ulm university universität ulm

Universität Ulm | 89069 Ulm | Germany

Fakultät für Ingenieurwissenschaften und Informatik
Institut für Neuroinformatik

Lernverhalten von Robotern mit Neuronalen Netzen

Bachelorarbeit an der Universität Ulm

Vorgelegt von:
Simon Eckert
simon.eckert@uni-ulm.de

Vorgelegt am:
26. November 2009

Gutachter:
Prof. Dr. Günther Palm
Institutsleiter
Institut für Neuroinformatik

Betreuer:
Dr. Mohamed Oubbati
Wissenschaftlicher Mitarbeiter
Forschungsgruppe Neurobotics

Fassung 26. November 2009

Satz: PDF-LaTeX 2ε

Name: Simon Eckert Matrikelnummer: 602713

Erklärung

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

Ulm, den .....

Simon Eckert

Inhaltsverzeichnis

1 Einleitung

Künstliche Neuronale Netze bilden ein Teilgebiet der Neuroinformatik, die sich mit der biologischen Informationsverarbeitung befasst. Die Erforschung der Theorie und der Anwendbarkeit von künstlichen Neuronalen Netzen erlangt in sehr vielen Forschungsgebieten immer mehr an Wichtigkeit. Neben der Informatik sind dies die Psychologie, Philosophie, die Naturwissenschaften und die Medizin. Neuronale Netze sind vom biologischen Vorbild, dem menschlichen Gehirn, inspiriert. Vor allem in der Informatik, den Naturwissenschaften und Teilen der Medizin will man die Abstraktion des Nervenzellenverbandes nutzen um die hervorragende Eigenschaft, die Lernfähigkeit von Menschen, auf einen Computer zu adaptieren. Der Mensch kann ohne Vorkenntnisse nur durch Eindrücke von außen lernen. Damit kommt der Mensch zu viel besseren Problemlösungen als jeder Computer. Ein Computer kann nur genau definierte einfache Aufgaben in bekannter Umgebung lösen. Aber viele der aktuellen Problembereiche lassen sich nicht durch einen Algorithmus beschreiben. Diese Problembereiche hängen von vielen verschiedenen subtilen Faktoren ab, wie zum Beispiel das Vorausbestimmen von Börsenkursen. Bei solch einer Anwendung, die sehr komplexe Anforderungen an einen Computer stellt, ist eine gute Lern- und Adaptionsfähigkeit vonnöten. Ein Neuronales Netz muss nicht programmiert werden, es kann beispielsweise an Hand von Trainingsbeispielen oder durch Bestärkung lernen. Die Trainingsbeispiele beschreiben, wie ein Problem gelöst werden sollte. Beim bestärkten Lernen werden die Ergebnisse des Neuronalen Netzes bewertet und so ein Lernfortschritt herbeigeführt. Die Vorteile, die sich aus dem Lernvorgang ergeben, sind die Generalisierungs- und Assoziationsfähigkeit. Nach dem Training kann ein Neuronales Netz ähnliche Problemstellungen, die nicht trainiert wurden, mit Erlerntem assoziieren und nachvollziehbare Lösungen finden. Neuronale Netze können sich also an Veränderungen anpassen. Die Fehlertoleranz von Neuronalen Netzen gegenüber verrauschten Daten ist eine weitere Eigenschaft, die es sich lohnt auf Computer zu übertragen. In der Industrie kommen Neuronale Netze schon reichlich zum Einsatz. Bei Banken werden sie beim Lesen von Schecks und anderen Dokumenten und bei der Bewertung von Kreditanträgen benutzt. In der Medizin werden Neuronale Netze in vielen Bereichen eingesetzt, wie beispielsweise beim Design von Prothesen, bei der Analyse von Brustkrebszellen oder auch zur Aufwandsverminderung. In der Nachrichtentechnik werden sie eingesetzt zur Bilder- und Datenkompression oder Echtzeitübersetzung von Sprache. Speziell bei der Sprache wird die Klassifikation von Vokalen und die Text zu Sprache Synthese mit Hilfe von Neuronalen Netzen realisiert. Wie man sieht gibt es jede Menge Anwendungen, wobei die aufgeführten nur ein kleiner Teil von vielen sind. In der Robotik gibt es einige Möglichkeiten Neuronale Netze einzusetzen. Man kann Neuronale Netze für die Positionsregelung und Geschwindigkeitsregelung einsetzen, da Neuronale Netze gut bei nichtlinearen Problemen funktionieren. Weitere Einsatzmöglichkeiten sind Zielerfassung, Zielverfolgung,

das Vermeiden von Hindernissen, die Lokalisation in unbekanntem Gelände und viele mehr.
Das Thema der vorliegenden Arbeit ist das Lernverhalten von Multilayer Perceptron Netzwerken an einem autonomen mobilen Roboter zu untersuchen. Ein Multilayer Perceptron (MLP) Netzwerk ist eine Art von Neuronalen Netzen. Neuronale Netze ist ein Überbegriff für eine Vielzahl von Ideen und Ansätzen um einem Computer das Lernen zu ermöglichen. In dieser Arbeit habe ich ausschließlich Multilayer Perceptrons verwendet. Der Roboter, auf dem die MLPs angewandt werden sollten, sollte verschiedene Verhalten lernen um in unbekanntem Gelände navigieren zu können. Das wichtigste Verhalten ist die Hindernissvermeidung. Ein mobiler autonomer Roboter sollte Hindernisse frühzeitig erkennen und ihnen gegebenenfalls ausweichen können. Eine weitere Aufgabe war, dass der Roboter in der Lage ist ein vorher festgelegtes Ziel aufzufinden. Da hierfür Hindernissen auf geeigneter Weise ausgewichen werden muss, ist auch hier die Hindernissvermeidung von größter Wichtigkeit. Das letzte Verhalten sollte so aussehen, dass der Roboter bei Gefahr flüchtet. Gefahr gilt, wenn sich dem Roboter ein Objekt nähert. Mit flüchten ist gemeint, dass der Roboter in die entgegengesetzte Richtung zum sich nähernden Objekt fährt. Auch hier muss Hindernissen ausgewichen werden.
Die Arbeit ist in fünf Kapitel aufgeteilt. Nach der Einführung wird im zweiten Kapitel auf Grundlegendes zur Robotik eingegangen. Es wird ein kleiner Überblick geschaffen, was eine elektrische Maschine zu einem Roboter macht und wozu es das Gebiet der Robotik überhaupt gibt. Im dritten Kapitel werden die Grundlagen künstlicher Neuronaler Netze erörtert. Es wird auf den Aufbau und den mathematischen Hintergrund Neuronaler Netze eingegangen. Dabei wird beschrieben wie die Daten, die in ein Neuronales Netz eingegeben werden, in den Neuronen weiterverarbeitet werden und wie die Ausgabe zu Stande kommt. Es gibt einen kurzen Überblick über die Lernverfahren und im Speziellen wird der Backpropagation-Algorithmus erklärt. Das vierte Kapitel umfasst die Implementierung und beginnt mit der ausführlichen Beschreibung des e-puck Roboters. Der e-puck ist der in dieser Arbeit verwendete autonome mobile Roboter mit dem die Aufgaben bearbeitet wurden. Danach werden die verschiedenen Lösungsansätze vorgestellt und das Vorgehen zur Lösungsfindung beschrieben. Dieses Kapitel wird durch die Diskussion der Probleme bei der Lösungsfindung abgeschlossen. Das letzte Kapitel beinhaltet das Resümee der Arbeit und gibt Vorschläge für weiterführende Arbeiten.

2 Robotik

2.1 Was ist ein Roboter?

2.1.1 Definition

Ein Pionier der industriellen Robotik, Joseph Engelberger, sagte einst:

> *I can't define a robot, but I know one when I see one.*
>
> Ich kann einen Roboter nicht definieren, aber ich erkenne einen, wenn ich einen sehe.

Jeder Mensch hat intuitiv eine Vorstellung was ein Roboter ist und was ihn ausmacht. Diese Vorstellungen können stark von einander abweichen. Überraschenderweise trifft das nicht nur auf die Definitionen von jedermann zu, sondern auch auf die offiziellen Definitionen. Von Land zu Land und Organisation zu Organisation können diese sich stark unterscheiden. Der Begriff Roboter taucht zum ersten mal 1917 in einer Kurzgeschichte von Karel Capek auf, der im Jahre 1920 ein Theaterstück schrieb mit dem Titel RUR (Rosum's Universalroboter), welches große Bekanntheit erlangte und somit auch der Begriff Roboter. Der Name Roboter kommt von dem tschechischen Wort „robota" und bedeutet so viel wie Zwangsarbeit oder Plackerei. Als 1954 ein Patent mit dem Titel „programmierter Artikeltransport" von einem Ingenieur namens Joseph Engelberger angemeldet wurde, konnte man zum ersten mal von einem realen Roboter sprechen. Wie schon erwähnt, gibt es eine Reihe von Definitionen des Begriffes Roboter.

Begriffsbestimmung Roboter

> *Im allgemeinen Sprachgebrauch wird heute als Roboter jedes automatische System mit bestimmten sensorischen und adaptiven Eigenschaften zur Ausführung manipulatorischer und ortsveränderlicher Vorgänge verstanden.*[7]

Begriffsbestimmung der ISO (International Organization for Standardization)

> Nach DIN EN ISO 8373 ist ein Industrieroboter ein: automatisch gesteuerter, frei programmierbarer Mehrzweck-Manipulator, der in drei oder mehr Achsen programmierbar ist und zur Verwendung in der Automatisierungstechnik entweder an einem festen Ort oder beweglich angeordnet sein kann.

Wobei als Anmerkung gilt, dass der Roboter den Manipulator (einschließlich Aktoren) und die Steuerung (sowohl Software wie auch Hardware) enthalten muss.

Begriffsbestimmung JRA (Japan Robot Association)

> Der JRA nach ist ein intelligenter Roboter: die am höchsten entwickelte Roboterart, die verschiedene Sensoren besitzt um auf Veränderungen der Werkstücke oder der Umgebung, ihren Möglichkeiten nach, selbständig aber der Situation angepasst zu handeln.

2.2 Wozu Roboter?

Eine Frage, die sich immer wieder stellt ist: „Wozu gibt es eigentlich Roboter?“. Dies ist eigentlich einfach zu beantworten, denn Roboter können unter Anderem dazu dienen die menschlichen Fähigkeiten zu erweitern und menschliche Unzulänglichkeiten auszumerzen. Was im Hinblick darauf aber von größter Wichtigkeit ist, ist die Sicherheit des Menschen. Zu diesem Thema gibt es von Isaac Asimov, ein vor allem durch seine Science-Fiction Bücher berühmter Schriftsteller und Biochemiker, die drei Gesetze der Robotik.

1. Ein Roboter darf kein menschliches Wesen verletzen oder durch Untätigkeit zulassen, dass einem menschlichen Wesen geschadet wird.
2. Ein Roboter muss dem vom Mensch gegebenen Befehl gehorchen, es sei denn, der Befehl verstößt gegen Regel eins.
3. Ein Roboter muss seine Existenz beschützen, solange sein Handeln dadurch nicht im Widerspruch zu den vorigen beiden Regeln steht.

2.3 Klassifikation

Je nach Einsatzgebiet werden verschiedene Anforderungen an einen Roboter gestellt. Das Hauptmerkmal, nach dem man Roboter klassifizieren kann, ist die Mobilität. Deshalb kann man Roboter in zwei Hauptgruppen einteilen. Zum Einen in die Gruppe der stationären Roboter, zum Anderen in die Gruppe der mobilen Roboter.

Mobile Roboter Ein mobiler Roboter ist nicht standortgebunden, sondern kann sich mehr oder weniger frei in seiner Umgebung bewegen. Er kann seinen Standort selbständig ändern. Der Grad der Bewegungsfreiheit hängt stark von der Umgebung und von der Bauweise des Roboters ab. Die Hauptaufgabe eines mobilen Roboters ist die Navigation in seiner Umgebung. Dies soll in bekannter wie auch unbekannter Umgebung gelingen.

Stationäre Roboter Wie das Wort stationär (von lat.*statio*: Stillstehen) schon sagt, ist ein stationärer Roboter stillstehend oder auch standortgebunden. Das heißt der Roboter ist nicht in der Lage seinen Standort ohne Einwirken von außen zu verändern. Bei dieser Art von Roboter mit einem festen Bezugssystem geht es meistens darum kollisionsfreie Bewegungen der Arme oder Greifer zu realisieren, Objekte zu erkennen, sie zu verfolgen und deren Lage, Position und Entfernung im Bezug auf den Standort des Roboters zu bestimmen. Stationäre Roboter werden im Zusammenhang mit eindeutigen und sich wiederholenden Tätigkeiten eingesetzt. Dies trifft in der Regel auf die Fertigungs- und Produktionsindustrie zu. Einsatzgebiete sind z.B. Punkt- und Bahnschweißen, Montage, Handling, Palettieren, Beschichten und vieles mehr .

2.4 Bestandteile eines Roboters

Ein Roboter besteht aus Sensoren zur Wahrnehmung der Umwelt, Aktoren die für jegliche Art der Bewegung zuständig sind und der Steuereinheit, die die Inputs ausliest, auswertet und an Hand dieser die Aktoren ansteuert. Dies ist in Abbildung 2.1 verbildlicht, wobei in dieser Abbildung die Steuereinheit aus einem Neuronalen Netz (siehe Kapitel 3) besteht. Andere Bestandteile sind in der Regel ein Chassi und eine Stromversorgung, meist in Form von Batterien.

2.4.1 Sensoren

Ein Sensor, auch Messnehmer oder Umwandler, ist ein technisches Bauelement in einer Messkette, welches eine variable Eingangsgröße in eine zur Weiterverarbeitung geeignete Messgröße umwandelt. Sensoren nehmen physikalische und chemische Signale aus ihrer Umwelt auf, wie Druck, Temperatur, Licht, Magnetfelder oder chemische Zusammensetzungen in der Luft und wandeln sie meist in eindeutige elektrische Signale um. Dabei sollten Sensoren relativ einfach gehalten werden und in Echtzeit auswertbar sein. Eine weitere wichtige Eigenschaft für Sensoren ist die Robustheit, da schon geringe äußere Einflüsse, wie zum Beispiel ein leichter Stoß, die Messwerte verfälschen können. Sensoren können verschieden klassifiziert werden. Es gibt interne und externe Sensoren, wie auch aktive und passive Sensoren.

Interne Sensoren überwachen den Zustand eines Systems im Inneren. Sie können dazu dienen den Batteriezustand zu erfassen, die Temperatur wichtiger Bauelemente festzustellen oder die Odometrie des Roboters zu bestimmen.

Externe Sensoren sind Sensoren um die Umgebung wahrzunehmen. Eine Kamera oder ein Bodensensor ist zum Beispiel ein externer Sensor.

Aktive Sensoren sind Spannungserzeuger die nichtelektrische Signale in elektrische Signale umwandeln. Sie senden Energie aus und messen die Reaktion der Um-

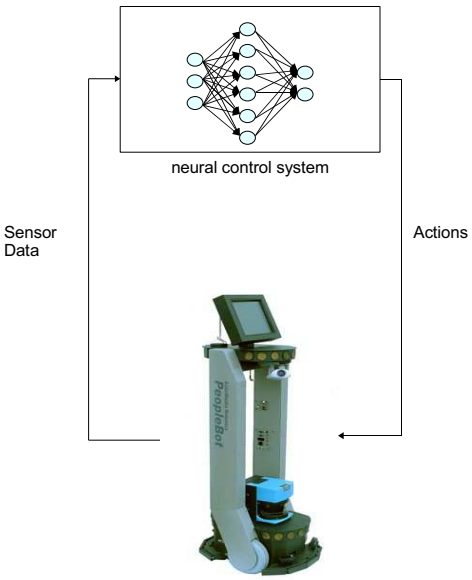


Abbildung 2.1: Ein Roboter, der die Sensordaten einliest und über ein Neuronales Netz die Aktoren ansteuert [12].

gebung. Somit wertet ein aktiver Sensor die Änderung der zu messenden Größe aus.

