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

Sensorische und sensomotorische Karten —
ein Vergleich unüberwachter Algorithmen zur Dimensionsreduktion und Vektorquantisierung.

Diplomarbeit
zur Erlangung des akademischen Grades Diplom-Informatiker

Berlin, 8. Juni 2010

eingereicht von:	André Stephan
Gutachter:	Prof. Dr. Hans-Dieter Burkhard
	Prof. Dr. Bernd-Holger Schlingloff
Betreuer:	Dr. Manfred Hild

Zusammenfassung

Dimensionsreduktion und Vektorquantisierung sind wesentliche Methoden der Informationskomprimierung zur Bewältigung von Informationsprozessen mit den jeweils verfügbaren technischen Ressourcen.

Die vorliegende Arbeit gibt eine Übersicht über bestehende Verfahren aus dem Umfeld der unüberwachten Verarbeitung sensorischer bzw. sensomotorischer Daten. Die Diplomarbeit ist in drei Teile gegliedert, die jeweils in Theorie und Praxis unterteilt sind.

In Teil I wird der Themenbereich der Dimensionsreduktion zur Visualisierung hochdimensionaler Sensordaten bearbeitet. In der theoretischen Einführung werden verschiedene mathematisch untersetzte Verfahren erläutert. Diese werden anschließend auf 37-dimensionale Bewegungsdaten eines humanoiden Roboters angewendet. Die mehrdimensionalen Bewegungsdaten werden auf zwei Dimensionen reduziert und dann anhand von ausgewählten Gütekriterien verglichen und analysiert.

Teil II behandelt die neuronalen Gase im Rahmen der künstlichen neuronalen Netze. Zunächst werden verschiedene Algorithmen des unüberwachten vektorbasierten neuronalen Lernens theoretisch analysiert. Danach werden die sensorischen Karten als Ergebnis dieser Algorithmen, basierend auf konkreten Bewegungsdaten des Roboters, hinsichtlich ihrer Qualität miteinander verglichen.

In Teil III wird das Toussaint-Modell einer sensomotorischen Karte analysiert. Zuerst wird erläutert wie dieses Modell eine sensomotorische Karte erzeugt und anschließend zur unüberwachten und zielgerichteten Bewegungssteuerung zu beliebigen Punkten verwendet werden kann. Die bisherigen theoretischen Ergebnisse von Toussaint werden im praktischen Teil durch die Resultate einer implementierten unüberwachten Steuerung des Roboterarmes erweitert. Dabei werden eine trainierte sensomotorische Karte und die daraus resultierenden Bewegungssteuerungen des Roboterarmes zu zufälligen Punkten untersucht.

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung und Motivation

Die moderne Robotik weist eine große Vielfalt in der Morphologie der Maschinen auf. Sowohl stationäre Roboter wie der weit verbreitete Industrieroboter *Programmable Universal Machine for Assembly* (PUMA) mit sechs aktiven Gelenken [Kur05] als auch mobile Roboter wie etwa der Krabbelroboter Oktavio mit 24 aktiven Gelenken [Hil07] können über eine hohe Anzahl von Sensoren verfügen. Dies hat zur Folge, dass einerseits die hochdimensionalen sensorischen Daten und die Ergebnisse weiterführender Verarbeitungsschritte für den Menschen nicht mehr visualisiert werden können. Andererseits sinkt die Güte von nachfolgenden Datenanalysen für eine gegebene Menge an Datenpunkten ab einer gewissen Dimensionsgröße. Dieses Phänomen wird auch als „Fluch der Dimensionalität“ bezeichnet [Olg06].

Für die Lösung dieses Problemkomplexes und speziell das Visualisieren von verschiedenen Roboterposen und zugehörigen Sensordaten ist die Dimensionsreduktion von besonderer Bedeutung. Zur Dimensionsreduktion können überwachte wie auch unüberwachte Algorithmen eingesetzt werden.

Die Einteilung von Algorithmen hinsichtlich der Merkmale „überwacht“ und „unüberwacht“ erfolgt in Abhängigkeit von der Kenntnis und Verwertung der Qualität der eingehenden und/oder der zu berechnenden Daten. In dem Falle, dass Algorithmen für die eingehenden Daten keine zusätzlichen Informationen verwenden, spricht man von unüberwachten Algorithmen. Darüber hinaus implizieren überwachte Algorithmen oder überwachte Verfahren stärker den Aspekt einer Beschränkung oder Beschränktheit als unüberwachte Algorithmen und unüberwachte Verfahren.

Um sich bei der Visualisierung von Roboterbewegungssequenzen nicht unnötig einschränken zu müssen, wurden ausschließlich unüberwachte Algorithmen zur Dimensionsreduktion gewählt.

Ein weiteres Kerngebiet der Robotik ist die unüberwachte Verarbeitung von sensomotorischen Informationen von Robotern mit dem Ziel der autonomen sinnhaften Bewegungssteuerung. Diese unüberwachte Verarbeitung wird als Lernprozess bezeichnet.

Mit Hilfe solcher Lernprozesse können gegenwärtige zielorientierte Bewegungssteuerungen, die einen komplexen Überwachungsprozess beinhalten, so verändert werden, dass eine Kontrolle oder ein Eingreifen in diese Bewegungssteuerung durch

den Menschen oder andere Systeme nicht mehr erforderlich sind. Solche gängigen Bewegungssteuerungen sind z. B. die Transitionen zwischen festgelegten Körperpositionen, den *Keyframes*, oder die Transitionen zwischen variablen Körperpositionen mit Hilfe der inversen Kinematik [Bur07].

Im Zusammenhang mit komplexen nichtlinearen Roboterbewegungen müssen in beiden Varianten die wesentlichen Informationen über den Konfigurationsraum (alle erreichbaren Winkelpositionen der aktiven Gelenke) und den Arbeitsraum (Bewegungsraum) des Roboters sowie die notwendigen Daten zur Motoransteuerung aufwendig zugeführt werden.

Spezielle neuronale Lernverfahren ermöglichen es einem Roboter, seinen Konfigurationsraum zu explorieren und dabei ein kompaktes Wissen über die Grenzen des Konfigurationsraumes und die dabei getätigten Motoransteuerungen in Form von sensomotorischen Karten zu entwickeln und abzuspeichern.

Ein Verfahren zur komprimierten Beschreibung des Konfigurationsraumes durch Analyse von Datenballungszentren ist die Vektorquantisierung. Um den Lernprozess möglichst wenig zu begrenzen und weitgehendst unvorhersehbare Lernergebnisse zu erzielen, wurden in dieser Arbeit nur die unüberwachten Algorithmen zur Vektorquantisierung betrachtet.

Den Lernprozess als einen permanenten Prozess mit potentiell unendlicher Folge von Dateneingaben zu betrachten, führt zu dem logischen Schluss, für die Vektorquantisierung unüberwachte Algorithmen einzusetzen, die insbesondere auch unendliche Datenströme verarbeiten können. Solche Verfahren werden als „Online“-Lernverfahren bezeichnet.

Wenn sich im Folgenden in dieser Arbeit auf Lernverfahren bezogen wird, sind damit ausschließlich „Online“-Lernverfahren gemeint.

Diese Lernverfahren bilden eine freie ungebundene neuronale Netzstruktur aus, die sich optimal an die Eingabedatenverteilung anpasst. Die Ungebundenheit der Struktur und die Art und Weise der Ausbreitung des neuronalen Netzes ähneln denen von Gasen. Aufgrund dieser Ähnlichkeit werden diese speziellen Netzstrukturen als „Neuronale Gas“-Netzwerke und dementsprechend die zugehörigen Lernverfahren als Neuronale Gase bezeichnet [MS91].

Das Netzwerk eines neuronalen Gases, das anhand sensorischer Daten ausgebildet wurde, wird aufgrund der räumlichen Ausprägung seiner Struktur auch als sensorische Karte bezeichnet. Werden neben den sensorischen auch motorische Daten im

neuronalen Netzwerk gespeichert, spricht man in diesem Spezialfall von sensomotorischen Karten. Eine solche erlernte sensomotorische Karte ermöglicht es dem Roboter, sich völlig autonom über seinen Konfigurationsraum zielgerichtet im Arbeitsraum zu bewegen.

1.2 Zielstellung

Neuere nichtlineare unüberwachte dimensionsreduzierende Algorithmen ([TdSL00], [RS00]) können im hochdimensionalen Datensatz nichtlineare Unterräume (mit geringerer Dimensionalität) errechnen, auf denen die Daten des Datensatzes verteilt sind.

Ein Beispiel eines solchen Unterraumes ist die zweidimensionale Oberfläche der Erde. Die Oberfläche der Erde ist dreidimensional angeordnet, kann aber für sehr viele praktische Anwendungen mit einem zu vernachlässigenden Informationsverlust in eine zweidimensionale Weltkarte transformiert werden.

Das erste Ziel dieser Arbeit ist das Erstellen und Untersuchen zweidimensionaler linearer und nichtlinearer Abbildungen mit Hilfe von unüberwachten dimensionsreduzierenden Algorithmen zur Visualisierung mehrdimensionaler Sensordaten. Für diese Arbeit werden die benötigten Sensordaten verschiedenen Posen eines humanoiden Roboters im Rahmen verschiedener Bewegungsabläufe entnommen.

In wissenschaftlichen Arbeiten wurden diverse Algorithmen für die unterschiedlichsten Lernverfahren erarbeitet. Bisher wurden diese Algorithmen aber noch nicht hinsichtlich ihrer Anwendbarkeit auf sensorische Karten von humanoiden Robotern der Humboldt Universität zu Berlin überprüft.

Das zweite Ziel ist der Vergleich ausgewählter Lernverfahren. In dieser Arbeit sollen die Algorithmen neuronaler Gase hinsichtlich der Qualität der zugehörigen sensorischen Karten auf Grundlage von Sensordaten einer humanoiden Roboterbewegung verglichen werden.

Basierend auf den neuronalen Gasen hat Marc Toussaint einen neuen dynamischen Ansatz entwickelt [Tou06], der während einer zufällig angesteuerten Punktbewegung eine sensomotorische Karte ausbildet. Danach können von diesem Modell auf der Grundlage der erlernten sensomotorischen Karte zufällige Orte im Konfigurationsraum gezielt angesteuert werden.

Das dritte Ziel der vorliegenden Arbeit ist die Ermittlung der Eignung des Modells einer sensomotorischen Karte von Marc Toussaint für die Ansteuerung zweier

Motoren eines humanoiden Roboterarmes.

1.3 Aufbau der Arbeit

Zentrales Thema der Diplomarbeit ist das Studium neuronaler Gase unter verschiedenen Aspekten mit dem Schwerpunkt auf wachsende neuronale Netzwerkstrukturen.

Die theoretischen Erkenntnisse diese Studiums werden an einem konkreten humanoiden Roboter überprüft und die praktische Anwendbarkeit der erarbeiteten Verfahren am Roboter visualisiert. Durch die Wahl dieser Hardware-Plattform für die Überprüfung der theoretischen Erkenntnisse stellte sich zwangsläufig das Problem der Handhabung hochdimensionaler Sensorräume. Der humanoide Roboter besitzt 21 aktive Gelenke und acht zweiachsige Beschleunigungssensoren. Zu jedem Zeitpunkt ist eine Pose des Roboters in seinem Arbeitsraum durch 37 Sensorwerte bestimmt.

Daher widmet sich Teil I der Dimensionsreduktion. Zur Überprüfung der Anwendbarkeit der gewählten dimensionsreduzierenden Algorithmen wurden die Sensordaten verschiedener Bewegungssequenzen des humanoiden Roboters gewählt. Teil II hat die Algorithmen der neuronalen Gase und die Visualisierung der Arbeitsweise dieser im zweidimensionalen Raum zum Inhalt. In praktischer Anwendung des in Teil II erlangten Wissens wird in Teil III das Toussaint-Modell einer sensomotorischen Karte für die Ansteuerung zweier Motoren eines humanoiden Roboterarmes untersucht.

Zum Gegenstand der einzelnen Kapitel:

- Im Kapitel 2 werden die wichtigsten Grundlagen dimensionsreduzierender Algorithmen dargelegt und ausgewählte Algorithmen im Detail diskutiert.

- Kapitel 3 befasst sich mit der Ermittlung der Qualität des Ergebnisses der Dimensionsreduktion anhand zweier Gütebestimmungsverfahren.

- Im Kapitel 4 werden die in Scilab implementierten dimensionsreduzierenden Algorithmen bezüglich ihrer Ausführungsgeschwindigkeit untersucht. Dazu werden dreidimensionale Diagramme erarbeitet, die den Zeitbedarf der rechentechnischen Abarbeitung der jeweiligen Algorithmen in Abhängigkeiten vom Eingabedatenvolumen sowie diversen Eingabeparametern visualisieren.

- Das Kapitel 5 thematisiert die Experimente und Resultate der Datenvisualisierung auf der Basis sensorischer Daten einer 37-dimensionalen Bewegungssequenz des humanoiden Roboters. In diesem Zusammenhang wird eingangs

die Notwendigkeit und Sinnhaftigkeit der Filterung des Datensatzes zur gezielten Reduzierung des Datenaufkommens mit Hilfe unterschiedlicher Grafiken unterlegt.

- Kapitel 6 fasst die Ergebnisse des gesamten ersten Schwerpunktes dieser Arbeit zusammen.
- Das Kapitel 7 ordnet die vektorbasierten neuronalen Netze in das große Umfeld der künstlichen neuronalen Netze ein.
- Im Kapitel 8 werden die in der vorliegenden Arbeit verwandten wissenschaftlichen Schriften kurz umrissen und die wichtigsten Begriffe und Ziele vektorbasierter neuronaler Netzwerke erörtert.
- Das Kapitel 9 legt die mathematischen Beschreibungen der Algorithmen neuronaler Gase und die Ergebnisse der Anwendung der jeweiligen Algorithmen auf verschiedene Eingabedatenverteilungen ausführlich dar.
- Im Kapitel 10 werden zur praktischer Umsetzung die Resultate der Anwendungen der Algorithmen neuronaler Gase auf die sensorischen Daten der Roboterbewegungssequenz „Stehen-Knien-Stehen“ diskutiert.
- Kapitel 11 fasst die Ergebnisse des gesamten zweiten Schwerpunktes dieser Arbeit zusammen.
- Im Kapitel 12 werden zur Einführung in den dritten Schwerpunkt dieser Arbeit die wesentlichen Grundlagen des Toussaint-Modells einer sensomotorischen Karte detailliert ausgeführt und mathematisch unterlegt.
- Das Kapitel 13 dokumentiert umfangreich die praktische Anwendung des Toussaint-Modells auf einen humanoiden Roboterarm zum Erlernen zielgerichteter Bewegungen innerhalb seines Arbeitsraumes.
- Im Kapitel 14 werden die Ergebnisse des gesamten dritten Schwerpunktes dieser Arbeit zusammengefasst.
- Den Abschluss bildet das Kapitel 15. In ihm werden die Kernaussagen der vorliegenden Arbeit im Extrakt als Fazit formuliert und Anregungen zur weiteren Bearbeitung von ausgewählten, aus dieser Arbeit resultierenden interessanten Problemstellungen gegeben.

1.4 Notation

In dieser Arbeit wird eine in sich konsistente Schreibweise verwendet. In der unten dargestellten Tabelle sind die benutzten mathematischen Schreibformen aufgelistet.

Symbol	Bedeutung
c, C	Zwei Skalare
$\mathbf{v}$	Ein Spaltenvektor
$\mathbf{v}^T$	Ein transponierter Vektor (Zeilenvektor)
v_i	Das i-te Element des Vektors $\mathbf{v}$
$\mathbf{M}$	Eine Matrix
m_{ij}	Das Element der i-ten Zeile und j-ten Spalte der Matrix $\mathbf{M}$
$\mathbf{m}_i$	Der i-te Spaltenvektor der Matrix $\mathbf{M}$
$\mathbf{m}^i$	Der i-te Zeilenvektor der Matrix $\mathbf{M}$

Teil I

Datenvisualisierung hochdimensionaler Daten durch dimensionsreduzierende Algorithmen

2 Erläuterung ausgewählter dimensionsreduzierender Algorithmen

2.1 Grundlagen

2.1.1 Mathematische Grundlagen

Die Visualisierung hochdimensionaler Daten ist ein großer Problemkomplex im Rahmen der Datenanalyse. Die Aufgabe der Datenvisualisierung besteht darin, eine ein- bis dreidimensionale Abbildung höherdimensionaler Datenräume zu erzeugen, die dem Menschen ein Verständnis über die Datenstruktur vermittelt. Für die Dimensionsreduktion existieren zwei größere Herangehensweisen. Es werden entweder die wesentlichen Merkmale (Dimensionen) selektiert oder neue Merkmale durch Datentransformationen extrahiert. In dieser Arbeit werden ausschließlich unüberwachte Algorithmen zur Extraktion untersucht. Diese erzielen bessere Resultate, da die Gesamtinformation der Daten einbezogen wird.

In der Datenanalyse bezeichnet das Merkmal eine spezielle Eigenschaft der statistischen Einheiten (auch Merkmalsträger genannt) [CK07]. Eine experimentell bestimmte Datenreihe, bestehend aus n Datenpunkten oder statistischen Einheiten der Dimension d, kann dann in der Tabelle oder Datenmatrix X zusammengefasst werden:

i-te statistische Einheit	j-tes Merkmal: 1	2	$\cdots$	d
1	x_{11}	x_{12}	$\cdots$	x_{1d}
2	x_{21}	x_{22}	$\cdots$	x_{2d}
$\vdots$	$\vdots$	$\ddots$		$\vdots$
$\vdots$	$\vdots$		$\ddots$	$\vdots$
n	x_{n1}	x_{n2}	$\cdots$	x_{nd}

und

$$\mathbf{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1d} \\ x_{21} & x_{22} & \cdots & x_{2d} \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ x_{n1} & x_{n2} & \cdots & x_{nd} \end{pmatrix}.$$

Die Datenpunkte sind damit als Zeilenvektoren mit x^i und die Variablen oder Merkmale als Spaltenvektoren mit x_j gegeben.

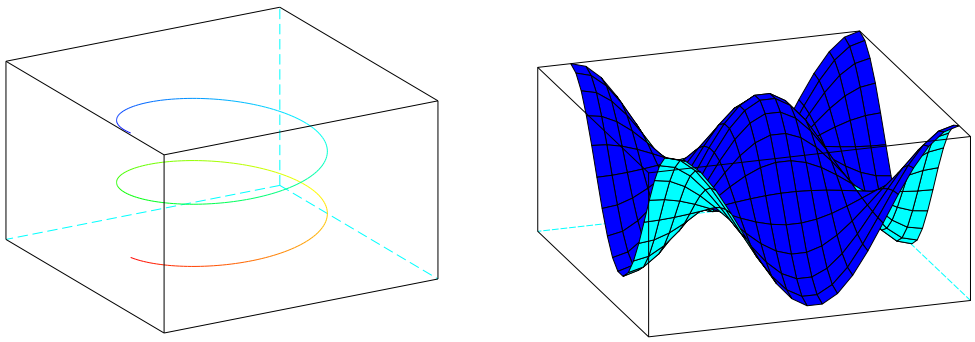
Die Klasse der Merkmalsextraktion beinhaltet lineare und nichtlineare Transformationen, d. h. dass die Eingabedatenpunkte mit Hilfe von linearen oder nichtlinearen Verfahren in ihrer Dimension reduziert werden. Beide Transformationstypen erzeugen eine Abbildung der originalen Daten in einem Raum geringerer Dimensionalität mit möglichst minimalem Informationsverlust, durch das Eliminieren redun-

danter Informationen. Dieser Raum mit geringer Dimensionalität wird als Mannigfaltigkeit bezeichnet.

Die Mannigfaltigkeit steht für ein Objekt, das in einem n-dimensionalen Raum $\mathbb{R}^n$ eingebettet ist und lokal durch den euklidischen Raum $\mathbb{R}^m$ mit $m \leq n$ modelliert werden kann [Lee00]. Eine Mannigfaltigkeit kann linear oder nichtlinear geformt sein. Sie ist genau dann linear, wenn sie auch global durch den euklidischen Raum $\mathbb{R}^m$ beschrieben werden kann. Für nichtlineare Mannigfaltigkeiten lässt sich diese globale Abhängigkeit in $\mathbb{R}^m$ nicht finden.

Beispiele für eindimensionale Mannigfaltigkeiten sind Kurven jeglicher Form. Eine Linie ist dabei der lineare Vertreter, da alle Punkte dieses eindimensionalen Objektes in einer linearen Abhängigkeit zueinander stehen. Anders ist es bei einer Schraubenlinie. Dort sind nur benachbarte Punkte annähernd linear voneinander abhängig. Deswegen ist die Schraubenlinie eine nichtlineare eindimensionale Mannigfaltigkeit.

In der Abbildung 2.1.1 sind Beispiele für ein- und zweidimensionale nichtlineare Mannigfaltigkeiten, eingebettet in $\mathbb{R}^3$, gegeben.



(a) Schraubenlinie (b) Cosinuslandschaft

Abbildung 2.1.1: *Beispiele für ein- und zweidimensionale nichtlineare Mannigfaltigkeiten.*

2.1.2 Verwandte wissenschaftliche Arbeiten

Die Grundsteine der linearen Transformation legte Pearson im Jahr 1901 mit dem Verfahren *Principal Component Analysis* (PCA) [Pea01]. Seine PCA-Technik ermöglichte die Transformation von maximal dreidimensionalen Punktwolken. Die Transformation realisierte er durch das Finden weniger neuer Koordinatenachsen, die rechtwinklig zueinander stehen und die meiste Variation der Punkte abdecken.

Im Jahre 1933 entwickelte Hotelling die Technik weiter, so dass beliebig mehrdimensionale Punktwolken in ihrer Dimension algorithmisch reduziert werden konnten [Jol86].

Eine ganze Klasse von anderen linearen Transformationen wird mit *Multidimensional Scaling* (MDS) bezeichnet. Dazu gehört das *Classical Scaling* (CS), welches im Jahr 1952 von Torgerson entwickelt wurde [CC94]. Mit dem CS-Algorithmus kann aus einer gegebenen Distanzmatrix von Punkten deren Koordinaten im euklidischen Raum berechnet werden.

Einen weiteren MDS-Algorithmus entwickelte Shepard im Jahr 1962. Seine iterative Technik versucht die Monotonie der Reihenfolge der Abstände nach der Transformation beizubehalten, jedoch versäumte es Shepard eine Gütefunktion zu definieren [Kru64a].

Im Jahr 1964 erweiterte Kruskal den MDS-Algorithmus von Shepard um eine Gütebestimmung und konnte dadurch weitere Optimierungen einbinden [Kru64a, Kru64b].

Erst 36 Jahre später wurden die ersten nichtlinearen Verfahren entdeckt. So entwickelte Tenenbaum im Jahr 2000 den Algorithmus *Isometric Feature Mapping* (ISOMAP) [TdSL00]. Er verwendet die CS-Technik mit einer angepassten Distanzmatrix. Für jeden hochdimensionalen Punkt werden alle seine Nachbardistanzen und anschließend für die restlichen Distanzen die kürzesten Pfaddistanzen gemessen.

Im gleichen Jahr wurde *Locally Linear Embedding* (LLE) als ein weiterer nichtlinearer Algorithmus von Saul und Roweis entwickelt [RS00]. Dieser kann auf die rechenintensive Bestimmung der kürzesten Pfaddistanzen in ISOMAP verzichten. Dazu wird zuerst jeder Datenpunkt als Linearkombination seiner nächstliegenden Nachbarpunkte dargestellt. Anschließend werden die neuen wenigen Koordinaten mit minimalem Konstruktionsfehler berechnet.

2.2 Linearer Algorithmus: Principal Component Analysis

Bei der *Principal Component Analysis* (PCA) wird der Datensatz $\mathbf{X}$, bestehend aus d Merkmalen bzw. Variablen $\mathbf{x}_1, \ldots, \mathbf{x}_d$ (siehe Abschnitt 2.1), in einen Datensatz mit p Variablen transformiert. Im Sinne der Dimensionsreduktion gilt: $p \leq d$. Jede dieser neuen Variablen wird auch als *Principal Component* (PC) bezeichnet. Um möglichst viel Information zu erhalten, sind die PCs unkorreliert und es werden nur die p PCs mit der größten Aussagekraft (Varianz) gewählt.

Die Berechnung der Transformation wird in den folgenden Absätzen beschrieben. Um die PCs bestimmen zu können, müssen zuerst die Varianzen (Aussagekraft) der einzelnen Variablen und eine Schätzung der Korrelation (Maß für lineare Abhängigkeit) der Variablen berechnet werden. Dafür wird die $(d \times d)$ Kovarianzmatrix $\mathbf{S}$ verwendet. Jedes Element s_{jk} enthält die Kovarianz der Variablen $\mathbf{x}_j$ und $\mathbf{x}_k$. Es gilt [Jol86]:

$$s_{jk} = \frac{1}{n-1} \sum_{i=1}^{n} (\mathbf{x}_{ij} - \bar{\mathbf{x}}_j)(\mathbf{x}_{ik} - \bar{\mathbf{x}}_k) \ .$$

Der Durchschnittsvektor $\bar{\mathbf{x}}$ ist definiert als:

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}^i \ .$$

Die Matrix $\mathbf{S}$ ist somit durch die Matrixoperation

$$\mathbf{S} = \frac{1}{n-1} \tilde{\boldsymbol{X}}^T \tilde{\boldsymbol{X}}$$

gegeben und $\tilde{\mathbf{X}}$ ist der zentrierte Datensatz:

$$\tilde{\mathbf{x}}^i = \mathbf{x}^i - \bar{\mathbf{x}} \ . \tag{2.1}$$

Demnach enthalten die Diagonaleinträge der Matrix $\mathbf{S}$ genau die Varianzen der einzelnen Merkmale $\mathbf{x}_1, \ldots, \mathbf{x}_d$ und die restlichen Einträge die Kovarianzen zweier Merkmale $\mathbf{x}_i$ und $\mathbf{x}_j$ mit $i \neq j$. Dabei gibt die Kovarianz die Tendenz einer linearen Abhängigkeit beider Variablen an, jedoch keine exakte Stärke des Zusammenhangs [CK07].
Als nächstes werden die neuen Merkmale bzw. Variablen (PCs) $\mathbf{z}_1, \ldots, \mathbf{z}_d$ extrahiert, die die größten Varianzen im ausgehenden Datensatz $\mathbf{X}$ abdecken. Für die $(n \times d)$ Matrix $\mathbf{Z}$ gilt die lineare Transformation[Jol86]:

$$\mathbf{Z} = \tilde{\mathbf{X}} \mathbf{V}_d \ .$$

Die $(d \times d)$ Matrix $\mathbf{V}_d$ mit $\mathbf{v}_1, \ldots, \mathbf{v}_d$ entspricht der Eigenvektormatrix der Kovarianzmatrix $\mathbf{S}$. Dabei wird ein Vektor $\mathbf{v}_i$ mit

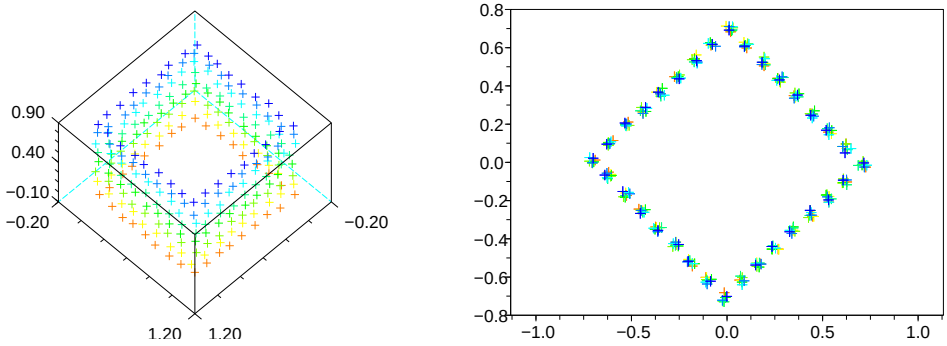
$$\mathbf{S}\mathbf{v}_i = \lambda_i \mathbf{v}_i$$

als Eigenvektor der Kovarianzmatrix $\mathbf{S}$ bezeichnet. Alle d Eigenvektoren sind zueinander orthogonal und spannen einen d-dimensionalen Raum auf. Die zugehörigen Eigenwerte $\lambda_1, \ldots, \lambda_d$ der Eigenvektoren geben dabei die Varianz der zugehörigen PCs $\mathbf{z}_1, \ldots, \mathbf{z}_d$ an.
Mit diesem Wissen kann nun eine lineare, möglichst mit geringem Informationsverlust behaftete Dimensionsreduktion vollzogen werden. Dafür werden nur die p Eigenvektoren $\mathbf{v}_1, \ldots, \mathbf{v}_p$ als $(d \times p)$ Basis $\mathbf{V}_p$ gewählt, die die p maximalen positiven Eigenwerte $\lambda_1 > \ldots > \lambda_p > 0$ besitzen. Das Resultat der Multiplikation $\tilde{\mathbf{X}}\mathbf{V}_p$ ist die $(n \times p)$ Ausgabematrix $\mathbf{Y}$, zusammengesetzt aus den p PCs mit maximaler Varianz des Ausgangsdatensatzes $\mathbf{X}$.
Die Transformation der $(n \times d)$ Datenmatrix $\mathbf{X}$ in den p-dimensionalen Raum ist durch

$$PCA(\mathbf{X}, p) = \tilde{\mathbf{X}}\mathbf{V}_p$$

gegeben.

Als Beispiel wird eine leicht verrauschte Würfeloberfläche (siehe Abbildung 2.2.1(a)) PCA-transformiert. Gut zu erkennen ist, dass die beiden PCs den Diagonalen der Grundfläche des Würfels entsprechen (siehe Abbildung 2.2.1(b)).



(a) Vor PCA in $\mathbb{R}^3$ (b) Nach PCA in $\mathbb{R}^2$

Abbildung 2.2.1: *Beispiel für eine PCA-Transformation einer verrauschten Würfeloberfläche.*

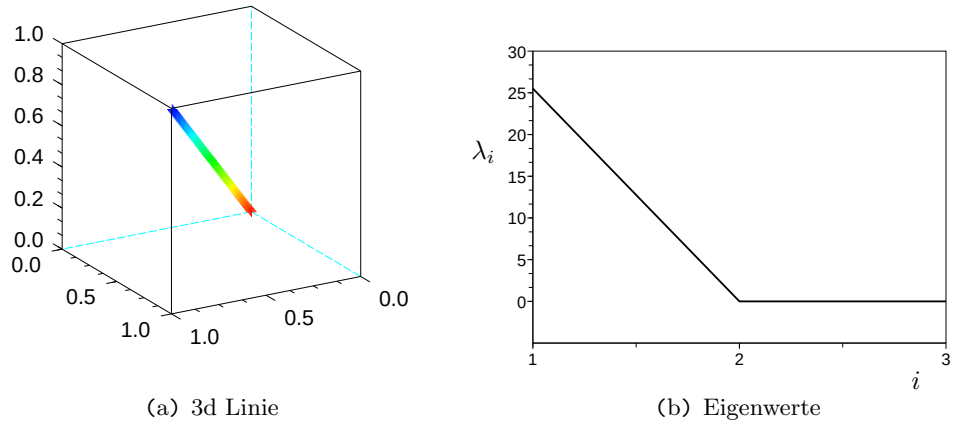
Aufgrund der Bestimmung der Kovarianzmatrix und ihrer Eigenwerte bzw. Eigenvektoren besitzt der Algorithmus die Zeitkomplexität[1] $O(nd^2 + d^3)$ [Dhi98].

[1] Die Zeitkomplexität eines Algorithmus gibt das Verhalten des Zeitverbrauchs eines rechentechnisch umgesetzten Algorithmus in Abhängigkeit der wesentlichen Kenngrößen an. Je höher die Komplexität ist, umso mehr Zeit benötigt die entsprechende Programmausführung.

Die Bestimmung der Dimensionalität m der linearen Mannigfaltigkeit, auf der die Datenpunkte liegen, ist mit der Größe der Eigenwerte der PCs verbunden. So sollen die ersten m-größten Eigenwerte eine Varianz aufsummieren, die 80% bis 90% der gesamten Varianz besitzt. Da die Summe der Varianz der Principal Components der Summe der entsprechenden Eigenwerte gleicht, kann der prozentuale Anteil der Varianz der ersten m PCs an der Gesamtvarianz über alle p PCs durch die folgende Gleichung bestimmt werden [Jol86]:

$$t_m = 100 \frac{\sum_{i=1}^{m} \lambda_i}{\sum_{j=1}^{p} \lambda_j} \ . \tag{2.2}$$

Jolliffe hat für eine geeignete Wahl der Dimensionsgröße m empfohlen, dass die Varianz der m PCs mindestens 80% der Gesamtvarianz abdecken sollte [Jol86]. Bei einer Datenpunktverteilung entlang einer Linie im dreidimensionalen Raum (siehe Abbildung 2.2.2(a)) beträgt der prozentuale Anteil der Variation der ersten PC gerundet 100% (siehe Abbildung 2.2.2(b)). Damit ergibt sich die korrekte interne Dimension von Eins.



(a) 3d Linie (b) Eigenwerte

Abbildung 2.2.2: *Eigenwerte der Kovarianzmatrix einer Linie im dreidimensionalen Raum.*

2.3 Klasse linearer Algorithmen: Multidimensional Scaling

2.3.1 Einführung

Mit *Multidimensional Scaling* (MDS) wird eine ganze Klasse von dimensionsreduzierenden Algorithmen bezeichnet, die für eine gegebene Anzahl n von Objekten deren paarweise Unterscheidungsmerkmale berücksichtigen, um diese n Objekte in einem p-dimensionalen Raum mit möglichst geringer Dimension anzuordnen. Dabei verwenden alle MDS-Algorithmen als Eingabe die Distanzen δ_{ij} aller n Objekte.

Falls die Objekte aus einem n-dimensionalen euklidischen Raum stammen, können die euklidischen Distanzen mit:

$$\delta_{ij} = \sqrt{\sum_{s=1}^{d}(x_{is} - x_{js})^2} = \|(\mathbf{x}^i)^T - (\mathbf{x}^j)^T\| \tag{2.3}$$

berechnet werden. Alle Distanzen müssen daher vor der Transformation vorliegen oder extra bestimmt werden und sind in der symmetrischen Distanzmatrix $\mathbf{\Delta}$ enthalten:

$$\mathbf{\Delta} = \begin{pmatrix} \delta_{11} & \cdots & \delta_{1n} \\ \vdots & \ddots & \vdots \\ \delta_{n1} & \cdots & \delta_{nn} \end{pmatrix}.$$

Das MDS wird in metrisches und nichtmetrisches Skalieren unterschieden. Beim metrischen Ansatz werden die Distanzen d_{ij} nach der Transformation den Distanzen δ_{ij} vor der Transformation angeglichen.

Im Gegensatz dazu versucht der nichtmetrische Ansatz ausschließlich die Ranginformationen der Distanzen vor und nach der Transformation beizubehalten. So soll möglichst für jede monotone Folge $\delta_{ij} \leq \delta_{kl} \leq \delta_{mn}$ von Distanzen δ_{ij} vor der Transformation eine monotone Folge $d_{ij} \leq d_{kl} \leq d_{mn}$ von Distanzen d_{ij} nach der Transformation existieren.

