-based localization in the sony four-legged robot league. In: *RoboCup 2003: Robot Soccer World Cup VII* (2004), S. 262–273

[37] Röfer, Thomas ; Laue, Tim ; Graf, Colin ; Kastner, Tobias ; Fabisch, Alexander ; Thedieck, Christian: B-Human Team Description for RoboCup 2010. In: Chown, Eric (Hrsg.) ; Matsumoto, Akihiro (Hrsg.) ; Ploeger, Paul (Hrsg.) ; Solar, Javier R. (Hrsg.) ; RoboCup Federation (Veranst.): *RoboCup 2010: Robot Soccer World Cup XIV Preproceedings, Singapore* RoboCup Federation, 2010

[38] Schlieder, Christoph ; Werner, Anke: Interpretation of Intentional Behavior in Spatial Partonomies. In: Freksa, Christian (Hrsg.) ; Brauer, Wilfried (Hrsg.) ; Habel, Christopher (Hrsg.) ; Wender, Karl (Hrsg.): *Spatial Cognition III* Bd. 2685. Springer Berlin / Heidelberg, 2003, S. 1035–1035

[39] Tulving, E.: *Elements of episodic memory.* New York: Oxford, 1983

[40] Tulving, E.: Episodic memory and autonoesis: Uniquely human. In: *The missing link in cognition: Origins of self-reflective consciousness* (2005), S. 3–56

[41] Villarreal, Mariana R.: *Complete neuron cell diagram de.* Publiziert auf Wikipedia Commons. `http://commons.wikimedia.org/wiki/File:Complete_neuron_cell_diagram_de.svg`. Version: Mai 2008

[42] Xu, Yuan ; Mellmann, Heinrich ; Burkhard, Hans-Dieter: An Approach to Close the Gap between Simulation and Real Robots. In: Ando, Noriaki (Hrsg.) ; Balakirsky, Stephen (Hrsg.) ; Hemker, Thomas (Hrsg.) ; Reggiani, Monica (Hrsg.) ; Stryk, Oskar von (Hrsg.): *2nd International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR)* Bd. 6472, Springer Berlin / Heidelberg, 2010 (Lecture Notes in Computer Science), 533-544

Abbildungsverzeichnis

Selbständigkeitserklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Berlin, den 16.08.2011 Thomas Krause