Passive Sensoren nutzen die Umgebungsenergie, sie wandeln die Erregungsenergie in elektrische Signale um. Hierbei kann aber in der Regel nur die Veränderung einer Messgröße festgestellt werden.

2.4.2 Abstandsensoren

In meinem praktischen Teil der Bachelorarbeit habe ich die Aufgabe der Hindernisvermeidung behandelt. Dazu eignen sich Abstandssensoren sehr gut. Außerdem sind Abstandssensoren auch bei der Lokalisierung und bei der Navigation sehr wichtig. Die Sensoren, die ich zur Abstandsmessung benutzt habe sind Infrarot Sensoren. In diesem Abschnitt möchte ich die Funktionsweise der Infrarot Sensoren, wie auch der anderen üblichen optischen Abstandssensoren erörtern. Abstandssensoren können den Abstand des Roboters zu einem Objekt bestimmen. In der Regel ohne das Objekt zu berühren. Ausnahmen sind Bumper (Berührungssensoren) und Whiskers (Fühler), die auf Berührung reagieren und so dem Roboter den Abstand mitteilen. Die genannten Sensoren

sind aber zur Kollisionsvermeidung nicht geeignet. Der Aufbau eines Abstandssensors ist trotz einer großen Vielfalt von optischen Sensoren und Messsystemen im Grunde wie folgt:

- Strahlungsquelle
- Beleuchtungs- und Abbildungsoptik
- Detektoreinheit
- Signal- und Datenauswerteeinheit

Es gibt auch noch induktive Abstandssensoren, die auf der Wechselwirkung von einem elektromagnetischen Feld einer Spule und einem metallenem Objekt beruhen. Sie sollen hier aber nur der Vollständigkeit halber erwähnt sein.

2.4.2.1 Licht-/Infrarotsensor

Lichtsensoren bestehen aus Leuchtdioden (LED - Light Emitting Diode), die das Licht im gewünschten Spektralbereich emittieren. Entweder im roten (sichtbaren) oder im infraroten (unsichtbaren) Spektralbereich. Der Lichtempfänger ist eine Fotodiode, die ein fotoelektrisches Signal erzeugt, wenn Licht einfällt. Dieses Signal wird dann in Spannung, Strom oder Frequenzen umgewandelt. Die Stärke des Signals hängt von der einfallenden Lichtstärke ab. Eine weiter Möglichkeit besteht darin, einen Fotowiderstand (LDR - Light Dependent Resistor) als Lichtempfänger zu nutzen. Ein Fotowiderstand ist ein lichtabhängiger Widerstand, der je nach einfallendem Licht seinen elektrischen Widerstand ändert. Infrarotsensoren arbeiten auf gleiche Weise wie Lichtsensoren nur, dass sie infrarotes Licht verarbeiten. Infrarotsensoren werden eher im Nahbereich eingesetzt.

2.4.2.2 Lasersensor

Die Lichtquelle von Lasersensoren sind Laserdioden. Mit Hilfe von Laserdioden können weit entfernte Objekte mit einer hohen Genauigkeit detektiert werden. Wegen dem gebündelten Lichtstrahl, der eine hohe Leuchtdichte aufweist, gilt es beim Einsatz von Lasersensoren jedoch entsprechende Schutzmaßnahmen zu beachten. Laserstrahlen können vor allem für Augen und Haut gefährlich sein. Auch bei den Lasersensoren ist der Lichtempfänger eine Fotodiode. Doch hier wird die Entfernung an Hand der Laufzeit eines Lichtpulses berechnet. Also der Zeit, die der Lichtpuls vom Sender zum Hindernis und wieder zurück zum Empfänger benötigt. Dabei wird die Proportionalität zwischen der gemessenen Zeit und dem Weg bis zum Hindernis ausgenutzt. Dieses Verfahren basiert auf dem TOF (Time of Flight) Prinzip.
Die Distanz zwischen einem Roboter und einem Hindernis ist gegeben durch

$$D = \frac{c * t}{2}$$

mit der Lichtgeschwindigkeit c und der Zeit t die der Lichtpuls vom Sender bis er wieder beim Empfänger ist benötigt.

2.4.2.3 Ultraschall

Die Entfernungsmessung bei Ultraschall basiert auf der Laufzeit des Schalls. Der Ultraschall wird durch dynamische oder elektrostatische Lautsprecher erzeugt. Dies gelingt am Besten mit Lautsprechern, die den piezoelektrischen Effekt verwenden. Um die Laufzeit des Schalls zu ermitteln wird ein Schallsignal ausgesendet und gewartet bis das Echo, welches von einem Objekt reflektiert wird, zurückkommt. Das Schallsignal liegt im für den Menschen nicht mehr hörbaren Bereich. Aus der Zeit, die der Schall gebraucht hat kann der Abstand ermittelt werden. Die Entfernung wird wie folgt berechnet:

$$D = \frac{c * t}{2}$$

mit der Schallgeschwindigkeit c ($c = 343$ m/s bei Raumtemperatur in Luft) und der Zeit t.
Bei der Entfernungsmessung mit Hilfe des Ultraschalls gibt es jedoch einige Probleme. Zum einen kann wegen der kegelförmig ausgestrahlten Schallwellen nur ein Hindernis detektiert werden, nicht aber der Winkel in dem das Hindernis zum Roboter steht. Zum anderen können sogenannte Scheinechos auftreten. Scheinechos können zum Beispiel auftreten, wenn ein Echo einer alten Messung während einer neuen Messung von einem sehr weit entfernten Objekt zurückkommt. In diesem Fall wird die Entfernung zu kurz eingeschätzt. Eine weitere Möglichkeit für Scheinechos ist, wenn mehrere Ultraschallsensoren im Einsatz sind können die von einem Sensor ausgestrahlten Schallwellen als Echos bei einem anderen Sensor empfangen werden.

2.4.3 Aktoren

Die Aktoren eines Roboters sind dazu da um physikalische Aktionen durchzuführen. Das heißt, Aktoren sind mechanische Bauteile zur Erzeugung von Bewegung. Bei Robotern gibt es eine Vielzahl von Möglichkeiten damit sie sich fortbewegen können. Meist kommt ein Getriebemotor zum Einsatz um mechanische Energie zu erzeugen. Somit benötigt ein Roboter einen Motor, ein Getriebe und Bauteile, die die Erzeugte Energie in Bewegung umsetzen. Solche Bauteile sind:

- Räder in unterschiedlicher Größe und Anzahl
- Beine in unterschiedlicher Größe und Anzahl
- Fahrketten
- Propeller für Unterwasserroboter oder Roboter in der Luft

Auch unkonventionelle Arten zur Fortbewegung sind denkbar. Es ist zum Beispiel auch möglich einen Roboter als Hovercraft zu entwickeln oder die Fortbewegung durch

den gesamten Körper des Roboters zu erreichen wie beispielsweise bei einem Reptil. Typischerweise werden Roboter durch Elektromotoren angetrieben.

2.5 Einsatzgebiete der Robotik

Robotik ist eine Wissenschaft, die die Technik, Entwicklung und Herstellung von Robotern umfasst. In diesen Bereich fallen viele Teilgebiete von anderen Wissenschaften wie zum Beispiel der Informatik (vor allem die künstliche Intelligenz), Elektrotechnik, Maschinenbau und immer mehr auch die Psychologie und Biologie. Die Einsatzgebiete der Robotik sind genauso vielseitig wie die Anzahl der vereinten Teilgebiete der anderen Wissenschaften. Heutzutage ist nicht nur die Industrie an Robotern und der daraus folgenden Möglichkeit Arbeitsschritte zu automatisieren interessiert, sondern auch andere Interessengruppen. In der Forschung bekommt die Robotik immer einen größeren Stellenwert und auch die Medizin zeigt immer größeres Interesse. Andere Einsatzgebiete von Robotern sind natürlich auch im Militär und im eigenen Heim zu sehen.

3 Künstliche Neuronale Netze

Ein großer Nachteil von Computern ist, dass sie nicht lernfähig sind, keine Fehlertoleranz haben und dass die Programme nicht generalisierungsfähig sind. Das menschliche Gehirn zeichnet sich gerade durch die genannten Fähigkeiten aus. Somit ist es naheliegend, dass man diese Eigenschaft auch auf einen Computer oder Roboter adaptieren will. Künstliche Neuronale Netze sind also vom biologischen Vorbild inspiriert und bestehen wie das Vorbild aus kleinen Einheiten (Neuronen), die durch Verbindungen zusammen ein Netzwerk ergeben. Lernfähigkeit bedeutet, dass das Neuronale Netz nur an Hand von Trainingsdaten lernt, wie die Inputs im Zusammenhang mit den Outputs stehen. Im Training wird dem Netz nur mit Hilfe von Beispieldaten nicht aber mit Informationen über den zu erlernenden Zusammenhang das gewünschte Verhalten erlernt. Das bringt uns zum nächsten Punkt, der Generalisierungsfähigkeit. Generalisierungsfähigkeit bedeutet, das Künstliche Neuronales Netze (KNN) verallgemeinerbar sein sollen. Das heißt ein Künstliches Neuronales Netz soll nach dem Training auch für neue, unbekannte Eingabedaten die richtigen Outputs erzeugen. Hiermit ist auch schon ein Schritt zur gewünschten Fehlertoleranz getan. Durch die Generalisierung können fehlerhafte oder verrauschte Eingabedaten plausiblen Ausgabedaten zugeordnet werden. KNN finden überall dort ihren Einsatz wo herkömmliche Methoden nicht funktionieren oder an ihre Grenzen stoßen. Es gibt Aufgabenstellungen, die nur gelöst werden können, wenn man viele variable Faktoren zur Lösungsfindung hinzuzieht und bei denen die Zusammenhänge unklar sind. Die herkömmlichen Algorithmen sind dafür nicht geeignet oder nur eingeschränkt einsetzbar. Auch bei Aufgabenfeldern bei denen nur mangelhaftes Wissen zur Problemlösung vorliegt, kann man herkömmliche Algorithmen ausschließen. Bei der Abbildung nichtlinearer Zusammenhänge sind Neuronale Netze am besten geeignet. Vor allem bei der Abbildung beliebiger, stetig nichtlinearer Funktionen stechen die Mehrschichtigen Neuronalen Netze (Abschnitt 3.4) hervor. Wegen der genannten Gründe sind KNN ein bewährtes Instrument in fast allen wissenschaftlichen Disziplinen. Bei allen Möglichkeiten, die einem ein Neuronales Netz beschert, gibt es auch Einschränkungen. Nachteile sind, dass man das erlernte Wissen von KNN nicht extrahieren kann, man nicht weiß was das Netz kann und wo eventuelle Fehler liegen. Eine weitere Einschränkung ist die schwierige Lernphase, denn hier gibt es eine Vielzahl von Parametern, die eingestellt werden müssen, die aber nicht von vornherein klar zu bestimmen sind. Diese Parameter sind in erster Linie die Anzahl von Neuronen, die Anzahl von Schichten, Verbindungen zu anderen Neuronen und einige mehr.

3.1 Das Neuron

Eine einzelne Nervenzelle wird auch Neuron genannt und ist die elementare funktionelle Einheit des zentralen Nervensystems. Ein künstliches Neuron (auch Perceptron) ist an das biologische Vorbild angelehnt und ist die kleinste Einheit in einem KNN. Das Neuron kann auch als einfache Recheneinheit eines neuronalen Netzes bezeichnet werden. Eine Verbildlichung eines Neurons ist in Abbildung 3.1 zu sehen. Die Neuronen

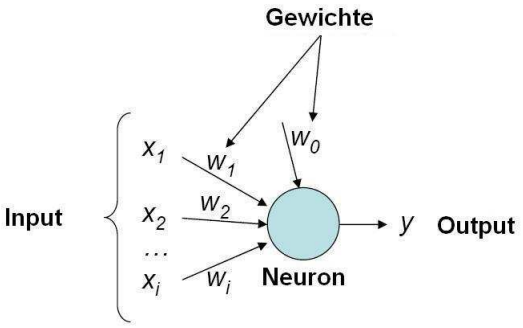


Abbildung 3.1: Ein Neuron [11]

sind durch Verbindungen untereinander vernetzt. Die Eingangssignale eines Neurons sind Ausgangssignale von anderen Neuronen oder die Eingangsdaten des Netzes. Aus diesen Daten berechnet das Neuron sein eigenes Ausgangssignal, das wiederum als Eingangssignal von anderen Neuronen oder als Output des Netzes dient. Ein Neuron kann beliebig viele Inputs verarbeiten, aber erzeugt nur ein Ausgangssignal. Die Verbindungen zwischen den Neuronen sind gerichtet und mit einem Faktor ω gewichtet (multipliziert). Gewichtungen können hemmend oder verstärkend wirken. Die gewichteten Inputs werden aufsummiert. Dieser Vorgang wird Propagierung genannt. Das Ergebnis ist die Netzeingabe *net*.

$$net = \sum_{i=1}^{n} x_i \omega_i = \omega^T x$$

mit dem Inputvektor x und Gewichtsvektor ω

$$x = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix}, \quad \omega = \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \vdots \\ \omega_n \end{pmatrix}$$

Von dem Ergebnis *net* wird dann der Schwellwert b des Neurons subtrahiert und dient der Aktivierungsfunktion φ (auch Transferfunktion) als Argument. Der Schwellwert eines künstlichen Neurons ist ein Zahlenwert um den die Aktivierungsfunktion besonders sensibel reagiert. Die Aktivierungsfunktion, ist in der Regel der tangens hyperbolicus (Kürzel: tansig, Gleichung (3.1)) oder eine logistisch sigmoide Funktion (Kürzel: logsig, Gleichung (3.2)).

$$\varphi(\mu) = \tanh(\mu) = \frac{e^{\mu} - e^{-\mu}}{e^{\mu} + e^{-\mu}} \tag{3.1}$$

$$\varphi(\mu) = \frac{1}{1 + e^{-\mu}} \tag{3.2}$$

mit

$$\mu = net - b \tag{3.3}$$

Es gibt auch noch weitere mögliche Funktionen wie etwa die Heaviside-Funktion (Gleichung (3.4)), die nur zwei Werte annehmen kann die am Schwellwert wechseln oder stochastische Aktivierungsfunkionen, die zufällig in Abhängigkeit von den Inputs erstellt werden.

$$\varphi(\mu) = \sigma(\mu) = \begin{cases} 0: & \mu \leq 0 \\ 1: & \mu > 0 \end{cases} \tag{3.4}$$

Nach dem die Aktivierungsfunktion berechnet wurde, wird das Ergebnis einer weiteren Funktion übergeben, der Ausgabefunktion. Die Ausgabefunktion dient der Berechnung des Wertes der an die folgenden Neuronen weitergegeben wird oder als Output ausgegeben wird. Sie kann zur Skalierung der Ausgabe dienen, ist aber oftmals die selbe Funktion wie die Aktivierungsfunktion und verändert den übergebenen Wert somit nicht. Der Output y ergibt sich dann durch

$$y = \varphi(\mu) \tag{3.5}$$

bei Gleichheit der Aktivierungs- und Ausgabefunktion.