2.3.2 Linearer Algorithmus: Classical Scaling

Eine Variante des metrischen MDS ist das *Classical Scaling* (CS). Hier wird nach einer Anordnung von n Datenpunkten in einem p-dimensionalen Raum $\mathbf{Y}$ mit $p < (n-1)$ gesucht. Dabei sind nur die Distanzen δ_{ij} gegeben, die zwischen den Datenpunkten y^i und y^j angestrebt sind. Ähnlich zur PCA besteht die Zielkonfiguration

$\mathbf{Y}$ aus den p Variablen $\mathbf{y}_1, \ldots, \mathbf{y}_p$, die die größten Varianzen in $\mathbf{Y}$ abdecken. Genauer gesagt, gilt im Falle von euklidischen Distanzen δ_{ij}, dass das Ergebnis des CS gleich dem Ergebnis der PCA ist [CC94]. Die folgenden Berechnungsschritte werden von Cox und Cox übernommen [CC94].
Gesucht wird die innere $(n \times n)$ Produktmatrix $\mathbf{B}$:

$$b_{rs} = \mathbf{y}^r(\mathbf{y}^s)^T \text{ und damit } \mathbf{B} = \mathbf{Y}\mathbf{Y}^T \ , \tag{2.4}$$

bestehend aus den Skalarprodukten aller gesuchter n Datenpunkte $\mathbf{y}^r$ und $\mathbf{y}^s$ mit $(r, s = 1, \ldots, n)$ der Dimension p. Aus $\mathbf{B}$ können die p-dimensionalen Datenpunkte berechnet werden. Für die gegebenen euklidischen Distanzen δ_{rs} gilt die Abhängigkeit:

$$\begin{aligned} \delta_{rs}^2 &= (\mathbf{y}^r - \mathbf{y}^s)(\mathbf{y}^r - \mathbf{y}^s)^T \\ &= \mathbf{y}^r(\mathbf{y}^r)^T + \mathbf{y}^s(\mathbf{y}^s)^T - 2\mathbf{y}_r(\mathbf{y}^s)^T \ . \end{aligned} \tag{2.5}$$

Es wird vorausgesetzt, dass die Datenpunkte $\mathbf{y^i}$ über dem Koordinatenursprung zentriert sind mit

$$\sum_{r=1}^{n} y_{ri} = 0 \ (i = 1, \ldots, p) \ . \tag{2.6}$$

Weitere Zusammenhänge ergeben sich aus den Gleichungen 2.5 und 2.6 :

$$\frac{1}{n}\sum_{r=1}^{n} \delta_{rs}^2 = \frac{1}{n}\sum_{r=1}^{n} \mathbf{y}_r(\mathbf{y}^r)^T + \mathbf{y}_s(\mathbf{y}^s)^T \ , \tag{2.7}$$

$$\frac{1}{n}\sum_{s=1}^{n} \delta_{rs}^2 = \mathbf{y}_r(\mathbf{y}^r)^T + \frac{1}{n}\sum_{s=1}^{n} \mathbf{y}_s(\mathbf{y}^s)^T \ , \tag{2.8}$$

$$\frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} \delta_{rs}^2 = \frac{2}{n}\sum_{r=1}^{n} \mathbf{y}_r(\mathbf{y}^r)^T \ . \tag{2.9}$$

Die Gleichungen 2.4, 2.5und 2.7 bis 2.9 ergeben:

$$\begin{aligned} b_{rs} &= -\frac{1}{2}\left(\delta_{rs}^2 - \mathbf{y}_r(\mathbf{y}^r)^T - \mathbf{y}_s(\mathbf{y}^s)^T\right) \\ &= -\frac{1}{2}\left(\delta_{rs}^2 - \frac{1}{n}\sum_{r=1}^{n} \delta_{rs}^2 - \frac{1}{n}\sum_{s=1}^{n} \delta_{rs}^2 + \frac{1}{n^2}\sum_{r=1}^{n}\sum_{s=1}^{n} \delta_{rs}^2\right) \ . \end{aligned} \tag{2.10}$$

Damit lässt sich **B** durch die Matrizen **A** und **H** berechnen:

$$\mathbf{B} = \mathbf{HAH}$$

mit

$$a_{rs} = -\frac{1}{2}\delta_{rs}^2 \text{ und } \mathbf{H} = \mathbf{I} - n^{-1}\mathbf{1}\mathbf{1}^T \ .$$

Hierbei ist $\mathbf{I}$ die $(n \times n)$ Einheitsmatrix und $\mathbf{1}$ ein Vektor mit n Einsen. Für die Matrix $\mathbf{B}$ werden die p größten Eigenwerte $\lambda_1 > ... > \lambda_p > 0$ und zugehörigen Eigenvektoren $\mathbf{v}_1, ..., \mathbf{v}_p$ bestimmt. Denn es gilt für die Zielkonfiguration $\mathbf{Y}$:

$$\mathbf{Y} = \mathbf{V}\mathbf{\Lambda}^{\frac{1}{2}} \ .$$

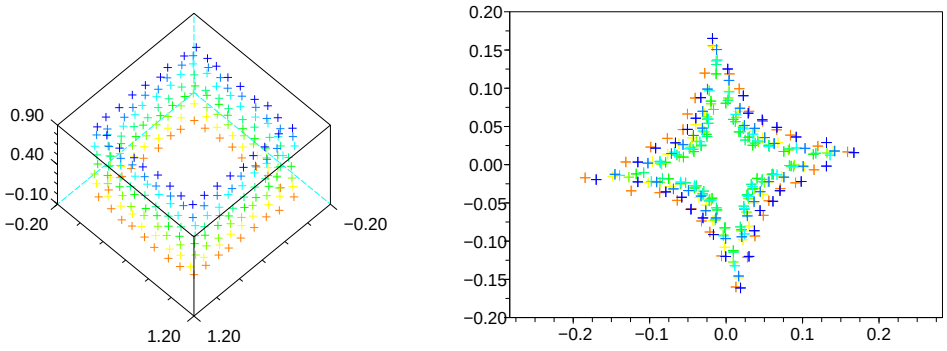
Wie bei der PCA geben die λ_i die Stärke der Varianzen (Information) entlang der Variablen $\mathbf{y}_i$ an. Damit lässt sich die Transformation zusammenfassen:

$$CS(\mathbf{\Delta}, p) = (\mathbf{v}_1, ..., \mathbf{v}_p) \begin{pmatrix} \sqrt{\lambda_1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \sqrt{\lambda_p} \end{pmatrix} \ .$$

Als Beispiel ist die CS-Transformation einer verrauschten Würfeloberfläche in der Abbildung 2.3.1 dargestellt. Für die Unähnlichkeiten wurden die kubischen euklidischen Distanzen zwischen allen Objekten i, j berechnet:

$$\delta_{ij} = (\sum_{s=1}^{t}(x_{is} - x_{js})^2)^{\frac{3}{2}}.$$

Die Zeitkomplexität des CS-Algorithmus ohne Berechnung der Distanzmatrix $\mathbf{\Delta}$ ist aufgrund der Bestimmung der größten Eigenvektoren mit $O(dn^3)$ gegeben. Genauso wie bei der PCA kann auch hier die Dimensionsgröße der Mannigfaltigkeit, auf der die Datenpunkte liegen, anhand des prozentualen Anteils der Summe der ersten p Eigenwerte an der gesamten Summe aller positiven Eigenwerte bestimmt werden (siehe Gleichung 2.2).



(a) Vor CS in $\mathbb{R}^3$ (b) Nach CS in $\mathbb{R}^2$

Abbildung 2.3.1: *Beispiel für eine metrische MDS-Transformation einer Würfeloberfläche.*

2.3.3 Linearer Algorithmus von Kruskal

Bei diesem nichtmetrischen MDS-Algorithmus soll nicht die Distanz- sondern die Ranginformation aller Objektdistanzen δ_{ij} nach der Transformation möglichst gut erhalten bleiben. Der nichtmetrische Vertreter der im Folgenden erläutert wird, ist das Kruskal-Verfahren. Darin wird ausgehend von einer initialen Startkonfiguration $\mathbf{Y}_0$ der Datenpunkte im p-dimensionalen Raum iterativ die Datenpunkte entsprechend einer Gütefunktion (Stress-Funktion) so lange verschoben, bis ein Abbruchkriterium erreicht wird. Die Stress-Funktion gibt dabei an, wie gut die aktuelle Konfiguration mit den ursprünglichen Daten δ_{ij} übereinstimmt. So ist die Berechnung durch Kruskal schematisch im Algorithmus 2.3.1 festgehalten [Kru64b].

Algorithmus 2.3.1: *Kruskal*

```
1  function Kruskal(Δ,p) {
2    Bestimme Startkonfiguration Y_0;
3    while (Abbruchbedingung gilt nicht) {
4      Normalisiere Konfiguration Y_k;
5      Berechne Distanzen d_ij;
6      Berechne Disparitäten d̂_ij;
7      Berechne Gradient G;
8      Berechne Schrittgröße α;
9      Berechne neue Konfiguration Y_{k+1};
10   }
11   Ausgabe von Konfiguration Y_k;
12 }
```

Bei der Bestimmung der Startkonfiguration $\mathbf{Y}_0$ ist darauf zu achten, dass alle

n Objekte im p-dimensionalen Raum jeweils unterschiedlichen Punkten zugewiesen werden, da sich andernfalls nicht die partielle Ableitung der Differenz berechnen ließe. Weiter dürfen die Punkte nicht in einem Raum mit kleinerer Dimension als p liegen, sonst würde jede lineare Kombination zweier Vektoren, die durch den Algorithmus berechnet wird, diesen Unterraum nicht verlassen können. Um die Berechnung in den folgenden Schritten zu vereinfachen, wird die Konfiguration $\mathbf{Y}_k$ zuvor normalisiert. Die Normalisierung nach Kruskal beinhaltet erstens das Zentrieren der Punktwolke $\mathbf{Y}_k$ mit Hilfe des Mittelwertvektors $\bar{\boldsymbol{y}}_{\boldsymbol{k}}$ analog zur Gleichung 2.1 und zweitens das Skalieren der Matrix $\mathbf{Y}_k$ mit $\frac{1}{rms}$, wobei das quadratische Mittel oder *Root Mean Square* (RMS) mit

$$rms = \sqrt{\frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n}(d_{ij})} = 1$$

berechnet wird. Die Distanzen zwischen den Objekten i, j für eine Konfiguration $\mathbf{Y}_k$ werden als d_{ij} bezeichnet und sind euklidischen Ursprungs (siehe Gleichung 2.3).

Bei den Disparitäten $\hat{d}_{ij}$ handelt es sich um eine monoton steigende Sequenz von Zahlen, die einerseits die Distanzen d_{ij} nach der Transformation und andererseits die Reihenfolge der entsprechenden sortierten Distanzen δ_{ij} vor der Transformation berücksichtigt. Die formale Definition der Disparität ist die folgende:

$$\hat{d}_{ij} = f(\delta_{ij}) \ ,$$

wobei f eine monotone Funktion ist mit [Kru64b]:

$$\delta_{ij} < \delta_{kl} \rightarrow \hat{d}_{ij} \leq \hat{d}_{kl} \ , \tag{2.11}$$

$$\delta_{ij} = \delta_{kl} \rightarrow \hat{d}_{ij} = \hat{d}_{kl} \ . \tag{2.12}$$

Kruskal beschreibt eine Methode, die die Disparitäten einer Konfiguration berechnet [Kru64b]. Dabei werden zu Beginn alle Distanzen d_{ij} entsprechend der Rangfolge der Unähnlichkeiten δ_{ij} angeordnet. In dieser Reihenfolge wird jeder d_{ij}-Wert einzeln einem Block aus einer Blockliste übertragen. Die Distanzwerte werden im Folgenden verändert und als Disparitäten bezeichnet. Die Blöcke und deren Disparitäten werden so lange berechnet, bis die Disparitäten in der Reihenfolge ihrer Blöcke eine monotone Folge bilden. Dabei wird in jedem Schritt überprüft, ob für den aktuellen Block B die Auf- und Abwärtsmonotonie für beide Nachbarblöcke gilt. Ist eine von

beiden Monotonien nicht gegeben, wird der entsprechende Nachbarblock C mit B zu B' verschmolzen. Hierbei wird die Summe $d_{B'}$ aus den bisher akkumulierten Distanzsummen d_B, d_B beider Blöcke gebildet und die neue Disparität $\hat{d}_{B'}$ berechnet. Es gilt:

$$d_{B'} = d_B + d_C \text{ und } \hat{d}_{B'} = \frac{d_{B'}}{k+l}.$$

Die Summanden k und l geben die Anzahl der bereits verschmolzenen Blöcke von B und C an. Wenn der letzte Block berechnet wurde, ist die monotone Folge von Blöcken $B_1, B_2, ..., B_i$ mit deren Disparitäten $\hat{d}_{B_1} \leq \hat{d}_{B_2} \leq ... \leq \hat{d}_{B_i}$ gegeben.

Als Beispiel sind die Distanzen vor und Disparitäten nach der Monotonisierung in Abbildung 2.3.2 zu sehen. Es handelt sich hierbei um das Streudiagramm der Kruskal-Transformation nach 100 Iterationen (siehe Abbildung 2.3.3(c)), worauf zu jeder Distanz d_{ij} die entsprechende Unähnlichkeit (Rot) und zu jeder Disparität $\hat{d}_{ij}$ die dazugehörige Unähnlichkeit (Blau) eingezeichnet wurden. Dabei ist die Monotonieeigenschaft der Gleichungen 2.11 und 2.12 entlang der blauen Kurve ersichtlich.

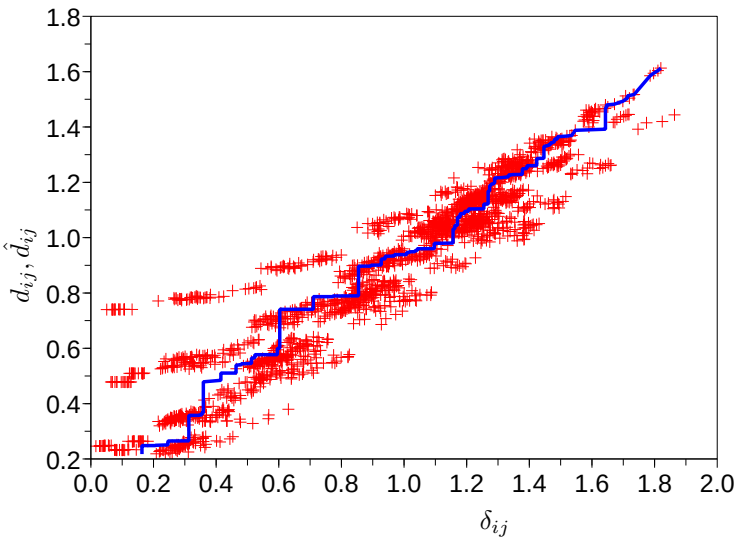


Abbildung 2.3.2: *Beispiel für ein Streudiagramm. Zu jeder Distanz δ_{ij} (vor der Transformation) werden die Distanzen d_{ij} nach der Transformation in Rot und die Disparitäten $\hat{d}_{ij}$ in Blau eingezeichnet.*

Nach der Berechnung der Disparitäten wird die Güte der aktuellen Konfiguration

bestimmt. Diese wird mit dem Stresswert

$$S = Stress(\mathbf{Y}_k)$$

quantifiziert. Kruskal definiert ihn durch [Kru64a]:

$$S^* = \sum_{i,j}(d_{ij} - \hat{d}_{ij})^2 \ ,$$

$$T^* = \sum_{i,j}(d_{ij}^2) \ ,$$

$$S = \sqrt{\frac{S^*}{T^*}} \ . \tag{2.13}$$

Ein Stresswert mit $S \geq 0.2$ gilt als nicht mehr ausreichend gute Annäherung der aktuellen Distanzen d_{ij} an die gegebenen Unähnlichkeiten δ_{ij}. Bei einer perfekten Konfiguration $\mathbf{Y}_k$ mit $S = 0$ entspricht somit die Rangfolge der Unähnlichkeiten der Rangfolge der Distanzen der Konfiguration. Mit Hilfe des Stresswertes wird ausgehend von einer Konfiguration $\mathbf{Y}_k$ eine neue Konfiguration $\mathbf{Y}_{k+1}$ berechnet. Diese wird mit Hilfe des Gradientenverfahrens so berechnet, dass sie in Richtung des negativen Gradienten liegt, mit maximal absinkenden Stresswert. Der negative Gradient ist durch die partiellen Ableitungen des Stresswertes über alle Elemente der Konfiguration $\mathbf{Y}_k$ definiert als:

$$\left(-\frac{\partial S}{\partial y_{11}}, ..., -\frac{\partial S}{\partial y_{1p}}, ..., -\frac{\partial S}{\partial y_{np}}\right).$$

Im Falle der euklidischen Distanz lässt sich der Gradient wie folgt berechnen:

$$g_{kl} = S\sum_{i,j}(\delta^{ki} - \delta^{kj})(\frac{d_{ij} - \hat{d}_{ij}}{S^*} - \frac{d_{ij}}{T^*})\frac{y_{il} - y_{jl}}{d_{ij}},$$

dabei sind δ^{ki}, δ^{kj} die Kronecker Symbole mit

$$\delta^{ki} = \begin{cases} 1, & \text{wenn } k = i \\ 0, & \text{andernfalls.} \end{cases}$$

Als nächstes wird die Schrittgröße α bestimmt, mit der die aktuelle Konfiguration

entlang des negativen Gradienten verschoben wird. Initial empfiehlt Kruskal einen Wert von 0.2 festzulegen. Bei den weiteren Wiederholungen berechnet sich α durch [Kru64b]:

$$\alpha_{\text{aktuell}} = \alpha_{\text{davor}} \cdot (\text{Winkelfaktor}) \cdot (\text{Entspannungsfaktor}) \cdot (\text{Glücksfaktor})$$

mit

$$(\text{Winkelfaktor}) = 4^{(\cos\theta)^3} \ ,$$

wobei θ der Winkel zwischen dem aktuellen Gradienten und dem Vorgängergradienten ist. Weiter gilt:

$$(\text{Entspannungsfaktor}) = \frac{1.3}{1 + (\text{5-Schritt-Verhältnis})^5} \ ,$$

$$(\text{5-Schritt-Verhältnis}) = min(1, \frac{\text{aktueller Stress}}{\text{Stress vor 5 Schritten}}) \ ,$$

$$(\text{Glücksfaktor}) = min(1, \frac{\text{aktueller Stress}}{\text{Stress davor}}) \ .$$

Falls noch keine fünf Zyklen durchlaufen sind, wird für „Stress vor 5 Schritten“ der Anfangswert genommen und analog für andere vorhergehende Werte verfahren. Wenn g'' der vorhergehende Gradient ist, ist der Winkel des neuen Gradienten durch

$$\cos\theta = \frac{\sum_{i,j} g_{ij} g''_{ij}}{\sqrt{\sum_{i,j} g_{ij}^2} \sqrt{\sum_{i,j} g''^2_{ij}}}$$

bestimmt. Nun kann die **neue Konfiguration** $\mathbf{Y}_{k+1}$ berechnet werden mit

$$y'_{ij} = y_{ij} + \frac{\alpha}{mag(\mathbf{G})} g_{ij}$$

über alle i, j und $mag(\mathbf{G})$ ist die Stärke des Gradienten:

$$mag(\mathbf{G}) = \sqrt{\frac{1}{n} \sum_{i,j} g_{ij}^2} \ .$$

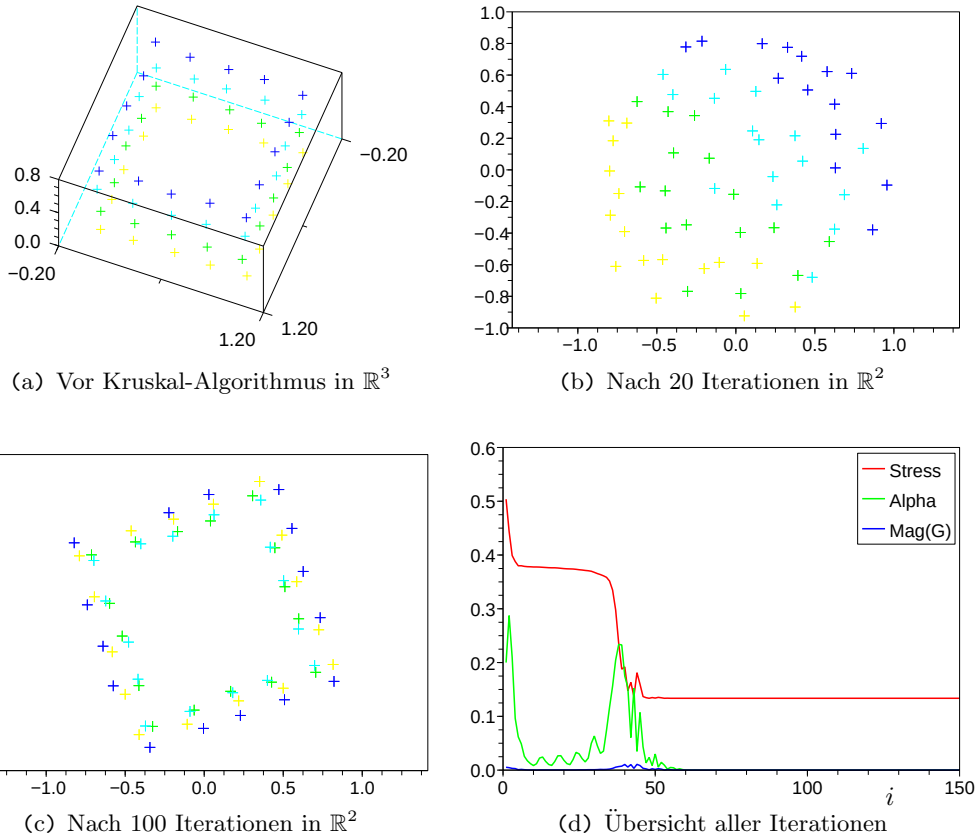
Nach jeder Wiederholung der Schleife verringert sich der Gradient und nähert sich somit einem Nullvektor an. Auch der Stresswert wird sukzessive kleiner und nähert sich einem lokalen, aber nicht zwangsweise globalen Minimum an. Als mögliche Ab-

bruchbedingungen können das Unterschreiten eines Grenzwertes für den Stresswert oder Gradientenbetrag oder einer Anzahl an Schleifenwiederholungen gelten. Der nichtmetrische MDS-Algorithmus von Kruskal hat zum Zeitpunkt des $k$ten Schleifendurchlaufes die Konfiguration Y_k zum Ergebnis:

$$\text{Kruskal}(\mathbf{\Delta}, p) = \mathbf{Y}_k \ .$$

Die Zeitkomplexität für jeden Iterationsschritt des Kruskal-Algorithmus ist wegen der aufwändigen Disparitätsbestimmung mit $O(dn^4)$ gegeben.

Als Beispiel dienen die Zwischenschritte der Transformation einer verrauschten Würfeloberfläche in den zweidimensionalen Raum (siehe Abbildung 2.3.3). Dabei ist in Abbildung 2.3.3(d) zu erkennen, dass sich nach 50 Wiederholungen die gefundene Konfiguration nicht mehr ändert.



(a) Vor Kruskal-Algorithmus in $\mathbb{R}^3$

(b) Nach 20 Iterationen in $\mathbb{R}^2$

(c) Nach 100 Iterationen in $\mathbb{R}^2$

(d) Übersicht aller Iterationen

Abbildung 2.3.3: *Beispiel für eine Kruskal-Transformation einer Würfeloberfläche.*

2.4 Nichtlinearer Algorithmus: Isometric Feature Mapping

Die *Isometric Feature Mapping* (ISOMAP) [TdSL00] zählt zu den nichtlinearen Transformationen und kann im Gegensatz zu den linearen Verfahren lineare als auch nichtlineare Mannigfaltigkeiten entdecken. Dafür müssen entsprechend der Definition der Mannigfaltigkeit (siehe Abschnitt 2.1) für jeden Datenpunkt im gegebenen Datensatz seine Nachbarpunkte bestimmt werden, von denen er linear abhängt.

Weiterhin muss der dabei entstehende Graph über alle Nachbarschaftsverbindungen zusammenhängend[2] sein (siehe analog zu LLE, Abschnitt 2.5).

Im Gegensatz zu anderen nichtlinearen Verfahren werden bei der ISOMAP die Abstände außerhalb der Nachbarschaft anhand der geodätischen Distanz (Abstand ihres kürzesten Pfades) bestimmt. Dadurch wird eine Distanzmatrix $\mathbf{\Delta}$ erzeugt, die lokale und globale lineare Abhängigkeiten besitzt. Auf die berechneten Distanzen wird die lineare CS-Transformation angewandt.

Der Algorithmus lässt sich in drei Schritte unterteilen. Als erstes werden die Distanzen δ_{ij} zwischen allen Datenpunkten $\mathbf{x}^i$ und deren benachbarten Datenpunkten $\mathbf{x}^j$ bestimmt, die entweder in einem festgelegten Radius r mit $\delta_{ij} < r$ oder innerhalb der K-Nachbarschaft von $\mathbf{x}^i$ liegen. Dabei wurde in dieser Arbeit für die Distanz der euklidische Abstand zwischen den beiden Datenpunkt $\mathbf{x}^i$ und $\mathbf{x}^j$ gewählt.

Im zweiten Schritt wird für jede unbestimmte Distanz die geodätische Distanz approximiert. Dafür kann ein beliebiger Algorithmus zur Bestimmung aller kürzesten Pfade genommen werden. In der vorliegenden Arbeit wurde der Floyd-Warshall-Algorithmus verwendet. Dieser basiert auf der Grundidee, dass für einen kürzesten Weg $w_1 = (i, ..., j)$ von Knoten i nach Knoten j und einen anderen kürzesten Weg $w_2 = (j, ..., k)$ von Knoten j nach Knoten k die Vereinigung beider Wege $w_3 = (i, ..., j, ..., k)$ wiederum ein kürzester Weg ist.

Der Ablauf der Floyd-Warshall-Methode ist prozedural im Algorithmus 2.4.1 implementiert. Dabei ist die Adjazenzmatrix die Matrix, in welcher der kürzeste Pfad von Knoten i nach Knoten j an Stelle a_{ij} gespeichert wird. Die Adjazenzmatrix wird initial mit den ermittelten Werten aus $\mathbf{\Delta}$ gefüllt. Alle noch nicht bekannten Distanzen werden dort auf unendlich gesetzt. Als Ergebnis wird die Adjazenzmatrix

[2] Wenn ausgehend von jedem Punkt p_i jeder andere Datenpunkt p_j in der K-Nachbarschaft von p_i verbunden wird, dann heißt eine Datenmenge oder Graph K-zusammenhängend, wenn jeder Punkt von jedem anderen Punkt aus über einen Pfad erreichbar ist. Ein Datenpunkt p_i liegt dabei in der K-Nachbarschaft von dem Datenpunkt p_j, wenn p_j zu den K-nächstliegenden Datenpunkten von p_i zählt.

Algorithmus 2.4.1: *FloydWarshall(Adjazenzmatrix A)*

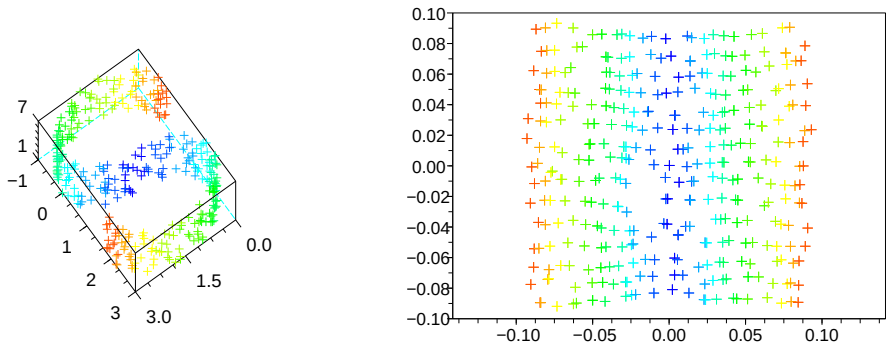
```
function FloydWarshall(Adjazenzmatrix A) {
  for (k=0; k<n; k++) {
    for (i=0; i<n; i++) {
      for (j=0; j<n; j++) {
        a[i][j]=min(a[i][j], a[i][k]+a[k][j]);
      }
    }
  }
}
```

A ermittelt, die für alle Datenpaare i, j den kürzesten Pfad a_{ij} enthält.

Im dritten Schritt werden aus den geodätischen Distanzen der Adjazenzmatrix mit Hilfe des CS-Algorithmus die Zielkonfiguration **Y** mit der Dimension p gesucht, die die Distanzen erhalten kann. Es gilt für den Algorithmus:

$$ISOMAP(\mathbf{X}, r, K) = CS(\mathbf{A}, p) \ .$$

Zur Demonstration wird eine ISOMAP-Transformation einer dreidimensionalen S-Kurve bestehend aus $n = 345$ Datenpunkten (siehe Abbildung 2.4.1(a)) mit Hilfe einer Nachbarschaft von $K = 8$ Datenpunkten in den zweidimensionalen Raum erzeugt (siehe Abbildung 2.4.1(b)). Es ist zu erkennen, dass die Fläche der S-Kurve „ausgebreitet“ wurde und nicht wie bei den linearen Transformationen ein „Schatten“ der Figur abgebildet wurde.



(a) Vor Transformation in $\mathbb{R}^3$ (b) Nach Transformation in $\mathbb{R}^2$

Abbildung 2.4.1: *Beispiel für eine ISOMAP-Transformation einer S-Kurve.*

Die Zeitkomplexität für die Berechnung der Distanzmatrix $\mathbf{\Delta}$, der Sortierung der Nachbardistanzen für jeden Datenpunkt, des Floyd-Warshall-Algorithmus und des CS-Algorithmus ist durch $O(dn^2 + n^2 \log_2 n + n^3 + n^3)$ gegeben. Das ergibt eine Gesamtkomplexität von $O(n^2(d + n))$.

2.5 Nichtlinearer Algorithmus: Locally Linear Embedding

Die *Locally Linear Embedding* (LLE) [RS00] [SR03] berechnet alle lokal linearen Einbettungen innerhalb einer Nachbarschaft jedes Datenobjektes und findet somit globale nichtlineare Strukturen in den Ausgangsdaten. Im Vergleich zu dem nichtlinearen ISOMAP-Algorithmus müssen hier nicht alle kürzesten Pfaddistanzen bestimmt werden. Der Algorithmus lässt sich in drei Schritte unterteilen:

Zuerst werden zu jedem Datenpunkt $\mathbf{x}^i$ die Nachbarpunkte $\mathbf{x}^j$ bestimmt, die entweder in einer K-Nachbarschaft oder in einem festgelegtem Radius r um $\mathbf{x}^i$ liegen.

Im zweiten Schritt werden die Gewichte w_{ij} (lineare Koeffizienten) bestimmt, die jeden Datenpunkt $\mathbf{x}^i$ als Linearkombination durch seine K Nachbarpunkte $\mathbf{x}^j$ am besten rekonstruieren lassen. Um das zu erreichen muss der Rekonstruktionsfehler

$$\varepsilon(\mathbf{W}) = \sum_{i=1}^{n} \parallel \mathbf{x}^i - \sum_{j=1}^{n} w_{ij}\mathbf{x}^j \parallel^2 \tag{2.14}$$

der Gewichtsmatrix $\mathbf{W}$ minimiert werden. Dabei ist $w_{ij} = 0$, wenn $\mathbf{x}^j$ kein K-Nachbar von $\mathbf{x}^i$ ist. Zur Berechnung der $(n \times n)$ Gewichtsmatrix über alle n Datenpunkte wird die lokale $(K \times K)$ Grammatrix $\mathbf{G}$ für die K Nachbarn jedes Datenpunkt $\mathbf{x}^i$ verwendet

$$g_{jk} = (\mathbf{x}^i - \boldsymbol{\eta}_j)(\mathbf{x}^i - \boldsymbol{\eta}_k) \ .$$

Dabei sind $\boldsymbol{\eta}_j$ und $\boldsymbol{\eta}_k$ die K-Nachbarn von Datenpunkt $\mathbf{x}^i$. Um die globale $(n \times n)$ Gewichtsmatrix $\mathbf{W}$ zu berechnen, muss für jeden der n Datenpunkte $\mathbf{x}^i$ die Linearkombination aus seinen Nachbarn bestimmt werden. Die Linearkombination $\mathbf{w}_i$ wird durch das Lösen des linearen Gleichungssystems

$$\mathbf{G}_i\mathbf{w}_i = \mathbf{1}$$

ermittelt. Dabei ist $\mathbf{1}$ ein Spaltenvektor bestehend aus K Einsen. Dafür muss sich $\mathbf{G}$ invertieren lassen. In dem Fall, dass $K > d$ gilt, ist $\mathbf{G}$ nicht invertierbar und es muss ein geringes Vielfaches der Identitätsmatrix zu $\mathbf{G}$ addiert werden. Bei einer Abwandlung des LLE kann die Bestimmung der Gewichtsmatrix $\mathbf{W}$ durch den Laplace-Beltrami-Operator [BN02] bestimmt werden. Die Gewichte berechnen sich

dann durch:

$$w_{ij} = \begin{cases} e^{\frac{-\|\mathbf{x}^i - \boldsymbol{\eta}_j\|^2}{4t}}, & \boldsymbol{\eta}_j \text{ sind } K\text{-Nachbarn von Datenpunkt } \mathbf{x}^i \\ 0, & \text{sonst.} \end{cases}$$

Dabei ist t eine reelle Zahl.

Im dritten Schritt wird die lineare Einbettung $\mathbf{Y}$ der Dimension p aus der Gewichtsmatrix $\mathbf{W}$ berechnet. Es werden dabei die $\mathbf{y}^i$ transformierten Datenpunkte der Dimension p bestimmt, die sich am besten durch die Gewichtsmatrix rekonstruieren lassen. Dafür muss der Fehler

$$\Phi(\mathbf{Y}) = \sum_{i=1}^{n} \left| \mathbf{y}^i - \sum_{j=1}^{n} w_{ij} \mathbf{y}^j \right|^2$$

der Zielkonfiguration $\mathbf{Y}$ minimiert werden. Darüber hinaus werden zwei Anforderungen an die Zielkonfiguration $\mathbf{Y}$ gestellt. Erstens soll sie im Koordinatenursprung zentriert sein:

$$\sum_i \mathbf{y}^i = 0 \ .$$

Und zweitens soll sie eine Kovarianz gleich der Einheitsmatrix $\mathbf{I}$ besitzen:

$$\frac{1}{n} \sum_i (\mathbf{y}^i)^T \mathbf{y}^i = \mathbf{I} \ .$$

Somit lässt sich die $(n \times n)$ Matrix

$$\mathbf{M} = (\mathbf{I} - \mathbf{W})^T (\mathbf{I} - \mathbf{W})$$

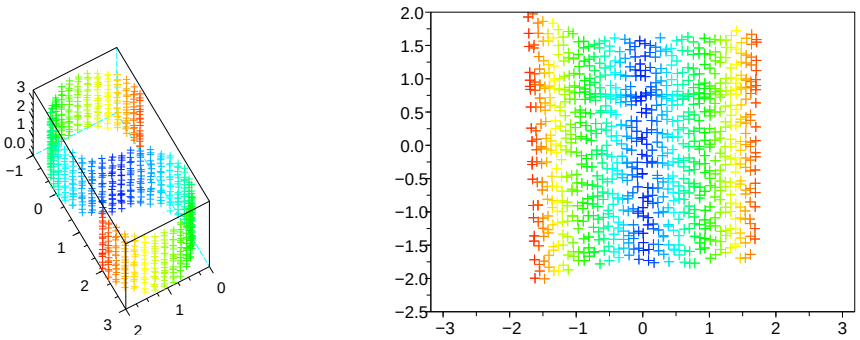
berechnen, dabei ist $\mathbf{I}$ die $(n \times n)$ Einheitsmatrix. Um die Einbettung $\mathbf{Y}$ zu erhalten, werden typischerweise die kleinsten $d + 1$ Eigenvektoren der Matrix $\mathbf{M}$ bestimmt, wobei der kleinste Eigenvektor $\mathbf{v}_0$ dem Einheitsvektor mit dem Eigenwert $\lambda_0 = 0$ entspricht und nicht berücksichtigt wird.
Somit gilt für den Algorithmus:

$$LLE(\mathbf{X}, r, K) = (\mathbf{v}_1, ..., \mathbf{v}_p), \text{ mit } 0 < \lambda_1 < ... < \lambda_p \ ,$$

$\mathbf{v}_i$ ist Eigenvektor und λ_i ist der entsprechende Eigenwert der Matrix $\mathbf{M}$. Die Zeit-

komplexität der Bestimmung der K nächsten Nachbarn eines jeden Punktes ist $O(dn^2 + n^2 \log_2 n)$. Die Berechnung der Gewichte verursacht eine Zeitkomplexität von $O(dnK^3)$ und die Berechnung der p kleinsten Eigenwerte inklusive Eigenvektoren hat eine Zeitkomplexität von $O(pn^2)$ [SR03]. Das ergibt eine Gesamtkomplexität von $O(n^2(d + log_2 n + p) + dnK^3)$

Zur Demonstration wird eine LLE-Transformation einer dreidimensionalen S-Kurve, bestehend aus $n = 624$ Datenpunkten (siehe Abbildung 2.5.1(a)), mit Hilfe einer Nachbarschaft von $K = 8$ Datenpunkten in den zweidimensionalen Raum präsentiert (siehe Abbildung 2.5.1(b)). Genauso wie der ISOMAP-Algorithmus findet der LLE-Algorithmus die Fläche der S-Kurve.



(a) Vor Transformation in $\mathbb{R}^3$ (b) Nach Transformation in $\mathbb{R}^2$

Abbildung 2.5.1: *Beispiel für eine LLE-Transformation einer S-Kurve.*

Bestimmung der optimalen Nachbarschaft

Bei der Verwendung der nichtlinearen LLE-Transformation wird zuerst eine Nachbarschaftsgröße festgelegt, innerhalb dieser dann zu jedem Datenpunkt seine Nachbarn erfasst werden. Dabei wurde in den vergangenen Textabschnitten nicht beschrieben, wie die optimale Größe von K oder r bestimmt wird.

Je nach Größe der Nachbarschaft ändert sich das Gesamtverhalten der Transformation. Wenn die Nachbarschaft zu klein ist, kann der Graph über allen kürzesten Pfaden nicht mehr K-zusammenhängend sein. In diesem Fall wird die LLE-Transformation generell schlechter. So müssen für alle K-zusammenhängenden Teilgraphen im Datensatz die LLE-Transformationen einzeln berechnet werden [PP01]. Falls eine zu große Nachbarschaft gewählt wird, werden die linearen Abhängigkeiten der

Nachbarn gebrochen und die Auffaltung einer nichtlinearen Mannigfaltigkeit ist nicht möglich.

Kouropteva hat eine hierarchische Methode angegeben, mittels dieser die optimale Nachbarschaftsgröße für den LLE-Algorithmus ermittelt werden kann [KOP02]. So besteht diese Methode aus folgenden Schritten:

Zuerst wird die maximale Nachbarschaftsgröße K_{max} festgelegt[3], so dass für alle $K_i \in \{1, K_{max}\}$ die Rekonstruktionsfehler $\varepsilon(\mathbf{W})$ (siehe Gleichung 2.14) berechnet und dann die K_i in S gespeichert werden, die jeweils ein Minimum von $\varepsilon(\mathbf{W})$ sind.

Für alle $K_i \in S$ wird die LLE-Transformation berechnet und das K_i, das eine minimale *Residual Variance* erzeugt, ist das K_{opt}. Dabei verwendet diese Varianz den Korrelationskoeffizienten R zwischen den Distanzen vor und nach der Transformation:

$$1 - R^2(\boldsymbol{\Delta}, \boldsymbol{D}) \ .$$

Für den Korrelationskoeffizient R nach Pearson gilt [Spe04]:

$$R(\boldsymbol{Z}, \mathbf{Y}) = \frac{\sum_{i=1}^{N}(\mathbf{z}_i - \bar{\boldsymbol{z}})(\mathbf{y}_i - \bar{\boldsymbol{y}})}{\sqrt{\sum_{i=1}^{N}(\mathbf{z}_i - \bar{\boldsymbol{z}})^2 \cdot \sum_{i=1}^{N}(\mathbf{y}_i - \bar{\boldsymbol{y}})^2}} \ , \tag{2.15}$$

$N = (n^2 - n)/2$ ist die Anzahl aller paarweisen Distanzen der Datenpunkte. Der Korrelationskoeffizient gibt die Stärke der linearen Abhängigkeit beider Variablen an. Das Ergebnis liegt im Intervall $[-1, +1]$. Dabei gilt, wenn $R = -1$ bzw. $R = +1$, dann ist die Variable Z perfekt negativ bzw. perfekt positiv linear abhängig von der Variable Y.