3.2 Netztopologien

Für künstliche Neuronale Netze gibt es eine Reihe von verschiedenen Netztopologien. In diesem Abschnitt werde ich die gängigsten Topologien aufführen. Es gibt viele Möglichkeiten einzelne Neuronen miteinander zu verbinden. Üblicherweise werden Neuronen in Schichten zusammengefasst. Was alle Topologien gemeinsam haben ist die Eingabeschicht, die verdeckten Verarbeitungsschichten und die Ausgabeschicht. Unterschiede gibt es in den Arten wie die Neuronen miteinander verbunden sind. Eine wichtige Netztopologie ist das Feedforward-Netz. Bei einem Feedforward-Netz dürfen Neuronen nur mit Neuronen der nächsten Schicht in Richtung der Ausgabeschicht verbunden sein. Ist jedes Neuron einer Schicht mit allen Neuronen der folgenden

Schicht verknüpft spricht man von vollverknüpften oder auch vollständig verbundenen Schichten. Eine Variante der Feedforward-Netztopologie ist ein Feedforward-Netz mit Short-Cut-Connections. Bei dieser Variante sind auch Verbindungen von Neuronen zu Neuronen irgend einer nachfolgenden Schicht in Richtung der Ausgabeschicht möglich. Einer weiteren Bauart gehören die rückgekoppelten Netze an. Von einer Rückkopplung spricht man, wenn für ein Neuron die Möglichkeit besteht sich durch irgendeine Verbindung selbst zu beeinflussen. Es gibt die direkte Rückkopplung und die indirekte Rückkopplung. Bei der direkten Rückkopplung kann ein Neuron mit sich selbst verbunden sein, bei der indirekten Rückkopplung kann ein Neuron mit einem Neuron einer Schicht in Richtung der Eingabeschicht verbunden sein. Es gibt vollständig verbundene rückgekoppelte Netze und auch teilweise verbundene rückgekoppelte Netze. Eine Übersicht ist in Abbildung 3.2 zu sehen.

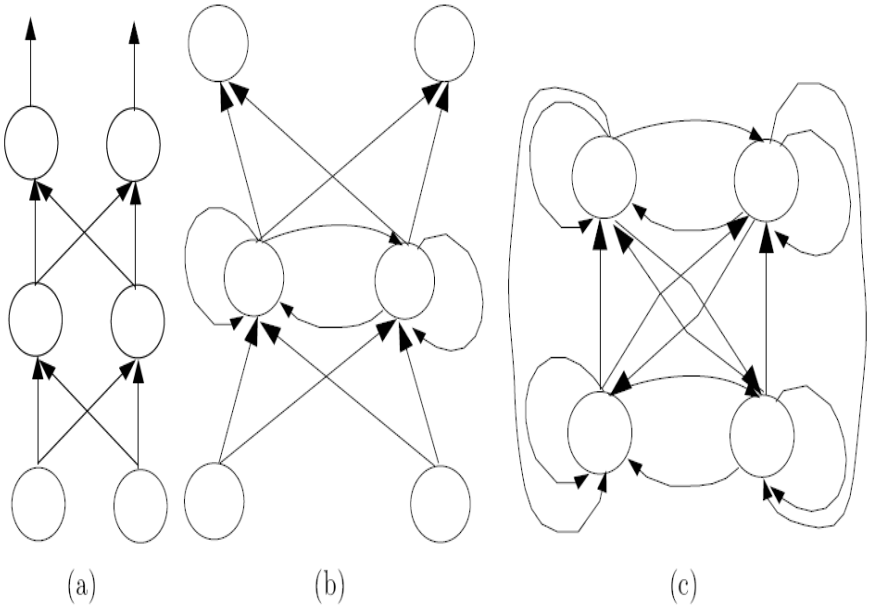


Abbildung 3.2: Drei Netztopologien a) Feedforward-Netz b) teilweise verbundenes rückgekoppeltes Netz c) vollständig verbundenes rückgekoppeltes Netz [11]

3.3 Lernverfahren

Da es verschiedene Arten des Lernens gibt, werde ich in diesem Abschnitt die möglichen Arten erörtern und dann im speziellen auf den Backpropagation-Lernalgorithmus eingehen. Zuerst steht die Überlegung an, wie man einem künstlichen Neuronalen Netz etwas beibringen kann. Im Allgemeinen gibt es die Möglichkeiten das Lernen zu simulieren, in dem man die Verbindungen modifiziert. Man kann sie löschen, neue bilden, die Gewichtung verstärken oder sie verringern. Eine weitere Möglichkeit ist die Modifizierung der Neuronen. Hier kann der Schwellwert, die Aktivierungsfunktion und die Ausgabefunktion verändert werden oder es werden neue Neuronen erzeugt oder bestehende Neuronen gelöscht. Üblicherweise wird die Gewichtsveränderung der Verbindungen und die Veränderung der Schwellwerte zum Lernen verwendet. Dies kann durch Lernalgorithmen realisiert werden, die durch konventionelles programmieren erstellt werden können. Es gibt drei wesentliche Arten des Lernens und zwei verschiedene Zeitpunkte und Arten der Gewichtsveränderung.

3.3.1 Überwachtes Lernen

Eine Art um künstlichen Neuronalen Netz etwas zu lernen ist das überwachte Lernen. Das überwachte Lernen ist zwar nicht so nah am biologischen Vorbild wie die anderen Verfahren, wird aber in der Praxis oft eingesetzt, weil es außergewöhnlich zielgerichtet ist. Das Vorgehen ist, dem Neuronalen Netz wird ein Trainingstupel[1] übergeben an Hand dessen die Ausgabewerte des Netzes zu jedem Eingabemuster mit den Soll-Ausgabewerten verglichen werden können. Die Differenz daraus liefert die Information, wie die Gewichte und Schwellwerte geändert werden müssen. Das Ziel liegt natürlich darin, dass das Netz generalisiert wird.

3.3.2 Unüberwachtes Lernen

Das unüberwachte Lernen ist dem biologischen Vorbild am nächsten, ist aber nicht bei allen Problemen einsetzbar. Dem Neuronalen Netz wird anders als beim überwachten Lernen nur das Eingabemuster übergeben. Das Netz sucht sich dabei Ähnlichkeiten und klassifiziert daran die Eingaben.

3.3.3 Bestärkendes Lernen

Bestärkendes Lernen ist eine andere Art des unüberwachten Lernens, da es dem Neuronalen Netz am Ende des Durchlaufes mitteilt ob das Ergebnis richtig oder falsch ist und in manchen Fällen wie falsch oder richtig es ist. Damit lassen sich Probleme um einiges besser lösen als mit dem unüberwachten Lernen.

[1] Ein Trainingstupel besteht aus Eingabemustern und den dazugehörigen Soll-Ausgabewerten

3.3.4 Zeitpunkt und Art der Gewichtsveränderung

Der Zeitpunkt an denen Gewichtsveränderungen vorgenommen werden ist entweder nachdem alle oder eine gewisse Anzahl von Eingabemuster das Netz durchlaufen haben. Die Fehler der Durchläufe werden zusammengezählt und dienen am Ende der Gewichtsveränderung. Diese Art wird mit offline- oder auch batch-Trainingsverfahren bezeichnet. Die andere Art wird als online-Trainingsverfahren bezeichnet und funktioniert wie folgt: Nach jedem einzelnen Eingabemuster werden die Gewichte angepasst.

3.4 Multilayer Perceptron

Ein mehrschichtiges Perceptron (kurz: MLP) ist ein Art der künstlichen Neuronalen Netze. Sie sind ohne Rückkopplung vernetzt und haben eine feste Eingabe- und Ausgabeschicht und beliebig viele Versteckte Schichten dazwischen (siehe Abbildung 3.3).Es gibt somit mehr als nur eine Schicht für die man die Gewichte und Schwellwerte ändern kann. Mit MLPs hat man erreicht, dass man Probleme lösen kann, die nicht linear separierbar sind. Meistens versteht man unter einem MLP ein FeedForward-Netz mit ShortCut-Connections (siehe hierzu Abschnitt 3.2).

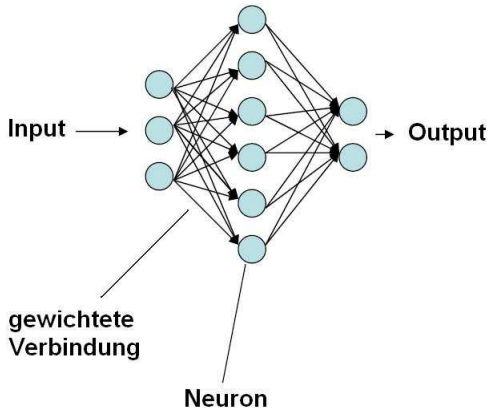


Abbildung 3.3: Beispiel eines MLP [11]

3.4.1 Backpropagation-Algorithmus

Um ein MLP zu trainieren gibt es mehrere Verfahren. Die am meisten verwendete Lernmethode für Neuronale Netze ist der Backpropagation-Algorithmus. Backpropagation, oder Fehlerrückführung wird der Gruppe der überwachten Lernverfahren zugeordnet.

Das Lernen kann nach dem online- wie auch nach dem batch-Lernverfahren geschehen. Aber in der Regel wird das online-Lernverfahren verwendet. Die Fehlerrückführung basiert auf einem Gradientenabstiegsverfahren (in [1] ausführlich beschrieben). Allgemein wird bei dem Backpropagation-Algorithmus der Output des Netzes für ein angelegtes Eingabemuster bestimmt und mit Hilfe der Fehlerfunktion (Gleichung 3.16) wird der Fehler des Netzes berechnet. In der Trainingsphase wird der Fehler des Outputs in Richtung der Eingabeschicht auf die Gewichte der Neuronenverbindungen und Schwellwerte aufgeteilt. Das Ziel ist einen möglichst kleinen Fehler am Output zu realisieren. Im Folgenden soll der Backpropagation-Algorithmus für ein MLP mit der Eingabeschicht i, einer versteckten Schicht j und der Ausgabeschicht k beschrieben sein.

Schritt 1: Initialisierung
Alle Gewichte ω und Schwellwerte b werden auf kleine zufällige Werte gesetzt.

Schritt 2: Berechnung der Netzausgabe y_k
Dazu wird erst der Netto-Input net der Neuronen in der Schicht j berechnet.

$$net_j = \sum_i \omega_{ij} x_i \tag{3.6}$$

Dabei ist ω_{ij} die Gewichtung zwischen Neuron i und Neuron j und x_i das Eingangssignal am Eingang i. Der Output der Neuronen in der Schicht j ist somit

$$y_j = \varphi(net_j - b_j). \tag{3.7}$$

Wobei φ die Aktivierungsfunktion ist. Der Netto-Input der Neuronen in der Schicht k ist

$$net_k = \sum_j \omega_{jk} y_j \tag{3.8}$$

und die Netzausgabe lautet

$$y_k = \varphi(net_k - b_k). \tag{3.9}$$

Schritt 3: Bestimmung des Fehlers am Netzausgang

$$\delta_k = (t_k - y_k)\varphi'(net_k - b_k) \tag{3.10}$$

Die Variable t_k ist hierbei die Soll-Ausgabe des Ausgabeneurons k.

Schritt 4: Fehlerrückführung

$$\delta_j = \varphi'(net_j - b_j) \sum_k \omega_{jk} \delta_k \tag{3.11}$$

Schritt 5: Lernen
Die Gewichte werden wie folgt berechnet:

$$\omega_{jk} = \omega_{jk} + \eta y_j \delta_k \tag{3.12}$$
$$\omega_{ij} = \omega_{ij} + \eta y_i \delta_j \tag{3.13}$$

Die Schwellwerte werden wie folgt berechnet:

$$b_k = b_k - \eta \delta_k \tag{3.14}$$
$$b_j = b_j - \eta \delta_j \tag{3.15}$$

Schritt 6: Ende Epoche
Solange noch Trainingsmuster vorhanden sind, wird bei Schritt 2 weitergemacht. Schritt

7: Trainingsende
Der quadratische Fehler wird durch

$$E = \frac{1}{2} \sum_k (t_k - y_k)^2 \tag{3.16}$$

berechnet. Solange E größer als ein zu Anfang bestimmter Fehler ist und die maximale Anzahl an Epochen noch nicht erreicht ist , wird wieder bei Schritt 2 weitergemacht. Mit Epoche ist ein Durchlauf von einem Trainingsmuster durch das MLP gemeint.

4 Implementierung

4.1 Der e-puck Roboter

Der e-puck ist ziemlich klein mit einem Durchmesser von 75mm, einer Höhe von 50mm (ohne Erweiterungen) und einem Gewicht von 200g (zu sehen in Abb: 4.1). Somit eignet er sich gut um mit ihm direkt vom Arbeitsplatz aus zu arbeiten, was die Effizienz beim experimentieren, lernen und entwickeln erhöht. Er hat einen große Anzahl von Funktionen. Er kann in vielen Bildungsbereichen eingesetzt werden, wie zum Beispiel in Signalverarbeitung, Regelung, verteilte Netze und natürlich in der Robotik als Ganzes. Das Benutzerinterface ist einfach und gut verständlich aufgebaut. Der Roboter ist billig und das e-puck Projekt ist open source sowohl die Hardware als auch die Software.

Abbildung 4.1: Der e-puck [9]

4.2 Bestandteile des e-puck

Der e-puck besteht aus 4 Plastikteilen, die den Körper bilden, ein Lichtstreuungsring aus Plastik und 2 Rädern. Im Inneren des Körpers ist der Motor angesiedelt, der die Räder direkt antreibt. Unter dem e-puck und zwischen den Rädern wird die Batterie eingesetzt. Auf dem Körper ist die meiste Elektronik angebracht und darauf wiederum der Lichterring und eine Standardplatine für Erweiterungen. Die mechanische Struktur des e-puck ist in Abbildung 4.2 zu sehen. Ein Überblick über die Elektronik ist in Abbildung 4.3 dargestellt.

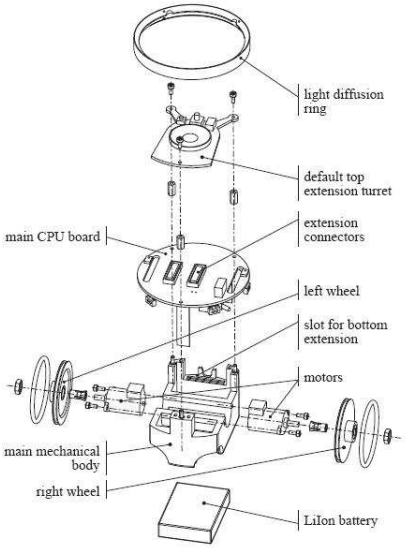


Abbildung 4.2: Die mechanische Struktur des e-puck in einer auseinandergezogenen Darstellung [9]

Mikrocontroller

Die Elektronik des e-puck ist um einen Mikrochip dsPIC Mikrocontroller aufgebaut. Der Mikrocontroller umfasst einen 16 bit Prozessor und einen digitalen Signalprozessor. Der Mikrocontroller hat 8kB RAM und 144kB Flash Speicher.

Sensoren und Aktoren

Der e-puck hat unterschiedliche Sensoren um verschiedene Gegebenheiten abdecken zu können (Audio, verschiedene visuelle Aspekte, Abstand zu Objekten). Die Aktoren sind zwei Räder, die von Schrittmotoren angetrieben werden.

IR- Sensoren

Der e-puck hat 8 Infrarotsensoren, die in der näheren Umgebung zu Abstandsmessungen eingesetzt werden können oder die Intensität des infraroten Lichts in der Umgebung messen können. Diese Art von Sensoren sind typisch für die Navigation in Umgebungen mit Hindernissen.

3D Beschleunigungsmesser

Der 3D Beschleunigungsmesser liefert den Beschleunigungsvektor des e-puck der genutzt werden kann um die Geschwindigkeit mit der sich der Roboter fortbewegt zu bestimmen wie auch die Neigung und Kollisionen.

Schrittmotor

Der e-puck wird von zwei Schrittmotoren mit einer Auflösung von 1000 Schritten pro Umdrehung angetrieben.

Mikrofone

Die Mikrofone sind verbaut um Geräusche zu erfassen. Da der e-puck 3 Mikrofone hat, kann er die Richtung aus der die Geräusche kommen herausfinden.

Kamera

Die CMOS Kamera hat eine Auflösung von 640x480 und kann farbige Bilder machen. Jedoch wird die Kamera vom Mikrocontroller eingegrenzt wegen dem geringen Speicher und der begrenzten Leistung. Der e-puck kann Farbbilder von 40x40 Pixel mit 4 Bilder pro Sekunde schießen. Bei Graustufenbildern verdoppelt sich dies.

Lautsprecher

Der Lautsprecher ist mit einem Audiodecoder verbunden und kann zusammen mit den Mikrofonen ein Kommunikationsnetzwerk bilden.

LEDs

Der e-puck ist mit einer Vielzahl von LEDs ausgestattet. Die LEDs dienen dem visuellen Zusammenspiel mit anderen e-pucks, der Entfernungsmessung bei Benutzung der Kamera und zur besseren Kommunikation mit dem Benutzer.

4.3 Schnittstellen

Es wurden verschiedene Schnittstellen in den Roboter integriert um eine möglichst vielseitige und einfache Benutzerinteraktion zu gewährleisten.

- Ein Infrarotempfänger für Infrarot-Fernbedienungen womit es dem Benutzer möglich ist den Roboter über solch eine Fernbedienung zu kontrollieren.

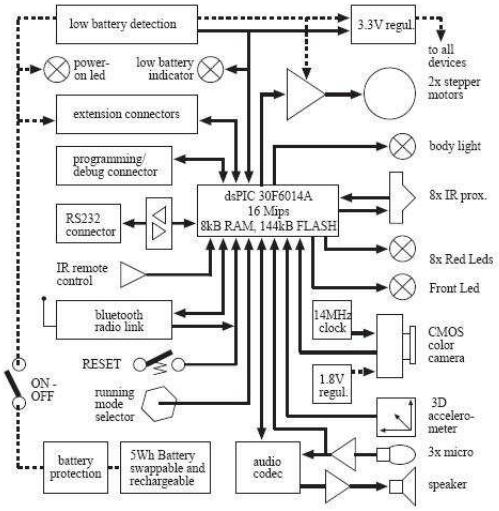


Abbildung 4.3: Ein Überblick über die Elektronik des e-puck [9].

- Ein Anschluss für ein in-circuit Debugger um den Flash- Speicher zu programmieren und zu debuggen.
- Eine Bluetoothschnittstelle um den Roboter vom PC zu kontrollieren oder um mit bis zu 7 weiteren e-pucks in Verbindung zu stehen.
- Ein Drehschalter mit 16 Positionen um mit Hilfe von einer 4 bit Nummer bestimmte Programme oder Parameter auszuwählen.
- Eine serielle RS232 Schnittstelle um mit einem PC zu interagieren.
- Ein Reset-Knopf

4.3.1 Erweiterungen

Um auf spezielle Gegebenheiten eingehen zu können, gibt es für den e-puck eine Reihe von Erweiterungen. Die Erweiterungen sind physischer Art und bieten die Möglichkeit die Sensoren und Aktoren auszubauen oder auch die Rechenleistung zu erhöhen. Angebracht werden diese mit Hilfe eines extra Buses. Es gibt drei verschiedene Arten von Erweiterungen. Eine Art sind die „top"-Erweiterungen, die keine anderen Erweiterungen über sich zulassen, wie zum Beispiel die Standardplatine mit ihrem Mikrofon, der seriellen RS232 Schnittstelle, dem Resetknopf, einem Switch und einem Infrarot Empfänger. Weitere Beispiele sind:

- ein rotierender Infrarot Triangulationsscanner, der eine Reichweite bis zu 40cm hat. Diese Erweiterung ist nützlich für die Lokalisierung des Roboters, da die Basissensoren nur eine geringe Reichweite haben.

- ein Turm mit drei linearen Kameras für optische Bewegungserkennung.

Die „bottom“-Erweiterungen sind Leiterplatten, die in einen Einschubschlitz im vorderen Teil des Gehäuses eingesteckt werden. Da diese Leiterplatten sehr nah am Boden liegen, können sie zum Beispiel durch drei Sensoren die Farbe des Bodens erkennen. Die letzte Art von Erweiterungen wird „sadwitch“ genannt und befindet sich zwischen dem e-puck und einer „bottom“-Erweiterung. Es ist auch möglich mehrere „sadwitch“-Erweiterungen an den Roboter anzubringen. Beispiele hierfür sind:

- ein weitere LED-Ring, der RGB Farben darstellen kann.
- eine ZigBee Funkverbindung mit einstellbarem Bereich der Funkverbindung.

4.4 Eingebettete Software des e-puck

Der e-puck ist mit üblicher, einfacher Software ausgestattet. Es gibt einen Bootloader, der es möglich macht den e-puck über Bluetooth zu programmieren. Die low-level Programmbibliothek steuert die Hardware und für die Kommunikation mit einem Desktop-Computer gibt es einen Monitor[1].

Bootloader

Die Besonderheit des Bootloaders vom e-puck ist, dass er es erlaubt den Flash des Mikrocontrollers immer wieder neu über Bluetooth oder einen seriellen Port zu programmieren. Beim booten hört der Bootloader diese Ports eine kurze Zeitspanne nach Aktivitäten ab, wenn keine Aktivitäten festgestellt wurden werden die Anwendungen des Benutzers geladen. Innerhalb dieser Zeitspanne kann der Benutzer seine Programme in den Flashspeicher schreiben.

Low-Level Programmbibliothek

Die Low-Level Bibliothek ist eine Sammlung von Funktionen um die Hardware des e-puck zu steuern. Ohne diese Bibliothek wäre die Bedienung erschwert, da hierfür ein spezieller Code und sehr exaktes Timing von Nöten ist. In der Bibliothek stehen Befehle mit der Bedeutung „bewege beide Motoren mit einer Geschwindigkeit von jeweils 100“ oder „alle LEDs sollen leuchten“.

BTcom Protokoll

Um das Arbeiten mit dem e-puck via Bluetooth oder einem seriellen Port von einem Desktop Computer aus zu gewährleisten, wurde ein Remot Controll Protokoll mit der

[1] Ein Monitor ist ein programmiersprachliches Konzept in der Informatik, welches Zugriffsüberschneidungen bei nebenläufigen Prozessen auf gleiche Datenstrukturen und Ressourcen synchronisiert.

Bezeichnung „(“BTcom Protokoll) implementiert. Dieses Protokoll bietet die gewünschte Funkfernsteuerung vollständig und bietet somit die Möglichkeit vom Desktop Computer aus den e-puck zu steuern und die Daten der Sensoren einzulesen.

4.5 Interaktion mit dem e-puck

Um mit dem e-puck zu arbeiten gibt es ein vollständiges Framework für Matlab. Will man den e-puck mit Basisbefehlen steuern, kann man auf eine sehr gute Benutzeroberfläche, die in Abbildung 4.4 zu sehen ist, zugreifen. Mit Hilfe der Oberfläche kann in Echtzeit in alle Sensordaten eingesehen werden, man kann eine gewünschte Geschwindigkeit des e-puck über zwei Eingabefelder oder einen Joystick setzen, Bilder mit der Kamera schießen, die LEDs bedienen und den gefahrenen Pfad aufzeichnen lassen. Ein zweiter Weg um das Framework zu nutzen, ist über die Kommandozeile von Matlab. Damit der e-puck gewünschte Verhalten ausführen kann ist eine Controller Funktion implementiert. Diese lässt sich über die Benutzeroberfläche ausführen. Der Controller verhält sich wie ein Matlab Skript womit alle Befehle um den e-puck zu kontrollieren in ihm verwendet werden können.

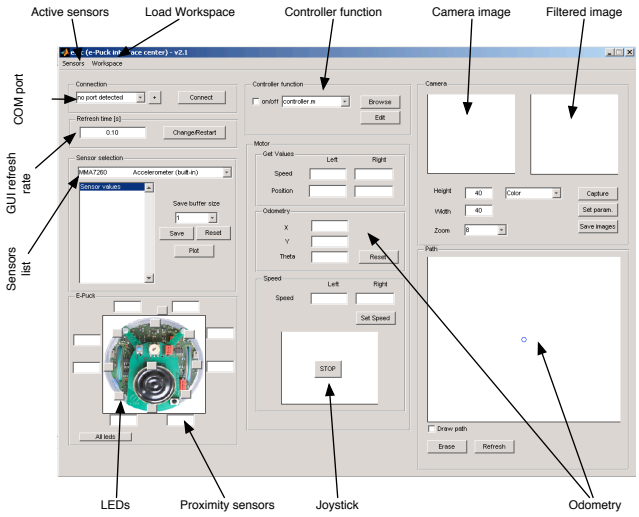


Abbildung 4.4: Benutzeroberfläche um mit dem e-puck zu interagieren [14]

4.6 Kinematik

Die Kinematik ist ein wichtiger Teil der Robotik und befasst sich mit der Bewegung eines Roboters im Bezug auf Zeit und Raum, ohne die wirkenden Kräfte zu beachten. Weil ich in dieser Arbeit mit dem e-puck gearbeitet habe, möchte ich kurz auf den Differentialantrieb eingehen. Dazu ist in Abbildung 4.5 ein kinematisches Modell des Differentialantriebs dargestellt. Beim Differenzialantrieb gilt für die Translationsgeschwindigkeit der einzelnen Räder:

$$v_l = (r \pm d)\omega \tag{4.1}$$

$$v_r = (r \mp d)\omega \tag{4.2}$$

r ist der Radius um das Drehzentrum des Roboters und d ist der Abstand vom Mittel-

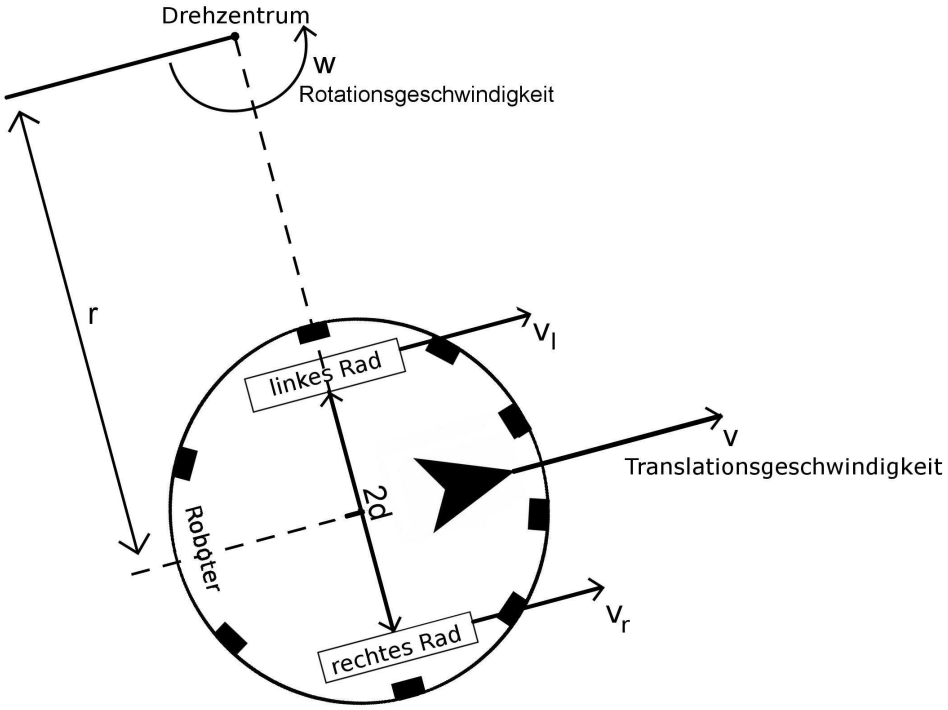


Abbildung 4.5: kinematisches Modell des Differentialantriebs

punkt des Roboter bis zum Rad. Um direkt mit der Translationsgeschwindigkeit und der Rotationsgeschwindigkeit des e-puck rechnen zu können, muss eine Transformation von (v_l, v_r) nach (v, ω) durchgeführt werden. Bei den folgenden Gleichungen wird von einem Drehzentrum links vom Roboter ausgegangen.
Transformation $(v_l, v_r) \rightarrow (v, \omega)$:

$$v_r - v_l = 2d\omega \rightarrow \omega = \frac{v_r - v_l}{2d} \tag{4.3}$$

$$v_r + v_l = 2\omega r \rightarrow r\omega = \frac{v_r + v_l}{2} \tag{4.4}$$

mit $v = r\omega$

$$v = \frac{v_r + v_l}{2} \tag{4.5}$$

$$r = d\frac{v_l + v_r}{v_r - v_l} \tag{4.6}$$

Des Weiteren gilt:

$$v_l = v_r \Rightarrow \omega = 0 \tag{4.7}$$

$$v_l = -v_r \Rightarrow v = 0 \tag{4.8}$$

Die Gleichungen 4.5 und 4.3 können zu einer Gleichung zusammengefasst werden:

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ -\frac{1}{d} & \frac{1}{d} \end{pmatrix} \begin{pmatrix} v_l \\ v_r \end{pmatrix} \tag{4.9}$$

Rücktransformation $(v, \omega) \rightarrow (v_l, v_r)$:

$$v_l = r\omega + d\omega \rightarrow v_l = v + d\omega \tag{4.10}$$

$$v_r = r\omega - d\omega \rightarrow v_r = v - d\omega \tag{4.11}$$

Die Gleichungen 4.10 und 4.11 können zu Gleichung

$$\begin{pmatrix} v_l \\ v_r \end{pmatrix} = \begin{pmatrix} 1 & -d \\ 1 & d \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix} \tag{4.12}$$

zusammengefasst werden. Liegt das Drehzentrum rechts, müssen entsprechende Vorzeichenwechsel durgeführt werden. In den Programmen die ich geschrieben habe, verwende ich Gleichung 4.7, um geradeaus zu fahren und Gleichung 4.8, um den Roboter auf der Stelle zu drehen.

4.7 Aufgabenstellung

Das Augenmerk dieser Bachelorarbeit lag darin, das Lernverhalten mit MLP Netzwerken näher zu untersuchen. Diese Untersuchung sollte an einem Roboter, dem e-puck, durchgeführt werden. Dem e-puck sollten bestimmte Verhaltensweisen mit Hilfe von MLP-Netzwerken beigebracht werden. Die gestellten Aufgaben waren:

- das Fluchtszenario (siehe Abschnitt 4.11)
- die Hindernissvermeidung (siehe Abschnitt 4.9)
- das Erreichen eines Ziels bei gleichzeitiger Hindernisvermeidung (siehe Abschnitt 4.10)

Alle Aufgaben sollten in unbekanntem Gelände realisiert werden.

4.8 Design des verwendeten MLP

Die MLPs, die ich trainiert habe, sind alles Feedforward-Netze mit Backpropagation. Die Trainingsfunktion der Netze basiert auf der Levenberg-Marquardt Optimierung [10] mit der Adaptionsfunktion nach dem Gradientenabstieg mit Momentum. Die Performanzfunktion ist die Sum squared error Performanzfunktion [3]. Die Transferfunktionen habe ich immer logarithmisch sigmoid gewählt und die Ausgabefunktion ist eine lineare Transferfunktion. Die Gewichte und Schwellwerte wurden vor Beginn des Trainings zufällig gewählt. Der Backpropagation-Lernalgorithmus wird im online-Modus betrieben. Also nach jedem dem MLP präsentierten Trainingsmuster wird der Fehler des Netzes berechnet und die Gewichte der Verbindungen angepasst.

4.9 Hindernisvermeidung

Das Wichtigste wozu ein mobiler Roboter in der Lage sein muss ist, dass er sich bewusst und ohne Einwirken von einem Mensch in bekannter wie auch unbekannter Umgebung bewegen kann. Die Herausforderung bei diesem Verhalten liegt darin mögliche Kollisionen des Roboters mit Hindernissen zu vermeiden. Das heißt Hindernisse müssen frühzeitig erkannt werden und es muss ihnen gegebenenfalls ausgewichen werden. Jedes Objekt, welches von dem verwendeten Roboter nicht bezwungen werden kann oder nicht befahrbare Bereiche stellen ein Hindernis dar.

4.9.1 Erster Lösungsansatz

Mein erster Lösungsansatz zur Hindernisvermeidung ist ein bisschen an den einfachen Bug-Algorithmus angelehnt. Es gibt verschiedene Variationen vom Bug-Algorithmus, aber im Groben sagt er aus:

Hindernisvermeidung: Algorithmus I

1. bewege dich auf direktem Weg zum Ziel bis ein Hindernis im Weg ist
2. bewege dich am Hindernis rechts oder links entlang bis man wieder in die Richtung des Ziels fahren kann
3. beginne wieder bei 1.