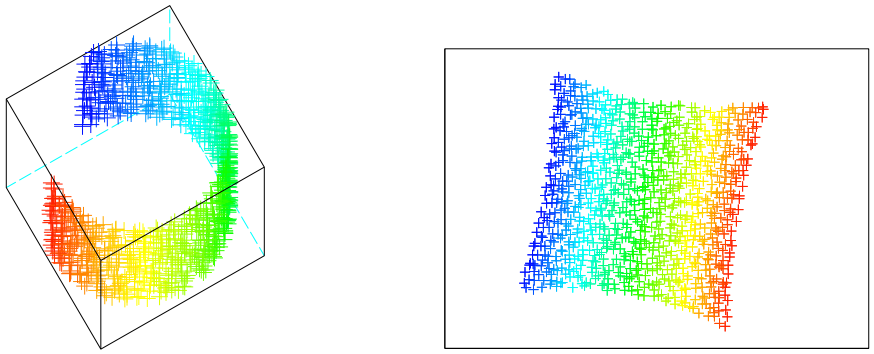
Die Resiual Variance bildet das Ergebnis des Korrelationskoeffizienten auf das Intervall $[0, 1]$ ab. Bei $R = 0$ besteht ein perfekt linearer Zusammenhang und bei $R = 1$ existiert kein linearer Zusammenhang.

Bestimmung der Dimension der Mannigfaltigkeit

Für die Anwendung des LLE-Algorithmus sowie anderer nichtlinearer Verfahren muss jeweils eine Zieldimension definiert werden, in die die Daten transformiert werden. Diese Dimension sollte der Dimensionsgröße der Mannigfaltigkeit entsprechen, auf

[3] Genauso wie K_{max} kann auch der maximal Radius r_{max} verwendet werden.

der die Datenpunkte verteilt sind, um den geringsten Informationsverlust zu gewährleisten. Zur Bestimmung dieser Dimension gibt es mehrere Varianten. Anhand der Transformation einer C-Kurve (siehe Abbildung 2.5.2) sollen zwei Verfahren vorgestellt werden.



(a) Vor Transformation (b) Nach LLE-Transformation

Abbildung 2.5.2: *Beispiel für eine LLE-Transformation einer C-Kurve.*

In der Arbeit von Polito und Perona [PP01] wird eine Möglichkeit gegeben, um die Dimension der Mannigfaltigkeit einzugrenzen. So soll die Dimension d kleiner als die Anzahl z gewählt werden. Dabei ist z die Anzahl der Eigenvektoren mit Eigenwert nahe Null, die von der Matrix $\mathbf{M}$ bestimmt wurden. Dem wird vorausgesetzt, dass die Datenpunkte K-zusammenhängend sind. So sollen bei nicht exakter lokaler Linearität oder Anwesenheit von Rauschen in den Ausgangsdaten entsprechend die Eigenwerte von 0 abweichen. Saul und Roweis weisen in ihrer Arbeit [SR03] darauf hin, dass diese Methode nur bei sehr künstlichen Daten gute Resultate liefert.

In Abbildung 2.5.3 ist die Verteilung der Eigenwerte der LLE-Transformation der C-Kurve mit $K = 7$ logarithmisch abgebildet. Der kleinste Eigenwert ist dabei dem Einheitsvektor zugeordnet und wird nicht betrachtet. Nur mit Mühe kann $z = 2$ detektiert werden.

Eine weitere Methode, um die Dimension der Mannigfaltigkeit zu bestimmen, ist die Untersuchung der Eigenwerte der lokalen Kovarianzmatrizen für jeden Datenpunkt in seiner K-Nachbarschaft [SR03]. Dafür werden für jede K-Nachbarschaft die PCs berechnet und die Summe der ersten Eigenwerte der Kovarianz (siehe Gleichung 2.2) untersucht. Entsprechend der 80%-Regel von Jolliffe [Jol86] kann die Dimension der Mannigfaltigkeit geschätzt werden.

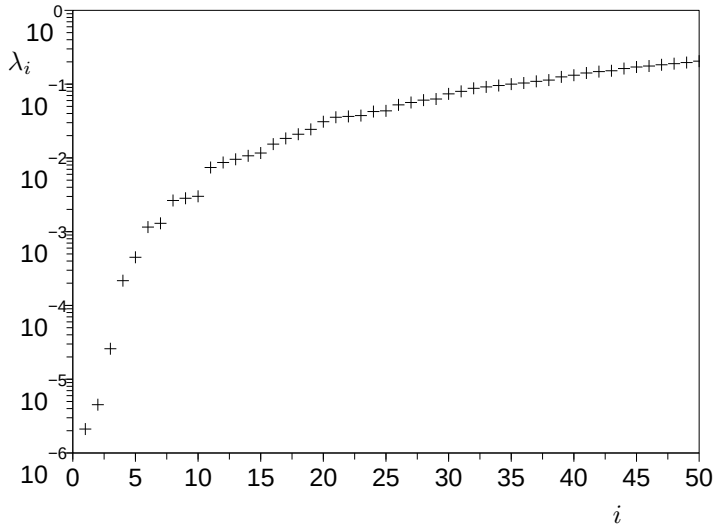


Abbildung 2.5.3: *Beispiel für die Eigenwerte einer LLE-Transformation einer C-Kurve.*

In Abbildung 2.5.4 sind die durchschnittlichen prozentualen Anteile der Summe über alle Eigenwerte an der Gesamtsumme für $k \in [2, 10]$ abgebildet. Die Eigenwerte wurden den lokalen Kovarianzen für die K-Nachbarschaft jedes Datenvektors entnommen.

Anhand dieser Methode ist gut zu erkennen, dass für $K \in \{3, 10\}$ die ersten beiden Eigenwerte eine Varianz von 100% und damit $\geq 80\%$ abdecken. Die Dimension der Mannigfaltigkeit wird korrekt mit zwei erkannt. Aufgrund der guten Resultate wird in dieser Arbeit zur Bestimmung der Dimension der nichtlinearen Mannigfaltigkeiten diese Variante verwendet.

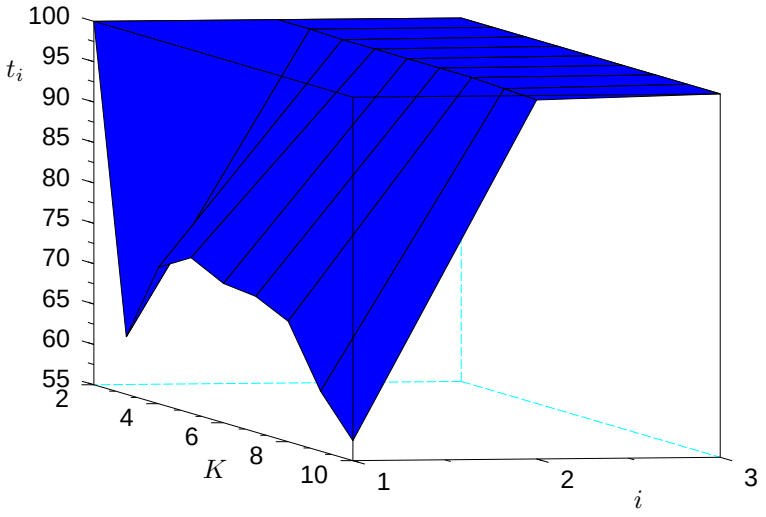


Abbildung 2.5.4: *Beispiel für den durchschnittliche Anteil der Summe der ersten Eigenwerte an der Gesamtsumme der lokalen Kovarianzen einer C-Kurve.*

3 Verfahren zur Gütebestimmung linearer Transformationen

3.1 Metrisches Verfahren: Procrustes-Analyse

Bei der Procrustes-Analyse [LdVC95] werden zuerst die beiden zu vergleichenden Konfigurationen $\mathbf{X}$ und $\mathbf{Y}$ bestmöglich deckungsgleich übereinander gelegt. Die Güte wird dann durch die summierten Längendifferenzen der Datenvektoren beider Konfigurationen berechnet.

Zu Beginn werden die $d-p$ fehlenden Dimensionen in der Konfiguration $\mathbf{Y}$ spaltenweise mit Nullen aufgefüllt. Anschließend werden $\mathbf{X}$ und $\mathbf{Y}$ zentriert (siehe Gleichung 2.1). Nun wird $\mathbf{Y}$ relativ zu $\mathbf{X}$ rotiert:

$$\mathbf{Y} = \mathbf{YQ} \ .$$

Dabei ist $\mathbf{Q}$ die orthogonale Matrix

$$\mathbf{Q} = \mathbf{VU}^T$$

und $\mathbf{U\Lambda V}^T$ die Singulärwertzerlegung der Matrix $\mathbf{X}^T\boldsymbol{Y}$. Anschließend wird $\mathbf{Y}$ um den Faktor c skaliert. Es gilt:

$$c = \frac{\mathbf{Spur}(\mathbf{\Lambda})}{\mathbf{Spur}(\mathbf{YY}^T)}.$$

Dabei ist $\mathbf{Spur}(\mathbf{\Lambda})$ die Spur der Eigenwertmatrix und durch

$$\mathbf{Spur}(\mathbf{\Lambda}) = \sum_{i=1}^{n} \lambda_{ii}$$

gegeben. Nun kann der Messwert mittels euklidischer Distanz (siehe Gleichung 2.3) bestimmt werden:

$$Procrustes(\mathbf{X}, \mathbf{Y}) = \sum_{i=1}^{n} \|(\mathbf{x}^i)^T - (\mathbf{y}^i)^T\|^2 \ .$$

Je kleiner das Ergebnis der Berechnung ist, desto besser gleichen sich die Konfigurationen und desto besser ist die lineare Transformation zu bewerten.

Aufgrund der linearen Berücksichtigung der globalen und lokalen Distanzen durch ausschließlich lineare Transformationen handelt es sich um ein Gütebestimmungsverfahren, dass für die Bestimmung der Güte von metrischen linearen Transformationen geeignet ist.

3.2 Nichtmetrisches Verfahren: Spearman's Rho

Spearman's Rho ist ein Produkt-Moment-Korrelationskoeffizient ρ_S, der die Abweichung der Ränge aller Distanzen der Datenpunkte vor und nach der Transformation verwendet. Somit benutzt diese Funktion einen ähnlichen Ansatz wie die Berechnung des Stress-Wertes von Kruskal.

In seiner ursprünglichen nichtnormalisierten Form berechnet er sich die Güte mit [Spe04]

$$\rho_S(\mathbf{\Delta}, \mathbf{D}) = 1 - \frac{3\sum_{i=1}^{N} \bar{d}_i}{N^2 - 1} \ .$$

Dabei ist $\bar{d}_i$ die Rangdifferenz aller $N = (n^2 - n)/2$ paarweisen Distanzen der Datenpunkte. Damit der Wert auf den Bereich von $[-1, 1]$ normalisiert ist, lautet die Definition [Slo05]:

$$\rho_S(\mathbf{\Delta}, \mathbf{D}) = 1 - \frac{6\sum_{i=1}^{N} \bar{d}_i^2}{N(N^2 - 1)} \ .$$

Im Falle von Gleichheit mehrerer Distanzen in $\mathbf{X}$ oder $\mathbf{Y}$ wird der Messwert ungenau, da die Ränge sich nicht gleichen. Darum berechnet sich in diesen Fällen der Rang aus dem arithmetischen Mittel der Ränge für alle gleichen Distanzen.

Das Ergebnis für $\rho_S(\mathbf{\Delta}, \mathbf{D})$ ist 0, wenn keine Korrelation zwischen den Rängen der Differenzen besteht. Wenn das Ergebnis +1/-1 lautet, dann liegt eine perfekte positive/negative Korrelation vor und die Transformation konnte alle/keine Ränge der Differenzen erhalten.

Dieses Verfahren ist für die Gütebestimmung von nichtmetrischen linearen Transformationen geeignet, da die Ränge aller Distanzen im Raum berücksichtigt werden.

4 Ausführungsgeschwindigkeiten der in Scilab implementierten dimensionsreduzierenden Algorithmen

In diesem Abschnitt werden die Algorithmen bezüglich ihres Zeitverbrauchs für verschiedene Datensatzgrößen und Parametersätze untersucht. Dieser Zeitfaktor ist eine wesentliche Kenngröße eines Algorithmus. Insbesondere soll festgestellt werden, für welche Datensatzgröße und Wahl der Parameter die einzelnen Verfahren ein Ergebnis zeitnah berechnen können. Aufgrund der Vielfalt der Matrixoperationen und der graphischen Ausgabe wurden alle dimensionsreduzierenden Algorithmen wurden im Rahmen dieser Arbeit in Scilab implementiert.[4] Bei den folgenden Zeitmessungen wurden jeweils n zufällig generierte Datenpunkte mit $n <= 1000$ der Dimension $d <= 1000$ in den zweidimensionalen Raum transformiert.

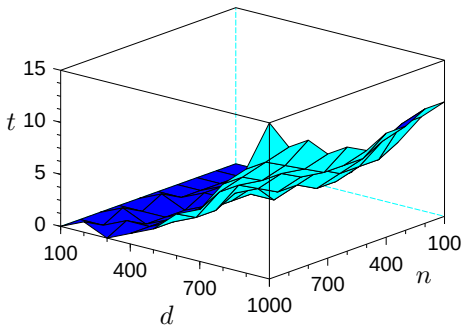
Für die linearen PCA-, CS- und Kruskal-Transformationen konnten die Ausführungsgeschwindigkeit in Abhängigkeiten von d und n in Sekunden gemessen werden (siehe Abbildung 4.0.1). Dabei ist festzuhalten, dass die PCA die höchste Ausführungsgeschwindigkeit bei $d \ll n$ aufweist. Bei einer Dimensionsgröße mit $d \leq 100$ ist der Zeitverbrauch sogar vernachlässigbar gering. Beim CS-Algorithmus zeigen die Ergebnisse, dass die Ausführungsgeschwindigkeit für $d \in [1, 1000]$ bei festem n als konstant betrachtet werden kann. Bei einer Anzahl von $n > 1000$ Datenpunkten können die Berechnungen jedoch äußerst zeitintensiv werden.

Der Kruskal-Algorithmus hat den größten Zeitverbrauch. Transformationen von Datenpunkten sind ab der Anzahl von $n > 200$ mit Scilab nicht zeitnah berechenbar, da zusätzlich noch bis zu 100 Wiederholungen pro Datensatz benötigt werden, um eine ausreichend gute Transformation zu erhalten [Kru64a].

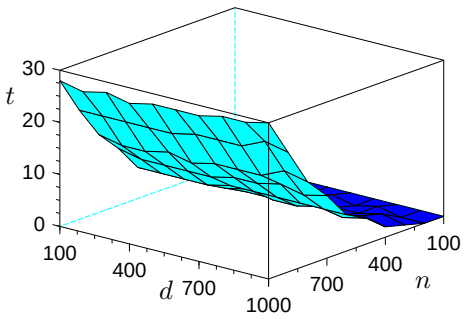
Bei den nichtlinearen ISOMAP- und LLE-Transformationen wurden die zeitlichen Abhängigkeiten der Parameter n, d und K in Sekunden gemessen (siehe Abbildung 4.0.2). Die Nachbarschaftsgröße K ist hier als prozentualer Anteil an der Anzahl n der Datenpunkte angegeben.

Der Zeitverbrauch des ISOMAP-Algorithmus ist von K unabhängig. Weiterhin kann die Dimensionsgröße d im Intervall $n = [1, 1000]$ als unabhängiger Faktor betrachtet werden, so dass nur n als wesentliche Größe den Zeitverbrauch bestimmt.

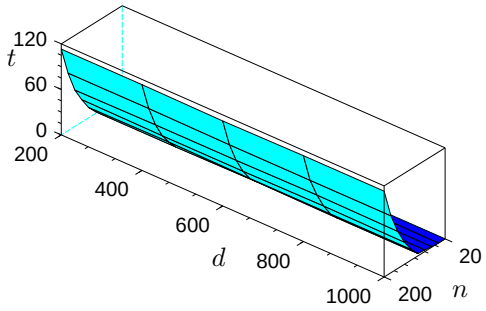
[4] Softwarepakete und Informationen zu Scilab sind auf der Seite `http://www.scilab.org` kostenlos erhältlich.



(a) PCA

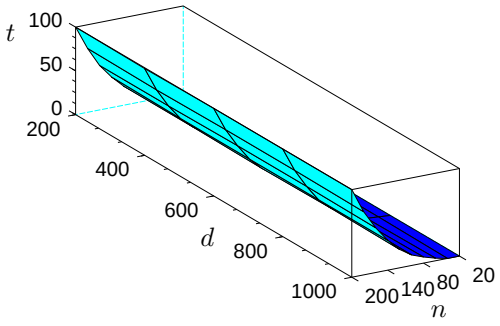


(b) CS

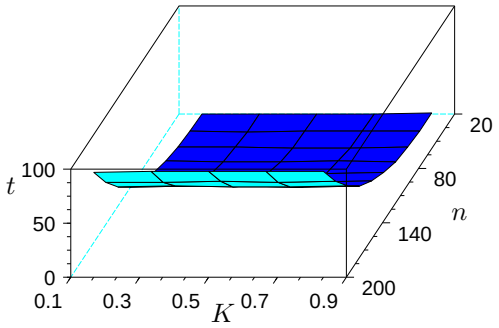


(c) Kruskal pro Iteration

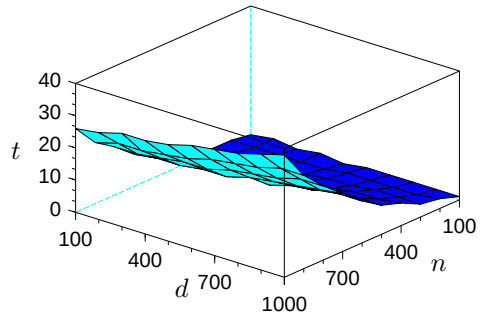
Abbildung 4.0.1: *Ausführungsgeschwindigkeit der linearen Dimensionsreduktionsalgorithmen in Abhängigkeit von n und d in Sekunden. Für die Messungen wurden zufällige Daten verwendet.*



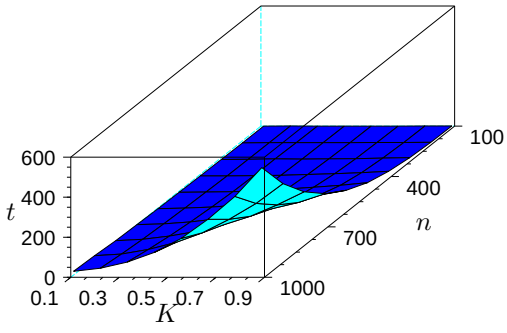
(a) ISOMAP mit $K = 14$



(b) ISOMAP mit $d = 100$



(c) LLE mit $K = 50$



(d) LLE mit $d = 100$

Abbildung 4.0.2: *Ausführungsgeschwindigkeit der nichtlinearen Dimensionsreduktionsalgorithmen in Abhängigkeit von n, d und K in Sekunden.*

Die ISOMAP-Implementierung innerhalb dieser Arbeit ist nur auf eine Anzahl von $n \leq 500$ anwendbar. Darüber hinaus kann der Grafik entnommen werden, dass Berechnungen von LLE-Transformationen mit einer Anzahl von $n > 100$ vor allem für große K viel Zeit benötigen. Wenn jedoch eine geringe Nachbarschaft mit $K < 50$ und eine geringe Dimensionsgröße mit $d \leq 100$ verwendet werden, sind Transformationen mit bis zu 1000 Datenpunkten zeitnah berechenbar.

5 Experimente und Resultate der Datenvisualisierung

5.1 Ursprung und Filterung der sensorischen Rohdaten

Als Quelle der hochdimensionalen Sensordaten, dient die als A-Serie bezeichnete humanoide Roboterplattform (siehe Anhang) des *Nerorobotics Research Laboratory* (NRL). Dieser besitzt 21 aktive Gelenke und acht zweidimensionale Beschleunigungssensoren. Gemäß des Hardwaretakts kann alle zehn Millisekunden ein 37-dimensionaler Datenvektor ausgelesen werden.

Für die folgenden Experimente wird eine Sequenz von Datenvektoren verwendet, die während verschiedener Körperbewegungen des humanoiden Roboters aufgezeichnet wurden. Für die Generierung der Bewegungen wurden fertige Netze von aneinandergereihten Körperhaltungen abgespielt. Jede Haltung ist dabei über alle Gelenkwinkel definiert.

Für die visuelle Zuordnung jedes Datenvektors zu der entsprechenden Körperpose des Roboters wurden während der Aufzeichnung der Robotersensorik die Bewegungen mit einer Kamera aufgenommen. Jeder ausgelesene Sensorwert (Gelenkwinkel und Beschleunigungswert) ist dabei einheitlich auf das Intervall $[-1, 1]$ skaliert, wodurch alle Sensorwerte als gleichrangig betrachtet werden.

Eine exemplarische Bewegungssequenz, bei der der Roboter seine Arme bewegt, ist in Abbildung 5.1.1 zu sehen. Darin beginnt der Roboter mit dem folgenden Bewegungsablauf:

1. Stehphase bis Zeitschritt $t \approx 400$
2. Gleichmäßiges Heben des linken Armes bis Zeitschritt $t \approx 600$
3. Stehphase mit gehobenen linken Arm bis Zeitschritt $t \approx 700$
4. Gleichmäßiges Absenken des linken Armes bis Zeitschritt $t \approx 900$

Bevor die Daten einer Dimensionsreduktion unterzogen werden können, müssen sie zuvor den Bedürfnissen entsprechend angepasst werden. Im Folgenden werde zwei für die Experimente notwendige Datenanpassungen beschrieben:

1. Ein Schritt der Vorverarbeitung ist die Fehlerkorrektur der Sensormessungen. Je nachdem, wie genau das Ergebnis der Untersuchung sein soll, müssen auch die Fehler in den Daten bereinigt werden. Dabei wird die Messgenauigkeit der Sensoren in der vorliegenden Arbeit als hinreichend genau vorausgesetzt.

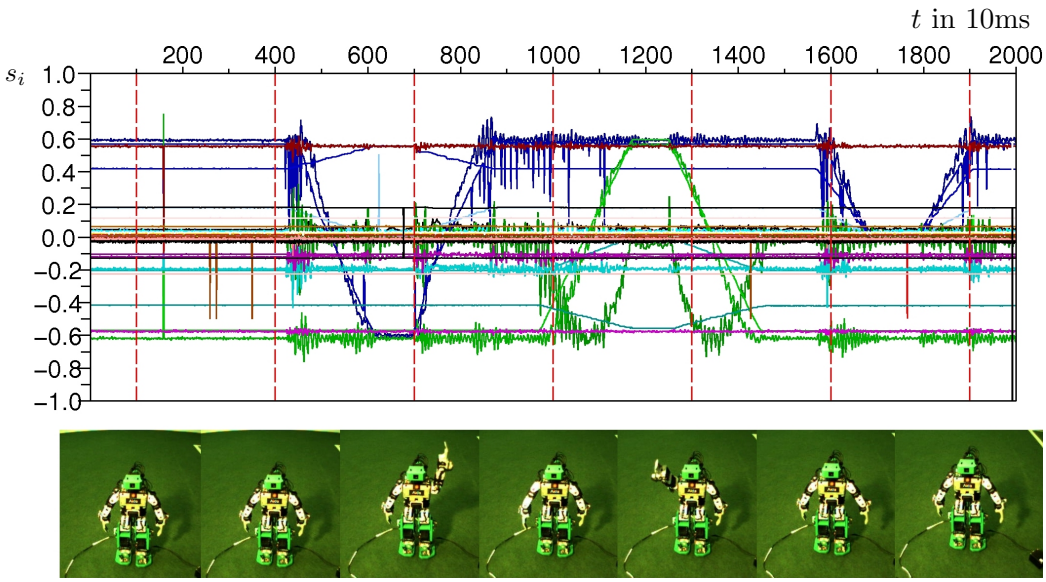
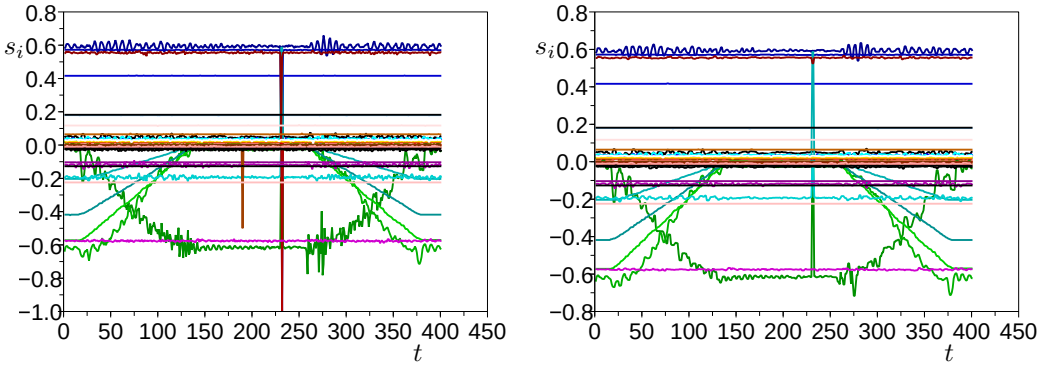


Abbildung 5.1.1: *Für jeden Zeitschritt* $t \in [1, 2000]$ *ist jeder der* 37 *Sensorwerte* s_i *als farbige Kurve dargestellt. Zu jedem Zeitschritt* $z \in \{100, 400, 700, 1000, 1300, 1600, 1900\}$ *(rote senkrechte Linien) ist darunter das zugehörige Kamerabild abgebildet.*

Jedoch wurden nach dem Auslesen der Sensorwerte vereinzelt Bitfehler detektiert, die einen Fehlerwert von bis zu 50% des maximalen Messbereiches aufweisen. Da innerhalb der Experimente nicht die Robustheit der Transformationen untersucht wird, werden zur Eliminierung dieser Ausreißer ein Medianfilter verwendet, der für jeden Sensorwert innerhalb einer Fenstergröße (typischerweise ungerade Größe) alle bis auf den mittleren der geordneten Werte herausfiltert.

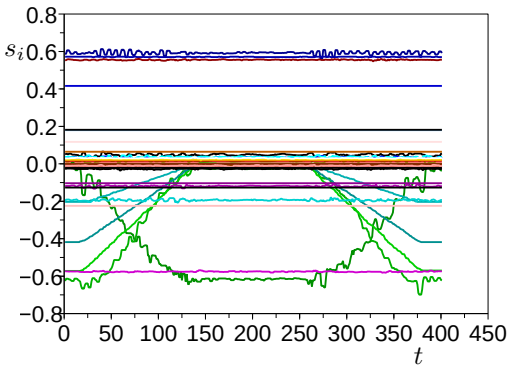
In Abbildung 5.1.2 sind die Beispielsensorwerte einer Roboterbewegung ungefiltert und gefiltert dargestellt. Innerhalb dieser Arbeit wurde eine Fenstergröße von fünf verwendet. Der Nachteil einer jeden Filterung besteht jedoch darin, dass je nach Stärke der Filterung eine entsprechend starke Informationsreduktion einhergeht.

2. Für die Minimierung des Rechenaufwandes der bevorstehenden Dimensionsreduktionen ist die Reduktion der Datenpunktanzahl des Messdatensatzes ein geeignetes Mittel. Bei einer mehrere Minuten langen Roboterbewegung kann die Messreihe entsprechend der 100Hz-Taktfrequenz aus mehreren Zehntausend Datenvektoren bestehen. Durch die rechenintensiven Algorithmen könnte die Ermittlung einer Transformation somit Tage dauern (siehe Abschnitt 4).



(a) Ohne Median

(b) Median 3



(c) Median 5

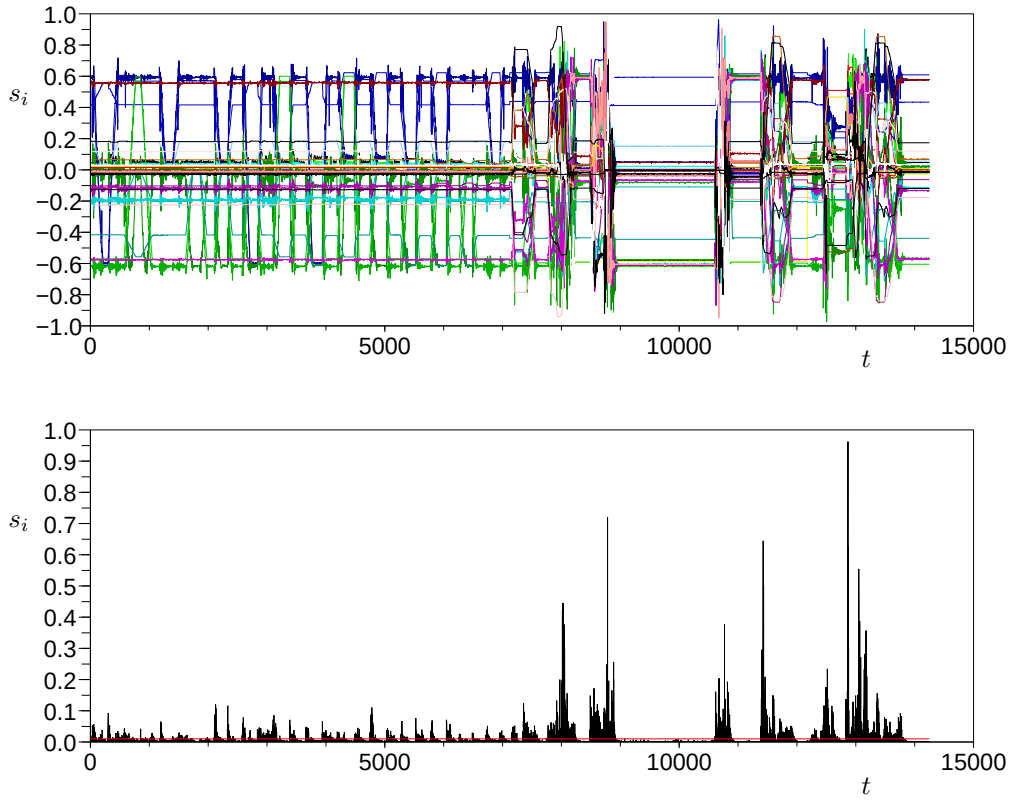
Abbildung 5.1.2: *Medianfilter mit unterschiedlicher Größe angewandt auf Sensordaten. Die Daten der* 37 *Sensoren wurden über die* 400 *Zeitschritte jeweils in einer Farbe eingezeichnet.*

Es gibt mehrere Möglichkeiten, dies zu umgehen. Die erste Option ist, aus den Messungen nur jeden x-ten Datenpunkt zu berücksichtigen. Dadurch kann der Datensatz relativ genau in seiner Zielgröße n bestimmt werden. Jedoch wird dann eine informationsarme Ruhephase in der Bewegung genauso „zerstückelt“ wie eine informationsreiche Bewegungsphase. Das Ergebnis ist ein „zackiger“ Kurvenverlauf, dem die „weichen“ Zwischenstufen der Bewegungsphase fehlen.

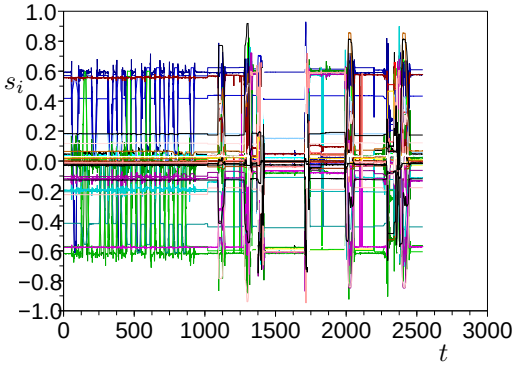
Die zweite Option ist die Berücksichtigung ausschließlich der Daten, die markante Sprünge bzw. einen Informationsanstieg aufweisen. Dafür werden zuerst die $n - 1$ euklidischen Distanzen aller aufeinanderfolgender Sensordaten bestimmt. Wenn dann eine Distanz zum nächsten Sensorvektor einen bestimmten Schwellwert t unterschreitet, wird dieser Vektor aus der Datenmenge entfernt. Damit können die informationsarmen Phasen auf nur wenige Datenpunkte reduziert werden. Diese Variante eignet sich jedoch nicht für eine genaue Festlegung auf eine Zielgröße von n Datenpunkten.

Da die zweite Variante den Datensatz auf die informationsreichen Datenpunkte reduziert, eignet sie sich mehr als die erste Methode und wird daher in dieser Arbeit zur Datenanpassung verwendet. Als Grenzwert zweier benachbarter Sensorwerte wurde $t = 0.01$ experimentell ermittelt.

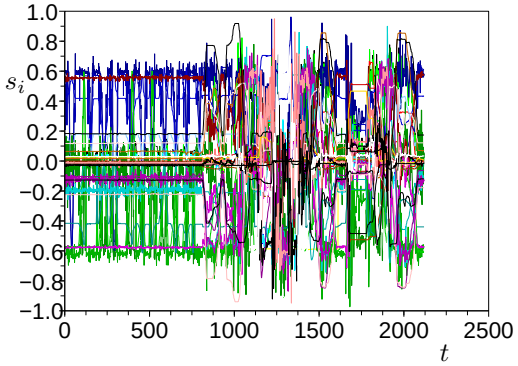
In Abbildung 5.1.3 sind die Ausgangsdaten und die $n-1$ euklidischen Nachbardistanzen abgebildet. Darin ist mit Rot der Grenzwert eingezeichnet, der über den Nachbardistanzen der „Ruhephasen“ der Roboterbewegung liegt, sich aber unterhalb der Nachbardistanzen der „Bewegungsphasen“ befindet. Darüber hinaus zeigt die Abbildung die Ergebnisse beider Datenreduktionsvarianten. Es ist erkennbar, dass die letzte Variante (siehe Abbildung 5.1.3(c)) mehr Information erhalten konnte als die erste (siehe Abbildung 5.1.3(b)).



(a) Ausgangsdaten und Nachbardistanzen mit rotem Grenzwert



(b) Jeder 8. Datenpunkt



(c) Nur Daten mit Nachbardistanzen > 0.01

Abbildung 5.1.3: *Zwei Varianten die Anzahl der Datenpunkte zu reduzieren.*

5.2 Zweidimensionale Visualisierungen der sensorischen Daten

In diesem Kapitel werden die verschiedenen dimensionsreduzierenden Algorithmen für die Erstellung einer zweidimensionalen Transformation von Bewegungsdaten des humanoiden Roboters (siehe Anhang) angewendet und untersucht.

Die 37-dimensionalen Beschleunigungs- und Winkelwerte des Roboters wurden in jeder Dimension intervallskaliert und mediangefiltert. Der mediangefilterte Datensatz mit ca. 14000 Datenpunkten wurde anschließend auf 2119 Datenpunkte reduziert (siehe Abschnitt 5.1). Aufgrund des hohen zeitlichen Aufwandes ab 2000 Datenpunkten werden zur Erstellung der zweidimensionalen Transformationen der ISOMAP- und der Kruskal-Algorithmus nicht verwendet. Somit werden die linearen Algorithmen PCA und CS und der nichtlineare Algorithmus LLE zur Darstellung der Sensordaten des humanoiden Roboters genutzt.

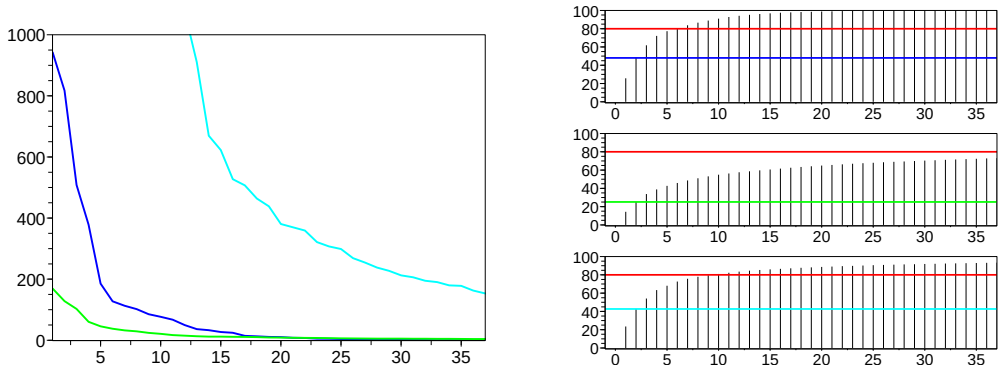
Lineare Transformationen

Beide linearen Algorithmen PCA und CS suchen nach den Eigenwerten, die die größtmögliche Variation im Datensatz abdecken. Für den CS-Algorithmus werden die Manhattan- und Chebyshev-Distanzen verwendet. Der Gebrauch der euklidischen Distanz würde andernfalls das gleiche Ergebnis wie die PCA bewirken. Es gilt:

$$\text{Manhattan: } \delta_{ij} = \sum_{s=1}^{d} \|x_{is} - x_{js}\| \ ,$$

$$\text{Chebyshev: } \delta_{ij} = \max_{i} \|x_{is} - x_{js}\|.$$

In Abbildung 5.2.1(a) sind für den PCA und die CS-Algorithmen die 37 größten Eigenwerte dargestellt. In Abbildung 5.2.1(b) sind die prozentualen Anteile der ersten Eigenwerte an der Gesamtvariation für jeden Algorithmus abgebildet. Die rote horizontale Linie zeigt die empfohlene 80%-Grenze an, die die ersten p Eigenwerte kumulieren. Die anderen horizontalen Linien zeigen den prozentualen Anteil an der Gesamtvariation für die ersten beiden Eigenwerte (Dimensionen) an. Demnach besitzt die PCA-Transformation im zweidimensionalen Raum mit ca. 50% die meiste Variation, gefolgt von der CS-Transformation mit Manhattan-Distanz mit ca. 40% und der CS-Transformation mit der Chebyshev-Distanz mit ca. 30%.



(a) Eigenwerte (b) Prozentualer Anteil an Gesamtvariation

Abbildung 5.2.1: *Die Eigenwerte des PCA- (Blau) und beider CS-Algorithmen (Chebyshev-Distanz: Grün, Manhattan-Distanz: Türkis) inklusive ihres prozentualen Anteils an der Gesamtvariation über alle* 1- *bis* 37*-dimensionale Zieltransformationen. Die roten Linien dienen zur grafischen Bestimmung der ersten Eigenwerte mit einer Gesamtvariation von über* 80%*.*

Unter den linearen zweidimensionalen Transformationen kann somit die PCA-Transformation die meiste Variation des Ausgangsdatensatzes abbilden. Anhand der 80%-Grenze [Jol86] kann mit Hilfe der PCA-Reduktion eine lineare Mannigfaltigkeit der Dimension $d \geq 5$ identifiziert werden. Bei den CS-Transformationen ist die Dimension der linearen Mannigfaltigkeit jedoch mit $d \geq 10$ gegeben.