Für den Anfang wollte ich aber nicht erreichen, dass der Roboter sein Ziel findet. Deshalb ist dieser Aspekt auch noch nicht in meiner Überlegung enthalten. Der erste Schritt war, dass der Roboter immer nach links fahren soll bei erkanntem Hindernis in Front. Der Roboter behält dann diese Richtung solange bei bis ein erneutes Hindernis im Weg ist.
Um diesen Lösungsansatz umzusetzen, habe ich mir überlegt, dass man einem Neuronalen Netz die Werte der Infrarotsensoren als Inputs übergibt und das Netz die Geschwindigkeit der beiden Räder als Output ausgibt. Das Ziel war, dem e-puck nicht vorgeben zu müssen wie er fahren soll. Der e-puck soll eigenständig die Geschwindigkeiten der Räder so einstellen, dass eine Kurvenfahrt bei Hindernis in Front möglich ist. Bei diesem Lösungsansatz habe ich mich entschieden reale Daten als Trainingsdaten zu benutzen. Dazu wurde ein kleines Programm geschrieben, dass die Geschwindigkeiten der Räder und die Werte der Infrarotsensoren während einer realen Fahrt ständig speichert. Um ein Hindernis zu erkennen, ist es nötig die zwei Infrarotsensoren vorne am e-puck zu beachten. Durch auslesen der Sensordaten habe ich festgelegt, dass ein Hindernis ab einem bestimmten Wert, dem in etwa 2cm Abstand zum Hindernis entsprechen, erkannt wird. Das Programm arbeitet wie folgt:

Hindernisvermeidung: Algorithmus II

1. wenn die zwei Infrarotsensoren vorne unter dem Wert zur Hinderniserkennung liegen (jeder für sich), soll der e-puck mit konstanter Geschwindigkeit gerade aus fahren.
2. liegen die Sensorwerte knapp über dem Wert zur Hinderniserkennung wird die Geschwindigkeit gedrosselt
3. liegt der Werte einer der Sensoren über dem bestimmten Wert soll der e-puck anhalten
4. steht der e-puck still soll er auf der Stelle nach links drehen bis der Sensor vorne rechts kein Hindernis mehr erkennt.
5. weitermachen bei 1.

Für ein gut funktionierendes MLP reicht es das Programm für zwei bis drei unterschiedliche Szenarien (eins davon zu sehen in Abbildung 4.6) auszuführen und die Daten zu sammeln. Die Daten wurden dann mit einer Toolbox von Matlab für Neuronale Netze[2] auf verschiedenen MLPs trainiert. Zum Erlernen des gewünschten Verhaltens

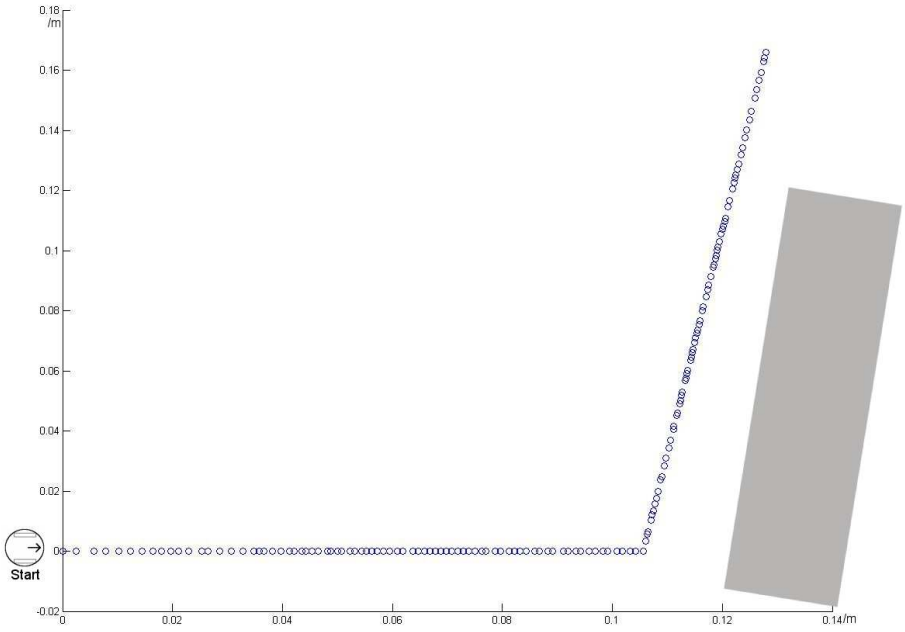
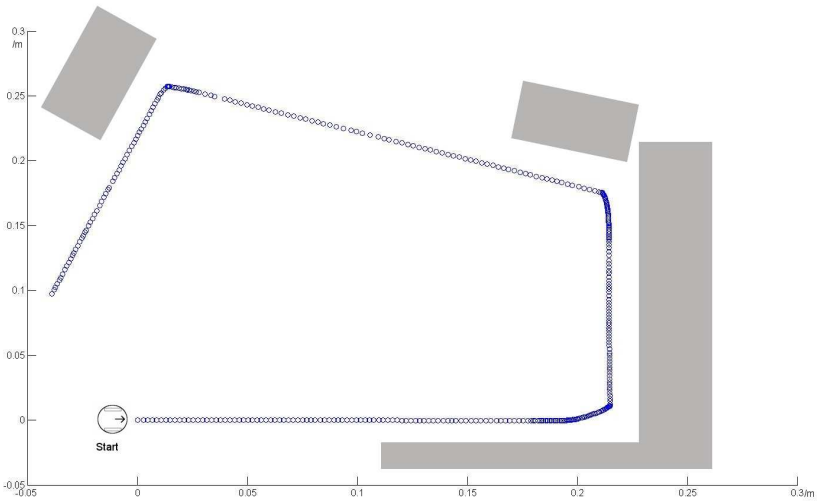


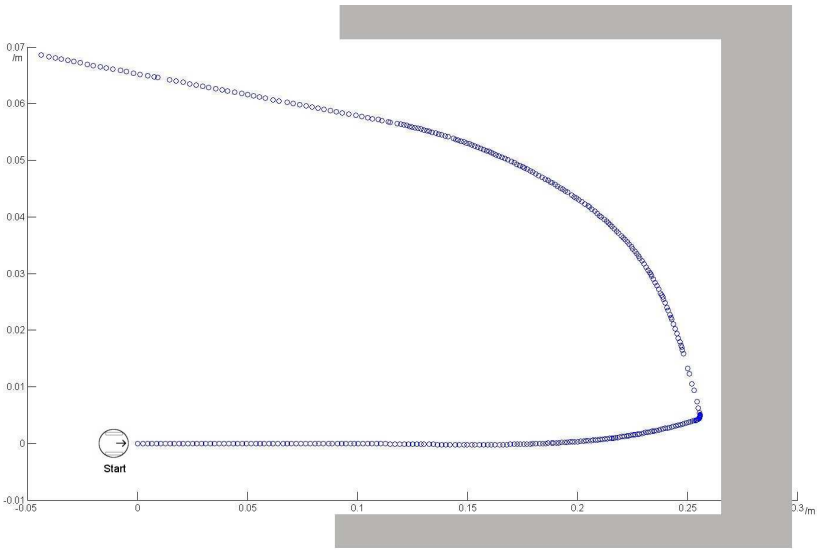
Abbildung 4.6: Zu sehen ist ein Lernszenario und die Fahrt des e-puck um die benötigten Trainingsdaten zu sammeln

diente ein Feedforward-Netz mit Backpropagation. Die Inputs bestehen aus den acht Sensordaten und die entsprechenden Zieloutputs sind die Geschwindigkeitsangaben der zwei Räder. Die Trainingsfunktion, die die Verbindungsgewichte und Schwellwerte aktualisiert basiert auf der Levenberg-Marquardt[10] Optimierung mit der Adaptionsfunktion nach dem Gradientenabstieg mit Momentum. Für die Performanzfunktion habe ich mich nach einigen Tests für die Sum squared error[3] (Abweichung der Quadratsumme) Performanzfunktion entschieden. Zur Architektur des MLP-Netzes probierte ich verschiedene Varianten aus. Am Ende entschied ich mich für zwei versteckte Schichten (Hidden Layers). Die erste Schicht besteht aus vier, die zweite aus zwei Neuronen. Die Transferfunktionen sind jeweils logarithmisch sigmoid (in Matlab: logsig) und die Ausgabefunktion ist eine lineare Transferfunktion (in Matlab: purelin). Beim Testen der MLPs mit verschiedener Neuronenanzahl der Schichten hat sich herausgestellt, dass die Netze mit wenigen Neuronen viel besser funktionieren als die mit großer Neuronenzahl. Beispielsweise ein Netz mit jeweils 20 und 10 Neuronen in den versteckten Schichten hat total unzufriedenstellende Outputs geliefert. Mein erstes MLP war ein [8 3 2 2] Netz (acht Inputs, erste versteckte Schicht drei Neuronen, zweite versteckte Schicht zwei Neuronen und zwei Outputs). Das Verhalten war aber

nicht optimal, da bei erkanntem Hindernis zwar die Geschwindigkeit gedrosselt wurde und dem Hindernis oftmals ausgewichen wurde, aber der Roboter auch sehr oft an den Kanten der Hindernisse hängen geblieben ist. Ein neues [8 7 3 2] MLP hat eigentlich ganz gute Ergebnisse geliefert, nur Hindernisse die im spitzen Winkel zur Front stehen werden nicht als Hindernis erkannt. Im Detail, wenn nur der linke vordere Sensor angesprochen ist, wird kein Hindernis erkannt. Mit anderen Architekturen sind Fehler, wie ungleiche Geschwindigkeiten der Räder bei freier Fahrt, zu weites Drehen nach links, wenn einem Hindernis ausgewichen werden soll oder endloses Drehen in engem Gelände aufgetreten. Eine Architektur mit nur einer versteckten Schicht ist auch möglich und funktioniert auch zufriedenstellend, aber wie erwähnt hat sich das [8 4 2 2] Netz für mich am besten bewährt. Das MLP-Netz generalisiert das Erlernte sehr gut. Bei leichten wie auch bei starken Veränderungen des Szenarios werden fast immer die richtigen Outputs geliefert. Das Verhalten vom e-puck ist in Abbildung 4.7 für zwei Szenarien zu sehen. Natürlich kommt es auch mal vor, dass sich der Roboter total falsch verhält, aber das hängt meist von den Sensoren ab, die nicht immer richtig funktionieren. Es hat sich aber herausgestellt, dass die Steuerung der Geschwindigkeiten an Hand der Sensordaten keine gute Lösung ist. Für schwierigere Aufgabenstellungen, wie zum Beispiel das Fluchtszenario, das Umfahren von Hindernissen oder das Auffinden eines Ziels mit Hindernisvermeidung ist dieser Lösungsansatz viel zu aufwendig und unklar, ob sich das Ergebnis dann gut generalisieren lässt. Da ich nicht davon ausgegangen bin, dass es sich lohnt diesen Lösungsansatz weiter zu verfolgen, habe ich mich den verschiedenen Ansätzen mit Verwendung von Sektoren gewidmet.



(a)



(b)

Abbildung 4.7: Zwei Bilder: a) Fahrt des e-puck in einem Gelände mit 3 Hindernissen
b) Fahrt des e-puck in eine Sackgasse

4.9.2 Sektoren-Strategie

Ein weiterer Lösungsansatz ist die Sektoren-Strategie. Die Idee liegt darin, die Umgebung des Roboters in mehrere Sektoren einzuteilen. Die Einteilung in Sektoren kann je nach Gebrauch und der Ausstattung an Sensoren des Roboters vorgenommen werden. Eine Einteilung in beispielsweise vorne, hinten, links und rechts, aber auch in kleinere Bereiche wie in Abbildung 4.8 zu sehen ist, wäre möglich. Um den Lösungsansatz

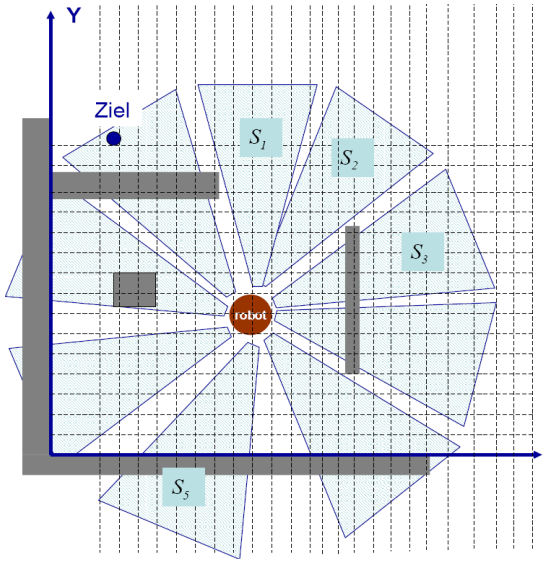


Abbildung 4.8: Mögliche Einteilung der Umgebung in Sektoren [11].

mit zur Hilfenahme von Sektoren anzuwenden, muss man sich erst einmal entscheiden, wie man die Umgebung des Roboters in Sektoren aufteilen will. Hierzu muss man die Ausstattung an Sensoren von dem verwendeten Roboter beachten und sich überlegen, wie man die vorhandenen Sensoren geschickt nutzt um Sektoren zu bilden. Die acht Sensoren des e-puck überschneiden sich mit ihren Blickfeldern. Somit kann man die Sektoren nicht einzelnen Sensoren zuordnen, sondern muss sie im Gesamten betrachten. Zur Einteilung der Sektoren habe ich mir zuerst überlegt vier Sektoren den Himmelsrichtungen entsprechend zu wählen (zu sehen in Abb: 4.9). Im Groben sollen jeweils zwei Sensoren einen Sektor ergeben. Wie aber schon erwähnt überlagern sich die Sensoren in deren Blickfeld und deshalb muss man beim Betrachten der einzelnen Sektoren immer auch die Sensoren beachten, die nicht dem betrachteten Sektor zugeordnet sind. Eine weiter Frage, die sich stellt ist: „Wie sollen die Trainingsdaten aussehen?“. Entweder man nimmt reale Werte aus Beispielszenarien als Trainingsdaten wie im ersten Lösungsansatz ohne Sektoren oder man erstellt die Trainingsdaten selbst. Auch eine Kombination aus selbst erstellten und realen Daten wäre möglich. Um den e-puck besser zum Ziel zu navigieren ist es womöglich besser die Sektoren-

anzahl zu erhöhen. Zu erst habe ich an acht gleichmäßig verteilte Sektoren gedacht aber diese Anzahl ist wahrscheinlich sehr aufwendig zu realisieren und da man die hinteren Sektoren sowieso nicht so dringend braucht, wären fünf gleichgroße Sektoren für Norden, Nordwest, Nordost, Osten ,Westen sowie einen Sektor für Hinten wohl die beste Aufteilung beim e-puck. Doch weil der Aufwand und die Schwierigkeit beim Erstellen der Trainingsdaten mit der Anzahl der Sektoren steigt, habe ich erst einmal eine Lösung mit vier Sektoren gesucht.

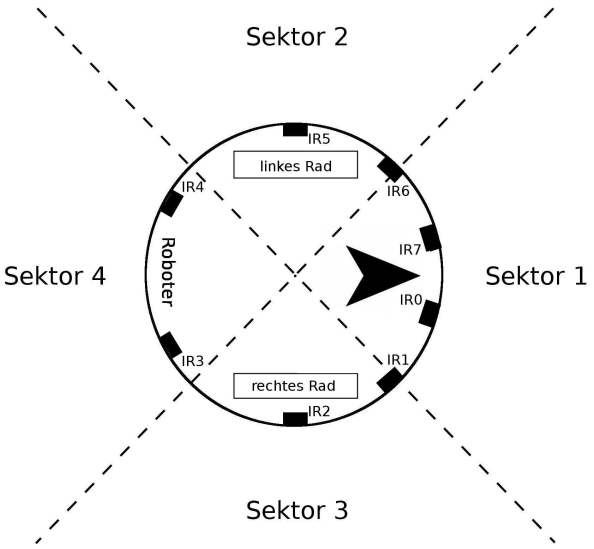


Abbildung 4.9: Aufteilung der Sektoren

4.9.2.1 Erste Variante

Zur reinen Hindernisvermeidung habe ich in den Anfangszeiten dieser Arbeit probiert ein einziges Neuronales Netz zu entwickeln, das eine direkte Sektorauswahl trifft in welche Richtung gefahren werden soll. Für den Versuch dies umzusetzen, teilte ich die Sektoren wie in Abbildung 4.9 zu sehen auf. Die grobe Zuordnung der Sensoren zu den Sektoren ist in Tabelle 4.1 dargestellt. Als Inputs in das Neuronale Netz sollen die Daten der Infrarotsensoren dienen. Die Outputs sollen den Sektor darstellen in

Infrarotsensor	IR0	IR1	IR2	IR3	IR4	IR5	IR6	IR7
Sektor	1	3	3	4	4	2	2	1

Tabelle 4.1: Zuteilung der Sensoren zu den Sektoren

dessen Richtung gefahren werden soll. Hierfür habe ich mich für zwei binäre Outputs entschieden, die Outputs sind also 0 oder 1. Die Auswahl der Sektoren soll dann wie in Tabelle 4.2 zu sehen ist definiert sein. Das trainierte MLP-Netz soll somit direkt

Output 1	**Output 2**	**Sektor**	**Auswahlkriterium**
0	0	1	Sektor 1 ist frei
0	1	2	Sektor 1 ist belegt und Sektor 2 ist frei
1	0	3	Sektor 1 und 2 belegt und Sektor 3 ist frei
1	1	4	Sektor 4 ist frei und alle anderen sind belegt

Tabelle 4.2: Auswahl des Sektors an Hand der Outputs des MLP

bestimmen in welche Richtung zu fahren ist, damit eine Kollision mit einem Hindernis vermieden wird. Die Trainingsdaten habe ich durch Zufallswerte und Extremwerte erstellt. Die Zufallszahlen liegen dabei in bestimmten Wertebereichen, die ich durch Tests der Sensoren festgelegt habe (siehe hierzu Tabelle 4.3). Also wenn die Sensoren

Sensorwerte	IR0	IR1	IR2	IR3	IR4	IR5	IR6	IR7
Sektor 1	< 300	-	-	-	-	-	-	< 300
Sektor 2	> 300	-	-	-	-	< 100	< 100	> 300
Sektor 3	> 300	< 100	< 100	-	-	> 100	> 100	> 300
Sektor 4	> 300	> 100	> 100	< 100	< 100	> 100	> 100	> 300

Tabelle 4.3: Sektoren gelten als frei wenn die Sensorwerten in bestimmten Wertebereichen liegen

0 und 7 kleinere Werte als 300 und die Sensoren 1 und 6 kleiner Werte als 800 liefern, dann wird Sektor 1 ausgewählt. Unter Extremwerten sind extreme Sensorwerte gemeint, wie zum Beispiel alle Sensorwerte sind im Bereich ihrer Minimas, alle Sensoren liefern Werte im Maximalbereich oder die Werte sind gerade die Grenzwerte, ab denen ein anderer Sektor gewählt wird. Für jeden Sektor habe ich $2300 * 8$ Sensorwerte, also insgesamt $9200 * 8$ Inputs generiert und dazu $9200 * 2$ Outputs für die entsprechende Sektorauswahl. Bei der Wahl der Architektur des MLP-Netzes habe ich wieder unterschiedliche Arten ausprobiert. Die besten Ergebnisse wurden mit einem Netz mit zwei versteckten Schichten erzielt, wobei bei nur einer Schicht oftmals sehr schlechte Ergebnisse herauskamen. Das vorerst Beste MLP war ein [8 5 3 2] Netz. Doch die Ausgaben waren nicht sonderlich zufriedenstellend. Bei anderen Architekturen traten teilweise total unbrauchbare Ausgaben auf. Die größten Ursachen für die nicht ausreichend funktionierenden MLPs sind meiner Meinung nach die Verwendung

von selbst erstellten Trainingsdaten und der Fakt, dass die Outputs gerundet werden müssen, damit Nullen oder Einsen herauskommen. Die Outputs werden wegen der linearen Ausgabefunktion nicht als 0 oder 1 ausgegeben, sondern liegen nach dem Training in einem Bereich um Null bis knapp über Eins. Bei manchen MLPs beginnt dieser Bereich bei knapp wenige als minus Eins. Somit kann nach dem Runden bei einigen MLPs in bestimmten Situationen sogar eine Ausgabe auftreten, die eine -1 enthält. Die Fehleranfälligkeit von den MLPs ist durch das Runden viel höher, denn nicht nur die normalen Abweichungen eines Neuronalen Netzes kommen zu tragen, sondern die Rundungsfehler verschlechtern die Ausgabe noch zusätzlich. Die Verwendung einer Heaviside-Funktion, mit der eine Ausgabe von 0 oder 1 möglich wäre, ist leider bei Verwendung des Backpropagation-Algorithmus nicht möglich. Die Trainingsdaten tragen möglicherweise zu den schlechten Ergebnisse bei, da sie fehlerhaft durch Denkfehler beim Erstellen der Daten sein können oder die Abgrenzung der Daten für die Sektorauswahl nach Tabelle 4.3 ist zu ungenau. Es ist auch möglich, dass durch die Zufallszahlen Kombinationen der Sensorwerte zu Stande kommen, die niemals real vorkommen würden. Nachdem ich einen anderen Lösungsansatz ausprobiert hatte und ich die für mich und die Problemstellung wohl beste Architektur des MLP gefunden hatte, habe ich noch einmal probiert diesen Lösungsansatz zu realisieren. Nach dem ich die Trainingsdaten erneut erstellt hatte, hat ein [8 10 6 2] MLP bessere Outputs geliefert als die anderen Architekturen, aber sie sind immer noch nicht befriedigend. Zur Hindernisvermeidung hatte ich eigentlich ein zweites Neuronales Netz geplant. Dem zweiten Netz sollten als Inputs die Outputs vom ersten Netz (also dem ausgewählten Sektor) und die Angabe des Ziels übergeben werden. Der Output sollte der zu befahrende Sektor sein, der frei ist und mit dem man auf schnellstem Wege an das Ziel kommt. Aber wegen den genannten Problemen der trainierten MLPs habe ich mich damit nicht weiter beschäftigt. Außerdem ist es wohl kaum zu realisieren, dass der zu befahrende Sektor bestimmt werden kann, wenn er schon im ersten Netz ohne Angabe des Ziels bestimmt wurde. Besser ist, wenn man die Sektorauswahl nicht schon im ersten Neuronalen Netz ausführt, sondern nur die Belegung der Sektoren ausgibt. Mit dieser Überlegung arbeitete ich dann weiter.

4.9.2.2 Zweite Variante

Wegen der unbefriedigenden Ergebnisse der ersten Variante mit Sektoren die Hindernisvermeidung zu realisieren, habe ich eine andere gewählt. Das Abtrennen der Entscheidungsfindung in welche Richtung gefahren wird von dem Herausfinden ob die Sektoren von einem Hindernis belegt oder aber frei sind, führt direkt zum nächsten Lösungsansatz. Um die Belegung der Sektoren zu ermitteln, kann man weiterhin ein Neuronales Netz nutzen. Wenn man weiß welche Sektoren frei und welche belegt sind, muss man nur noch entscheiden in welche Richtung sich der Roboter bewegen soll. Zur Auswahl des Sektors kann man einen Entscheidungsalgorithmus schreiben, der nach einer gewissen Regel vorgeht. Eine Regel könnte etwa so aussehen:

Sektoren-Strategie: Algorithmus zweite Variante

1. ist der Sektor nach vorne frei, dann fahre gerade aus
2. ist ein Hindernis in Front, also der Sektor nach vorne belegt, dann entscheide
   a) ist der linke Sektor frei, fahre nach links
   b) ist der linke Sektor belegt, fahre nach rechts
   c) ist der linke und rechte Sektor belegt, fahre nach hinten
3. weiter bei 1.

Zur Umsetzung der zweiten Variante gibt es wieder vier Sektoren, die gleich gewählt wurden wie bei der ersten Variante (Abb: 4.9 auf Seite 33). Die Trainingsdaten sind im Vergleich zu denen der ersten Variante real. Das trainierte MLP soll ausgeben, ob ein Sektor belegt ist oder nicht und nicht schon die Entscheidung vorwegnehmen in welche Richtung sich der Roboter bewegen soll. Ein mögliches MLP ist in Abbildung 4.10 verbildlicht. Die Inputs des MLP Netzes sind die Werte der acht Infrarotsensoren

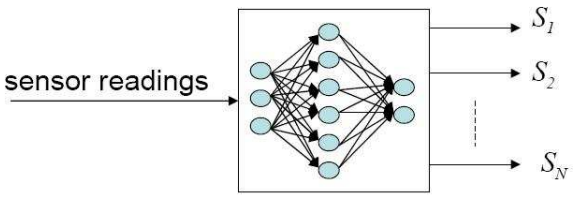
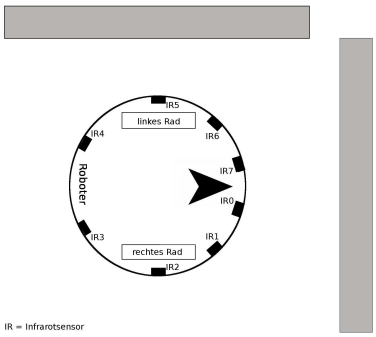


Abbildung 4.10: Ein MLP mit Sensordaten als Inputs und den Outputs S_1 bis S_N, die die Belegung der Sektoren 1 bis N ausgeben [11].

und es gibt vier Outputs. Jeder Output repräsentiert einen Sektor und gibt an, ob der Sektor belegt ist oder ob er frei ist. Die Outputs sollten im Wertebereich zwischen null und eins liegen, null für einen freien Sektor und eins für einen belegten Sektor. Die Werte zwischen null und eins repräsentieren dann wie stark der jeweilige Sektor belegt ist. Wird ein Hindernis erkannt, aber es befindet sich noch in weiter Entfernung zum Roboter, ist der Wert des Outputs gering, zum Beispiel $0,2$. Ist das Hindernis nicht mehr weit entfernt wird beispielsweise $0,6$ ausgegeben und bei einem Hindernis in unmittelbarer Nähe dann die 1 oder $0,9$. Da das Lernen mit realen Trainingsdaten beim ersten Lösungsansatz ganz gut funktioniert hat und das Lernen mit Zufallswerten bei der ersten Variante mit Sektoren eher nicht, habe ich mich entschieden dieses mal wieder reale Sensorwerte als Inputs zu benutzen. Nun hat sich die Frage gestellt, wie diese Sensorwerte wohl am besten aufgenommen werden. Im ersten Versuch habe ich wie beim ersten Lösungsansatz ein Programm in Matlab geschrieben, das mir die Werte der Infrarotsensoren liefert und die entsprechenden Outputs speichert. Das Programm

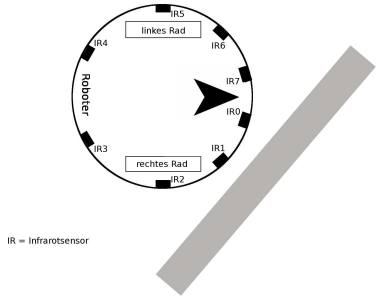
ist so aufgebaut, dass es ab bestimmten Sensorwerten die Outputs für den entsprechenden Sektor auf belegt setzt oder eben auf frei. Die Zuordnung der Sensoren zu den Sektoren ist wie bei der ersten Variante (siehe Tabelle 4.1). Doch die Entscheidung, ob die Sektoren frei oder belegt sind, wird anders gehandhabt, da ja nur die Belegung ausgegeben werden soll. Die Belegung des ersten Sektors wird an Hand der Sensoren 0 und 7 bestimmt, die des dritten Sektors an Hand der Sensoren 1 und 2 und so weiter. Um die Daten zu sammeln, wurden verschiedene Szenarien aufgebaut, die dem e-puck präsentiert wurden (ein Trainingsaufbau ist in Abb: 4.11(a) zu sehen). Einige Szenarien hat der e-puck abgefahren, andere wurden bei stillstehendem Roboter aufgenommen. Bei stillstehendem Roboter habe ich die Hindernisse während der Aufnahme der Sensorwerte leicht verschoben um verschiedene Werte zu bekommen. Die Szenarien wurden immer nach den verschiedenen Möglichkeiten der Sektorbelegung aufgebaut. Also jeder Sektor einzeln belegt und alle anderen frei und sämtliche Kombinationen die vorkommen können. Es hat sich aber herausgestellt, dass die gewonnenen Trainingsdaten nicht sonderlich gut sind. Wie in Abbildung 4.11(d)) zu sehen ist, sind die Outputwerte zu dem Trainingsaufbau in Abbildung 4.11(c) falsch. Rechts wird zwar ein Hinderniss mit einem Belegungsgrad von ca. 0.25 ausgegeben aber das ist zu wenig für ein so nahe stehendes Hindernis. Sektor 1 wird als frei angezeigt obwohl er belegt sein müsste und Sektor 4 gilt als belegt obwohl er frei ist. Die schlechten Trainingsdaten kommen wahrscheinlich daher, dass die Sensoren den entsprechenden Sektoren nicht genau zugeordnet werden können, da sie sich in ihrem Blickfeld überschneiden. Außerdem sind die festgelegten Grenzwerte für belegt oder nicht belegt wahrscheinlich ungenau und ungenügend, weil sie sehr schwer zu bestimmen sind. Deshalb habe ich einen viel einfacheren Weg gewählt. Man kann die Sensorwerte in den verschiedenen Szenarien aufnehmen und die Belegung der Sektoren einfach vorgeben. Dazu ist auch kein Programm mit mehreren if-Anweisungen mehr nötig. Es reicht Szenarien aufzubauen, diese dem e-puck zu präsentieren, die Sensorwerte aufzunehmen und dann den Output zu jedem Sektor zu bestimmen. Das Sammeln der Sensorwerte läuft dabei gleich, wie schon beschrieben, ab. Die Architektur der zu erstellenden MLPs war bei dieser Variante zehn Neuronen in der ersten und sechs Neuronen in der zweiten versteckten Schicht. Mit dieser [8 10 6 4] Architektur habe ich, zu Beginn der Testphase der erstellten Netze, zufriedenstellende Ergebnisse erhalten und habe deshalb daran festgehalten. Doch bis ich das finale MLP-Netz erstellen konnte, war ein langer Weg zu beschreiten. Dem ersten Netz wurden nur Daten aus einfachen Szenarien zum Training übergeben. Unter einfachen Szenarien ist gemeint, die Sektoren werden vollständig von Hindernissen ausgefüllt. Zum Beispiel Sektor 1 und 2 sind beide vollständig belegt oder ähnliches. Ein komplexeres Szenario wäre, wenn ein Hindernis seitlich schräg vor dem Roboter steht und dadurch die Sektoren nur teilweise belegt sind, aber als vollständig belegt gelten müssen, weil das befahren nicht möglich ist. Zum Testen der Netze habe ich mir einfach die Outputs ausgeben lassen und diese in bekannten wie auch neuen Situationen überprüft. Solange die Testszenarien einfach gehalten wurden, hat das trainierte Netz gute Outputs geliefert. Aber von einer perfekten Ausgabe konnte noch nicht die Rede sein. Wenn nur ein Sensor angesprochen wurde, kamen total falsche Outputs zu Stande. Auch bei Hindernissen, die schräg angefahren wurden, kam Unsinn heraus. Das MLP hat dabei nicht nur den vorderen, sondern auch den hinteren

Sektor als belegt ausgegeben. Auch Hindernisse schräg hinter dem e-puck wurden nicht richtig erkannt. Der Sektor nach hinten war sowieso ein Problem, da die Sensoren hinten nicht gerade nach hinten stehen, sondern schräg zur Seite. Somit wurde Sektor 4 immer als belegt ausgegeben wenn ein Hindernis seitlich nah genug am Roboter stand.