In der Abbildung 5.2.2 ist die zweidimensionale PCA-Transformation dargestellt und die Abbildungen 5.2.3 und 5.2.4 zeigen die zwei Ergebnisse der zweidimensionalen CS-Transformationen. Die grünen Verbindungen zwischen allen Datenpunkten sind die Verbindungen zwischen dem Vorgänger und Nachfolger im Datensatz. Dazu sind zu 22 Datenpunkten (Rot umkreist) die zugehörige Körperhaltung (nächstliegendes Foto) gegeben.

Dass die Dimensionsreduktion selbst bei der PCA-Reduktion die Information nicht optimal erhalten konnte, zeigt sich exemplarisch an dem unteren Bild, in welchem der Roboter eine spagatähnliche Haltung aufweist. Dieser Datenvektor wurde in unmittelbarer Nähe zu der Körperstellung angeordnet, bei der der Roboter auf dem Rücken liegt. Bei einer optimalen Transformation hätten beide Datenpunkte weiter entfernt angeordnet werden müssen, da sich Gelenkstellung und Beschleunigungswerte stark unterscheiden.

Speziell benachbarte Datenpunkte mit großer Entfernung zeigen, dass die lineare Reduktion fehlerbehaftet ist, denn benachbarte Punkte im hochdimensionalen

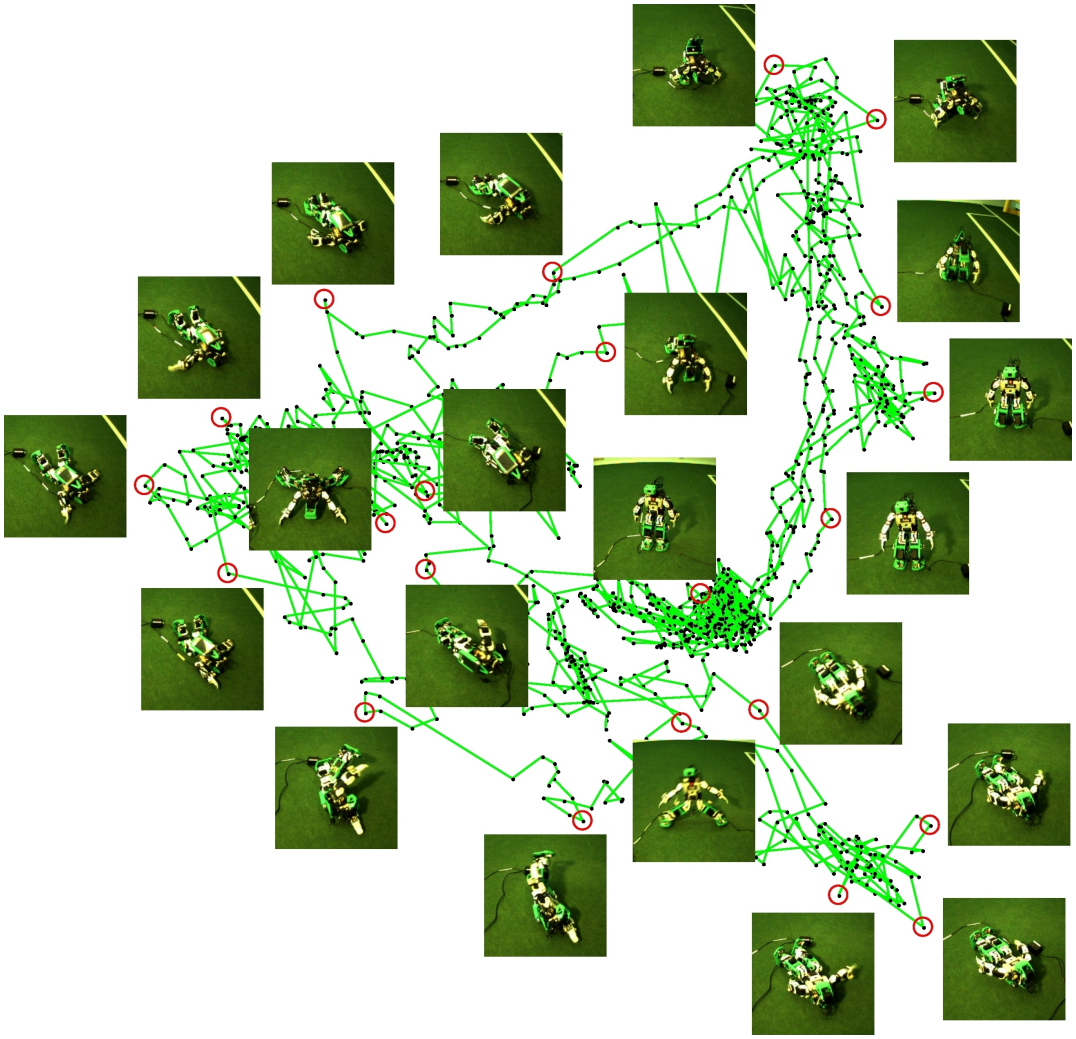


Abbildung 5.2.2: *Zweidimensionale PCA-Transformation aller Datenpunkte mit zugehörigen Fotos.*

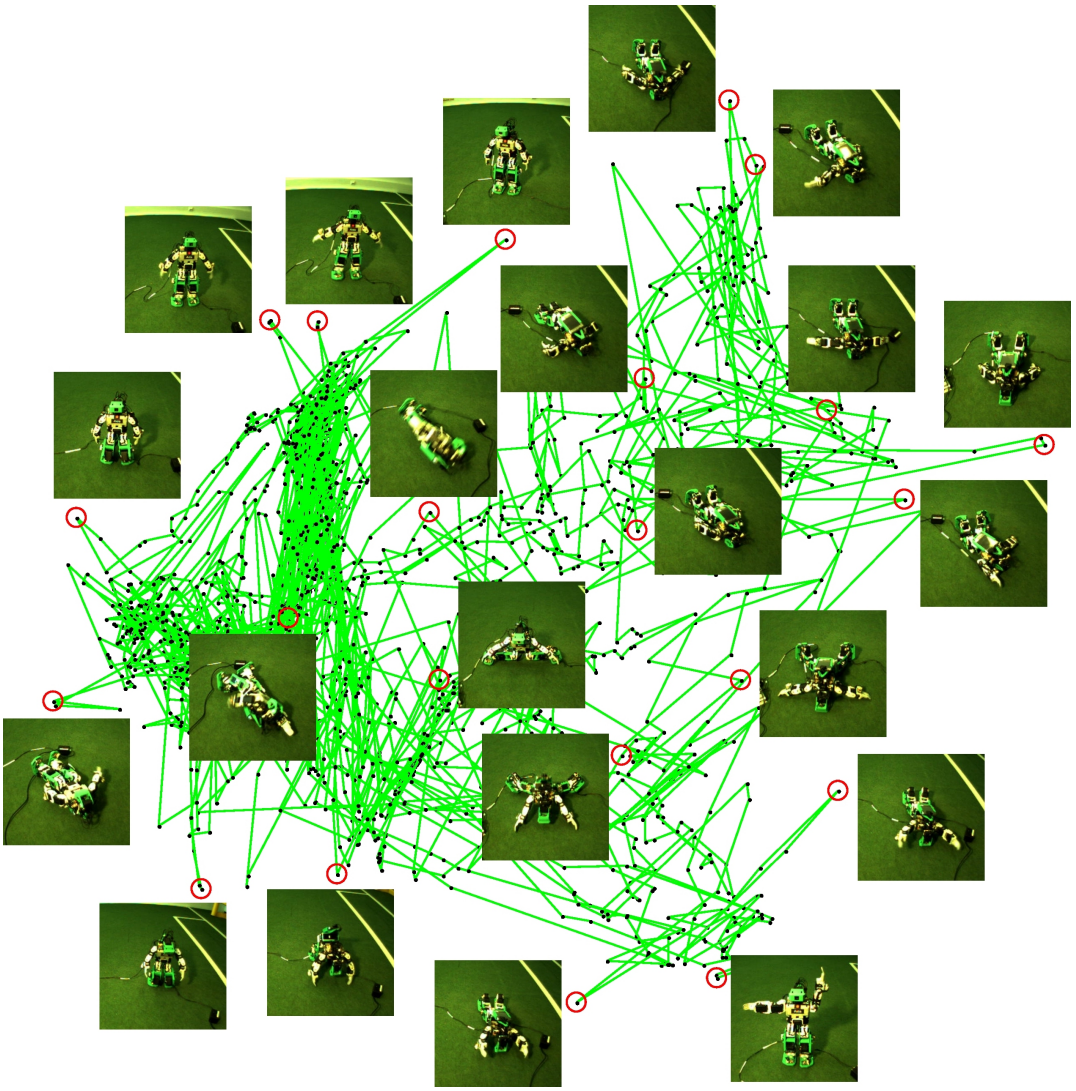


Abbildung 5.2.3: *Zweidimensionale CS-Transformation mit der Chebyshev-Distanz aller Datenpunkte mit zugehörigen Fotos.*

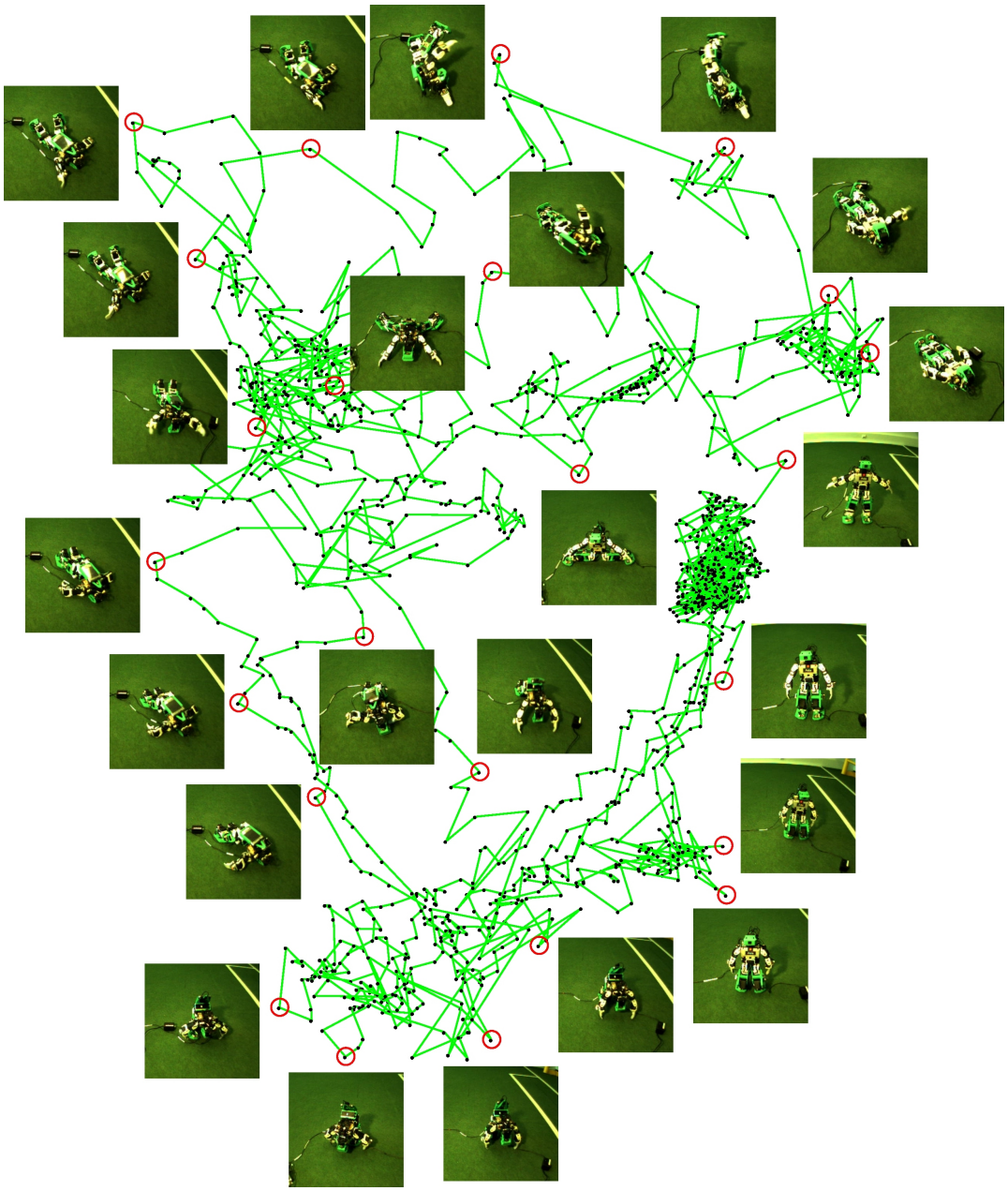


Abbildung 5.2.4: *Zweidimensionale CS-Transformation mit der Manhattan-Distanz aller Datenpunkte mit zugehörigen Fotos.*

Raum müssten auch nach der Transformation geringe Distanzen aufweisen. Mit diesem Wissen ist sofort ersichtlich, dass die CS-Reduktion mit der Chebyshev-Distanz das schlechteste Resultat liefert. Bei genauer Betrachtung der PCA- und CS-Transformation mit der Manhattan-Distanz können jeweils die gleichen drei markanten Körperhaltungen identifiziert werden, die die Netze aufspannen. Dazu zählen: Die Rückenlage, die Bauchlage mit gespreizten Armen und das Stehen mit vornübergebeugtem Rumpf.

Nichtlineare LLE-Transformation

Im Folgenden wird eine zweidimensionale Transformation mit Hilfe des nichtlinearen LLE-Algorithmus ermittelt. Anschließend wird die Dimension der nichtlinearen Mannigfaltigkeit der Daten näherungsweise bestimmt. Zur Erfassung der optimalen Nachbarschaft K (siehe Abschnitt 2.5) wird im Intervall von $K_i \in [1, 250]$ jedes Minimum des Rekonstruktionsfehlers der Gewichtsmatrix M bestimmt und anschließend das K_i gewählt, für das die LLE-Transformation die kleinste Residual Variance besitzt. Für die optimale K-Nachbarschaft ergibt sich $K_{opt} = 91$ (siehe Abbildung 5.2.5).

In der Abbildung 5.2.6 ist die zweidimensionale LLE-Transformation der Roboterdaten dargestellt. Zur Veranschaulichung wurden vier Pfade (Blau, Rot, Schwarz und Magenta) über die benachbarten Datenpunkte gewählt. Es wurden lokale, jedoch nicht direkte Nachbardatenpunkte (durch eine Linie miteinander verbunden) für den Pfad verwendet, da direkt benachbarte Datenpunkte zwangsweise ähnliche Körperstellungen aufweisen.

Für jeden dieser Pfade wurden die zugehörigen Roboterstellungen in die Grafik eingefügt. Der schwarze Pfad beginnt mit der Roboterstellung: Liegt auf dem Bauch, die Arme und Beine sind geschlossen. Der Pfad endet mit: Liegt auf der rechten Seite, die Arme und Beine sind geschlossen. Den Fotofolgen kann entnommen werden, dass für viele der dimensionsreduzierten Datenpunkte die lokalen Abhängigkeiten eingehalten werden konnten. Manche Datenpunkte wurden jedoch schlecht durch den Algorithmus in die zweidimensionale Karte eingebettet. So sind beispielsweise auf dem schwarzen Pfad eine seitlich liegende Körperhaltung und eine Bauchlage benachbart.

Im Allgemeinen sind all jene Datenpunkte durch den Algorithmus schlecht trans-

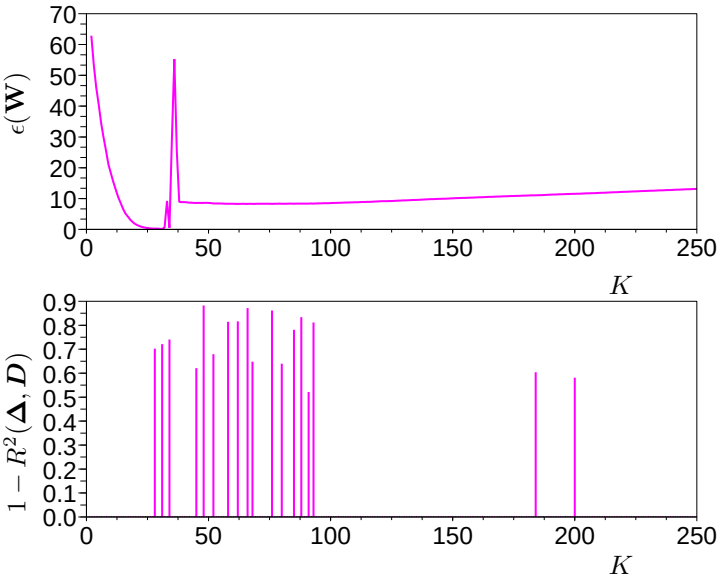


Abbildung 5.2.5: *Oberes Bild zeigt die Rekonstruktionsfehler für alle K_i und unteres Bild veranschaulicht die Residual Variance der LLE-Transformationen der K_i mit Rekonstruktionsfehlerminimum.*

formiert worden, deren direkte Nachbarn weit voneinander entfernt sind. Dies ist damit zu begründen, dass diese Datenpunkte $10ms$ nacheinander aufgenommen wurden, falls sie nicht bei der Datenfilterung entfernt wurden. Selbst im Falle einer Filterung stehen die großen Distanzen direkt benachbarter Datenpunkte im Kontrast zu dem durchschnittlich geringen Abstand zweier direkt benachbarter Datenpunkte. Durch die automatische Bestimmung der relativ großen Nachbarschaft mit $K_{opt} = 91$ werden nicht nur die lokalen linearen Abhängigkeiten im Datensatz gefunden, sondern globale lineare Abhängigkeiten. Das führt dazu, dass sich die LLE-Transformation mit steigendem K dem Ergebnis der PCA-Transformation anpasst. Jedoch ist gut zuerkennen, dass die Armbewegungen (rote Punkte) des Roboters im Vergleich zu den linearen Verfahren aufgerollt werden konnten.

Als nächstes wird die Dimension der Mannigfaltigkeit mit $K = 91$ näherungsweise bestimmt, auf der die Datenpunkte liegen. Dadurch kann die Eignung einer zweidimensionalen LLE-Transformation zur Datenvisualisierung beurteilt werden.

Dazu werden die Eigenwerte der lokalen Kovarianzen betrachtet (siehe Abschnitt 2.5). In Abbildung 5.2.7 wurde für alle K-lokalen Kovarianzen (aller n Datenpunkte)

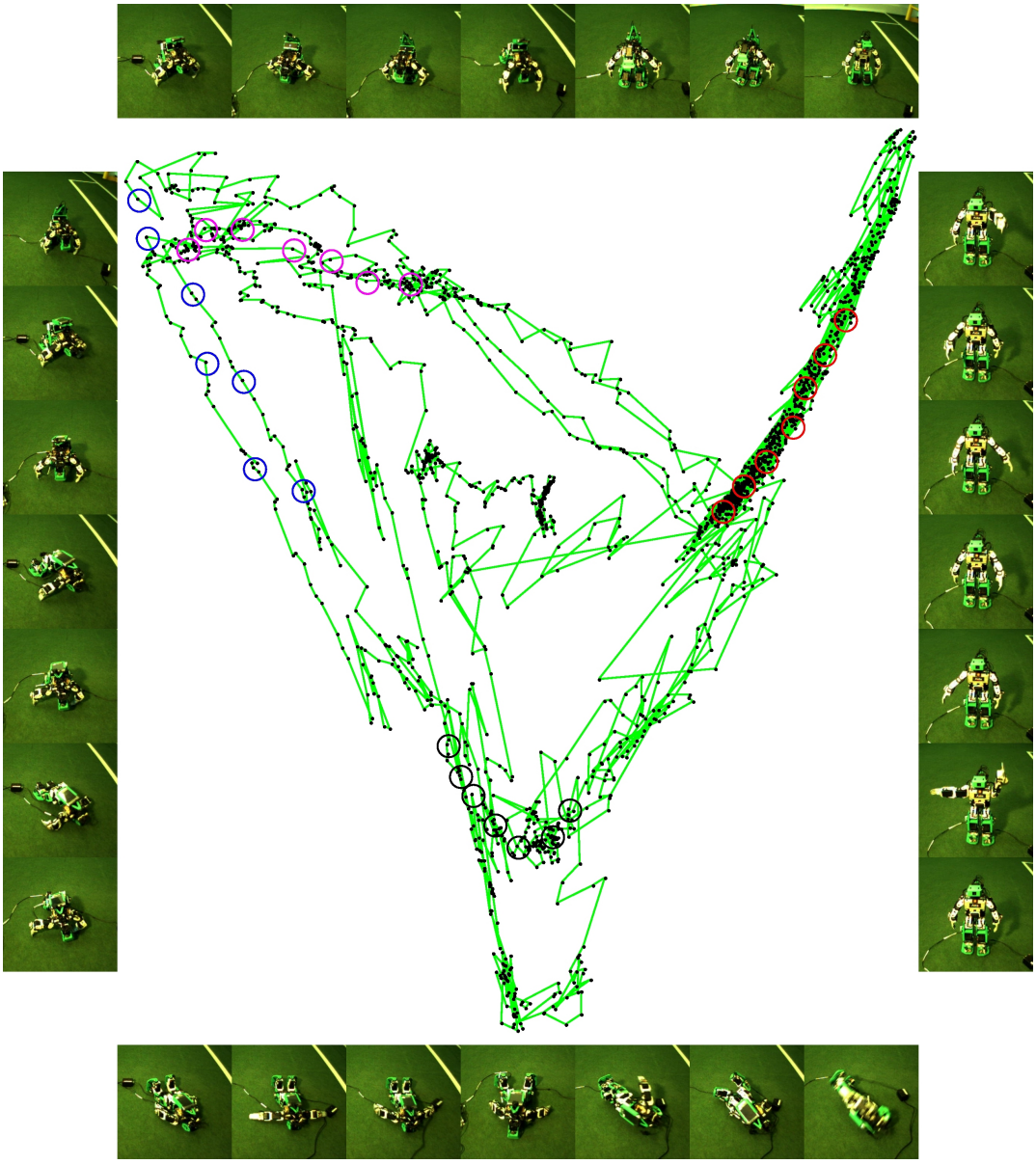


Abbildung 5.2.6: *Zweidimensionale LLE-Transformation aller Datenpunkte mit zugehörigen Fotos. Jede Fotofolge (links, unten, rechts und oben) entspricht den farblich markierten Datenpunkten (Blau, Schwarz, Rot und Magenta) in der vertikalen bzw. horizontalen Reihenfolge. Jede grüne Kante verbindet zwei zeitlich aufeinander folgende Datenpunkte.*

der durchschnittliche Anteil der ersten kumulierten Eigenwerte an der Gesamtsumme berechnet und ausgegeben (blaue Kurve). Für $K = 91$ beträgt die durchschnittliche Summe der ersten beiden Eigenwerte 66% und erst ab dem vierten Eigenwert über 80% der Gesamtvariation.

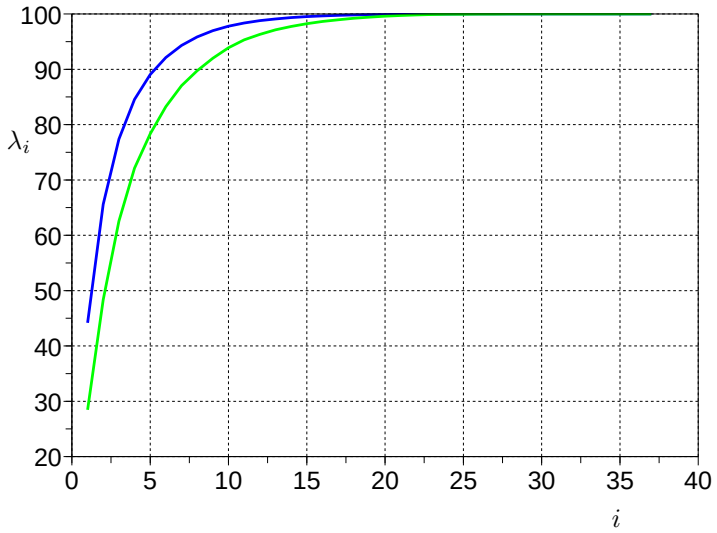


Abbildung 5.2.7: *Durchschnittlicher Anteil der $K = 91$ lokalen Varianz der ersten 40 Eigenwerte, über alle Punkte (Blau) und über die Punkte mit lokaler Kovarianz mit maximaler Dimension (Grün).*

Zur Detektion von möglichen Überkreuzungen nichtlinearer Mannigfaltigkeiten bzw. vereinzelt maximalen, Dimensionen der Mannigfaltigkeit wurden die K-Nachbarschaften gewählt, die eine maximale Dimension der Mannigfaltigkeit andeuten (grüne Kurve). Dafür wurde die durchschnittliche Summe der ersten d Eigenwerte ausgegeben, für die der Anteil der ersten Eigenwerte ein maximales d mit $t_d >= 80$ ergab. Für $K = 91$ beträgt der Anteil der Summe der ersten $d = 6$ Eigenwerte an der Gesamtsumme über 80%. Werden die durchschnittlichen und die maximalen durchschnittlichen Dimensionsgrößen der lokalen Kovarianzen berücksichtigt, ergibt sich eine Dimension der Mannigfaltigkeit von $4 \leq d \leq 6$.

Trotz der Bestimmung einer großen Nachbarschaft mit $K_{opt} = 91$ ist an dem „Ausbreiten“ der Armbewegungen des Roboters (Datencluster in der Nähe der roten Kreise) zu erkennen, dass dort die Daten auf einer stark nichtlinearen Mannigfaltigkeit verteilt sind.

6 Zusammenfassung von Teil I

Im theoretischen Teil des ersten Themenbereichs wurden ausgewählte dimensionsreduzierende Algorithmen nach dem folgenden Schema analysiert:

- Schrittweise mathematische Beschreibung
- Gegenüberstellung eines Datensatzes und seiner Transformation
- Bestimmung der Zeitkomplexität des Verfahrens
- Analyse der Zieldimension der Transformation

Die linearen dimensionsreduzierenden Verfahren *Principal Component Analysis* (PCA), *Classical Scaling* (CS) und das Kruskal-Verfahren extrahieren je nach Parameterwahl in unterschiedlicher Qualität die lineare Mannigfaltigkeit (mit geringer Dimensionalität) eines höherdimensionalen Datenraumes. Jeder dieser Algorithmen weist Stärken auf, die für verschiedene Anwendungen sehr vorteilhaft sind:

1. Ein PCA-transformierter Datensatz weist unter den linearen Algorithmen entsprechend der euklidischen Metrik den geringsten Informationsverlust auf. Das PCA-Verfahren besitzt die geringste Zeitkomplexität im typischen Fall, d. h. wenn die Anzahl n der Datenpunkte größer ist, als die zu reduzierende Dimension d.

2. Das CS bietet die Möglichkeit, die Datenstruktur durch nachträgliches Gewichten von Distanzen gezielt anzupassen. Im untypischen Fall, wenn $n < d$ ist, besitzt das CS die geringste Zeitkomplexität der linearen Algorithmen.

3. Die gradientenbasierte Methode von Kruskal ermöglicht das Erzeugen einer Transformation mit einer vorab definierten Zielgüte, so denn diese erreichbar ist. Dieses Verfahren besitzt stets die höchste Zeitkomplexität der linearen Algorithmen.

Die nichtlinearen Verfahren *Isometric Feature Mapping* (ISOMAP) und *Locally Linear Embedding* (LLE) extrahieren je nach Parameterwahl in unterschiedlicher Qualität die nichtlineare Mannigfaltigkeit (mit geringer Dimensionalität) eines höherdimensionalen Datenraumes. Beide Verfahren unterscheiden sich dabei hinsichtlich der Transformation der lokalen bzw. globalen Distanzen der Mannigfaltigkeit:

1. Aufgrund der Verwendung von geodätischen Distanzen kann die ISOMAP-Methode gewährleisten, dass entfernte Distanzen (in der Mannigfaltigkeit) nach der Transformation erhalten bleiben. Jedoch werden dabei die lokalen Abstände verzerrt.

2. Durch den lokalen Ansatz des LLE-Verfahrens verhält sich dieser genau entgegengesetzt. Lokale Abstände werden nach der Transformation erhalten und globale verzerrt.

Die Zeitkomplexität der Berechnung ist bei der LLE im typischen Fall mit $K < n$ geringer als bei der ISOMAP.

Im praktische Teil führten nähere Untersuchungen der Ausführungsgeschwindigkeit der in Scilab implementierten Algorithmen und der erforderlichen Filterungen der sensorischen Rohdaten zu folgenden Erkenntnissen:

1. Nach den Zeitmessungen zu den einzelnen dimensionsreduzierenden Algorithmen mit zufälligen Daten sind der Kruskal- und der ISOMAP-Algorithmus für die Dimensionsreduktion mit einer Datenanzahl $n >= 500$ unter den gegebenen rechentechnischen Voraussetzungen ungeeignet. Die jeweiligen Programme hätten für die Berechnung der Transformation einen Zeitbedarf von mehr als einem Tag benötigt.

2. Alle Sensorwerte sollten mit einer Fenstergröße von fünf Datenpunkten medianangefiltert werden.

3. Die ursprünglichen 14000 Datenpunkte mit längeren Steh- und Liegephasen des Roboters können durch Filterungen informationserhaltend auf die 2119 Datenpunkte reduziert werden, die signifikante Informationssprünge aufweisen.

Unter Berücksichtigung der oben genannten Einschränkungen konnte erfolgreich die vorverarbeitete 37-dimensionale Datensequenz der humanoiden Roboterbewegung in einen zweidimensionalen Raum transformiert werden. Folgende Ergebnisse wurden erzielt:

1. Bei den linearen Dimensionsreduktionen erzielt die PCA die besten Ergebnisse. Sie kann ca. 50% der globalen Variation bzw. Information der Ausgangsdaten in zwei Dimensionen reproduzieren. Um eine globale Variation von ca. 80% abzudecken, sind jedoch mindestens fünf Dimensionen nötig.

Transformation	Global/Lokal	Procrustes	ρ_S	$1 - R^2(\boldsymbol{\Delta}, \boldsymbol{D})$
$CS_{Chebyshev}$	Global	4790.9	0.6156	0.5935
	Lokal	8.4	0.6085	0.5911
$CS_{Manhattan}$	Global	3166.1	**0.8420**	0.2944
	Lokal	6.9	0.6128	0.5931
PCA	Global	**2920.5**	0.8369	**0.2895**
	Lokal	**6.5**	0.6039	0.5892
LLE	Global	5184.7	0.6649	0.5213
	Lokal	8.7	**0.6560**	**0.5500**

Tabelle 6.0.1: *Bewertung der Güte der Transformationen lokal und global mit Procrustes, Spearman's Rho und Residual Variance. Die besten Fehlerwerte wurden fett hervorgehoben.*

2. Das nichtlineare LLE-Verfahren kann mit der Nachbarschaft von $K = 91$ eine durchschnittliche lokale Variation der ersten beiden Eigenwerte von 66% erzielen. Die optimale Dimension d der nichtlinearen Mannigfaltigkeit, auf der die Daten verteilt sind, ist jedoch erst mit mindestens $4 \leq d \leq 6$ gegeben.

   Ungeachtet des guten Ergebnisses mit PCA konnte durch das LLE-Verfahren trotz Dimensionsreduktion auf die gleiche Dimension (zwei) im Bereich der Armbewegung eine signifikante Information bereitgestellt werden. Genauer gesagt konnte eine nichtlineare Mannigfaltigkeit im Bereich der Armbewegungen gefunden und „ausgebreitet“ werden.

Der Tabelle 6.0.1 sind die globale und lokale Güte je nach Verfahren zu entnehmen. Die lokalen Gütewerte ergeben sich aus dem arithmetischen Mittel der Gütewerte bestimmt für jeden Datenpunkt zuzüglich seiner K Nachbarschaftsdatenpunkte. So bestätigen die Gütewerte das PCA-Verfahren als bestes lineares und das LLE-Verfahren als einziges praktisch einsetzbares nichtlineares Transformationsverfahren.

Teil II

Ausbildung sensorischer Karten durch neuronale Gase

7 Einteilung der künstlichen neuronalen Netze

Als künstliche neuronale Netze werden die Modelle bezeichnet, welche mit dem Ziel der Beschreibung biophysikalischer Prozesse zwischen Neuronen in der Informationsverarbeitungskette im Gehirn konzipiert wurden. Trotz der großen Anzahl an neuronalen Ansätzen lassen sich einige Merkmale extrahieren, die alle Modelle besitzen:

- Identische Funktionseinheiten (die Neuronen) mit Ein- und Ausgängen.
- Zwischen den Funktionseinheiten existieren Verbindungen.
- Die Ausgabesignale werden aus den Eingabesignalen berechnet.

Das erste neuronale Modell wurde 1933 von Rashevsky entwickelt. Er verwendete ein zeitkontinuierliches Modell, in welchem jedes Neuron mittels zweier linearer Differentialgleichungen das Eingabesignal anregen und hemmen kann. Zur Ausgabe wird ein Grenzwertoperator benutzt, der entweder 0 oder 1 als Ausgabe berechnet [Cul07]. Im Jahr 1943 entwickelten McCulloch und Pitts [MP43] ein zeitdiskretes statisches Modell, bei dem jedes Neuron die Eingabesignale gewichtet aufsummiert und mittels eines Grenzwertoperators entweder 0 oder 1 als Ausgabe erzeugt. Dieses Modell wurde 1958 durch das Perzeptron von Rosenblatt [Ros58] erweitert, bei welchem sich die Gewichte der Eingänge entsprechend verschiedener Lernregeln innerhalb einer gewissen Trainingszeit anpassen können. Darüber hinaus muss die Ausgabefunktion nicht strikt boolescher Natur sein. Die Perzeptronen können zu ein- oder mehrschichtigen Netzen angeordnet werden. Kohonen hat die neuronalen Netze in drei Klassen wie folgt unterteilt[Koh01]:

1. **Signal-Transfer-Netzwerk.** Dieser Netztyp transformiert die Eingangssignale $\mathbf{x} = (x_1, x_2, ..., x_n)^T$ zu den Ausgangssignalen $\mathbf{y} = (y_1, y_2, ..., y_m)^T$. Dabei ist jedes Ausgangssignal $y_i(t)$ zum Zeitschritt t von den aktuellen und allen vorhergehenden Eingangssignalen $x_i(t'), t' \leq t$ abhängig.

2. **Zustand-Transfer-Netzwerk.** Bei diesem Netztyp werden die Ausgangssignale y_i als Eingangssignale erneut ins Netz eingekoppelt, so dass gilt:

$$y(t+1) = f[\mathbf{x}(t), \mathbf{y}(t)].$$

   Damit können für das gleiche Eingabesignal verschiedene stabile Ausgabesignale erzeugt werden, diese werden Attraktoren genannt. So wird ein Startzustand

zum Zeitschritt $t = 0$ mit $\mathbf{y} = \mathbf{y}(0)$ festgelegt. Durch Einspeisung der Eingabesignale vollzieht der Zustand von $\mathbf{y}$ eine Reihe von Transitionen oder endet in einem Attraktor, welcher das Endergebnis darstellt.

3. **Wettbewerbslernen-Netzwerke.** Dieser Netztyp besteht aus einer Anzahl von Neuronen, die topologisch miteinander verbunden sind. Alle Neuronen erhalten die gleichen Eingabesignale. Mit Hilfe von lateralen Interaktionen wird ein „Gewinnerneuron“ (oder ein lokaler Bereich von benachbarten Gewinnerneuronen) aktiviert, das das Eingabesignal lernt. Nach Anwendung mehrerer Eingaben wird jede Netzregion für eine spezielle Domäne von Eingaben sensibilisiert. Eine mathematische Umsetzung dieses neuronalen Modells ist das vektorbasierte neuronale Netz. Dieses besteht aus einer begrenzten Anzahl von n-dimensionalen Vektoren, den Codebuch- oder Referenzvektoren. Diese besitzen die gleiche Dimensionalität wie das Eingangssignal $\mathbf{x}$. Um das Netz zu trainieren, wird zu jedem Eingabevektor $\mathbf{x}$ der nächstliegende Referenzvektor $\mathbf{w}_c$ bestimmt und angeglichen.

8 Vektorbasierte neuronale Netze

8.1 Verwandte wissenschaftliche Arbeiten

Ein vektorbasiertes neuronales Netzwerk ist das 1982 von Kohonen entwickelte Verfahren *Self-Organizing Map* (SOM) [Koh01]. Dieses Netzwerk besteht aus einer festen Anzahl von Referenzvektoren und einer festen Verbindungsstruktur zwischen den Referenzvektoren. Das Ziel des SOM-Netzwerkes ist die optimale Verteilung der Referenzvektoren entlang der Eingabedatenstruktur.

Ein anderes vektorbasiertes neuronales Netzwerk ist das Modell *Neural Gas* (NG). Es wurde 1991 von Martinetz und Schulten entwickelt [MS91]. Die Aufgabe dieses Netzes besteht in der Quantisierung und topologischen Beschreibung des Eingaberaumes. Im Gegensatz zum SOM wird eine variable Netzstruktur verwendet.

Fritzke entwickelte im Jahr 1995 das Modell *Growing Neural Gas* (GNG) [Fri95] und im Jahr 1997 das Modell *Growing Neural Gas with Utility criterion* (GNG-U) [Fri97]. Beide Netzwerke sind jeweils eine Erweiterung des NG-Netzwerkes und besitzen eine wachsende variable Netzwerkstruktur.

Auf Basis wachsender neuronaler Netzwerke hat Toussaint in den Jahren 2004 [Tou04] und 2006 [Tou06] ein Modell entwickelt, das eine sensomotorische Karte während einer Trainingsphase im Eingabedatenraum ausprägt und diese zur zielgerichteten Punktbewegung in der Planungsphase nutzt (siehe Kapitel 12).

Für die kontinuierliche Dynamik seines System verwendet Toussaint die dynamischen Felder. Das Verhalten von dynamischen Feldern mit lateraler Hemmung wurde erstmals 1977 von Amari [Ama77] untersucht und charakterisiert.