(a) Trainingsaufbau

(b) Outputs des trainierten MLP



(c) Testaufbau

(d) Outputs des trainierten MLP

Abbildung 4.11: a) Ein möglicher Trainigsaufbau für Sektor 1 und Sektor 2 belegt. b) Outputs (o = Sektor 1 bis 4 von links nach rechts gelesen) des trainierten MLP zu einem Hindernissaufbau ähnlich wie in a) trainiert. c) Testaufbau um zu sehen ob das MLP diese Situation richtig erkennt. d) Falsche Outputs (o = Sektor 1 bis 4 von links nach rechts gelesen) des trainierten MLP zu dem Testaufbau c).

Eine weitere Fehlausgabe ergab sich bei der Fahrt auf eine zu schmale Gasse oder im Allgemeinen bei der Fahrt auf eine Gasse zu. Aber die Trainingsdaten waren ja auch noch nicht sonderlich umfangreich. Um die Fehlausgabe bei einem zum Roboter schräg stehenden Hindernis in Front auszumerzen, wurde das nächste Netz mit neuen Daten aus solchen Situationen erweitert. Diese Maßnahme hat gut geholfen. Auch Probleme, die nicht extra trainiert worden sind, wie das Problem mit einem Hindernis schräg hinten, wurden jetzt richtig gelöst. Ein zu schmaler Durchgang wurde mit dem neuen Netz als zu schmal erkannt und der Sektor wurde als belegt ausgegeben. Aber der hintere Sektor hat immer noch Probleme bereitet. Wenn der e-puck vor einer schmalen Gasse stand, wurde Sektor 4 als belegt bestimmt. Bei der Fahrt auf eine schmale Gasse zu, kurz vor der Einfahrt, wurde Sektor 1 als belegt ausgegeben und während der Fahrt durch die Gasse war der Sektor hinten immer belegt. Sonst wurde im linken Sektor ein Hindernis zu spät erkannt was aber nur ein kleines Problem darstellt. Im Vergleich zu dem ersten MLP lief das zweite aber deutlich besser. Um eine weiter Verbesserung herbeizuführen,erweiterte ich die Trainingsdaten noch einmal. Die neuen Daten wurden aufgenommen, indem der e-puck durch eine Gasse gefahren ist. Wie gehabt wurden die Sensordaten aufgenommen und die Outputs vorgegeben. Das neue MLP hat keine neuen Probleme mit sich gebracht und war somit eine Verbesserung, aber hatte immer noch kleine Fehler. Aber die Fehler des Netzes waren erst einmal zu vernachlässigen. Erst wollte ich das Netz testen, ob es schon ausreichend trainiert ist, um ein Ziel mit gleichzeitiger Hindernisvermeidung erreichen zu können.

4.10 Erreichen eines Ziels mit Hindernisvermeidung

Erreichen eines Ziels mit Hindernisvermeidung bedeutet, dass ein gegebenes Ziel erreicht werden soll ohne das Hindernisse ein großes Problem darstellen. Hindernisse, die sich auf dem Weg befinden, sollen umfahren werden. Bei der Vermeidung der Hindernisse sollte diesen möglichst so ausgewichen werden, dass man sich zum Ziel hin bewegt und nicht etwa endlos im Kreis fährt ohne jemals das Ziel zu erreichen. Es gibt zwei Möglichkeiten ein Ziel zu finden bei gleichzeitiger Hindernisvermeidung. Die erste Möglichkeit ist die bereits erwähnte Lösung zur Hindernisvermeidung (siehe hierzu Abschnitt 4.9.2.2) mit einer entsprechenden Erweiterung des Entscheidungsalgorithmus. In die Entscheidungsfindung muss das Ziel mit einbezogen werden und kann dann so ablaufen:

Erreichen eines Ziels mit Hindernisvermeidung: Algorithmus I

1. fahre auf direktem Weg zum Ziel
2. ist ein Hindernis im Sektor nach vorne
   a) drehe in die Richtung eines freien Sektors
   b) fahre solange in die gewählte Richtung bis der Sektor in dem das Ziel liegt wieder frei ist
   c) drehe in die Richtung des Sektors in dem das Ziel liegt
   d) fahre solange gerade aus bis
      i. das Ziel erreicht ist und springe zu 3.
      ii. das Ziel in einem anderen freien Sektor liegt und springe zu c.
      iii. ein Hindernis im Sektor nach vorne detektiert wird, dann mache weiter bei a.
3. ist das Ziel erreicht, dann halte an

Die zweite Möglichkeit wäre ein weiteres Neuronales Netz, dem die Sektorbelegungen und das zu erreichende Ziel als Inputs dienen und das als Output den günstigsten zu befahrenden Sektor ausgibt. Da ich noch keine genauen Vorstellungen hatte, wie ich ein zweites Neuronales Netz realisieren sollte und ich erst einmal probieren wollte, ob das Erreichen eines Ziels mit dem bestehenden Netz funktioniert, habe ich mich für eine Lösung mit herkömmlichen Programmieren entschieden.

4.10.1 Ziel erreichen mit Positionsregler

Die erste Überlegung war, mit Hilfe eines Positionsreglers die Navigation zum Ziel durchzuführen und mit meinem schon erstellten MLP aus Abschnitt 4.9.2.2 Hindernisse zu vermeiden.

ohne MLP zur Hindernisvermeidung

Der verwendete Positionsregler ist im Umfang des Frameworks für den e-puck enthalten [14]. Dem Regler wird das Ziel in Form von den X- Y-Koordinaten und einem Winkel, in dem zum Ziel gefahren werden soll, übergeben. Die Fahrt zum Ziel wird dann vom Regler übernommen. Der Positionsregler läßt den Roboter, durch Regelung der Geschwindigkeit der Räder, entlang sanfter Kurven zum Ziel fahren (siehe Abb: 4.12). Eine Hindernisvermeidung ist aber nicht gegeben.

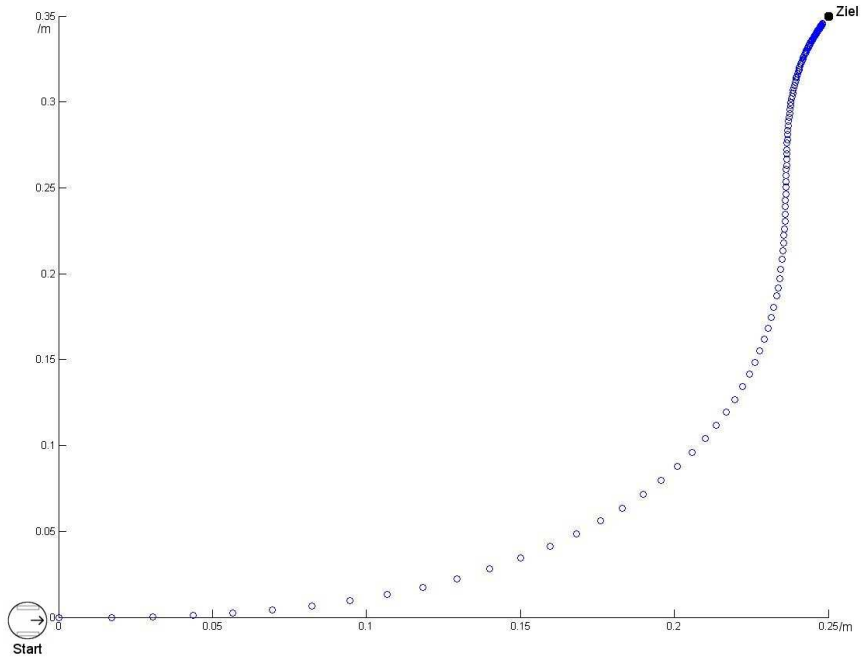


Abbildung 4.12: Auf dieser Abbildung ist zu sehen, wie der Roboter mit Hilfe des Positionsreglers auf sanften Kurven zum Ziel fährt.

mit MLP zur Hindernisvermeidung

Zur Kombination des MLP zur Hindernisvermeidung mit dem Positionsregler habe ich ein Programm geschrieben, das wie folgt abläuft:

Erreichen eines Ziels mit Hindernisvermeidung: Algorithmus II

1. ist Sektor 1 frei
   a) dem Positionsregler werden die Parameter des Ziels übergeben
   b) der Positionsregler regelt die Geschwindigkeit der Räder um an das Ziel zu gelangen
2. ist Sektor 1 belegt
   a) je nach Sektorbelegung der anderen Sektoren wird eine Richtung ausgesucht die am „freisten“ ist
   b) dann wird in den ausgewählten Sektor gedreht
3. weiter bei 1.

Beim Testen des Programms ist ein kleinerer schon bekannter Fehler des MLP aufgefallen. Damit das MLP-Netz keine falschen Outputs ausgibt, musste das Verhalten, wenn nur einer der Sensoren 1 oder 6 (vorne schräg links/rechts) angesprochen wird, noch erlernt werden. Die Lösung mit dem Regler hat nach der kleinen Verbesserung des MLP-Netzes auch funktioniert (siehe dazu Abb: 4.13).

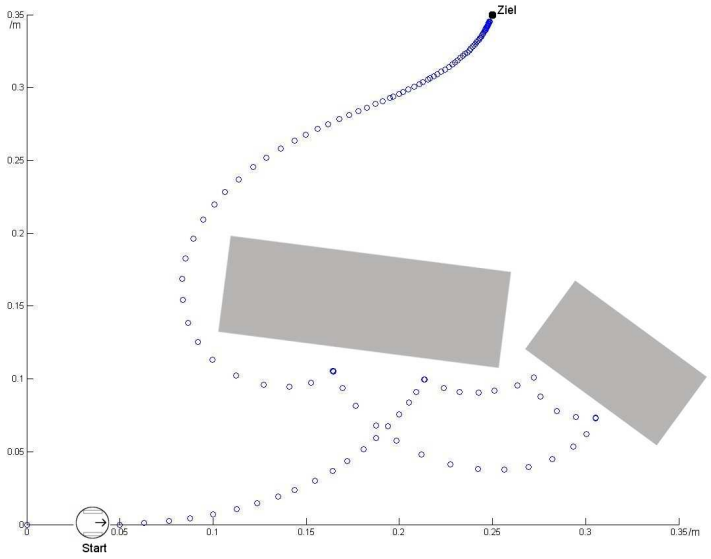


Abbildung 4.13: Auf dieser Abbildung kann man sehen, wie den Hindernissen ausgewichen wird und das Ziel erreicht wird.

4.10.2 Ziel erreichen mit Hilfe der Sektoren-Strategie

Mein Ziel war nicht nur die Hindernisse durch die Auswahl geeigneter Sektoren zu vermeiden sondern die Sektoren auch zum Erreichen der Zielkoordinaten zu benutzen. Um das umzusetzen habe ich ein weiteres Programm geschrieben. Das Programm ist an den Algorithmus aus Abschnitt 4.10 angelehnt. Dem Programm werden die Koordinaten des Ziels übergeben, womit der Winkel vom e-puck zum Ziel berechnet werden kann. Um nun an das Ziel zu gelangen, muss der Roboter um den errechneten Winkel gedreht werden und soll dann auf direktem Weg zum Ziel fahren. Die ganze Fahrt über wird der Winkel vom Roboter relative zum Ziel berechnet und es wird ständig überprüft in welchem Sektor der Winkel liegt (die Zuteilung der Winkel zu den Sektoren ist in Tabelle 4.4 dargestellt). Die Sektoren 2 und 3 wurden etwas größer gewählt

Sektor	1	2	3	4
Winkel in Grad	$315° - 45°$	$45° - 160°$	$200° - 315°$	$160° - 200°$

Tabelle 4.4: Die Tabelle zeigt die Zuteilung der Winkel zu den Sektoren. 0° ist gerade aus und die Gradzahlen erhöhen sich in Richtung des Uhrzeigersinns

um den hinteren Sektor kleiner zu machen. Dies wurde so definiert, damit der hintere Sektor selten bis gar nie ausgewählt wird. Damit kann ein frühzeitiges und meistens unnötiges Umdrehen des e-puck verhindert werden. Der Sektor, in dem der berechnete Winkel liegt ist der, der am nächsten zum Ziel liegt. Nach dem Drehen des e-puck zum Ziel hin ist der Winkel also 0° und liegt im ersten Sektor. Wird die Fahrt durch ein Hindernis gestört (Sektor 1 belegt), dann wird an Hand der Outputs des MLP zur Hindernisvermeidung ein freier Sektor ausgewählt. Sind mehrere Sektoren frei, wird der Sektor ausgewählt der am nächsten zum errechneten Winkel liegt. Kommen dazu zwei Sektoren in Frage wird der gewählt, der den geringsten Outputwert angibt. In die Richtung des ausgewählten Sektors wird dann solange gefahren bis der Sektor, der am nächsten zum Ziel liegt wieder frei ist. Es wird so also der zu befahrende Sektor ausgewählt mit dem das Ziel am schnellsten erreicht werden kann. Die erzielten Ergebnisse mit diesem Programm sind wie erwartet. Beim Testen hat sich herausgestellt, dass ein Ziel in einem Gebiet mit Hindernissen fast immer erreicht werden kann. Es kommt zwar auch mal vor, dass das Ziel verfehlt wird, aber in der Regel funktioniert es gut. Warum das Erreichen eines Ziels manchmal nicht klappt ist mir unklar. Fehler könnten im Programm liegen, die Updates der Odometrie setzen manchmal aus oder sind fehlerhaft, die Sensorwerte sind umgebungsabhängig und das MLP ist auch nicht perfekt. Für wenige Situationen gibt das MLP-Netz fehlerhafte Outputs. Doch im Großen und Ganzen sind diese Fehler zu vernachlässigen. Für einfache, nicht all zu komplexe Szenarien wird ein Ziel sogar mit der ersten MLP aus Abschnitt 4.9.2.2 auf Seite 35 erreicht. Das Ergebnis war für die wenigen Trainingsdaten sehr gut. Die Graphen dazu sind in Abbildung 4.14 zu sehen.