Unter dem Aspekt der Erzeugung von Bewegungsparametern wurden weitere Ergebnisse für die Verwendung der dynamischen Feldern im Jahre 2002 von Erlhagen und Schöner [ES02] zusammengetragen.

8.2 Netzwerkstruktur und Gewinnerneuron

In diesem und den folgenden beiden Unterabschnitten werden die allgemeinen Eigenschaften der verktorbasierten neuronalen Netze beschrieben und damit eng verbundene Begriffe erklärt. Fritzke hat in seiner Arbeit [Fri98] eine ausführliche Einführung zu den vektorbasierten neuronalen Netzen gegeben, in der viele der folgenden Definitionen verwendet wurden.

Das vektorbasierte neuronale Netz besteht aus einer Menge

$$\mathcal{A} = \{c_1, ..., c_n\}$$

von N Neuronen. Jedem Neuron $c \in \mathcal{A}$ ist ein d-dimensionalen Referenzvektor

$$\mathbf{w}_c \in \mathbb{R}^d$$

zugeordnet. Die Referenzvektoren besitzen die gleiche Dimension wie die Eingangssignale bzw. Eingabevektoren.

Zwischen den Neuronen können ungewichtete Verbindungen der Menge

$$\mathcal{C} \subset \mathcal{A} \times \mathcal{A}$$

existieren. Diese Verbindungen sind symmetrisch und es gilt:

$$(i, j) \in C \Longleftrightarrow (j, i) \in \mathcal{C} \ .$$

Über diese Verbindungen können Nachbarschaftsrelationen beschrieben werden. So besitzt jedes Neuron c eine Menge

$$\mathcal{N}_c = \{i \in \mathcal{A} | (c, i) \in \mathcal{C}\}$$

von direkt benachbarten Neuronen i.

Die auf das neuronale Netz angewendeten Trainingsdaten liegen entweder als endliche Menge

$$\mathcal{D} = \{\boldsymbol{\xi}_1, ..., \boldsymbol{\xi}_M\},\ \boldsymbol{\xi}_i \in \mathbb{R}^d$$

von Eingabesignalen bzw. -Vektoren vor oder mit einer Wahrscheinlichkeitsdichte

$$p(\boldsymbol{\xi}),\ \boldsymbol{\xi} \in \mathbb{R}^d \ .$$

Die zentrale Aufgabe des Wettbewerbslernens ist die Bestimmung des Gewinnerneurons. So wird das Gewinnerneuron $s(\boldsymbol{\xi})$ für die Eingabe $\boldsymbol{\xi}$ mit

$$s(\boldsymbol{\xi}) = \arg\min_{i \in \mathcal{A}}(\|\boldsymbol{\xi} - \mathbf{w}_i\|)$$

berechnet. Es lässt sich zu jeder Eingabe genau ein Gewinner bestimmen, falls jedoch mehrere Referenzvektoren den gleichen Abstand zum Eingabesignal besitzen, wird das Neuron als Gewinner deklariert, das den kleinsten Index besitzt. Dadurch werden die vektorbasierten Netze auch als distanzbasierte neuronale Netze bezeichnet.

8.3 Voronoigebiet und Delaunay-Triangulation

Als Voronoigebiet $\mathcal{V}_i$ eines Referenzvektors $\mathbf{w}_i$ wird die Menge

$$\mathcal{V}_i = \{\boldsymbol{\xi} \in \mathbb{R}^d | i = \arg\min_{j \in \mathcal{A}} \|\boldsymbol{\xi} - \mathbf{w}_j\|\}$$

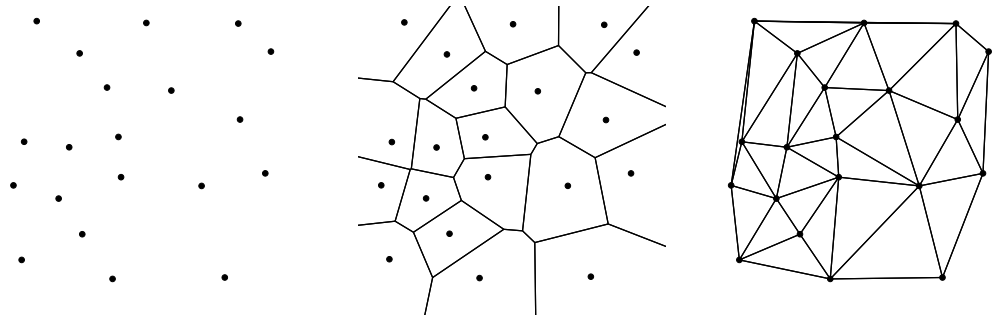
aller Punkte aus $\mathbb{R}^d$ definiert, für die $\mathbf{w}_i$ der nächstliegende Referenzvektor ist. Analog wird das Voronoigebiet $\mathcal{V}_c$ eines Neurons c als das Voronoigebiet des entsprechenden Referenzvektors definiert. Den Eingabevektoren, die mehrere nächstliegende Referenzvektoren besitzen, wird der Referenzvektor mit dem kleinsten Index zugeordnet. Die Partition des Eingaberaumes $\mathbb{R}^d$ durch alle Voronoigebiete $\mathcal{V}_i$ wird als Voronoitesselation bezeichnet. Als Delaunay-Triangulation wird der Graph bezeichnet, der aus den Kanten zwischen allen Referenzvektoren mit benachbarten Voronoigebieten entsteht. Im Falle einer endlichen Datenmenge $\mathcal{D}$ wird die Voronoimenge des Neurons i als die Menge

$$\mathcal{R}_i = \{\boldsymbol{\xi} \in \mathcal{D} | s(\boldsymbol{\xi}) = i\}$$

aller Eingabevektoren aus $\mathcal{D}$ definiert, für die das Neuron i der Gewinner ist. Die Voronoimengen bilden somit eine Partition von $\mathcal{D}$. Es gilt:

$$\bigcup_{i \in \mathcal{A}} \mathcal{R}_i = \mathcal{D},\ \mathcal{R}_i \cap \mathcal{R}_j = \varnothing \text{ für } i, j \in \mathcal{A} \text{ und } i \neq j\ .$$

In Abbildung 8.3.1 ist der Zusammenhang zwischen Referenzvektoren und ihren Voronoigebieten zuzüglich ihrer Delaunay-Triangulation abgebildet.



(a) Referenzvektoren (b) Voronoigebiete (c) Delaunay-Triangulation

Abbildung 8.3.1: *Referenzvektorverteilung in $\mathbb{R}^2$, mit entsprechenden Voronoigebieten und dazugehöriger Delaunay-Triangulation [Fri98].*

8.4 Aktivierungsregion eines Neurons

Die Aktivierungsregion eines Neurons ist der Bereich des Eingaberaums $\mathbb{R}^n$ aus dem ein Eingabevektor stammen muss, um dieses Neuron zu aktivieren. Bei den vektorbasierten neuronalen Netzen mit hartem Wettbewerbslernen wird nur der Referenzvektor $\mathbf{w}_c$ des Gewinnerneurons c aktiviert bzw. zum Eingabesignal hin verschoben. Ein Beispiel einer kontinuierlichen Aktivierung eines Neurons in Abhängigkeit zum Abstand des Eingabevektor ist in der Arbeit[Tou04] von Toussaint durch die Gaußfunktion

$$h_c(\boldsymbol{\xi}) = \exp\left(-\frac{\|\boldsymbol{\xi} - \mathbf{w}_c\|^2}{2\sigma^2}\right) \tag{8.1}$$

gegeben. Im Falle eines weichen Wettbewerbslernens werden alle Referenzvektoren i bezüglich ihres Abstandes zum Eingabevektor unterschiedlich stark aktiviert. Dabei kann die Aktivierung vom Abstand des Gewinnerneurons c zu dem zu aktivierenden Neuron i abhängen[Koh01]:

$$h_i(\boldsymbol{\xi}) = \exp\left(-\frac{\|\mathbf{w}_i - \mathbf{w}_c\|}{2\sigma^2(t)}\right) .$$

Weiterhin kann der Rang k des zu aktivierenden Neurons innerhalb der sortierten Abstandsliste aller Referenzvektoren zu dem Eingabevektor die Aktivierung beeinflussen [Fri98]:

$$h_c(\boldsymbol{\xi}) = \exp\left(\frac{1-k}{\sigma(t)}\right) .$$

Dabei kann σ sowohl als ein konstanter als auch zeitabhängiger Parameter in der Aktivierungsfunktion definiert sein.

Zusammenfassend besitzen die vektorbasierten neuronalen Netze eine Aktivierungsregion in Form einer Glockenkurve, da nur lokale Eingabesignale ein Neuron stark aktivieren können (siehe Abbildung 8.4.1).

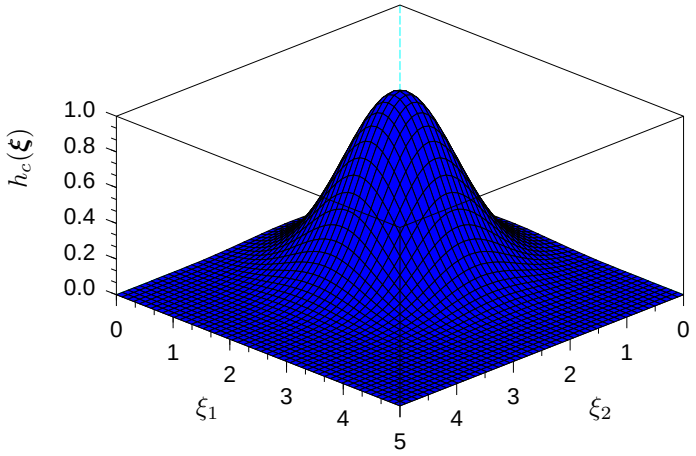


Abbildung 8.4.1: *Lokale Aktivierungsregion eines vektorbasierten Neurons (siehe Gleichung 8.1).*

8.5 Ziele vektorbasierter neuronaler Netzwerke

Es gibt verschiedene Ziele vektorbasierter neuronaler Netze. So verweist Fritzke auf folgende Ziele: Fehlerminimierung, Entropiemaximierung, Dichteschätzung, Clustering und Datenvisualisierung [Fri98]. In dieser Arbeit werden nur zwei Ziele der vektorbasierten Netzwerke unter dem Gesichtspunkt des unüberwachten Lernens mit variablen Netzwerkstrukturen behandelt. Die Fehlerminimierung ist ein entscheidendes wenn nicht sogar das entscheidendste Qualitätsmerkmal vektorbasierter neuronaler Netze. Die Entropiemaximierung ist ein Ziel solcher Netzwerke, das über einen Parameter im Algorithmus bestimmter neuronaler Gase (GNG und GNG-U) ein- und damit zwangsläufig die Fehlerminimierung ausgeschaltet wird.

Ein Ziel vektorbasierter Netzwerke ist die Fehlerminimierung der durchschnittlichen Distanzen zwischen jedem Eingabevektor $\boldsymbol{\xi}$ der Trainingsdaten $\mathcal{D}$ und dem Referenzvektor $\mathbf{w}_c$ des entsprechenden Gewinnerneurons c aus der Menge aller Refe-

renzvektoren $\mathcal{A}$. Falls die Eingabedaten als endliche Menge vorliegen, ist der durchschnittliche Fehler durch [Fri98]:

$$E = \frac{1}{|\mathcal{D}|} \sum_{c \in \mathcal{A}} \sum_{\boldsymbol{\xi} \in \mathcal{R}_c} \|\boldsymbol{\xi} - \mathbf{w}_c\|^2 \tag{8.2}$$

definiert. Im Falle, einer kontinuierlichen Verteilung der Datenvektoren gilt:

$$E = \sum_{c \in \mathcal{A}} \int_{\mathcal{V}_c} \|\boldsymbol{\xi} - \mathbf{w}_c\|^2 p(\boldsymbol{\xi}) d\boldsymbol{\xi} \ .$$

Mit Hilfe dieses Zieles wird eine Komprimierung der gegebenen Eingabedaten mit minimierten Informationsverlust erwirkt. So werden die $|\mathcal{D}|$ vielen Eingabedaten auf die $|\mathcal{A}|$ wenigen Referenzvektoren der entsprechenden Gewinnerneuronen mit dem Informationsverlust E reduziert.

Ein weiteres entscheidendes Merkmal ist die Entropie. Bei der Entropiemaximierung besteht im Gegensatz zur Fehlerminimierung die Aufgabe darin, die Referenzvektoren im Eingaberaum so zu verteilen, dass jedes Neuron c mit der gleichen Wahrscheinlich $P(c)$ das Gewinnerneuron für einen zufälligen Eingabevektor $\boldsymbol{\xi}$ ist [Fri98]:

$$P(c) = \frac{1}{|\mathcal{A}|}, \text{ mit } c = s(\boldsymbol{\xi}) \ .$$

Für eine Wahrscheinlichkeitsverteilung der Eingabedaten gilt analog für jedes Voronoigebiet $\mathcal{V}_c$:

$$\int_{\mathcal{V}_c} p(\boldsymbol{\xi}) d\boldsymbol{\xi} = \frac{1}{|\mathcal{A}|} \ .$$

Bei einer endlichen Datenmenge muss jede Voronoimenge $\mathcal{R}_c$ die gleiche Kardinalität besitzen und es gilt:

$$\frac{|\mathcal{R}_c|}{|\mathcal{D}|} = \frac{1}{|\mathcal{A}|} \ .$$

Nach Shannon [Sha48] lässt sich die gleichwahrscheinliche Referenzvektorverteilung als Maximierung der Entropie H über alle Neuronen c definieren:

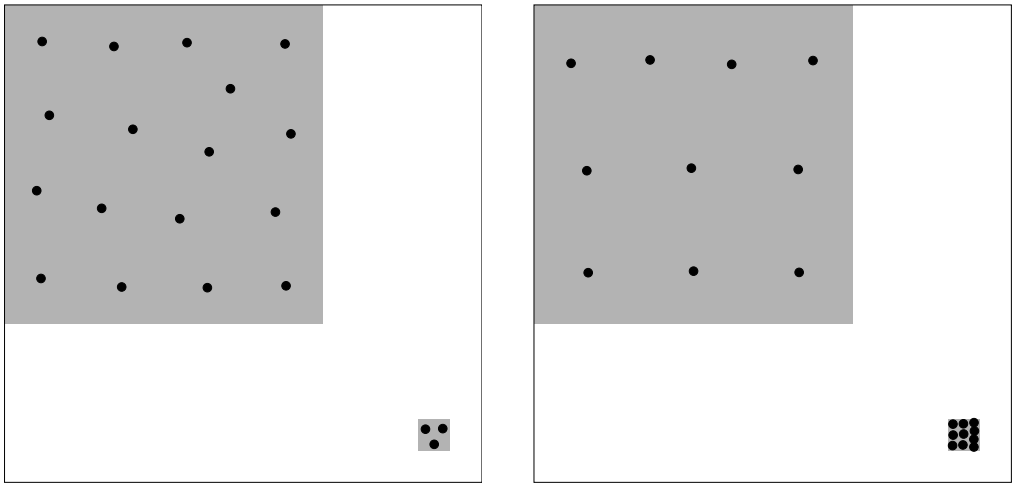
$$H = -\sum_{c \in \mathcal{A}} P(c) \log_b \left(P(c)\right) \ .$$

Die Einheit der Entropie wird entsprechend der Basis des Logarithmus in Bits ($b = 2$),

Nats ($b = \mathrm{e}$) oder dezimale Zahlen ($b = 10$) angegeben. Bei einer Gleichverteilung ist der maximale Entropiewert in gleich $\log_b(|\mathcal{A}|)$. Das Ergebnis der Verteilung der Referenzvektoren ergibt eine höhere Dichte von Referenzvektoren in den Regionen des Eingaberaumes, in denen die Eingabewahrscheinlichkeit höher ist.

Im Falle einer gleichverteilten Eingabewahrscheinlichkeit im Eingaberaum ist die Referenzvektorverteilung nach der Fehlerminimierung gleich der nach der Entropiemaximierung, andernfalls sind beide Ziele nicht miteinander vereinbar.

In der Abbildung 8.5.1 hat Fritzke ein Beispiel für die konkurrierenden Ziele der Fehlerminimierung und Entropiemaximierung konstruiert [Fri98]. Es wurden für zwei Referenzvektorverteilungen der Fehler E und die Entropie H für 20000 zufällig erzeugte Signale gemessen. Dabei wurde eine uniforme Wahrscheinlichkeitsdichte in beiden Quadraten gewählt. Die erste Referenzvektorverteilung besitzt zwar einen um 37% kleineren durchschnittlichen Fehler (siehe Abbildung 8.5.1(a)), aber eine um elf Prozent geringere Entropie als die zweite Referenzvektorverteilung (siehe Abbildung 8.5.1(b)).



(a) Fehlerminimierung: $E = 0.0024$, $H = 2.657$

(b) Entropiemaximierung: $E = 0.0038$, $H = 2.990$

Abbildung 8.5.1: *Unvereinbarkeit von Fehlerminimierung und Entropiemaximierung [Fri98]. Beide Quadrate besitzen die gleiche Wahrscheinlichkeitsdichte $p(\boldsymbol{\xi})$. Es wurden 20 Referenzvektoren entsprechend den beiden unterschiedlichen Ansätzen verteilt und der Fehler E sowie die Entropie H für* 20000 *zufällig erzeugte Eingabesignalen gemessen.*

9 Verfahren neuronaler Gase

9.1 Neuronales Gas

Das *Neural Gas* (NG) ist ein vektorbasierter neuronaler Netztyp und wurde von Martinetz und Schulten [MS91] entwickelt. Dessen Funktionalität liegt darin, eine feste Anzahl von Referenzvektoren gleichmäßig in einem beliebig geformten n-dimensionalen Eingaberaum zu verteilen und eine Topologie zwischen den Referenzvektoren auszubilden. Diese Verbindungen bilden einen Untergraph der Delaunay-Triangulation aus. Diese Triangulation hat die Besonderheit, dass sie optimal zur Funktionsinterpolation geeignet ist [Fri98] und damit optimal den Eingaberaum approximiert. Der NG-Algorithmus arbeitet nach einem iterativen Schema und arbeitet pro Zeitschritt die folgenden Schwerpunkte ab, bis eine maximale Anzahl t_max an Zeitschritten durchlaufen wurden:

1. Zu jedem Eingabevektor $\boldsymbol{\xi}$ werden die Abstände $\boldsymbol{\xi} - \mathbf{w}_i$ zu allen Referenzvektoren $\mathbf{w}_i$ aufsteigend in einer Rangliste sortiert. Somit ist der zum Eingabevektor am nächsten liegende Referenzvektor am Anfang der Liste und besitzt den Rang $k = 1$. Der am weitesten entfernte Referenzvektor besitzt den Rang $k = n$.

2. Die Lernrate eines Neurons ist durch die zeitabhängige monoton fallende Funktion
$$\epsilon(t) = \epsilon_i(\epsilon_f/\epsilon_i)^{t/t_{max}} \tag{9.1}$$
gegeben. Dabei muss ein maximale Anzahl an Zeitschritten t_{max} des neuronalen Netzes festgelegt werden. Somit liegt der Wert der Lernrate in Abhängigkeit von dem aktuellen Zeitschritt t zwischen einem Startwert ϵ_i und einem geringerem Endwert ϵ_f.

   In Abbildung 9.1.1 ist der monotone Charakter der zeitabhängigen Lernrate abgezeichnet.

3. Die Nachbarschaftsreichweite beeinflusst die Aktivierung des Neurons entsprechend der Position k in der Rangliste. Das erste Neuron der Liste wird am stärksten und das letzte am schwächsten adaptiert. Darin enthalten ist auch eine weitere zeitabhängige Größe $\lambda(t)$, die analog zur Funktion 9.1 definiert ist.

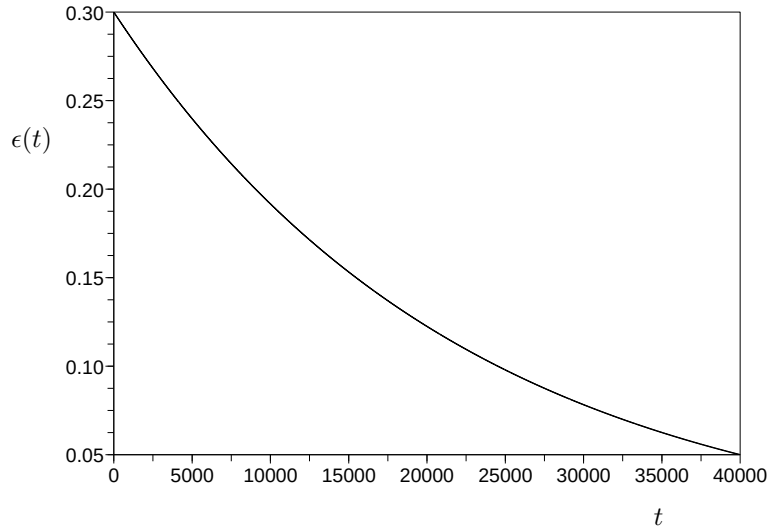


Abbildung 9.1.1: *Die Lernrate als zeitabhängige Funktion. Die Parameter sind:* $\epsilon_i = 0.3$, $\epsilon_f = 0.05$ *und* $t_{max} = 40000$.

Die Nachbarschaftsreichweite ist mit

$$h(t,k) = \exp\left(\frac{1-k}{\lambda(t)}\right)$$

gegeben. Damit werden mit fortschreitenden Zeitschritten immer weniger Neuronen adaptiert, die der Eingabe am nächsten liegen.

4. Für die gesamte Adaptionsstärke $\Delta\mathbf{w}_i$ jedes Referenzvektors i bei gegebenem Eingabevektor $\boldsymbol{\xi}$ gilt:

$$\Delta\mathbf{w}_i = \epsilon(t)h(t,k)(\boldsymbol{\xi} - \mathbf{w}_i) \ .$$

In Abbildung 9.1.2 ist der monotone Charakter der Adaptionsrate in Abhängigkeit des Zeitschrittes t und des Ranglistenplatzes k abgebildet. Gut erkenntlich ist, dass je mehr Zeitschritte vergangen sind, nur noch das Neuron, das dem Eingabevektor am nächsten liegt, adaptiert wird.

5. Die Hebb'sche Wettbewerbsregel von Martinetz und Schulten [Mar93] bildet eine topologische Struktur zwischen den Referenzvektoren aus. Diese Struktur entspricht dabei einem Untergraphen der Delaunay-Triangulation innerhalb

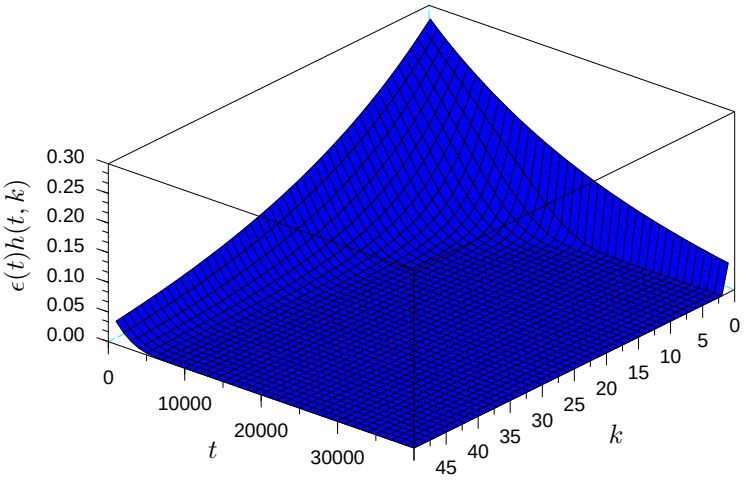


Abbildung 9.1.2: *Die zeitabhängige Adaptionsrate* $\epsilon(t)h(t,k)$ *mit den Parametern:* $\epsilon_i = 0.3$, $\epsilon_f = 0.05$, $\lambda_i = 30$, $\lambda_f = 0.01$ *und* $t_{max} = 40000$.

der Eingabedatenverteilung $p(\boldsymbol{\xi})$ [MS94]. Um diese Topologie zu erzeugen, werden zu jedem Eingabesignal $\boldsymbol{\xi}$ die beiden nächstliegenden Referenzvektoren $\mathbf{w}_0$ und $\mathbf{w}_1$ bestimmt. Zwischen beiden Neuronen s_0 und s_1 wird eine Verbindung (s_0, s_1) mit:

$$\mathcal{C} = \mathcal{C} \cup \{(s_0, s_1)\}$$

erzeugt, falls diese nicht schon existiert. Die Hebb'sche Wettbewerbsregel kann sowohl nach als auch während der Verteilung der Referenzvektoren im Eingaberaum angewendet werden. Für den letzteren Fall wird jeder existenten Verbindung $(s_i, s_j) \in \mathcal{C}$ ein Alter $age_{(s_i,s_j)}$ zugeordnet. Wenn dieses Alter ein maximales Alter $T(t)$ in Abhängigkeit des aktuellen Zeitschritts t überschreitet, wird die Verbindung gelöscht. Damit kann sich die Triangulation stets an die sich verschiebenden Referenzvektoren anpassen. Beim erstmaligen Hinzufügen einer Verbindung (s_0, s_1) oder wenn erneut s_0, s_1 die nächstliegenden Neuronen sind, wird das Alter der Verbindung auf Null gesetzt bzw. verjüngt:

$$age_{(s_0,s_1)} = 0 \ .$$

Für alle Neuronen c, die mit dem Gewinnerneuron s_0 über eine Verbindung

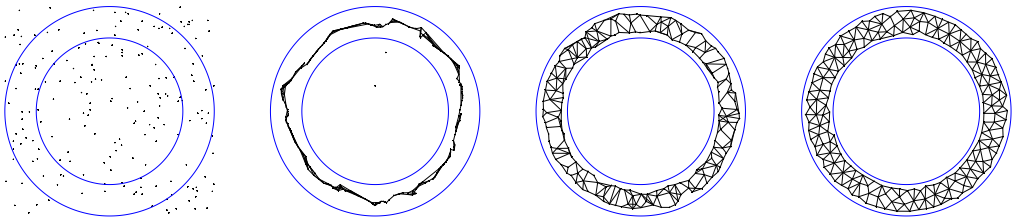
benachbart sind, wird das Alter inkrementiert:

$$age_{(s_0,c)} = age_{(s_0,c)} + 1, \forall c \in \mathcal{N}_{s_0} \ .$$

$\mathcal{N}_{s_0}$ entspricht der Menge aller Nachbarn von s_0. Das maximale Alter $T(t)$ ist ähnlich wie die Lernrate in Gleichung 9.1 definiert. Dabei ist jedoch die Anfangszeitspanne T_i größer als die Endzeitspanne T_f, da sich die Referenzvektoren mit jedem weiteren Zeitschritt immer weniger verschieben und folglich die Verbindungen zum Ende hin über einen größeren Zeitraum der Delaunay-Triangulation entsprechen. Martinetz und Schulten haben bewiesen, dass die mit der Hebb'schen Wettbewerbsregel erstellte Topologie einem Untergraph der Delaunay-Triangulation innerhalb der Mannigfaltigkeit der Eingabewahrscheinlichkeit entspricht, vorausgesetzt die Dichte der Referenzvektoren ist hoch genug[MS94].

Die Zeitkomplexität eines Berechnungsschrittes ist durch den aufwändigen Sortierungsschritt zur Bestimmung der Rangliste mit $O(n \log_2 n)$ gegeben.

In Abbildung 9.1.3 ist das Verhalten des neuronalen Gases exemplarisch dargestellt. Die Eingabedatenverteilung $p(\boldsymbol{\xi})$ ist gleichmäßig innerhalb des Blau umrandeten Kreisringes gegeben. Die Parameter sind: $n = 200$, $t_{max} = 40000$, $\epsilon_i = 0.5$, $\epsilon_f = 0.005$, $\lambda_i = 10$, $\lambda_f = 0.01$, $T_i = 20$, $T_f = 200$.



(a) 0 Zeitschritte (b) 1000 Zeitschritte (c) 10000 Zeitschritte (d) 40000 Zeitschritte

Abbildung 9.1.3: *Das zeitliche Verhalten während der Anpassung des neuronalen Gases mit Hebb'schem Wettbewerbslernen über* 40000 *Zeitschritte.*

9.2 Wachsendes neuronales Gas

Das *Growing Neural Gas* (GNG) wurde von Fritzke entwickelt[Fri95]. Im Vergleich zum NG werden hier keine zeitabhängigen Parameter verwendet. Die Besonderheit dieses Netzwerkes liegt in dem inhärenten Wachstumsprozess. So wird jeweils nach einer konstanten Zahl an Zeitschritten ein weitere Referenzvektor dem Netzwerk zugefügt, bis eine maximale Zahl an Referenzvektoren eingefügt wurden.

Der interne Wachstumsprozess ist jedoch nicht zufällig, sondern benutzt das über die steigende Anzahl an Zeitschritten akkumulierte Fehlermaß E_c eines jeden Neurons c. Es handelt sich dabei um den Quantisierungsfehler eines jeden Neurons. Dieser wird zu jedem Zeitschritt für das Gewinnerneuron s_0 um den Wert

$$\Delta E_{s_0} = \|\boldsymbol{\xi} - \mathbf{w}_{s_0}\|^2 \tag{9.2}$$

erhöht. Um das akkumulierte Fehlermaß

$$\sum_{c \in \mathcal{A}} E_c$$

zu minimieren, wird der Wachstumsprozess des Netzes so gerichtet, dass durch das Einfügen eines neuen Neurons r der Gesamtfehler abgesenkt wird:

$$\sum_{c \in \mathcal{A}} E_c > \sum_{c \in \mathcal{A} \cup \{r\}} E_c \ .$$

Das neue Neuron r wird zwischen dem Neuron q mit dem größten akkumulierten Fehler E_q und dessen benachbarten Neuron f mit dem größten akkumulierten Fehler aller benachbarten Neuronen eingefügt mit:

$$q = \arg\max_{c \in \mathcal{A}} E_c, \ f = \arg\max_{c \in \mathcal{N}_q} E_c \ .$$

Für die neue Referenzvektorposition $\mathbf{w}_r$ gilt:

$$\mathbf{w}_r = \frac{(\mathbf{w}_q + \mathbf{w}_f)}{2} \ .$$

Das neue Neuron bekommt den mittleren Fehler der beiden Neuronen q, f:

$$E_r = \frac{E_q + E_f}{2} \ .$$

Durch das Einfügen eines neuen Neurons wird der Gesamtfehler des Netzes gesenkt. Um diese Absenkung zu approximieren, wird einerseits der Fehler der Neuronen q, f um den Bruchteil α verringert:

$$\Delta E_q = -\alpha E_q, \ \Delta E_f = -\alpha E_f \ .$$

Andererseits werden die Fehler jedes Neurons c um den Bruchteil β verkleinert:

$$\Delta E_c = -\beta E_c \ .$$

Ein neues Neuron wird dabei alle λ Zeitschritte solange eingefügt, bis eine maximale Netzgröße n_{max} oder ein Quantisierungskriterium erfüllt ist.

Im Gegensatz zum NG werden die zeitabhängigen Parameter durch Konstanten ausgetauscht. So existiert keine Nachbarschaftsreichweite $h(t, k)$. Es werden nur der Referenzvektor des Gewinnerneurons s_0 mit Hilfe der Lernrate ϵ_1 um

$$\Delta \mathbf{w}_{s_0} = \epsilon_1(\boldsymbol{\xi} - \mathbf{w}_{s_0})$$

und die Referenzvektoren der direkten Nachbarneuronen i mit Hilfe der Lernrate ϵ_2 um

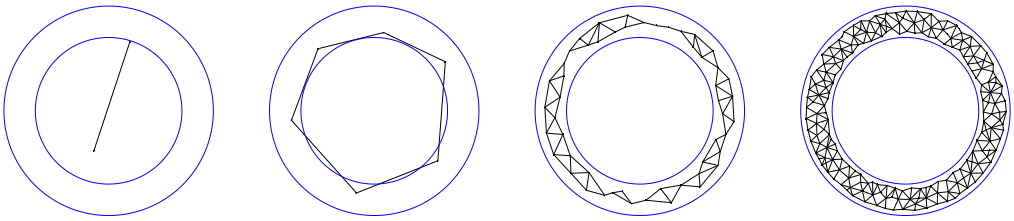
$$\Delta \mathbf{w}_i = \epsilon_2(\boldsymbol{\xi} - \mathbf{w}_i)$$

verschoben. Das zeitabhängige maximale Alter $T(t)$ einer Kante wird durch den konstanten Wert a_{max} ersetzt. Ein weiterer Unterschied zum NG besteht darin, dass Neuronen gelöscht werden, wenn sie keine Kanten mehr besitzen.

Da im Vergleich zum NG hier keine Rangliste nötig ist, ist die Zeitkomplexität hier durch die Anzahl der Referenzvektoren n mit $O(n)$ gegeben.

In Abbildung 9.2.1 ist das Verhalten des GNG exemplarisch dargestellt. Die Eingabedatenverteilung $p(\boldsymbol{\xi})$ ist gleichmäßig innerhalb des Blau umrandeten Kreisringes gegeben. Die Parameter sind: $n_{max} = 200$, $\epsilon_1 = 0.5$, $\epsilon_2 = 0.005$, $\alpha = 0.5$, $\beta = 0.0005$, $a_{max} = 88$ und $\lambda = 200$.

Falls nicht das Ziel die Fehlerminimierung des Netzwerkes ist, sondern die Entro-



(a) 100 Zeitschritte (b) 1000 Zeitschritte (c) 10000 Zeitschritte (d) 40000 Zeitschritte

Abbildung 9.2.1: *Das zeitliche Verhalten während der Anpassung des wachsenden neuronalen Gases mit Hebb'schem Wettbewerbslernen über* 40000 *Zeitschritte.*

piemaximierung (siehe Abschnitt 8.5), wird die Gleichung 9.2 durch

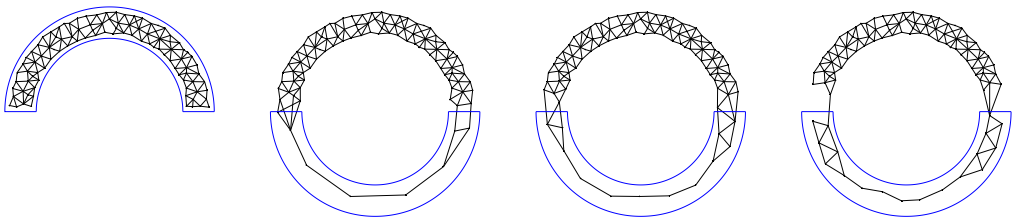
$$\Delta E_{s_0} = 1 \tag{9.3}$$

ersetzt. Damit werden alle Fehler der Gewinnerneuronen als gleichwertig betrachtet und neue Referenzvektoren werden vermehrt neben den Referenzvektoren positioniert, die häufiger die Gewinner waren. Dadurch wird die Entropie des Netzwerkes maximiert.

9.3 Wachsendes neuronales Gas mit Nützlichkeitsmaß

Das *Growing Neural Gas with Utility criterion* (GNG-U) ist eine Variante des GNG und wurde von Fritzke [Fri97] entwickelt. Diese Variante ermöglicht dem Netzwerk sich an nichtstationäre Eingabedatenverteilungen anzupassen. Da die Lernraten ϵ_1, ϵ_2 des GNG konstant sind, wird das Netzwerk um ein Nützlichkeitsmaß erweitert, das eine Anpassung an dynamische Eingabedatenverteilungen gewährleistet.

Wendet man das GNG auf eine nichtstationäre Eingabedatenverteilung an, kann das folgende Verhalten charakterisiert werden (siehe Abbildung 9.3.1). Nach den ersten 20000 Zeitschritten hat sich das GNG-Netz an die stationäre Eingabedatenverteilung der oberen Kreisringhälfte angepasst. Die nächsten 20000 Zeitschritte springt die Eingabedatenverteilung auf die untere Kreisringhälfte. Im Verlaufe dieser 20000 Zeitschritte werden jedoch nur die äußeren Neuronen in die untere Kreisringhälfte gezogen und die restlichen Neuronen verbleiben im oberen Bereich. Diese Neuronen werden auch als tote Neuronen bezeichnet. Die verwendeten Parameter sind: $n_{max} = 100$, $\epsilon_1 = 0.5$, $\epsilon_2 = 0.005$, $\alpha = 0.5$, $\beta = 0.0005$ maximales Alter einer Kante $a_{max} = 88$, ein neues Neuron wird alle $\lambda = 200$ Zeitschritte eingefügt.



(a) 20000 Zeitschritte (b) 21000 Zeitschritte (c) 25000 Zeitschritte (d) 40000 Zeitschritte

Abbildung 9.3.1: *Das zeitliche Verhalten während der Anpassung des GNG an eine nichtstationäre Eingabedatenverteilung über* 40000 *Zeitschritte.*

Das GNG-U versucht diese toten Neuronen zu identifizieren, um sie dann dem Einfügeprozess erneut zu übergeben. Dafür wird das Nützlichkeitsmaß U_c für jedes Neuron c verwendet. Die Nützlichkeit gibt an, wie stark der Fehler des zweitnächsten Neurons s_1 ansteigt, wenn man das Gewinnerneuron s_0 aus dem Netzwerk entfernt. Somit ist die Nützlichkeit des Gewinnerneurons s_0 durch die Differenz des Fehlers E_{s_0} des Gewinnerneurons und des Fehlers E_{s_1} des zweitnächsten Neurons s_1 gegeben:

$$U_{s_0} = U_{s_0} + \|\boldsymbol{\xi} - \mathbf{w}_{s_1}\|^2 - \|\boldsymbol{\xi} - \mathbf{w}_{s_0}\|^2 \ .$$

Die Nützlichkeit akkumuliert sich über die Zeit für alle Gewinnerneuronen. Da die toten Neuronen weit entfernt von der aktuellen Eingabedatenverteilung liegen, wird in jedem Zeitschritt die Nützlichkeit U_c aller Neuronen um einen Bruchteil β gesenkt:

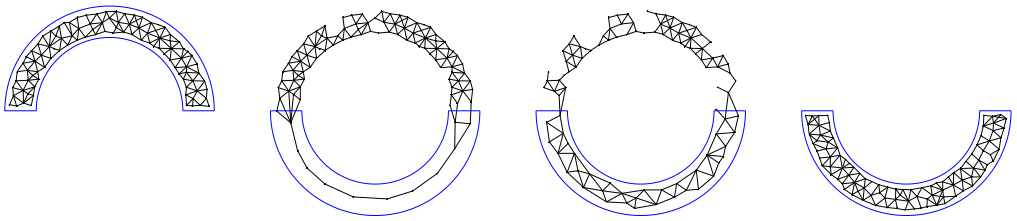
$$U_c = U_c - \beta U_c \ .$$

Damit stagniert die Nützlichkeit eines Neurons c, wenn der zugehörige Referenzvektor $\mathbf{w}_c$ von anderen Referenzvektoren (mindestens einen) sehr dicht umgeben ist oder wenn c zu einem toten Neuron wird.

Das Entfernen einer Einheit mit geringster Nützlichkeit wird nicht zu einem beliebigen Zeitschritt durchgeführt, da immer ein Neuron mit geringster Nützlichkeit existiert, auch wenn die Anpassung des Netzes an die Eingabedatenverteilung optimal ist. Das Löschen des Neurons i mit der geringsten Nützlichkeit U_i ist dann sinnvoll, wenn der maximale Fehler E_q eines Neurons q größer als ein Vielfaches k des Nutzens U_i ist:

$$E_q > kU_i \ .$$

Das Einfügen des Neurons i geschieht analog zu dem Einfügen eines neuen Neurons wie im GNG. Analog zu dem GNG besitzt auch das GNG-U eine Zeitkomplexität von $O(n)$ für jeden Zeitschritt.



(a) 20000 Zeitschritte (b) 21000 Zeitschritte (c) 25000 Zeitschritte (d) 40000 Zeitschritte

Abbildung 9.3.2: *Das zeitliche Verhalten während der Anpassung des GNG-U an eine nichtstationäre Eingabedatenverteilung über* 40000 *Zeitschritte.*

In Abbildung 9.3.2 ist die Anpassung des Netzes an die nichtstationäre Eingabedatenverteilung dargestellt. Dabei springt die Eingabedatenverteilung von der oberen Kreisringhälfte nach 20000 Zeitschritten in die untere Hälfte. Es ist gut zu erkennen, wie die toten Neuronen nach und nach entfernt werden. Die verwendeten Parameter sind: $n_{max} = 100$, $\epsilon_1 = 0.5$, $\epsilon_2 = 0.005$, $\alpha = 0.5$, $\beta = 0.0005$, $k = 3$, $a_{max} = 88$ und $\lambda = 200$.

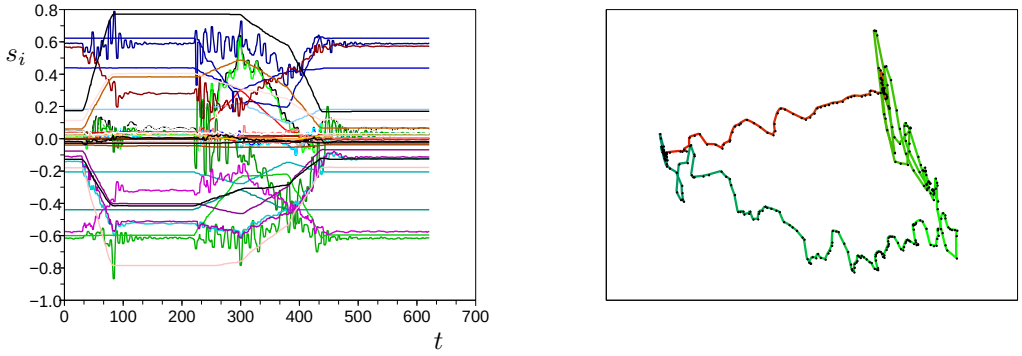
10 Experimente und Resultate

10.1 Einführung zu den Experimenten

In diesem und den folgenden Textabschnitten werden die Algorithmen von NG, von GNG und von GNG-U zur Ausbildung sensorischer Karten sowie von dem Toussaint-Verfahren zur Ausprägung einer sensomotorischen Karte als ein Spezialfall des GNG bezüglich ihres Fehlers (siehe Gleichung 8.2) miteinander verglichen.

Die Erweiterung des Vergleiches der sensordatenverarbeitenden Verfahren um das Toussaint-Verfahren ist notwendig, um die in Teil III erarbeiten Aussagen zum Toussaint-Modell im Vergleich zu untersetzen. Dabei werden die Netzwerkanpassungen an einen Datensatz mit originaler und zufällig durchmischter Reihenfolge untersucht. Anschließend wird die Anpassung des GNG-U-Netzwerkes an die nichtstationären und jeweils letzten 100 Datenpunkte desselben originalen Datensatzes untersucht.

Als Trainingsdaten werden mediangefilterte Robotersensordaten verwendet, welche während einer „Stehen-Knien-Stehen“ Bewegung aufgezeichnet wurden. Die 621 Sensorwerte sind in Abbildung 10.1.1 zu sehen. Zur zweidimensionalen Veranschaulichung werden sie mit dem PCA-Algorithmus (siehe Abschnitt 2.2) transformiert.



(a) Vor Transformation

(b) Nach PCA-Transformation.

Abbildung 10.1.1: *Trainingsdaten zur Verwendung für NG-Algorithmen vor und nach der PCA-Transformation. Auf Bild (a) sind alle* 37 *Sensorwerte s_i mit jeweils einer Farbe über alle Zeitschritte eingezeichnet. Auf Bild (b) ist die zweidimensionale PCA-Transformation des Datensatzes abgebildet. Dabei wird ein kontinuierlicher Farbübergang von Rot nach Grün nach Blau verwendet.*

10.2 Netzwerkanpassung an den gesamten Trainingsdatensatz

Zum Vergleich der Güte der Netzwerkanpassung durch die Algorithmen NG, GNG, GNG-U und Toussaint an die Trainingsdaten wird wie folgt verfahren:

Nach jedem Durchlauf des ganzen Trainingsdatensatzes wird für das derzeitig ausgeprägte neuronale Netz das Fehlermaß (siehe Gleichung 8.2) über die gesamten Trainingsdaten berechnet. Diese Trainingsdaten werden zehn Iterationen lang in die Algorithmen eingespielt. Dabei wird bei jedem Algorithmus der Einfachheit halber nur ein Parameter variiert und eine größtmögliche Übereinstimmung bei den verschiedenen Paramatersätzen eingehalten. Dieses Vorgehen soll eine möglichst allgemeine Aussage über das Verhalten und die Güte der Modelle liefern.

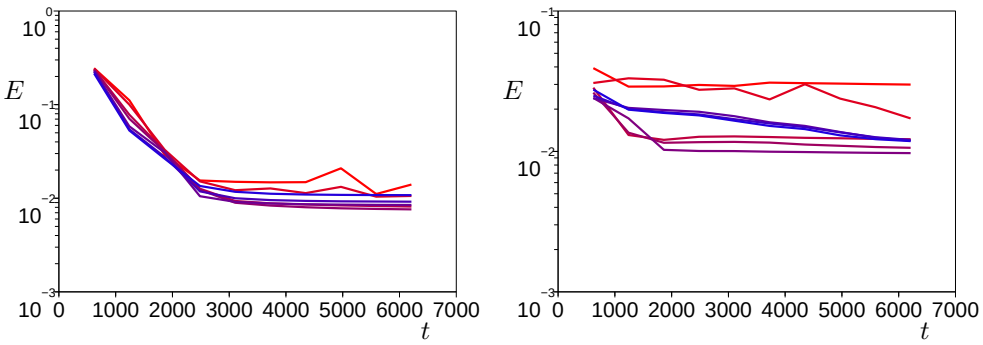
In Abbildung 10.2.1 sind die Fehlerwerte der Netzwerke nach jedem Iterationsschritt über alle Trainingsdaten eingezeichnet. Die konstanten und variierenden Parameter der Algorithmen sind in der Tabelle 10.2.1 eingetragen.

	n	ϵ_i	ϵ_f	λ_i	λ_f	T_i	T_f	
NG	10	0.1	$[\mathbf{0.1}, \mathbf{0.001}]$	10	0.01	20	200	
	n	α	β	λ	a_{max}	ϵ_1	ϵ_2	
GNG	10	0.5	0.0005	30	90	$[\mathbf{0.5}, \mathbf{0.01}]$	0.0001	
	n	α	β	λ	a_{max}	ϵ_1	ϵ_2	k
GNG-U	10	0.5	0.0005	30	90	0.016	0.0001	$[\mathbf{10^{-1}}, \mathbf{10^6}]$
	n	ν	σ_S					
Toussaint	10	0.5	$[\mathbf{0.1}, \mathbf{0.5}]$					

Tabelle 10.2.1: *Konstante und variierende (fettgedruckte Intervalle) Parameter der verschiedenen Algorithmen neuronaler Gase.*

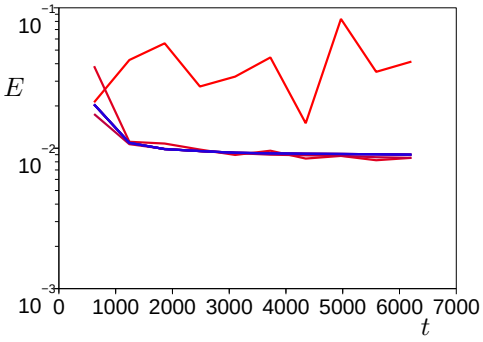
Die Variation der ausgewiesenen variablen Parameter (siehe Tabelle 10.2.1) erfolgt linear innerhalb der Intervalle und die resultierenden Netzwerkfehler sind grafisch durch die Färbung von Rot über Magenta nach Blau dargestellt. Die Auswahl der Parameter wird so getroffen, dass die Start- und Endwerte (Rot- und Blaufärbung) der Parameter möglichst sichtbar zu schlechteren Fehlerwerten als die mittleren suboptimalen Parameterwerte (Magentafärbung) führen.

Mit Hilfe der wenigen Resultate können jedoch schon einige Eigenschaften festgehalten werden. So weisen alle Algorithmen ähnliche Minima (Magentafärbung) auf und besitzen damit die gleiche bzw. sehr ähnliche Güte nach Abschluss der zehn-

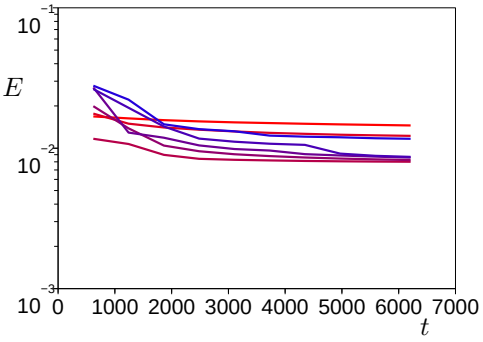


(a) NG

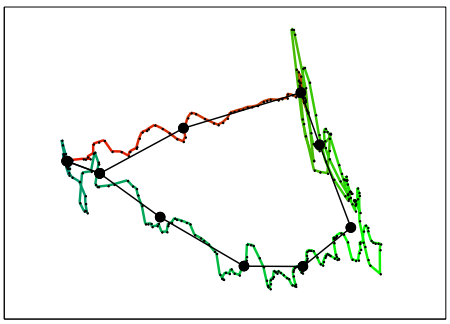
(b) GNG



(c) GNG-U



(d) Toussaint



(e) Transformierte Sensordaten und NG-Netz

Abbildung 10.2.1: *Netzwerkfehler jedes Algorithmus im Verlauf mehrerer Iterationen der Trainingsdaten. Die lineare Variation der variablen Parameter (siehe Tabelle 10.2.1) mit den resultierenden Netzwerkfehlern ist durch den Übergang von Rot über Magenta nach Blau visualisiert. Das adaptierte NG-Netz mit zehn Neuronen (schwarze Kreise) und Kanten (schwarze Linien) ist zusammen mit den Trainingsdaten eingezeichnet.*

ten Iteration. Es ist zu erkennen, dass der Fehlerwert nicht zwangsweise monoton gegen einen Endwert konvergiert, sondern auch einen sprunghaften sowie oszillierenden Charakter aufweisen kann (siehe dazu in Abbildung 10.2.1 die Fehler von NG, GNG und GNG-U).

Die NG-Ergebnisse weisen die größten Differenzen zwischen den Start- und Endwerten auf, was durch die anfänglich zufällige Positionierung der Referenzvektoren zu begründen ist.

Der Toussaint-Algorithmus besitzt die geringste Differenz zwischen Start- und Endfehlerwert. Dies ist durch die äquidistante Platzierung der Referenzvektoren zu erklären, wodurch sich die Referenzvektoren im weiteren Verlauf der Netzwerkausbreitung nur noch wenig verschieben.

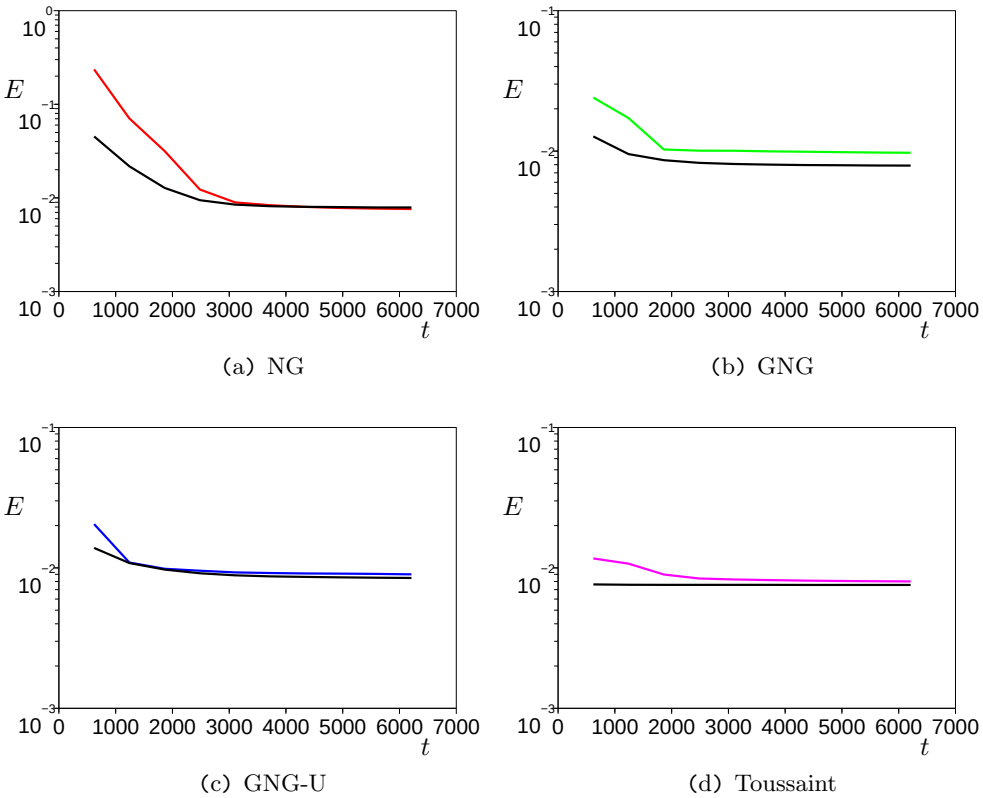
Bei dem GNG-U ist zu vermerken, dass wenn k zu klein definiert wird, die Neuronen mit geringster Nützlichkeit häufiger entfernt und an neue Positionen gesetzt werden, so dass das Netz sich an die jüngsten Trainingsdaten anpasst und eine Netzwerkanpassung an alle Trainingsdaten nicht gewährleistet ist. Eine weitere Eigenschaft des GNG-U-Algorithmus besteht darin, dass dieser die geringste Anzahl an Durchläufen benötigte, um annähernd die Endkonstellation des Netzes zu erreichen.

10.3 Netzwerkanpassung an den Trainingsdatensatz mit zufällig durchmischter Reihenfolge

In diesem Abschnitt wird untersucht, ob die zufällige Durchmischung der Reihenfolge der Trainingsdaten einen Einfluss auf die Güte der Netzwerkausprägung hat. Um diesen Einfluss experimentell zu messen, werden die 621 Eingabevektoren in ihrer Reihenfolge durchmischt. Dieser neue Datensatz wird dann mit zehn Wiederholungen in die jeweiligen Algorithmen eingespielt. Nach jeder Iteration wird der Gesamtfehler des bis dahin adaptierten Netzes bestimmt. Die Parameterwahl der Algorithmen ist gleich geblieben (siehe Tabelle 10.2.1).

In Abbildung 10.3.1 sind die jeweils besten Fehlerwerte der einzelnen Algorithmen eingezeichnet. Es zeigt sich, dass der zufällig durchmischte Datensatz eine offensichtliche Verringerung des Netzwerkfehlers bewirkt. Vor allem ist bis zu dem fünften Durchlauf der zufällig durchmischten Trainingsdaten das Netz um ein Vielfaches besser angepasst.

Dieser signifikante Einfluss resultiert aus der Beschaffenheit der Sensordaten. Die Sensordaten einer Roboterbewegung gleichen einer Punktbewegung im mehrdimen-



(a) NG (b) GNG

(c) GNG-U (d) Toussaint

Abbildung 10.3.1: *Der geringste Netzwerkfehler jedes Algorithmus im Verlauf mehrerer Wiederholungen des Datensatzes mit zufällig durchmischter Reihenfolge (schwarze Kurve) und mit Ausgangsreihenfolge (farbige Kurve).*

sionalen Raum. Damit liegen die Sensordaten zeitlich lokal betrachtet auf einer eindimensionalen Mannigfaltigkeit (siehe Abschnitt 2.1). Wenn diese Daten ohne zufällig durchmischte Reihenfolge in ein neuronales Netz eingespielt werden, wird das neuronale Netz zeitlich lokal betrachtet jeweils nur in eine Richtung gezogen. Hingegen das Einspielen von Daten mit zufällig durchmischter Reihenfolge, bewirkt ein gleichmäßiges Ausbreiten des Netzes in alle Richtungen. Dadurch ist das Netz von Beginn an besser verteilt und der Fehlerwert geringer als bei dem Netz, das ohne zufällig durchmischte Sensordaten trainiert wurde.

10.4 Netzwerkanpassung an die jüngsten Trainingsdaten

Im Folgenden wird die spezielle Eigenschaft der Anpassung an nichtstationäre Eingabedatenverteilungen des GNG-U untersucht. Die anderen NG-Algorithmen werden hier nicht betrachtet, da keine Anpassung an einen dynamischen Datensatz im Modell vorgesehen ist und damit stets schlechtere Ergebnisse resultieren würden (siehe Abbildung 9.3.1). Die verschiedenen Parametersätze des GNG-U werden auf die gleichen oben erwähnten Roboterdaten angewendet. Jedoch wird nun exemplarisch der Fehler der letzten 100 Zeitschritte während der Trainingsphase gemessen und nicht über alle 621. Somit wird eine zyklische nichtstationäre Verteilung von Sensordaten induziert.

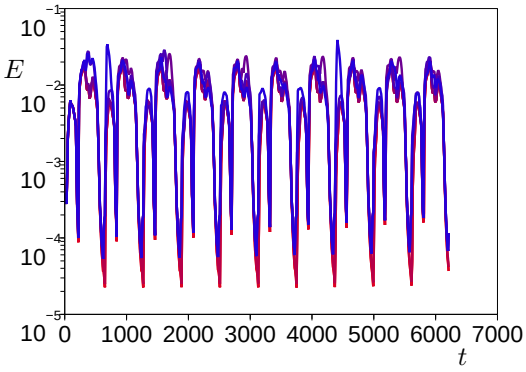
Die Wahl des variierenden Parameters fällt auf β (siehe Tabelle 10.4.1), da es sich hier um den Bruchteil des Nützlichkeitsmaßes handelt, um den in jedem Zeitschritt jedes Neuron verringert wird. Je größer dieser Wert ist, umso schneller fällt die Nützlichkeit eines Neurons, das nicht dem aktuellen Eingabevektor am nächsten liegt. Anstelle von β kann auch k, in einem entsprechend großen Intervall, variiert werden.

	n	α	β	λ	a_{max}	ϵ_1	ϵ_2	k
GNG-U	10	0.5	$[\mathbf{0.5}, \mathbf{0.005}]$	10	90	0.005	0.0001	1

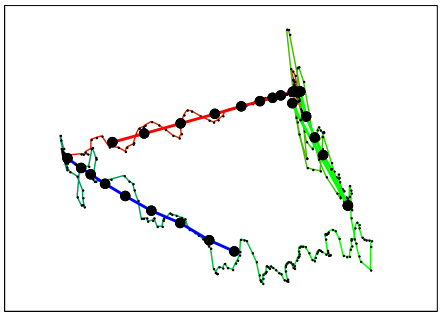
Tabelle 10.4.1: *Konstante und variierende (fettgedrucktes Intervall) Parameter des GNG-U-Algorithmus.*

In Abbildung 10.4.1(a) sind die Netzwerkfehler von sieben Parametersätzen (siehe Tabelle 10.4.1) eingezeichnet. Dabei wurde der β-Parameter linear gesenkt. Grafisch ist die Linearität durch den farblichen Übergang der entsprechenden Netzwerkfehlern von Rot über Magenta nach Blau dargestellt. So bewirkt der größte Wert mit $\beta = 0.5$ (rote Färbung) für alle Fehlerwerte die beste Netzwerkausprägung.

In Abbildung 10.4.1(b) sind der PCA-transformierte Datensatz und drei Zustände (nach 150, 300 und 500 Zeitschritten) des GNG-U Netzes mit $\beta = 0.5$ dargestellt. Die Neuronen sind durch Kreise und die Verbindungen in den Farben Rot, Grün und Blau gekennzeichnet.



(a) GNG-U: β ist variabel



(b) PCA-Transformation

Abbildung 10.4.1: *(a): Netzwerkfehler verschiedener GNG-U-Parametersätze im Verlauf über mehrere Iterationen der Trainingsdaten. Die lineare Variation des Parameter β ist durch den Übergang von Rot über Magenta nach Blau visualisiert. (b): Drei Netzwerkausprägung (Neuronen als Kreise und Verbindungen in Rot, Grün, Blau) eines Netzes mit $\beta = 0.5$ sind über die Trainingsdaten eingezeichnet.*

11 Zusammenfassung von Teil II

Im theoretischen Teil wurden die wettbewerbslernenden neuronalen Netze in das System der künstlichen neuronalen Netze eingeordnet. In diesem Zusammenhang wurden die Begrifflichkeiten und Definitionen der vektorbasierten neuronalen Netze geklärt. Die wettbewerbslernenden neuronalen Gase komprimieren und beschreiben die Eingabedatenstruktur unüberwacht optimal. In diesem Zusammenhang wurden die bedeutendsten Algorithmen neuronaler Gase nach dem folgenden Schema untersucht:

- Mathematische Beschreibung der Arbeitsschritte
- Erläuterung der Ausprägung der sensorischen Karte nach unterschiedlichen Zeitschritten
- Bestimmung der Zeitkomplexität der Algorithmen

Im Einzelnen wurden in der vorliegenden Arbeit die folgenden Unterscheidungsmerkmale der Algorithmen neuronaler Gase herauskristallisiert:

1. Das Verfahren *Neural Gas* (NG) ist das einzige, das zeitabhängige Parameter mit einem Start- und Endwert verwendet. Dadurch ist einerseits eine genaue Anpassung an eine statische Eingabedatenverteilung möglich, andererseits ist diese Netzstruktur nur für stationäre Datenräume mit absehbar vielen Datenpunkten geeignet, d. h. ab einer gewissen Zeit findet keine Anpassung an sich ändernde Datenkonzentrationen statt. Darüber hinaus besitzt der Algorithmus die höchste Zeitkomplexität.

2. Das Verfahren *Growing Neural Gas* (GNG) ist wegen der wachsenden Netzstruktur und den festen Lernraten für Eingabedaten mit statischer sowie leicht variierender Datenstruktur geeignet. Aufgrund des internen Fehlermaßes ist der durchschnittliche Fehler nach jedem Zeitschritt für das ausgeprägte neuronale Netzwerk abschätzbar. Die Zeitkomplexität ist geringer als die des NG-Verfahrens.

3. *Growing Neural Gas with Utility criterion* (GNG-U) ist eine Erweiterung von GNG und erzeugt eine Netzstruktur, die sich speziell an nichtstationäre Eingabedaten anpassen kann. Dieses Verfahren weist die gleiche Zeitkomplexität wie das GNG-Verfahren auf.

Im praktischen Teil wurden die Ausprägungen der neuronalen Netzwerke der Algorithmen NG, GNG, GNG-U und des Toussaint-Modells untersucht. Dafür wurde ein exemplarischer Datensatz des humanoiden Roboters einer „Stehen-Knien-Stehen“-Bewegungssequenz aufgezeichnet und von den Algorithmen verarbeitet. Die rechentechnische Umsetzung der Algorithmen zu den einzelnen Verfahren zeigte folgende Ergebnisse:

1. Nach zehnmaliger Einspeisung des vorgegebenen Datensatzes, d. h. nach zehnmaliger Trainingsphase der Bewegungssequenz, weisen alle neuronalen Gase keinen signifikanten Unterschied hinsichtlich des Netzwerkfehlers auf (siehe Tabelle 11.0.2). Dabei wurden alle Verfahren für diese Bewegungssequenz optimal parametrisiert, um eine Vergleichbarkeit zu gewährleisten.

2. Die Geschwindigkeitsunterschiede bei der Netzwerkausprägung der neuronalen Gase sind dabei signifikant. So benötigt der NG-Algorithmus mit vier die meisten Trainingsphasen und der GNG-U-Algorithmus mit nur zwei die wenigsten Trainingsphasen, um die Endkonstellationen der entsprechenden Netzwerkausprägung zu erreichen.

3. Bei der Ausprägung des neuronalen Netzwerkes des Toussaint-Modells werden die Referenzvektoren von Anfang an im Vergleich zu den anderen Verfahren mit der höchsten Genauigkeit positioniert. Selbst nach mehreren Trainingsphasen bleibt der Netzwerkfehler konstant, während die anderen Netzwerke der neuronalen Gase erst nach mehreren Trainingsphasen ihren Netzwerkfehlerendwert erreichen.

4. Das GNG-U-Netz kann sich erfolgreich an Eingabedatenverteilungen anpassen, deren örtliche Konzentration im Datenraum sich über mehrere Zeitschritte erheblich ändert (nichtstationären Eingabedatenverteilung).

5. Bei einer Verarbeitung der Sensordaten mit einer einheitlich verwendeten gemischten Reihenfolge, d. h. unabhängig von ihrer zeitlichen Ermittlung, erreicht die Ausprägung der meisten Netzwerke der neuronalen Gase nach wesentlich weniger Trainingsphasen ihren Netzwerkfehlerendwert. So senkt eine einheitlich verwendete gemischte Reihenfolge der Trainingsdaten den Netzwerkfehler der zu vergleichenden Netze um bis zu 90% nach dem ersten Durchlauf und 30% nach dem zehnten Durchlauf (siehe NG und GNG-U in Tabelle 11.0.2).

Trainingsdatenreihenfolge	$x$ter Durchlauf	NG	GNG	GNG-U	Toussaint
Nicht zufällig durchmischt	$x = 1$	0.2359	0.0172	0.0205	0.0117
	$x = 10$	0.0076	0.0097	0.0090	0.0080
Zufällig durchmischt	$x = 1$	0.0218	0.0127	0.0139	0.0076
	$x = 10$	0.0079	0.0079	0.0085	0.0076

Tabelle 11.0.2: *Die besten Netzwerkfehler der neuronalen Netze nach der ersten und zehnten Iteration über die Trainingsdaten in einheitlich verwendeter zufälliger oder ursprünglicher Reihenfolge.*

Teil III

Toussaint-Modell einer sensomotorischen Karte

12 Theoretische Grundlagen

12.1 Überblick

Toussaint stellt einen Algorithmus vor, der während der Trainingsphase eine sensomotorische Karte anhand einer masselosen Punktbewegung im Eingaberaum ausbildet [Tou06]. Für die sensomotorische Karte verwendet er ein wachsendes neuronales Netzwerk, das für den wachsenden Prozess Techniken der NG-Netzwerke verwendet (siehe Kapitel 9).

In der anschließenden Planungsphase werden die gelernten Bewegungssteuerungen zur Erzeugung von zielgerichteten Bewegungssteuerungen in Richtung zufälliger Punkte innerhalb des Eingaberaumes verwendet.

Da Toussaint in seinem Verfahren das bisher noch nicht eingeführte Modell der dynamischen Felder verwendet, werden die folgenden Schwerpunkte des Toussaint-Modells einer sensomotorischen Karte erläutert:

1. Dynamisches Feld
2. Modellierung der sensomotorischen Eingaben
3. Ausbildung der sensomotorischen Karte in der Trainingsphase
4. Erzeugung eines zielgerichteten Verhaltens in der Planungsphase

12.2 Dynamisches Feld

Für ein Feld von mehreren Einheiten kann die zeitkontinuierliche Aktivierung $u(x,t)$ der Einheit x zum Zeitschritt t durch die Lösung eines dynamischen Systems beschrieben werden. Die Aktivierungsänderung $\dot{u}(x,t)$ des Feldes ist eine Funktion der aktuellen Aktivierung der Einheiten des Feldes [ES02]:

$$\dot{u}(x,t) = f[u(x',t)] \ .$$

Die Aktivierungsänderung $\dot{u}(x,t)$ einer Einheit x kann dabei entweder von der Aktivierung aller restlicher Einheiten im Feld abhängen oder nur die eigene Aktivierung berücksichtigen.

Im Falle der unabhängigen Aktivierung ist das dynamische Feld durch die Funktion

$$\tau\dot{u}(x,t) = -u(x,t) + h + S(x,t)$$

gegeben. Mit τ ist die Geschwindigkeit des dynamischen Systems definiert, $S(x,t)$ entspricht der Eingabe der Einheit x zum Zeitschritt t und h ist der Ruhepegel. Das Verhalten in Abhängigkeit der Parameter τ und h bei einem gegebenem verzerrten Sinussignal ist in Abbildung 12.2.1 zu sehen.

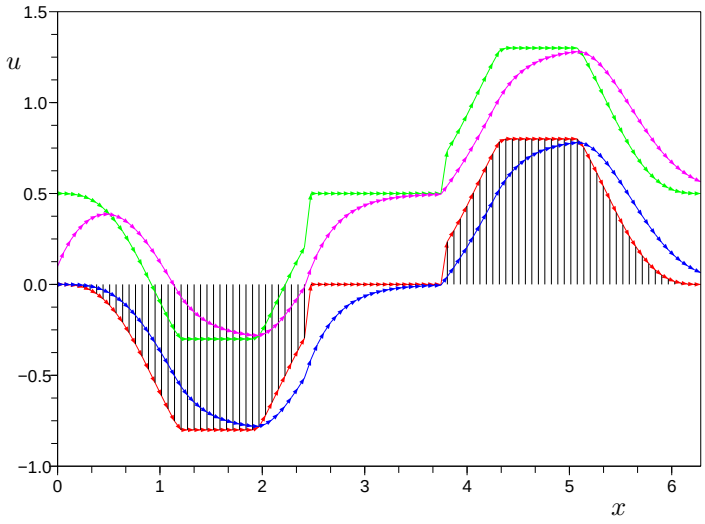


Abbildung 12.2.1: *Das Verhalten eines dynamischen Systems bestehend aus einem Element. Die senkrechten Linien geben die Stärke des Eingabesignal in jedem Zeitschritt t an. Die farbige Pfeillinie zeigt die Aktivierung der Einheit in dem entsprechendem Zeitschritt an. Rot: $h = 0, \tau = 1$, Blau: $h = 0, \tau = 5$, Grün: $h = 0.5, \tau = 1$, Magenta: $h = 0.5, \tau = 5$.*

Falls die Eingabe Null ist, konvergiert das System gegen den Ruhepegel h. Andernfalls konvergiert das System gegen $S(x,t) + h$. Selbst bei Eingabesprüngen bzw. -aussetzern zeigt das System kontinuierliche Aktivierungsänderungen (Hysterese-Effekte), solange $\tau > 1$ gilt.

Falls die Aktivierung einer Einheit von den restlichen Aktivierungen im Feld abhängt, gilt die Gleichung [Ama77]:

$$\tau \dot{u}(x,t) = -u(x,t) + h + S(x,t) + \int w(x - x') f[u(x',t)] dx' \ .$$

Dabei ist $w(x - x')$ die laterale Hemmung zweier Neuronen x und x', $f(u)$ ist eine

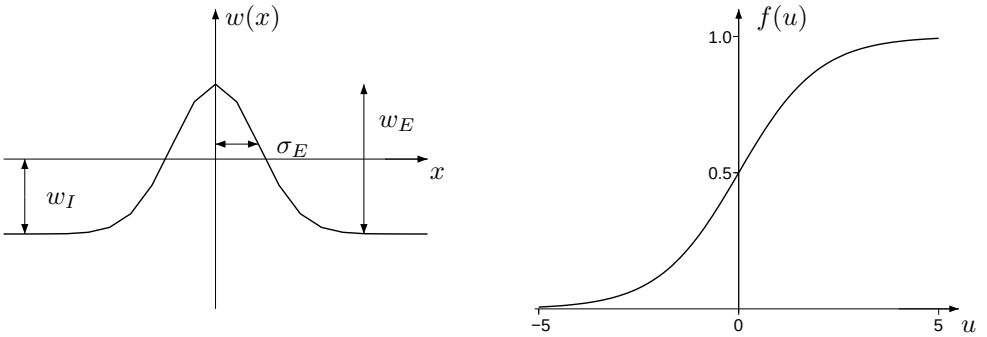
sigmoide Funktion. Die Hemmung kann durch eine Gaußkurve mit [ES02]:

$$w(x - x') = w_E \exp\left[\frac{-(x - x')^2}{2\sigma_E^2}\right] - w_I$$

gegeben sein. Die Funktion ist durch den Kurvenverlauf in Abbildung 12.2.2(a) vollständig beschrieben. So besitzt die Kurve eine lokale Anregung mit der Breite σ_E und Stärke $w_E + w_I$. Darüber hinaus existiert die globale Hemmung mit der Stärke w_I. Demnach haben nah gelegene Aktivierungen positiven und entfernte negativen Einfluss auf die Aktivierung einer Einheit. Die sigmoide Funktion [ES02]

$$f(u) = \frac{1}{1 + \exp[-\beta(u - u_0)]}$$

ist in Abbildung 12.2.2(b) dargestellt. Die Steigung der Funktion im Schnittpunkt mit der Ordinate wird durch β bestimmt und u_0 hat Einfluss auf die Schnittstelle $f(0)$. Diese Funktion gewährleistet, dass nur ausreichend aktivierte Einheiten betrachtet werden.



(a) Laterale Hemmung von 21 Einheiten (b) Sigmoidfunktion mit $\beta = 1$ und $u_0 = 0$

Abbildung 12.2.2: *Die laterale Hemmung und die Sigmoidfunktion.*

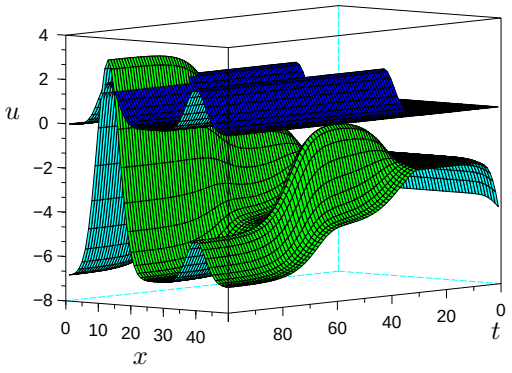
Aus der Kombination dieser Funktionen ergeben sich folgende globale Eigenschaften für ein solches dynamischen Feld:

1. Innerhalb der Zeit, in der keine Anregung existiert, ist die Stabilisierung eines Ruhepotentials gut zu erkennen. Bei ausreichend kleinem h entspricht dieses Potential dem Ruhepegel.

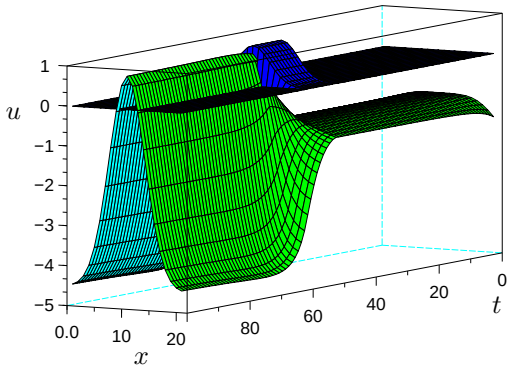
2. Ein lokales Signal $S(x,t)$ kann alle entfernt liegenden Signale unterdrücken. Dadurch wird in Anwesenheit eines (Sensor-)Rauschens ein Entscheidungsprozess induziert (siehe Abbildung 12.2.3(a)).

3. Eine lokale Erregung kann sich nach einem kurzzeitigen Signal dauerhaft stabilisieren (siehe Abbildung 12.2.3(b)).

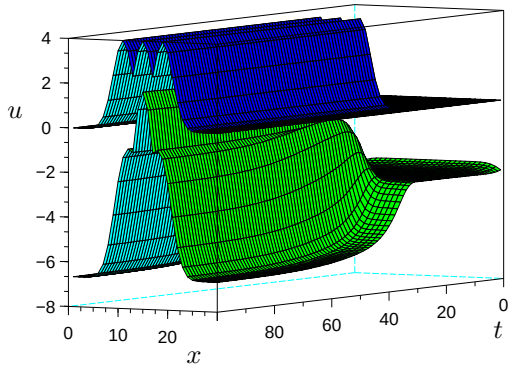
4. Benachbarte Eingangssignale werden zu einer starken Anregung fusioniert (siehe Abbildung 12.2.3(c)).



(a) Entscheidung und Hemmung



(b) Stabilisierung eines kurzzeitigen Signals



(c) Fusion benachbarter Signale

Abbildung 12.2.3: *Eigenschaften eines eindimensionalen dynamischen Systems. Die Aktivierung eines eindimensionalen Feldes $u(x,t)$ (in Grün) ist in Anwesenheit der Anregung $S(x,t)$ (in Blau) über die Zeit t eingezeichnet.*

Das zeitdiskrete System von Toussaint verwendet für die näherungsweise Bestimmung der zeitkontinuierlichen Aktivierungsänderung des dynamischen Feldes

$$\tau \dot{u}(x,t) = f[u(x',t)]$$

die Eulerintegration mit [Tou04]:

$$u(x,t) = u(x,t-1) + \frac{1}{\tau}(f[u(x',t-1)]) \ .$$

12.3 Sensomotorische Eingabe

Der zeitdiskrete Toussaint-Algorithmus verwendet eine gesteuerte Bewegung eines masselosen Punktes im zweidimensionalen Raum. Für die aktuelle Position $\mathbf{y}$ gilt: $\mathbf{y} \in [-1,1]^2$. Die Steuerung des Punktes basiert auf einem globalen 20-dimensionalen Motorvektor $\mathbf{m} \in [0,1]^{20}$. Jedes Element von $\mathbf{m}$ kodiert eine der Richtungen $\phi_i \in \{0°, 18°, ..., 342°\}$ im zweidimensionalen Raum. So sind alle 20 Motoreinheiten in einem Ring angeordnet (siehe Abbildung 12.3.1).

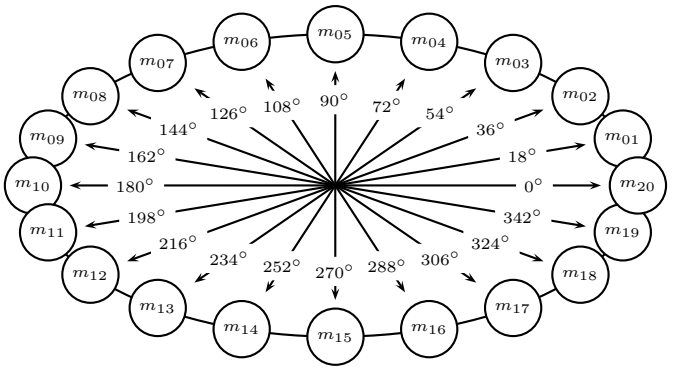


Abbildung 12.3.1: *Globale ringförmige Anordnung der Motoreinheiten m_i, die den anzusteuernden Richtungsvektor kodieren.*

Ausgehend von einer gaußverteilten Belegung des Motorvektors $\mathbf{m}$ wird die Geschwindigkeit $\dot{\mathbf{y}}$ bzw. die Positionsänderung für die Punktbewegung bestimmt:

$$\begin{pmatrix} \dot{y}_1 \\ \dot{y}_2 \end{pmatrix} = \sum_{i=1}^{20} \begin{pmatrix} \cos\phi_i \\ \sin\phi_i \end{pmatrix} m_i \ .$$

Je höher die aktuelle Gaußverteilung des Motorvektors **m** ist, umso höher ist die Geschwindigkeit $\dot{\mathbf{y}}$ in Richtung ϕ_i der Motoreinheit m_i mit maximalem Wert. Somit ergibt sich zum Zeitschritt t die aktuelle Position:

$$\mathbf{y}_t = \mathbf{y}_{t-1} + \dot{\mathbf{y}}_{t-1}.$$

Um eine zufällige und kontinuierliche Richtungsänderung zu gewährleisten, wird eine Richtungsaktivierung **a** mit dem gleichem Aufbau wie die Motoreinheit **m** verwendet. Diese Richtungsaktivierung gibt zu jedem Zeitschritt die angestrebte Richtung vor. Um eine kontinuierliche Richtungsänderung von der alten Motoraktivierung zu einer neuen zu erwirken, verwendet Toussaint ein dynamisches Feld analog zu Erlhagen [ES02]. Die Motoraktivierung ist mit dem folgenden dynamischen Feld beschrieben:

$$\tau_m m_i = -m_i + h_m + a_i + \sum_j w_{ij}\phi(m_j) + \xi \ . \tag{12.1}$$

Dabei gibt τ_m die Geschwindigkeit des dynamischen Systems an, bei der sich die Motoraktivierung an die Aktivierung a_i anpasst. Der Ruhepegel ist durch h_m gegeben und ξ ist ein normalverteilter Fehler. Damit sich eine Gaußverteilung um das Potential von a_i ausprägt, wird der Restterm verwendet. Die Interaktionsstärke **W** enthält die Anregung benachbarter und die Hemmung entfernter Motoreinheiten zueinander. Für die Elemente gilt:

$$w_{ij} = w_E \exp \frac{-(r_i - r_j)^2}{2\sigma_E^2} - w_I.$$

Da die Motoreinheiten in einem Ring angeordnet sind, kann für jede Motoreinheit die Position r_i angegeben werden. Die Distanz $r_i - r_j$ ist dabei die kürzeste Distanz im Ring. Weiterhin ist w_E die maximale Erregung und w_I die maximale Hemmung. Darüber hinaus ist $\phi(m)$ die stückweise lineare Sigmoidfunktion

$$\phi(m) = \begin{cases} 0, & m < 0 \\ m, & 0 \leq m \leq 1 \\ 1, & m > 1 \ . \end{cases}$$

12.4 Sensomotorische Karte

In der sensomotorischen Karte sind sowohl die Motoraktivierungen $\mathbf{m}$ als auch die Sensoraktivierungen $\mathbf{y}$ der Punktbewegung abgespeichert. So verwendet der Algorithmus zur Ausprägung der Sensorkarte eine Variante des wachsenden neuronalen Gases (siehe Abschnitt 9.2). Die Unterschiede dieser Variante im Vergleich zu Fritzkes GNG sind die folgenden:

Es wird immer ein neues Neuron j an die aktuelle Stelle $\mathbf{y}$ eingefügt, wenn der Fehler

$$\tau_e e_i = -e_i + (1 - S_i) \tag{12.2}$$

des Gewinnerneurons i einen Grenzwert $\nu \in [0, 1]$ überschreitet. Die Fehler beider Neuronen i, j werden anschließend auf Null gesetzt. Dabei ist die Sensoreingabe des Neurons i mit der Gaußkurve

$$S_i = \exp \frac{-(\mathbf{s}_i - \mathbf{y})^2}{2\sigma_S^2} \tag{12.3}$$

definiert und vergleicht die aktuelle Position $\mathbf{y}$ mit der Lage der Referenzvektoren aller existenten Neuronen. Alle Verbindungen sind gerichtet, damit die gerichtete Motoraktivierungen gespeichert werden kann. Das Alter age_{ij} jeder Verbindung von Neuron j zu Neuron i wird in jedem Zeitschritt um $h_{ij}\phi(x_j)$ mit

$$age_{ij}^{(t)} = age_{ij}^{(t-1)} + h_{ij}\phi(x_j) \tag{12.4}$$

erhöht. Ist j das nächstliegende und i das zweitnächste Neuron, wird das Alter auf Null verjüngt. Dabei ist das Skalarprodukt

$$h_{ij} = \langle \mathbf{g}_{ij}, \mathbf{m} \rangle \tag{12.5}$$

desto größer, je mehr die aktuelle Motoraktivierung $\mathbf{m}$ mit einem Motorgewichtsvektor $\mathbf{g}_{ij}$ übereinstimmt. So altert eine Verbindung umso schneller, wenn zu der aktuellen Position nahe liegende Verbindungen in die gleiche Richtung der aktuellen Motoraktivierung „zeigen“ und nicht mehr „abgefahren“ werden. Zur Anpassung der Referenzvektoren und der Motorvektoren der Topologie wird eine gewichtete Summe verwendet, um keine zusätzlichen Parameter zu integrieren. Der Motorgewichtsvektor speichert die durchschnittliche Motoraktivierung, die registriert wurde, um von Neu-

ron j aus das Neuron i anzusteuern. Der Durchschnitt des aktuellen Motorvektors $\mathbf{g}_{ij}^{(T)}$ berechnet sich mit:

$$\mathbf{g}_{ij}^{(T)} = \frac{1}{\sum_{t'=1}^{T} \alpha_{ij}^{(t')}} \sum_{t=1}^{T} \alpha_{ij}^{(t)} \mathbf{m}^{(t)} \ .$$

Die Gewichtung erfolgt mittels α_{ij}:

$$\alpha_{ij}^{(t)} = \begin{cases} 1 & \text{wenn } \dot{x}_i > 0 \text{ und } \dot{x}_j < 0 \\ 0 & \text{andernfalls.} \end{cases}$$

Dadurch wird nur dann das Motorgewicht der Verbindung (j, i) der aktuellen Motoraktivierung angeglichen, wenn die Bewegung weg von Neuron j und hin zu Neuron i erfolgt. Analog dazu werden die Referenzvektoren s_i angepasst:

$$\mathbf{s}_{i}^{(T)} = \frac{1}{\sum_{t'=1}^{T} \alpha_{i}^{(t')}} \sum_{t=1}^{T} \alpha_{i}^{(t)} \mathbf{y}^{(t)} \ .$$

Hier ist $\alpha_i^{(t)}$ genau dann 1, wenn das Neuron i zum Zeitschritt t das Gewinnerneuron war, andernfalls ist es Null. Die Aktivität x_i eines Neurons i ist mit dem dynamischen Feld

$$\tau_x x_i = -x_i + h_x + S_i + \eta \sum_j (h_{ij} \tilde{w}_{ij} - w_I) \phi(x_j)) + \xi \tag{12.6}$$

gegeben. Dabei ist h_x der Ruhepegel, S_i die Sensoreingabe (siehe Gleichung 12.3) und ξ ein normalverteilter Fehler. Der Restterm beschreibt die Antizipation der aktuellen Position durch benachbarte Neuronen der sensomotorischen Karte. Die Stärke der Antizipation ist durch η definiert. So wird mit diesem Term jedes Neuron i zusätzlich aktiviert, das in Richtung der aktuellen Motoraktivierung anfahrbar ist. Dabei ist $\tilde{w}_{ij}$ genau dann Null, wenn keine Verbindung von Neuron j nach Neuron i existiert. Andernfalls ist es Eins. Zur Bestimmung von h_{ij} siehe Gleichung 12.5. Die Hemmung ist mit w_I gegeben. Eine grafische Übersicht der sensomotorischen Karte zweier benachbarter Neuronen i und j ist in Abbildung 12.4.1 abgebildet.

Wird eine zufällige Punktbewegung, wie in Abschnitt 12.3 beschrieben, mit einer Dauer von 50000 Zeitschritten angewendet, resultiert die sensomotorische Karte (siehe Abbildung 12.4.2). Aufgrund der Wahl dieser zufälligen Punktbewegung steigt

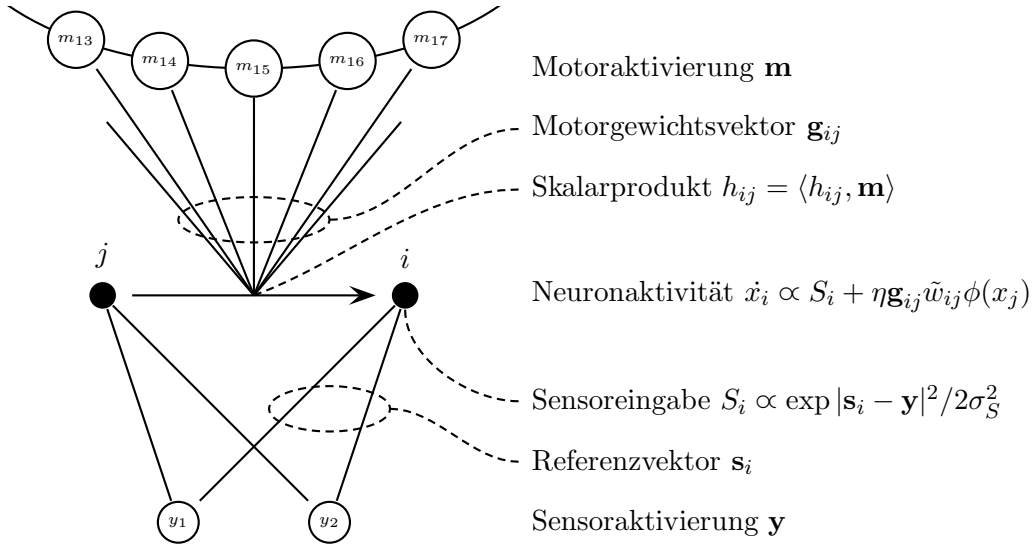


Abbildung 12.4.1: *Sensomotorische Karte und zugehörige Zusammenhänge für zwei benachbarte Neuronen i, j.*

die Fehlerrate der Motorwerte am Rand des Netzes. Dies ist daran zu erkennen, dass die roten Motorrichtungen von der Richtung der zugehörigen Verbindung abweichen. Dabei werden zur graphischen Veranschaulichung zu jeder topologischen Verbindung zwischen Neuron i und Neuron j die gespeicherte Motorbewegung als Geschwindigkeitsvektor (siehe Gleichung 12.3) in Rot auf 2/3 der gerichteten Verbindung eingezeichnet. Die verwendeten Parameter sind: $h_m = -1$, $\tau_m = 5$, $w_E = 1$, $w_I = 0.5$, $\sigma_E = 2$, $\tau_e = 10$, $\sigma_S = 0.2$, $h_x = 0$, $\eta = 0$, $\tau_x = 2$, $\nu = 0.2$ und $max_{age} = 300$.

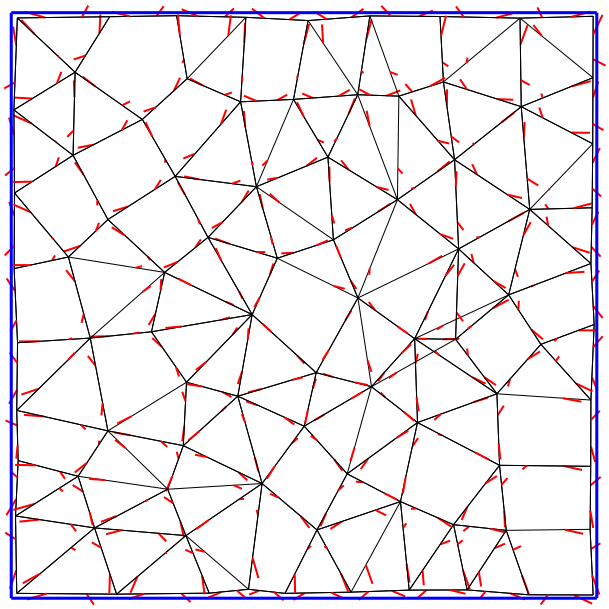


Abbildung 12.4.2: *Ausprägung einer sensomotorische Karte nach Verwendung einer zufälligen Punktbewegung (siehe Abschnitt 12.3).*

12.5 Zielgerichtetes Verhalten

Wenn die Trainingsphase abgeschlossen ist und sich eine sensomotorische Karte ausgeprägt hat, beginnt die Planungsphase des Algorithmus. In dieser Phase werden zufällig gewählte Neuronen ausgehend von der aktuellen Position angefahren. Um ein solches zielgerichtetes Verhalten zu organisieren, wird jedem Neuron i abhängig von dem Zielneuron k eine Anregung

$$r_i = \frac{1}{Z} \exp \frac{-(\mathbf{s}_i - \mathbf{s}_k)^2}{2\sigma_R^2}$$

zugewiesen. Dabei wird Z so gewählt, dass $\sum_i r_i = 1$ ist. Je näher ein Neuron i am Zielneuron k liegt, umso größer ist die Anregung r_i. Damit die Topologie des Netzes berücksichtigt wird, ist ein weiterer dynamischer Prozess integriert, der zu jedem Zeitschritt für jedes Neuron i zum einen die distanzbasierte Anregung r_i und zum anderen die maximale Anregung eines benachbarten Neurons j berücksichtigt. Der

dynamische Prozess ist durch

$$\tau_v v_i = -v_i + r_i + \gamma \max_j (w_{ji} v_j)$$

gegeben. Der Bruchteil γ gibt die Stärke des Einflusses des maximalen Potentials v_j des Nachbarneurons j über alle Nachbarneuronen an.

So verteilt sich von dem zufällig gewählten aktuellen Zielneuron bis hin zur aktuellen Position im Eingaberaum ein Potentialfeld $\mathbf{v}$ über alle Neuronen i. Zur Visualisierung des Einflusses der Parameter σ_R und γ auf das Potentialfeld inklusive der Anregung $\mathbf{r}$ dient die Abbildung 12.5.1.

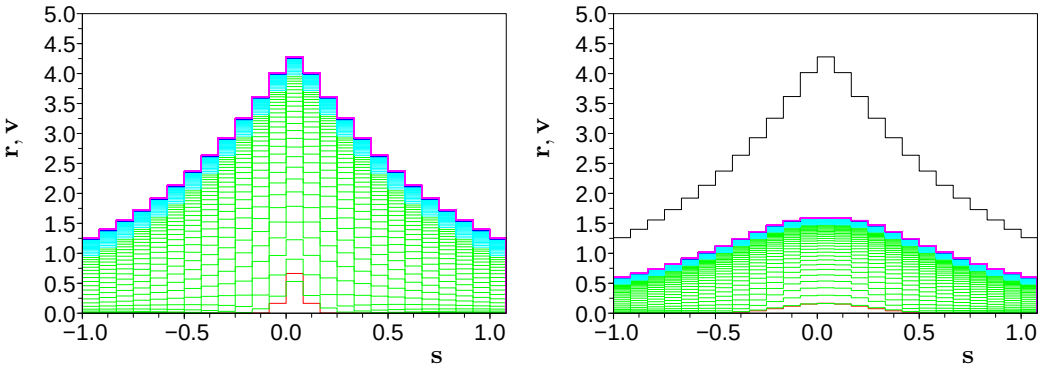
Dort sind für jedes Neuron i beide Werte (r_i in Rot und V_i in den restlichen Farben) abgebildet. Die Geschwindigkeit des dynamischen Feldes τ_v wird hier auf 5 festgelegt und das Zielneuron bleibt konstant das mittlere Neuron.

Für die drei vorliegenden Parametersätze kann ausgehend von der Anregung $\mathbf{r}$ (in Rot) die Ausbildung des Potentialfeldes $\mathbf{v}$ (von Grün nach Magenta) bis in den Fixpunkt $v_i^* = r_i + \gamma max_j(w_{ji} v_j^*)$ verfolgt werden. Die Wahl von σ_R beeinflusst die Gaußkurve von $\mathbf{r}$ und $\mathbf{v}$. Je größer σ_R ist, umso flacher und runder ist die Kurve (siehe Bild 12.5.1(b)). Je kleiner σ_R ist, umso höher und spitzer ist die Kurve. Der Parameter γ hat Einfluss auf den Fixpunkt. Je kleiner γ gewählt wird, umso mehr nähert sich das Potential $\mathbf{v}$ der Anregung $\mathbf{r}$ an und umso schneller konvergiert das Potential in den Fixpunkt. Je größer γ ist, umso größer wird das Potentialfeld und umso mehr Zeitschritte werden benötigt, um den Fixpunkt zu erreichen. Bei dem Fall $\gamma \geq 1$ divergiert das Potentialfeld für jedes Neuron i mit fortschreitender Zeit gegen Unendlich (siehe Bild 12.5.1(c)). Zum Vergleich ist in die Bilder 12.5.1(b) und 12.5.1(c) jeweils das Endpotential von Bild 12.5.1(a) schwarz eingezeichnet.

Über das so gewonnene Skalarfeld $\mathbf{v}$ lässt sich in jedem Zeitschritt mit Hilfe des Gradienten $v_j - v_i$ der aktuelle Zielwert der Motoraktivierung $\mathbf{a}$ berechnen. So wird eine entsprechend dem Gradienten gewichtete Summe

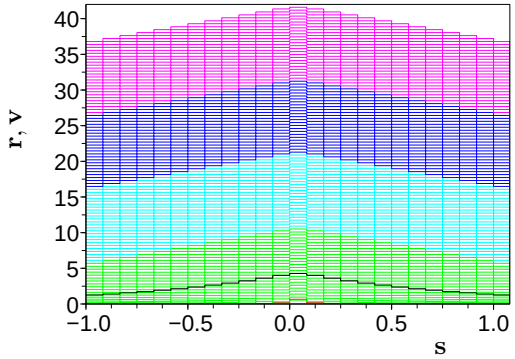
$$\mathbf{a} = \frac{1}{Z} \sum_{i,j} x_i w_{ji} (v_j - v_i) \mathbf{g}_{ji} \tag{12.7}$$

über alle gelernten Motorgewichtsvektoren $\mathbf{g}_{ji}$ berechnet. Es wird Z so gewählt, dass für den Betrag der Zielmotoraktivierung $|\mathbf{a}| = 1$ gilt. Mit diesem dynamischen System ist eine Kürzeste-Pfad-Suche über alle Referenzvektoren implementiert.



(a) $n = 25, \sigma_R = 0.05, \gamma = 0.9$

(b) $n = 25, \sigma_R = 0.2, \gamma = 0.9$



(c) $n = 25, \sigma_R = 0.05, \gamma = 1$

Abbildung 12.5.1: *Für eine Kette von* 20 *benachbarten Neuronen sind die Anregungen* r_i *und die Potentiale* v_i *für jedes Neuronen* i *über* 500 *Zeitschritte als Stufenfunktion angezeichnet. Die Neuronen sind äquidistant im Gesamtintervall von* $[-1, 1]$ *angeordnet. Für die Zeitintervalle* $[1, 125], [126, 250], [251, 375], [376, 500]$ *zeigen die Farben Grün, Cyan, Blau und Magenta die Stärke des Potentialfeld* $\mathbf{v}$ *an. In Rot ist die Anregung* $\mathbf{r}$ *abgebildet. Für jeden Zeitschritt ist das zentrale Neuron das Zielneuron. In den Abbildungen (b) und (c) ist das Potentialfeld von (a) nach 500 Zeitschritten zum Vergleich in Schwarz eingezeichnet.*

Das zielgerichtete Verhalten hat wegen der Berechnung des Skalarfeldes $\mathbf{v}$ eine Zeitkomplexität von $O(n^2)$ pro Zeitschritt und ist damit aufwändiger als die Netzwerkausbildung mit der Zeitkomplexität von $O(n)$. Verglichen mit dem Floyd-Warshall-Algorithmus der alle kürzesten Pfade mit der Zeitkomplexität von $O(n^3)$ bestimmt, hat sich das Potential ausgehend vom Zielneuron bis zum Gewinnerneuron nach durchschnittlich $n/2$ Zeitschritten aufgebaut.

13 Implementierung des Toussaint-Modells und Resultate

13.1 Aufbau des Experimentes

Als Experimentierplattform dienen die beiden Schultermotoren des rechten Armes des A-Serie-Roboters (siehe *ShoulderRRoll* und *ShoulderRPitch* in Abbildung A.1(b)).

Dieser ist auf einem Brett an der Hüfte fixiert. Auf dem Brett ist zusätzlich eine 10 cm lange Holzleiste befestigt, damit ein Überdrehen und die damit verbundene Überbelastung der Verbindungsleitungen des Pitch-Motors verhindert wird. Um die Beschädigung des rechten Armes durch Stöße an der Holzleiste zu minimieren, sind die Platine des rechten Armes und die Leiste gepolstert (siehe Abbildung 13.1.1).

Die Implementierung des Verfahrens des Toussaint-Modells und der Kommunikation mit den Motoren des Roboters wurden in C++ bewältigt und die Ausführung des Programms auf einen Desktop-PC ausgelagert.

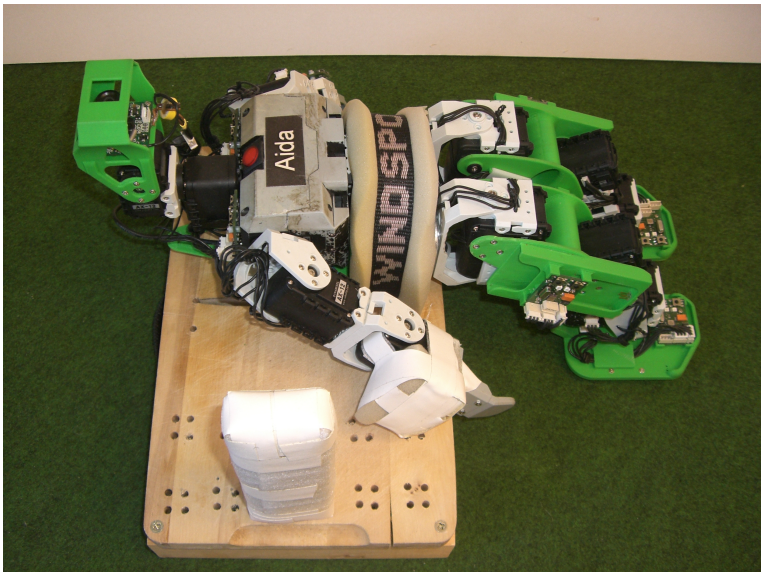


Abbildung 13.1.1: *Der fixierte humanoide Roboter.*

Das Ziel des Experimentes ist zum einen die Ausbildung einer sensomotorischen Karte in der Trainingsphase mit Hilfe einer zufälligen Motoransteuerung. Zum anderen sollen in der Planungsphase zufällig ausgewählte Neuronen der sensomotorischen Karte zielsicher angesteuert werden. Als Grundlage dienen das beschriebene Modell

von Toussaint (siehe Abschnitt 12) und seine Experimente [Tou06].

13.2 Anpassungen des Toussaint-Modells

Da die Ergebnisse von Toussaint ausschließlich auf einer theoretischen Motoransteuerung basieren [Tou06], werden für die Verwendung der gegebenen humanoiden Hardware die folgenden drei Anpassungen der Motorsteuerung vorgenommen:

1. Um eine geradlinige Armbewegung entsprechend des resultierenden Geschwindigkeitsvektors $\dot{\mathbf{y}}$ trotz Schwerkraft zu gewährleisten, werden die folgenden Ansteuerungen der beiden Motoren verwendet:

   (a) Der Motor $i \in [1, 2]$ wird mit der Wahrscheinlichkeit $p(i) = |\dot{y}_i| / \sum_{j=1}^{2} |\dot{y}_j|$ aktiviert.

   (b) Ist ein Motor aktiv, wird er mit einem konstanten Drehmoment angesteuert.

   (c) Falls ein Motor inaktiv ist, behält er seine aktuelle Position bei, um ein Abrutschen in Richtung der Gravitation zu verhindern.

   Die resultierende Bewegung entspricht einer stufenförmigen Approximation in Richtung des Geschwindigkeitsvektors. Falls beide Motoren aktiv sind, resultiert eine nahezu geradlinige Bewegung mit leichter Krümmung in Richtung der Gravitation.

2. Die nächste Anpassung dient der zeitlichen Verkürzung der Explorationsphase. Hier wird nicht zu jedem Zeitschritt mit einer bestimmten Wahrscheinlichkeit p die Richtungsaktivierung $\mathbf{a}$ zufällig gewählt, sondern eine rudimentäre Kollisionserkennung verwendet. Nach jeder detektierten Kollision mit der Konfigurationsraumgrenze wird die Richtungsaktivierung $\mathbf{a}$ so geändert, dass die neue Richtung zufällig im Intervall $[\phi_a + 90°, \phi_a + 270°]$ liegt, wobei ϕ_a die aktuelle Richtung ist. Die resultierende explorative Bewegung ist geradlinig und wird an den Konfigurationsraumgrenzen reflektiert.

3. Des Weiteren wird die Motoransteuerung während der Trainingsphase mit der Gleichung 12.1 auf das entsprechende dynamische Feld mit unabhängiger Aktivierung vereinfacht. Diese Entscheidung wird vor dem Hintergrund getroffen, dass die Richtung der Motoransteuerung keine Stabilisierungseffekte benötigt,

da jede mögliche Aktivierungseingabe a_i gleichmäßig berücksichtigt und nicht gehemmt werden soll. Es gilt nun:

$$\tau_m \dot{m}_i = -m_i + h_m + a_i.$$

Dabei wird entsprechend der aktuellen Zielrichtung ϕ_i um a_i ein Gaußpotential mit $\sigma_E = 2$ erzeugt. In Abbildung 13.2.1 sind exemplarisch drei verschiedene Motoraktivierungen über alle $N = 20$ Motoreinheiten eingezeichnet. Die verwendeten Parameter des dynamischen Feldes sind $\tau_m = 5$ und $h = 0$. Die resul-

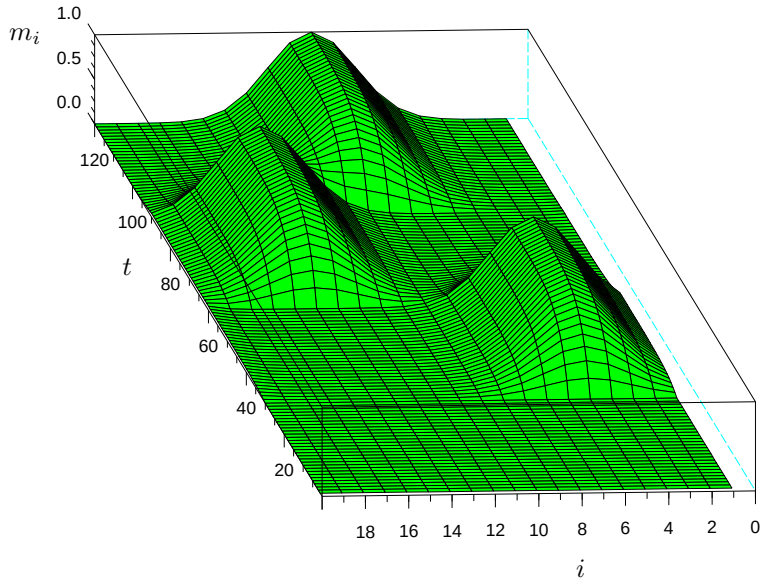


Abbildung 13.2.1: *Die Motoraktivierung* **m** *nach dreimaligen Wechsel der Richtungsaktivierung* **a**. *Nach den Zeitschritten* $30, 60, 90$ *wurde eine gaußverteilte Richtungsaktivierung über den Motoreinheiten* $5, 15, 10$ *ausgehend von keiner Richtungsaktivierung (Zeitschritte* 0 *bis* 30*) induziert.*

tierenden Motoraktivierungen liegen somit im Intervall $[0, 1]$ und zeigen einen kontinuierlichen Übergang zur Richtungsaktivierung **a**. Die erzielten Resultate benötigen weniger Rechenaufwand und sind identisch zu denen von Toussaint.[5]

Die Wahl des geeigneten Parametersatzes orientiert sich an den Ergebnissen von Toussaint und wird an die vorliegende Hardware angepasst. Da sich der Einfluss jedes

[5]Mehrere Resultate sind in Videodateien unter `http://user.cs.tu-berlin.de/~mtoussai/04-smm/index.html` zu sehen.

Parameters auf das Gesamtverhalten anhand der beschriebenen Grundlagen nachvollziehen lässt, werden die verwendeten Parametersätze nicht einzeln begründet. Einzig zu erwähnen ist die Wahl von $\eta = 0$. So würden für $\eta > 0$ nicht die örtlich lokalen Neuronen aktiviert werden, sondern die Neuronen die in Bewegungsrichtung liegen (siehe Abschnitt 12.4). Diese Antizipation wird in den Experimenten deaktiviert, um die Komplexität so gering wie möglich zu halten. Die Tabelle 13.2.1 enthält alle verwendeten Parameter für den angepassten Algorithmus.

Motorsteuerung	τ_m	h_m					
	5	-1					
Sensomotorische Karte	n	ν	σ_S	τ_e	τ_x	a_{max}	h_x
	60	0.2	0.15	10	2	2000	0
Zielverhalten	σ_R	τ_v	γ				
	0.05	5	0.9				

Tabelle 13.2.1: *Parameter der angepassten Implementierung des Toussaint-Modells.*

13.3 Wahl des Parametersatzes und Zeitanalyse des Systems

Bei der Wahl der Parameter müssen die beiden Konstanten σ_S und ν besonders sorgfältig gewählt werden. Diese Kenngrößen beeinflussen entsprechend der Gleichungen 12.2 den durchschnittlichen Abstand aller benachbarten Neuronen zueinander. Zusammen mit der Gleichung 12.6 bestimmen beide Parameter, wie viele Neuronen (in der Nähe der aktuelle Position im Konfigurationsraum) signifikant aktiviert sind. Dabei wird im weiteren Verlauf diese Signifikanz mit $x_i > 0.01$ festgelegt. Die Größe dieses signifikanten Aktivierungsradius bestimmt das Verhalten des Algorithmus an zwei Stellen.

1. In der Trainingsphase wird mit diesem Radius die Menge der alternden Neuronenverbindungen um die aktuelle Position im Konfigurationsraum bestimmt (siehe Gleichung 12.4). Ein zu großer Radius würde ein zu schnelles Altern und damit Löschen von Kanten zur Folge haben, wodurch das neuronale Netz nicht die optimale Annäherung an die Delaunay-Triangulation erreicht.

2. In der Planungsphase legt der Aktivierungsradius fest, wie viele gelernte Motorvektoren $\mathbf{g}_{ij}$ in der Nähe der aktuellen Position im Konfigurationsraum in die Berechnung der resultierenden Motoraktivierung $\mathbf{a}$ eingehen (siehe Gleichung 12.7).

Wird dieser Aktivierungsradius zu groß gewählt, können an den Grenzen des neuronalen Netzes falsche Motoransteuerungen berechnet werden. Zur Veranschaulichung des möglichen Fehlverhaltens bei einer zu starken Neuronenaktivierung $\mathbf{x}$ dient die Abbildung 13.3.1.

Solange sich innerhalb des Radius der Neuronaktivität $\mathbf{x}$ die Neuronen gleichverteilt befinden, resultiert in jedem Fall der korrekt gemittelte Geschwindigkeitsvektor $\dot{\mathbf{y}}$ (siehe 13.3.1(a)). Dieser zeigt dann in Richtung des Zielneurons.

Wenn sich jedoch die Roboterkonfiguration an der Konfigurationsraumgrenze und das Zielneuron etwas weiter im Konfigurationsraum befindet, allerdings nicht in der Peripherie des $\mathbf{x}$-Radius, kommt es zu einer fehlerhaften Richtungsbestimmung (siehe 13.3.1(b)). Darüber hinaus werden entsprechend der Gleichung 12.7 die Motoransteuerungen $\mathbf{g}_{ji}$ aufsummiert, die vom Zielneuron wegführen. Diese werden entsprechend dem Gradienten $(v_j - v_i)$ gewichtet. Dadurch wird der fehlerhafte Effekt verstärkt, so dass $\dot{\mathbf{y}}$ noch weniger mit der Richtung zum Zielneuron übereinstimmt. Der Übersichtlichkeit wegen wurden diese zurückführenden Motorvektoren nicht in die Grafik eingezeichnet.

Im Falle einer geringeren Neuronenaktivierung zeigt der resultierende Geschwindigkeitsvektor in die korrekte Richtung (siehe 13.3.1(c)). Nun bestimmen ausschließlich die direkt benachbarten gelernten Motoransteuerungen die aktuell anzusteuernde Richtung.

Nachdem die Experimentieranordnung und die Parameterwahl erläutert wurden, wird nun die Zeitmessung für die Berechnungen jedes Zeitschrittes durchgeführt. Damit wird überprüft, ob die Einspeisung neuer Sensorsignale in das Modell in gleichen Zeitabständen gewährleistet ist. Die Messung mit 50 Referenzvektoren ergibt, dass jeder Berechnungsschritt sowohl während der Trainingsphase als auch in der Planungsphase innerhalb von zwei Millisekunden (siehe Abbildung 13.3.2) fertiggestellt wird. Somit kann entsprechend dem 100Hz-Takt der Roboter-Hardware innerhalb jedes 10ms-Fensters ein ganzer „Sense-Think-Act“-Zyklus abgearbeitet werden. Wobei für den „Think“-Schritt jede Zeitgrenze von vier Millisekunden eingehalten werden kann.

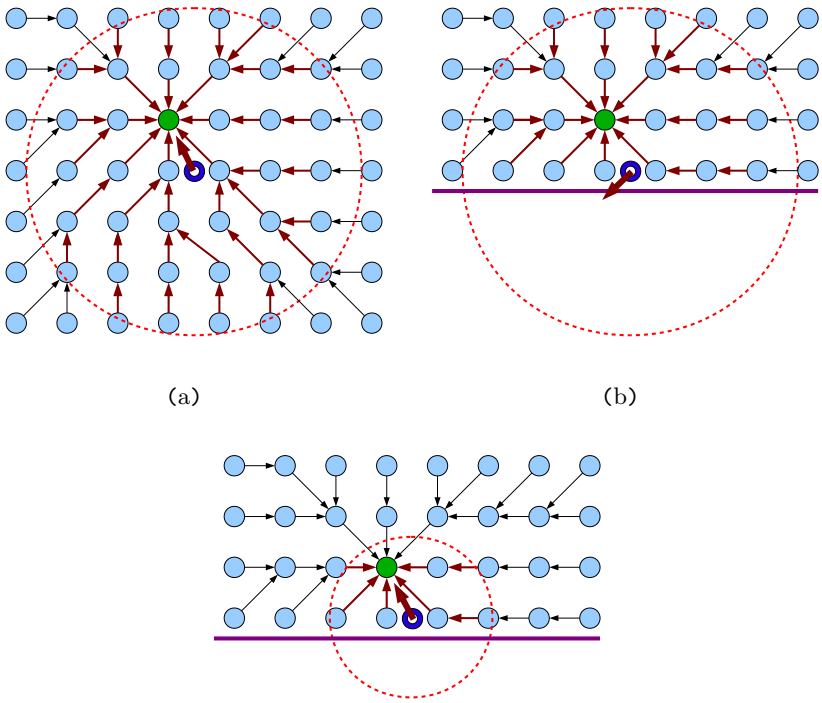


Abbildung 13.3.1: *(a)-(b): Der Toussaint-Algorithmus mit resultierendem Geschwindigkeitsvektor bei zu starker Neuronenaktivierung* $\mathbf{x}$*. Dieser Vektor kann am Rand des Eingaberaumes (b) in die falsche Richtung zeigen. (c): Der korrekte Richtungsvektor bei geringer Neuronenaktivierung* $\mathbf{x}$*. Der blaue Kreisring zeigt die aktuelle Roboterkonfiguration und sein roter Pfeil entspricht dem resultierenden Geschwindigkeitsvektor* $\dot{\mathbf{y}}$*. Die hellblauen Kreise lokalisieren die Neuronen im gelernten Eingaberaum, dabei ist das grüne Neuron das Zielneuron. Die dazugehörigen Pfeile sind die gelernten Motoransteuerungen, die in Richtung des Potentialfeldes* $\mathbf{v}$ *mit maximalen Gradienten* $(v_j - v_i)$ *zeigen. Die rote gestrichelte Kreislinie entspricht der signifikanten Aktivierung mit* $x_i > 0.01$ *der Neuronen. Nur die roten Pfeile gehen signifikant in die Rechnung ein. Die Motorvektoren, die nicht zum Zielneuron führen, wurden nicht eingezeichnet.*

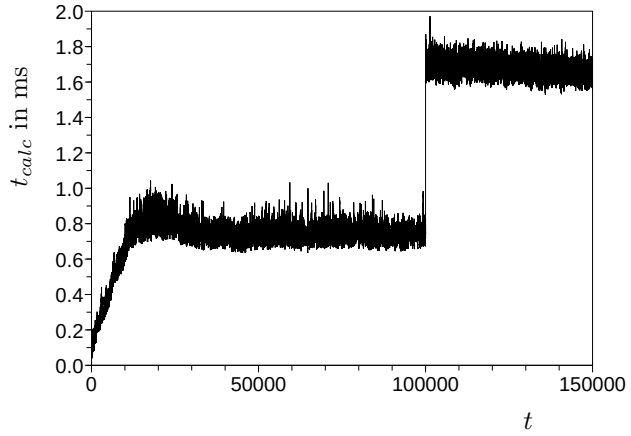


Abbildung 13.3.2: *Ausführungsgeschwindigkeit in Millisekunden aller Berechnungsschritte t. Ab $t =$ 100000 Zeitschritten setzt die Planungsphase ein.*

13.4 Resultate

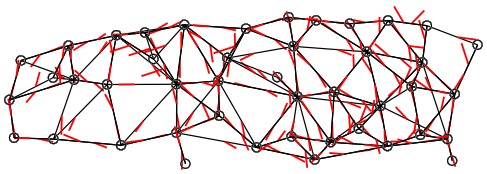
Der Verlauf der Ausprägung der sensomotorischen Karte in der Trainingsphase ist auf den Bildern 13.4.1(a) und 13.4.1(b) zu erkennen.

Als Gütefunktion der ausgeprägten sensomotorischen Karte wird der Fehler $E^{(t)}$ mit

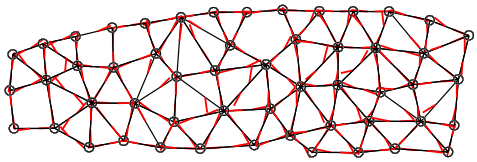
$$E^{(t)} = \frac{\sum_{i,j} |\measuredangle(i,j)^{(t)} - \measuredangle(\mathbf{g}_{ij}^{(t)})|}{|\mathcal{C}|}$$

berechnet. Dieser gibt zu jedem Zeitschritt t die durchschnittliche Differenz zwischen den Winkeln jeder Verbindung $(i,j) \in \mathcal{C}$ und dem Winkel des entsprechenden Motorgewichtsvektors $\mathbf{g}_{ij}$ an. Wie jedoch der Fehlerkurve aller Verbindungsfehler in Abbildung 13.4.1(c) entnommen werden kann, konvergiert der Fehler $E^{(t)}$ nicht gegen Null. Das liegt zum einen an der Motoransteuerung, die nicht zu hundert Prozent den aktuellen Geschwindigkeitsvektor realisieren kann. Zum anderen detektiert die Kollisionserkennung nach frühestens zehn Zeitschritten ein Hindernis. Dadurch registrieren am Rande liegende Neuronen auch Motoransteuerungen, die über die Grenzen des Eingaberaumes hinweg gerichtet sind. Der dritte Grund ist die ständige Anpassung der Delaunay-Triangulation. So sind die Motoransteuerungen neuer Kanten anfänglich ungenau (siehe Abbildung 13.4.1(a)).

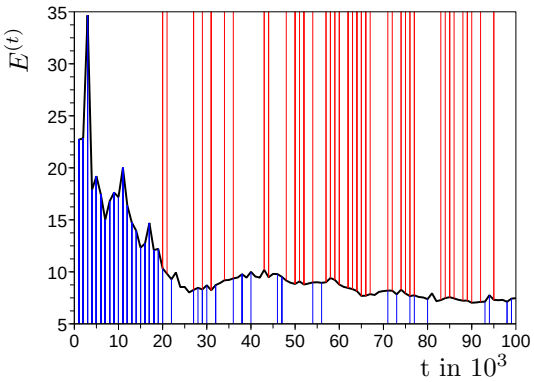
Nach der Trainingsphase wird der Roboterarm in der Planungsphase angesteuert. Hier fährt der Roboter mit Hilfe des Zielverhaltens (siehe Abschnitt 12.5) zufällig



(a) $t = 10000$



(b) $t = 100000$



(c) Fehlerkurve über die Zeitschritte

Abbildung 13.4.1: *(a)-(b):Ausprägung der sensomotorischen Karte zu den verschiedenen Zeitschritten. Auf jeder unidirektionalen Kante zwischen zwei Neuronen ist der gelernte Geschwindigkeitsvektor in Rot eingezeichnet. (c): Fehler $E^{(t)}$ nach jeweils* 1000 *Zeitschritten in Grad. Die senkrechten Linien zeigen eine neue (Blau) oder gelöschte (Rot) Verbindung im entsprechenden Zeitschritt an.*

gewählte Neuronen in seinem Konfigurationsraum an. Dabei wird die Netzwerkausbreitung weiterhin so wie in der Trainingsphase fortgeführt.

In Abbildung 13.4.2 ist der Netzwerkfehler während der Planungsphase eingezeichnet. Um eine Aussage über die Güte der Motorsteuerungen zum jeweiligen

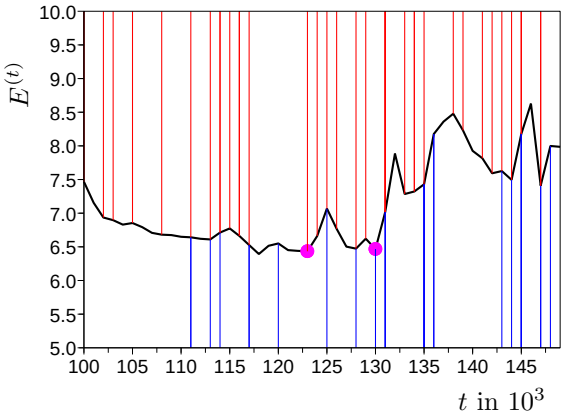


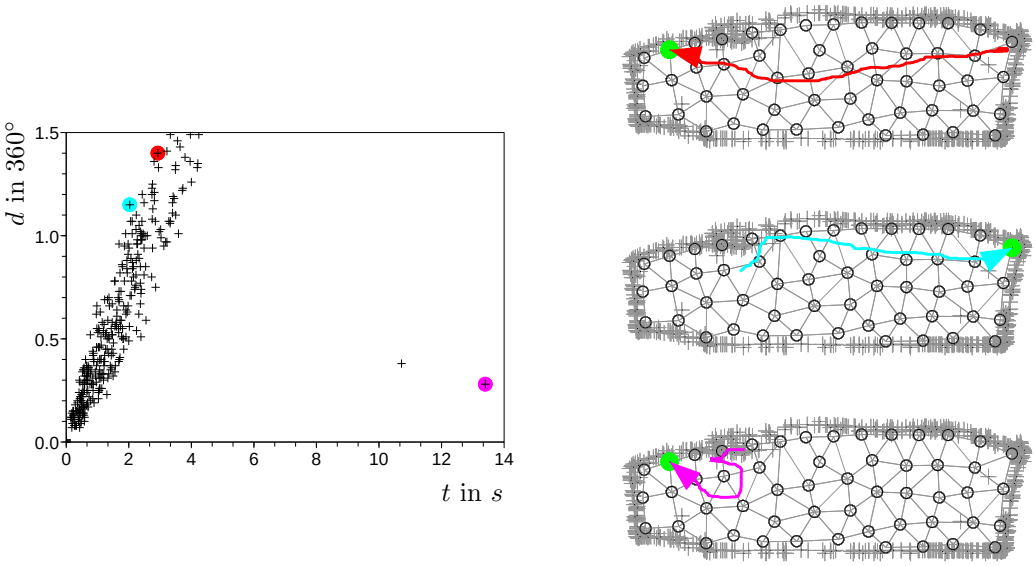
Abbildung 13.4.2: *Fehler $E^{(t)}$ der Planungsphase nach jeweils* 1000 *Zeitschritten in Grad. Die senkrechten Linien zeigen eine neue (Blau) oder gelöschte (Rot) Verbindung im entsprechenden Zeitschritt an. Die magentafarbenen Punkte zeigen die Zeitschritte an, zu denen der Roboter eine Kollision mit einem Hindernis hatte und dort kurzzeitig verweilte.*

Zielneuron treffen zu können, wird die verbrauchte Zeit bis zum neuen Zielneuron gegen die kürzeste Strecke im Konfigurationsraum eingezeichnet (siehe Abbildung 13.4.3(a)).

Da der Konfigurationsraum annähernd konvex ist, kann für die kürzeste Distanz der euklidische Abstand zwischen dem alten und dem neuen Zielneuron gemessen werden. Jedes angesteuerte Zielneuron konnte in dem Experiment erreicht werden. Für alle bis auf zwei angefahrene Zielneuronen kann eine lineare Abhängigkeit, innerhalb eines Fehlerintervalls, zwischen beiden Größen entdeckt werden. Somit kann festgehalten werden, dass sich die Roboterarmbewegung bzw. die zugehörige Bewegung im Konfigurationsraum an dem kürzesten Pfad zum Zielneuron orientiert.

Die beiden Ausreißer sind das Resultat von Kollisionen mit den Grenzen des Konfigurationsraumes. Erst nach der Löschung von Neuronenverbindungen konnte der Roboterarm innerhalb seines Konfigurationsraumes über eine Bewegungstrajek-

torie[6] in Form eines Kreisbogens zum Zielneuron gelangen. Die zugehörige Bewegungstrajektorie ist in Magentafarbe abgebildet (siehe Abbildung 13.4.3(b)). Dabei ist erkenntlich, dass an dieser Stelle der Konfigurationsraum keine harten Grenzen aufweist. Dadurch konnte sich das neuronale Netz bis hinter die „weichen“ Grenzen (Polsterung der Experimentierumgebung) ausbreiten.



(a) Benötigte Zeit und Distanz zum neuen Zielneuron

(b) Zugehörige Bewegungstrajektorien

Abbildung 13.4.3: *Auf Bild (a) sind für jedes erreichte Zielneuron die verbrauchte Zeit und die euklidische Distanz vom alten bis zum neuen Zielneuron im Konfigurationsraum der beiden Motoren des Roboterarmes eingezeichnet. Auf den restlichen Bildern sind drei korrespondierende Bewegungstrajektorien zum neuen Zielneuron (grüner Kreis) abgebildet. Jedes graue Kreuz steht für ein detektiertes Hindernis.*

Die Abbildung 13.4.4 zeigt die Richtung des Potentialfeldes $\measuredangle(\mathbf{v})$ in Abhängigkeit der aktuellen Roboterarmkonfiguration, die Richtung des resultierenden Geschwindigkeitsvektors $\measuredangle(\dot{\mathbf{y}})$ und die durch die Motorsteuerung gefahrene Richtung des Roboters im Konfigurationsraum.

Für $\measuredangle(\mathbf{v})$ werden in Abhängigkeit der Roboterkonfiguration die Winkel aller existenten Neuronenverbindungen im euklidischen Raum ohne den Einfluss der gemit-

[6] Wenn sich der Roboterarm in seinem Arbeitsraum bewegt, beschreibt eine Sequenz seiner Bewegungen eine Kurve im Konfigurationsraum. Diese Kurve wird im Folgenden als Bewegungstrajektorie bezeichnet.

telten gelernten Motoransteuerungen betrachtet. Es gilt:

$$\measuredangle(\mathbf{v}) = \measuredangle(\sum_i x_i \tilde{w}_{ji}(v_j - v_i)(\mathbf{s}_j - \mathbf{s}_i)).$$

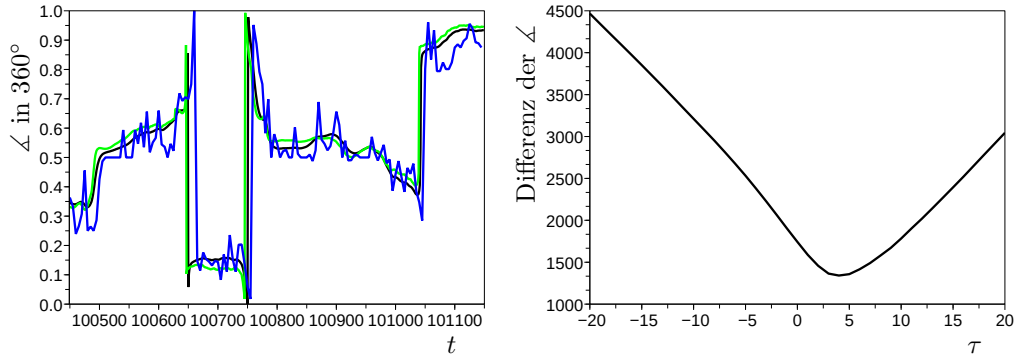
Zur Veranschaulichung wurde exemplarisch das Zeitschrittintervall $[100500, 101100]$ gewählt. Abbildung 13.4.4(a) zeigt die träge Anpassung der Richtung des angesteuerten Geschwindigkeitsvektors $\measuredangle(\dot{\mathbf{y}})$ (Schwarz) an die Zielrichtung $\measuredangle(\mathbf{v})$ (Grün). Die Richtungen liegen im periodischen Intervall von $[0°, 360°]$. Der Abbildung kann entnommen werden, dass die Richtungen der gelernten Motoransteuerungen nicht exakt den Zielrichtungen im euklidischen Raum entsprechen. Andernfalls dürfte das dynamische Feld $\measuredangle(\dot{\mathbf{y}})$ nicht den Wert von $\measuredangle(\mathbf{v})$ während eines monotonen Anstiegs überschreiten bzw. während eines monotonen Abstiegs unterschreiten. Des Weiteren ist der oszillierende Charakter der implementierten Motoransteuerung entlang der Richtung des Geschwindigkeitsvektors $\measuredangle(\dot{\mathbf{y}})$ anhand der blauen Kurve zu erkennen. Der Grund dieser Schwankungen liegt in der Umsetzung der oben beschriebenen, einmotorigen Ansteuerung der Motoren. In Abbildung 13.4.4(b) ist die quadratische Differenz

$$S = \sum_t \sum_\tau (\measuredangle(\mathbf{v}(t)) - \measuredangle(\dot{\mathbf{y}}(t+\tau)))^2$$

für die 50000 Zeitschritte innerhalb der Planungsphase eingezeichnet. Im Durchschnitt werden vier Zeitschritte benötigt, bis die Anpassung von $\measuredangle(\dot{\mathbf{y}})$ an $\measuredangle(\mathbf{v})$ erfolgt ist. Für die durchschnittliche Abweichung in Grad der Richtung des Geschwindigkeitsvektors von der Richtung des positionsabhängigen Potentialfeldes über alle m Zeitschritte, gilt:

$$\frac{\sum_{t=1}^{m} \measuredangle(\mathbf{v}(t)) - \measuredangle(\dot{\mathbf{y}}(t+4))}{m} = 5.94 \ .$$

Diese durchschnittliche Ungenauigkeit aller angesteuerten Richtungen des Roboterarmmotoren in der Planungsphase korreliert dabei mit dem Netzwerkfehler in der Planungsphase (siehe Abbildung 13.4.2) über alle Verbindungen.



(a) Richtungen von $\mathbf{v}$, $\dot{\mathbf{y}}$ und der Roboterkonfiguration

(b) Quadratische Differenz

Abbildung 13.4.4: *Auf Bild (a) sind die Richtungen des Potentialfeldes $\measuredangle(\mathbf{v})$ in Abhängigkeit der Roboterarmkonfiguration (Grün), des Geschwindigkeitsvektors $\measuredangle(\dot{\mathbf{y}})$ (Schwarz) und der letzten beiden Roboterarmkonfigurationen (Blau) eingezeichnet. Für das Zeitintervall $t \in [1, 50000]$ wurde auf (b) die quadratische Differenz zwischen der Richtung $\measuredangle(\mathbf{v}(t))$ und $\measuredangle(\dot{\mathbf{y}}(t+\tau))$ mit $\tau \in [-20, 20]$ eingezeichnet.*

14 Zusammenfassung von Teil III

Im theoretischen Teil wurde das Toussaint-Modell zur Ausprägung einer sensomotorischen Karte für die Erzeugung einer zielgerichteten Punktbewegung beschrieben. Das Verhalten des Toussaint-Modells kann in zwei Schritte unterteilt werden:

1. Während einer zufälligen Punktbewegung im Konfigurationsraum wird mit Hilfe eines neuronalen Gases eine sensomotorische Karte für diese Punktbewegung ausgebildet.

2. Ausgehend von einer beliebig vorgegebenen Punktposition wird eine theoretische Motorbewegungssequenz für einen kurzen Weg zu einer beliebigen Position jeweils im Eingaberaum erzeugt.

Das Toussaint-Modell weist folgende signifikante Merkmale auf:

1. Zur Ausprägung der sensomotorischen Karte wird eine Variante des Verfahrens *Growing Neural Gas* (GNG) mit wenigen Parametern verwendet.

2. Aufgrund der Implementierung von dynamischen Feldern zeigt das Verfahren sogar bei sprunghaften Eingaben kontinuierliche Verhaltensänderungen und ist damit für die Verarbeitung von ungefilterten Sensordaten gut geeignet.

3. Da die Explorationsphase der Punktbewegung für alle Zeitschritte der Trainingsphase zufällig ist, benötigt das Toussaint-Modell für große Konfigurationsräume wesentlich mehr Zeitschritte als bei einer gerichteten Exploration.

4. Die Zeitkomplexität der Bestimmung des Potentialfeldes, das den kürzesten Weg zum Zielneuron anzeigt, ist im Durchschnitt nur halb so hoch, wie die des Floyd-Warshall-Algorithmus zur Bestimmung aller kürzesten Pfade.

Im praktischen Teil wurde das Toussaint-Modell der Hardware eines Roboterarmes angepasst und angewendet. Dabei wurden die folgenden Ergebnisse erarbeitet:

1. Die Anwendbarkeit des angepassten Toussaint-Verfahrens ist gegeben, da in der Planungsphase alle zufällig gewählten Zielneuronen im gesamten Konfigurationsraum erfolgreich angesteuert werden konnten.

2. Durch die durchgängige Verwendung von dynamischen Feldern zeigt das System ein kontinuierliches Verhalten trotz schwankender oder verrauschter Eingaben.

3. Zeitmessungen zeigen, dass im Rahmen des 100Hz-Taktes der Motoren für jedes 10ms Zeitfenster eine vollständige Berechnung eines Zeitschrittes im Toussaint-Modell gewährleistet ist.

4. Durch die erarbeitete und implementierte Kollisionserkennung wird während der Trainingsphase die Explorationszeit verringert.

5. Die mit Abschluss der Trainingsphase ausgebildete sensomotorische Karte weist eine durchschnittliche Richtungsungenauigkeit der Motorvektoren von unter zehn Grad auf.

6. Die resultierenden Bewegungstrajektorien des Roboterarmes bzw. der entsprechenden Konfigurationsraumbewegung in der Planungsphase richten sich nach dem kürzesten Weg zwischen den Start- und Zielneuronen im Konfigurationsraum.

7. Bei einer anhaltenden Steuerung gegen das gleiche Hindernis, aufgrund einer falsch ausgeprägten sensomotorischen Karte, erfolgt eine um ca. zehn Sekunden verzögerte Anpassung dieser Karte, d. h. es wird mindestens eine der aktuellen Konfiguration nächstliegende Kante gelöscht. Erst dadurch ist das Verfahren

in der Lage den Roboterarm vom Hindernis weg- und erfolgreich zum Ziel hinzuführen.

8. Die Wahl des humanoiden A-Serie-Roboters führte dazu, dass das Toussaint-Modell hinsichtlich der Schwerkraft des Roboterarmes nachträglich angepasst werden musste.

9. Durch die Wahl der Experimentierplattform mussten folgende mit der Roboter-Hardware verbundene Probleme zeitaufwendig gelöst werden:

   - Erschwernisse im Umgang mit dem System aufgrund der kabelgebundenen Datenerfassung (Lösung: zeitweise manuelle Kabelführung und Behebung von Kabeldefekten)
   - Wiederholung zeitaufwendiger Messreihen aufgrund von Firmwarefehler der Motoren (Lösung: Beschaffung und Einspielung einer funktionierenden Firmware nach zeitraubender Fehlersuche)
   - Softwareseitige Synchronisation der Videosequenzen und zugehörigen Roboterbewegungen (Lösung: Programmierung von entsprechender Software)

15 Fazit und Ausblick

15.1 Fazit

Im Rahmen dieser Arbeit können folgende zentrale Aussagen getroffen werden:

1. Hinsichtlich der dimensionsreduzierenden Algorithmen:

   - Die in dieser Arbeit vorgestellten dimensionsreduzierenden Algorithmen berücksichtigen die Information jeder Dimension des Datensatzes und sind damit für die unüberwachte Datenverarbeitung von besonderem Interesse. Darüber hinaus zeigen die Ergebnisse, dass die Dimensionsreduktion mit dem Ziel der Datenvisualisierung für das Arbeiten mit hochdimensionalen Datensätzen für die menschliche Wahrnehmung und Verständlichkeit ein unverzichtbares Werkzeug ist.
   - Aufgrund ihrer geringen Komplexität und ihrer guten Ergebnisse eignen sich der lineare Algorithmus *Principal Component Analysis* (PCA) und der nichtlineare Algorithmus *Locally Linear Embedding* (LLE) für die zweidimensionale Datenvisualisierung der Bewegungsmuster des humanoiden A-Serie-Roboters am besten.

2. Hinsichtlich der neuronalen Gase:

   - Durch die rechentechnische Umsetzung der den Verfahren zugrunde liegenden Algorithmen sowie eines in dieser Arbeit erstellten visualisierenden Softwaresystems konnte der Prozess der Ausprägung von Netzwerken der neuronalen Gase im Eingabedatenraum und damit wesentliche Aspekte des Lernprozesses visualisiert werden.
   - Es bestätigte sich, dass sich die Netzwerke aller neuronalen Gase in Abhängigkeit von den konkreten Bedingungen, zwar durchaus in unterschiedlicher Zeit, aber letztlich immer mit annähernd gleichem Fehlerendwert im Eingabedatenraum ausbilden, d. h. von ähnlich guter Qualität sind.

3. Hinsichtlich des behandelten Toussaint-Modells:

   - Bei der Ausprägung einer sensomotorischen Karte durch das Toussaint-Modell werden die Referenzvektoren von Anfang an im Vergleich zu den Verfahren der anderen neuronalen Gase mit der höchsten Genauigkeit positioniert. Damit ist die Toussaint-Variante einer neuronalen Netzwerkstruktur gegenwärtig am besten für die Arbeit mit Eingabedaten unbekannter Anzahl geeignet.
   - Die vielversprechenden theoretischen Ergebnisse des Toussaint-Modells haben sich durch die praktischen Resultate, basierend auf den sensomotorischen Daten des Roboterhardwaresystems, bestätigt. Dieses sensomotorische Modell ermöglicht es, die erlernten Motoransteuerungen durch Bewegungen zu einem vorgegebenen Ziel sichtbar zu machen.

15.2 Ausblick

Ausgehend von dem Stand der im Rahmen dieser Arbeit erwirkten Ergebnisse scheint eine Weiterführung der begonnenen Arbeit für folgende Themen sinnvoll:

1. Bezüglich dimensionsreduzierender Algorithmen:

   Im Experiment zeigten die Transformationen durch den linearen PCA- und den nichtlinearen LLE-Algorithmus für bestimmte Bewegungssequenzen einen jeweils unterschiedlich hohen Informationsgehalt auf. Es wäre außerordentlich vorteilhaft für die Dimensionsreduktion, wenn man in der Lage wäre, Verfahren zu entwickeln, die es erlauben, hochdimensionale Eingabedatenräume zu zerschneiden, mit unterschiedlichen dimensionsreduzierenden Algorithmen zu bearbeiten und anschließend die einzelnen Transformationen wieder nahtlos aneinander anfügen zu können.

2. Bezüglich der Analyse von Lernprozessen:

   - Zur Analyse von Lernprozessen sollte das Toussaint-Modell in Experimenten weiter implementiert werden, da dieses Modell die erlernten Bewegungen eindrucksvoll demonstrieren kann. Ausgehend von den hier vorliegenden Ergebnissen, sollte das zweidimensionale Experiment um ein oder zwei Dimensionen erweitert werden. Dadurch könnte weiteres Wissen über das hochdimensionale Lernen mit Hilfe von sensomotorischen Karten erworben werden. Bei weiteren Arbeiten mit dem Modell müssen der Einfluss der Schwerkraft und der daraus resultierende Systemfehler berücksichtigt werden.
   - Eine weitere Möglichkeit der Verbesserung eines Lernsystems, sowohl hinsichtlich der Exploration des Konfigurationsraumes als auch der gezielten Bewegung im Arbeitsraum, ist die Implementierung einfacher oder komplexer Kollisionsbewältigungs- und Kollisionsvermeidungssysteme in das Lernsystem. Toussaint hat für die Kollisionsvermeidung eine Erweiterung seines Modells entwickelt, in der die Bewegungsrichtung des Roboters vom nächstliegenden Hindernis wegdreht. Genauer gesagt, wird für jede Richtung ϕ_i durch einen Abstandssensor (mit gemessenen Abstand d_i) eine Hemmung proportional zu $(1 - d_i)^3$ direkt in m_i aufsummiert [Tou06]. Diese Erweiterung sollte nach ihrer Implementierung hinsichtlich ihrer Eignung experimentell überprüft werden.
   - Ein weiterer Ansatzpunkt betrifft die Verkürzung der Explorations- oder Trainingsphase. Bei einer Ausweitung der Experimente mit dem Toussaint-Modell auf grosse, konkave Konfigurationsräume kann die verwendete zufällige lineare Exploration sehr viele Zeitschritte benötigen bis alle Teile des Raumes mit annähernd gleicher Häufigkeit abgefahren wurden. Um die Explorationszeit zu reduzieren, könnte ein inneres Verlangen (Bedürfnis) in das Toussaint-Modell eingebaut werden, um alle Referenzvektoren oder Kanten mit der gleichen Häufigkeit zu frequentieren. So könnte bei Überschreitung einer maximalen lokalen Häufigkeit die Explorationsrichtung des Roboters, in Richtung der seltener aufgesuchten Orte der bisher gelernten Karte, geändert werden.

Anhang

Die humanoide Plattform der Experimentalserien

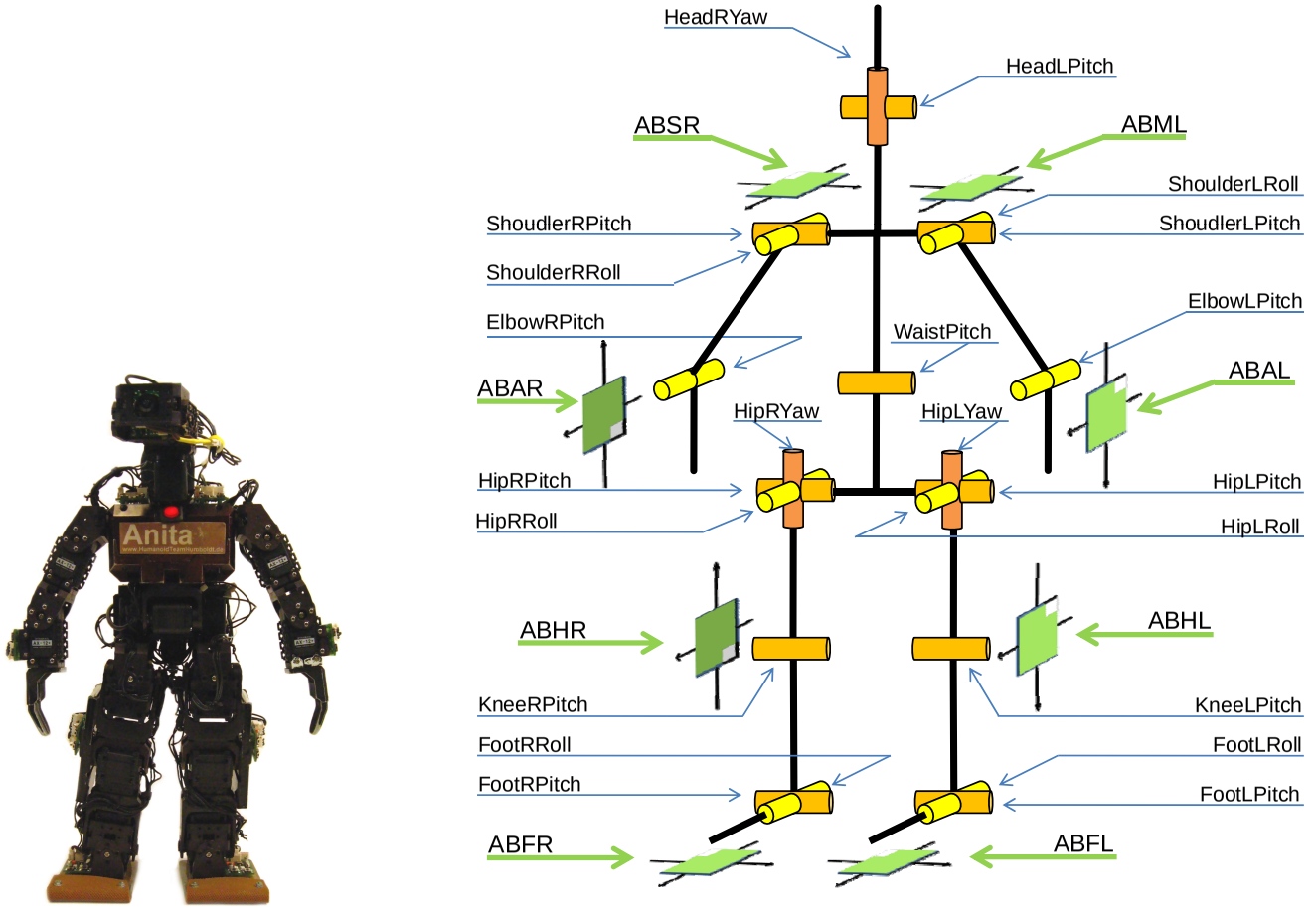
Der in dieser Arbeit verwendete humanoide Roboter stammt aus dem NRL des Instituts für Informatik der Humboldt-Universität zu Berlin. Im Folgenden werden die wesentlichen Hardware-Eigenschaften aufgelistet [Wer08]:

1. Der Roboter ist 47 cm hoch und wiegt 2.2 kg.
2. Der Roboter verfügt über 21 aktive Gelenke.
3. Die Gelenke sind Aktuatoren aus dem Roboterbaukasten Bioloid der Firma Robotis. Jeder Motor besitzt einen Winkelencoder mit einer Auflösung von 0.35°.
4. Auf dem Roboter sind acht zweiachsige Beschleunigungssensoren auf jeweils einer Platine, als *Accel Board* (AB) benannt, integriert.
5. Die Taktrate der Motorkommunikation beträgt 100 Hz.

In der Abbildung A.1(a) ist die frontale Ansicht des stehenden Roboters im schwarzen Design zu erkennen. Die Lage der ABs und der Gelenke ist in Abbildung A.1(b) dargestellt.

Für die Experimente dieser Arbeit wurden zum einen fertige Bewegungsmuster des Humanoiden abgespielt und die Sensorik zu jedem Zeitschritt ausgelesen. Diese Daten dienten dann für Untersuchungen der Dimensionsreduktionen und der neuronalen Gase-Netzwerke.

Zum anderen wurde die Hardware des Roboters im Echtzeitbetrieb durch die dynamischen Algorithmen des angepassten Toussaint-Modells einer sensomotorischen Karte gesteuert. Dabei konnte in jedem Motortakt ein vollständiger Sense-Think-Act-Zyklus des Verfahrens durchlaufen werden. Die Berechnung des Think-Schrittes wurde dafür in C++ implementiert und auf einen Desktop-PC ausgeführt.



(a) Roboter Frontalansicht

(b) Robotergelenke und -beschleunigssensorik

Abbildung A.1: *Auf Bild (a) ist die Frontalansicht und auf Bild (b) die Aktuatorik bzw. Sensorik des humanoiden Roboters zu sehen [Wer08].*

Literaturverzeichnis

[Ama77] Shun-Ichi Amari. Dynamics of pattern formation in lateral-inhibition type neural fields. *Biological Cybernetics*, 27:77–87, 1977.

[BN02] Mikhail Belkin and Partha Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering. In *Advances in Neural Information Processing Systems 14*, 2002.

[Bur07] Prof. Dr. Hans-Dieter Burkhard. Vorlesung "Kognitive Robotik: Bewegung". 2007.

[CC94] Trevor F. Cox and Michael A.A. Cox. *Multidimensional Scaling*. Chapman & Hall, 1994.

[CK07] Erhard Cramer and Udo Kamps. *Grundlagen der Wahrscheinlichkeitsrechnung und Statistik*. Springer-Verlag Berlin Heidelberg, 2007.

[Cul07] Paul Cull. The matimatical biophysics of nicolas rashevsky. *BioSystems*, 88:178–184, 2007.

[Dhi98] Inderjit Singh Dhillon. *A new O (N(2)) algorithm for the symmetric tridiagonal eigenvalue/eigenvector problem*. PhD thesis, University of California, Berkley, 1998.

[ES02] Wolfram Erlhagen and Gregor Schöner. Dynamic field theory of movement preparation. *Psychological Review*, 109:545–572, 2002.

[Fri95] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in Neural Information Processing Systems 7*, 1995.

[Fri97] Bernd Fritzke. A self-organizing network that can follow non-stationary distributions. In *ICANN*, 1997.

[Fri98] Bernd Fritzke. *Vektorbasierte Neuronale Netze*. PhD thesis, Friedrich-Alexander-Universität Erlangen-Nürnberg, Erlangen, 1998.

[Hil07] Manfred Hild. *Neurodynamische Module zur Bewegungssteuerung autonomer mobiler Roboter*. PhD thesis, Humboldt-Universität zu Berlin, 2007.

[Jol86] I. T. Jolliffe. *Principal component analysis*. Springer-Verlag, 1986.

[Koh01] Teuvo Kohonen. *Self-Organizing Maps*. Springer-Verlag, 2001.

[KOP02] Olga Kouropteva, Oleg Okun, and Matti Pietikäinen. Selection of the optimal parameter value for the locally linear embedding algorithm. In *FSKD*, 2002.

[Kru64a] Joseph B. Kruskal. Multidimensional scaling by optimizing goodness-of-fit to a non-metric hypothesis. *Psychometrika*, 29:1–27, 1964.

[Kru64b] Joseph B. Kruskal. Non-metric multidimensional scaling: a numerical method. *Psychometrika*, 29:115–129, 1964.

[Kur05] Thomas R. Kurfess. *Robotics and Automation Handbook*. CRC Press, 2005.

[LdVC95] Sofianto Li, Olivier de Vel, and Danny Coomans. Comparative Performance Analysis of Non-linear Dimensionality Reduction Methods. Technical report, James Cook University, 1995.

[Lee00] John M. Lee. *Introduction to Topological Manifolds*. Springer-Verlag New York, 2000.

[Mar93] Thomas Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. In *ICANN-93*, 1993.

[MP43] Warren. S. McCulloch and Walter H. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5:115–133, 1943.

[MS91] Thomas Martinetz and Klaus Schulten. A "neural-gas" network learns topologies. *Artificial Neural Networks*, I:397–402, 1991.

[MS94] T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.

[Olg06] Kayo Olga. *Locally Linear Embedding Algorithm Extensions and applications*. PhD thesis, 2006.

[Pea01] Karl Pearson. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, 2:559–572, 1901.

[PP01] Mariza Polito and Pietro Perona. Grouping and dimensionality reduction by locally linear embedding. In *Advances in Neural Information Processing Systems 14*, pages 1255–1262. MIT Press, 2001.

[Ros58] F. Rosenblatt. The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain. *Psychological Review*, 65:386–408, 1958.

[RS00] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290:2323 – 2326, 2000.

[Sha48] Claude Elwood Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423, 1948.

[Slo05] Douglas J. Slotta. *Evalutating Biological Data Using Rank Correlation Methods*. PhD thesis, 2005.

[Spe04] Charles Edward Spearman. The proof and measurement of association between two things. *The American journal of psychology*, 15:72–101, 1904.

[SR03] Lawrence K. Saul and Sam T. Roweis. Think globally, fit locally: unsupervised learning of low dimensional manifolds. *J. Mach. Learn. Res.*, 4:119–155, 2003.

[TdSL00] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290:2319–2323, 2000.

[Tou04] Marc Toussaint. Learning a world model and planning with a self-organizing dynamic neural system. In *NIPS 2003*, 2004.

[Tou06] Marc Toussaint. A sensorimotor map: Modulating lateral interactions for anticipation and planning. *Neural Computation*, 18:1132–1155, 2006.

[Wer08] Benjamin Werner. Sensomotorische Erzeugung eines Gangmusters für humanoide Roboter. Studienarbeit, Lehrstuhl für künstliche Intelligenz, November 2008.

Abbildungsverzeichnis

Abkürzungsverzeichnis

AB *Accel Board*

CS *Classical Scaling*

GNG *Growing Neural Gas*

GNG-U *Growing Neural Gas with Utility criterion*

ISOMAP *Isometric Feature Mapping*

LLE *Locally Linear Embedding*

MDS *Multidimensional Scaling*

NG *Neural Gas*

NRL *Nerorobotics Research Laboratory*

PC *Principal Component*

PCA *Principal Component Analysis*

PUMA *Programmable Universal Machine for Assembly*

RMS *Root Mean Square*

SOM *Self-Organizing Map*

Selbstständigkeits- und Einverständniserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmitteln angefertigt habe.

Ich gebe mein Einverständnis, dass die vorliegende Arbeit in der Bibliothek des Institutes für Informatik der Humboldt-Universität zu Berlin ausgestellt werden darf.

Berlin, den 8. Juni 2010

André Stephan