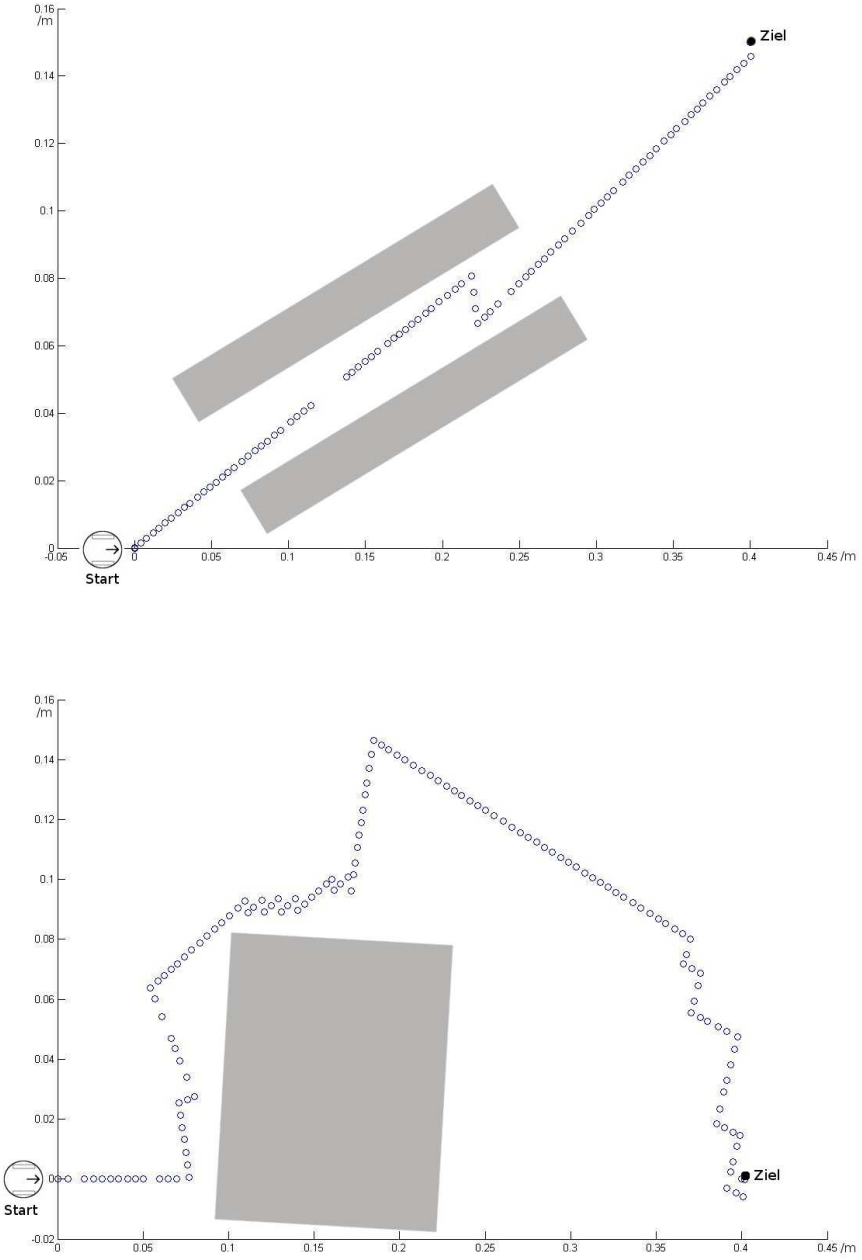


Abbildung 4.14: Beide Abbildungen zeigen die Fahrt des e-puck zu einem Ziel mit dem MLP aus Abschnitt 4.9.2.2 auf Seite 35.

Für komplexere Szenarien wurde das MLP, welches bei der Lösung mit dem Positionsregler verwendet wurde beim Versuch das Fluchtverhalten zu realisieren noch einmal verbessert. Dieses MLP-Netz funktioniert sehr zufriedenstellend. Die Ergebnisse damit sind in den Abbildungen 4.15 und 4.16 visualisiert.

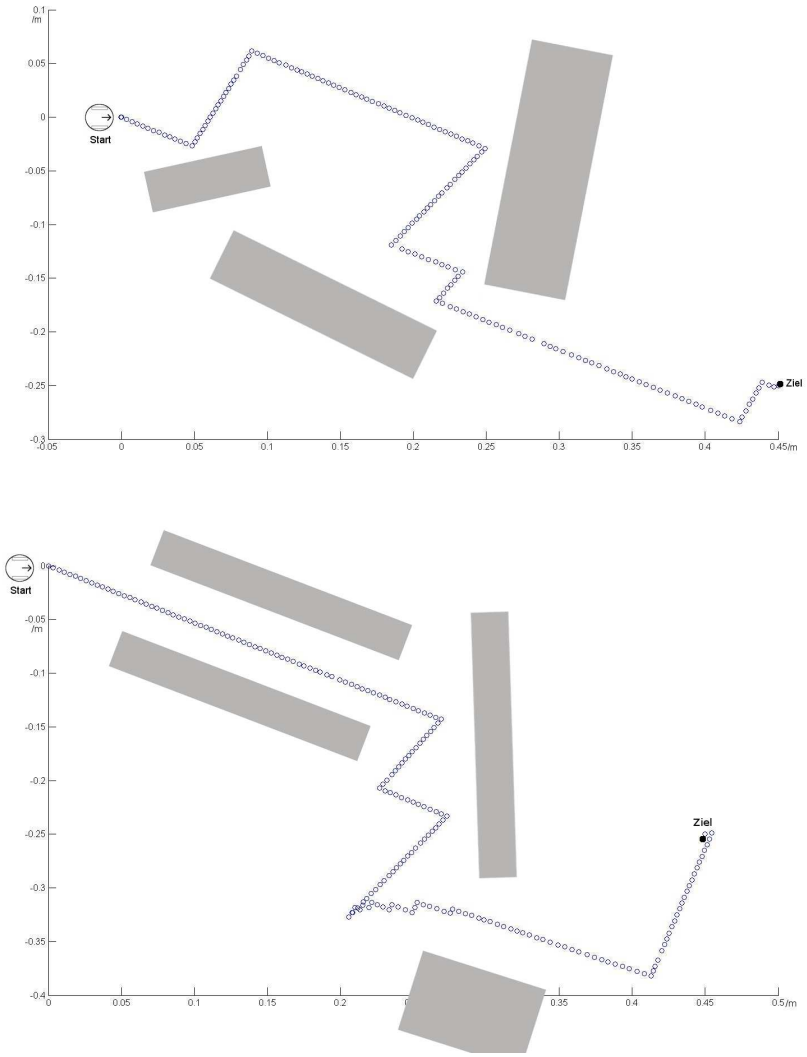


Abbildung 4.15: Auf beiden Bildern kann man sehen, wie der e-puck sein Ziel erreicht trotz Hindernissen auf dem Weg.

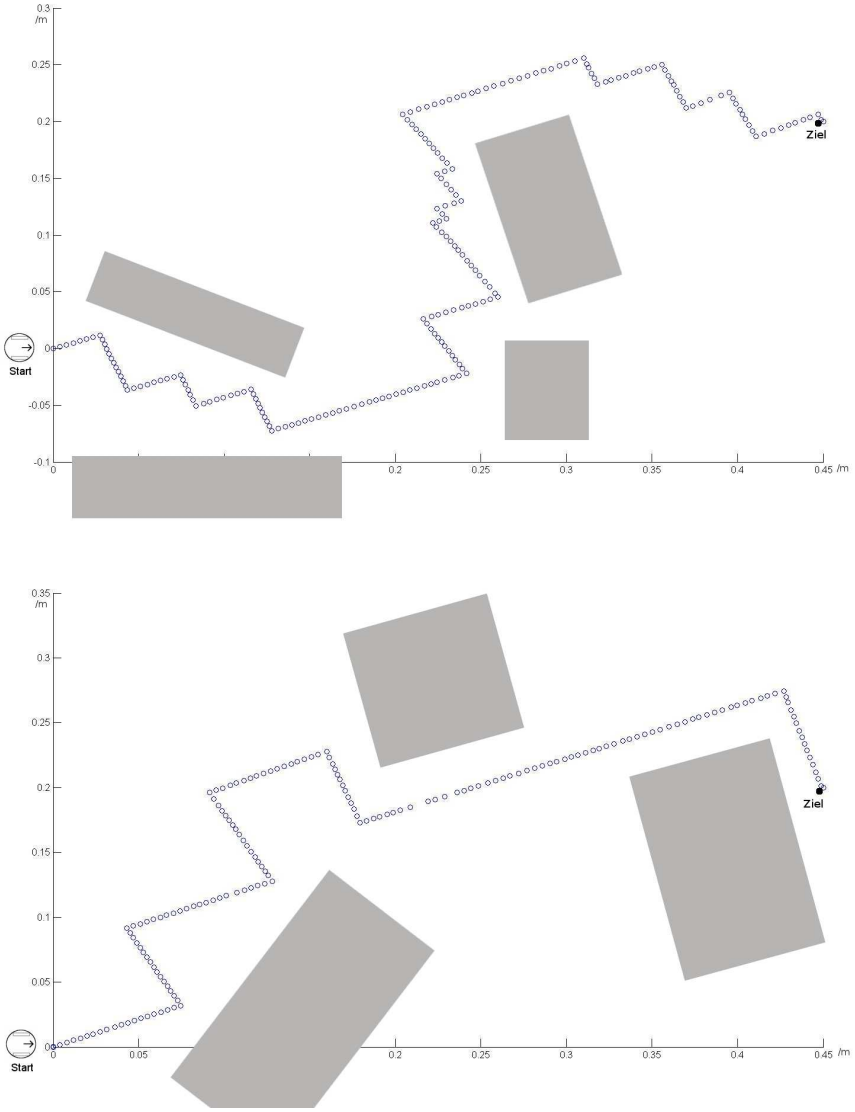
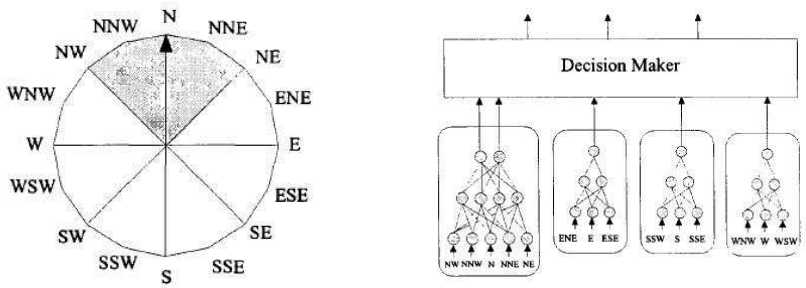


Abbildung 4.16: Der e-puck erreicht die gewünschten Ziele trotz Hindernissen.

4.10.3 MONODA (Modular Network for Obstacle Detection and Avoidance)

Ein weiterer Lösungsansatz könnte die MONODA Architektur[13] darstellen. Hier wird die Umgebung des Roboters auch in Sektoren aufgeteilt. Die MONODA Architektur funktioniert gut um Hindernisse zu erkennen und sie zu vermeiden. Sie besteht aus einem Neuronalen Netz, welches selbst aus mehreren modular strukturierten MLPs besteht (zu sehen in Abb: 4.17(b)). Um die Performanz beim Training des Neuronalen Netzes zu verbessern, wurde das Prinzip *devide and conquer* (teile und herrsche) zur Hilfe gezogen, womit es einfacher ist umfangreiche, komplexe Probleme zu lösen. Hierbei wird das Hauptproblem in mehrere kleinere und leichter zu lösende Probleme geteilt. Die Lösungen der Teilprobleme ergeben dann zusammen die Gesamtlösung. Durch die modulare Struktur bekommt man eine bessere Generalisierung zu Stande weil jedes der kleinen Module besser zu trainieren ist. Außerdem sind die Gewichte der Verbindungen weniger als bei einem vollständig verbundenen MLP. Der Roboter mit dem diese Architektur getestet wurde, war ein NOMAD der mit 16 Ultraschallsensoren und 16 Infrarotsensoren besser ausgestattet ist als der e-puck, der „nur“ 8 Infrarotsensoren besitzt. Um die Hinderniserkennung und Hindernisvermeidung zu realisieren, wurde das Problem in vier Teile zerlegt (vier Sektoren), die zusammen entscheiden in welche Richtung der Roboter fahren soll. Dieser Lösungsansatz war aber leider mit dem e-puck nicht umzusetzen, da einfach zu wenige Sensoren zur Verfügung stehen, damit es Sinn macht separate MLPs für jeden Sektor zu nutzen.



(a) Sektoreinteilung (b) Architektur

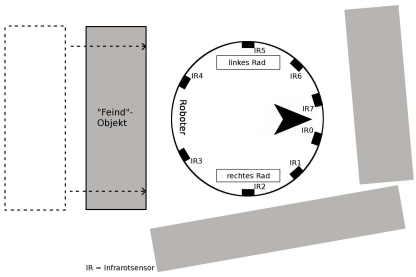
Abbildung 4.17: Zwei Bilder : a) Hier ist die Sektoreinteilung des NOMAD Roboters zu sehen. b) Hier ist die Architektur der modular aufgebauten Netze dargestellt. [13]

4.11 Flucht

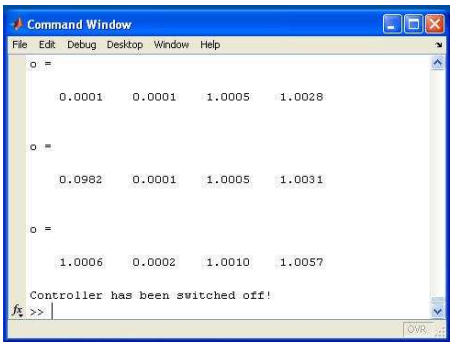
Beim Fluchtverhalten soll der Roboter flüchten, sobald sich ihm ein Objekt nähert. Die Flucht sieht so aus, dass der Roboter versucht in die entgegengesetzte Richtung zu fliehen. Befindet sich ein Hindernis auf dem Fluchtweg muss der günstigste der übrig gebliebenen Fluchtwege ausgewählt werden. Die Schwierigkeit bei der Flucht liegt darin, dass der Roboter erkennen muss, ob und von wo sich ein möglicher „Feind“ nähert und was passiert, wenn der Fall eintritt, dass sich ein Hindernis auf dem Fluchtweg befindet. Für die Flucht vor einem sich näherndem Objekt habe ich auch den Ansatz mit den vier Sektoren verwendet. Durch das Wissen über die Belegung der Sektoren kann man leicht einen Fluchtweg bestimmen, falls sich dem Roboter ein Objekt nähert. Das Fluchtverhalten des Roboters wurde mit demselben MLP-Netz realisiert wie das, welches zum Erreichen eines Ziels mit Hindernisvermeidung eingesetzt wurde. Ein sich näherndes Objekt wird durch das MLP zur Bestimmung der Sektorbelegung erkannt. Nähert sich dem e-puck etwas, kann das an den sich erhöhenden Outputwerten des MLP abgeleitet werden. Ab einem bestimmten Outputwert gilt ein Sektor als belegt. Sobald ein Sektor als belegt gilt dreht sich der e-puck in die Richtung des entgegengesetzten Sektors und fährt gerade aus. Ist der gewünschte Fluchtsektor belegt, dann wird in einen anderen freien Sektor ausgewichen. Der Sektor in den der Roboter ausweichen soll, wird durch eine Funktion bestimmt, die die Sektorbelegung übergeben bekommt und damit den günstigsten Sektor auswählt. Ein Beispiel zum Fluchtverhalten ist in 4.18 dargestellt. Die einzige Schwierigkeit bei dem Fluchtverhalten war der Fall, dass sich ein Hindernis in Front befindet und ein Objekt nähert sich von hinten. Für diesen Fall lieferte das trainierte MLP-Netz aus Abschnitt 4.10.2 zum Erreichen eines Ziels total falsche Ausgaben. Dieses Problem ist bei der Fahrt zu einem Ziel mit Hindernisvermeidung nie aufgetreten und deshalb auch nicht aufgefallen. Nach dem der genannte Fall trainiert wurde war auch dieses Problem gelöst. Nähern sich aus verschiedenen Richtungen Objekte wird immer vor dem Sektor mit dem höchsten Belegungsgrad geflüchtet.

4.12 Diskussion

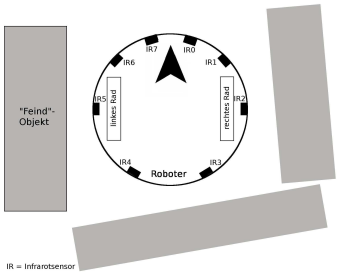
Das wohl größte Problem während dieser Bachelorarbeit waren die Fehler, die der e-puck verursacht hat. Das Framework des e-puck ist leicht zu verwenden, die Oberfläche um den e-puck zu bedienen und das Bedienen an sich waren zwar leicht, aber die Hardware des e-puck ist fehleranfällig. Ein Problem mit der Hardware sind die Schrittmotoren. Der e-puck mit dem ich die Trainingsdaten aufgenommen und auf dem ich die MLPs angewendet habe, konnte nur schwer dazu gebracht werden geradeaus zu fahren. Das Problem ist auch auf einem weiteren e-puck aufgetreten. Der Grund dafür ist, dass der linke Schrittmotor von Zeit zu Zeit einen oder auch mehrere Schritte auslässt und der Roboter somit nach links fährt. Vielleicht ist es auch ein Softwareproblem des e-puck, aber meinem Erachten nach ist der Schrittmotor kaputt. Aber der Fakt, dass der e-puck immer ein bisschen, mal mehr mal weniger, nach links fährt



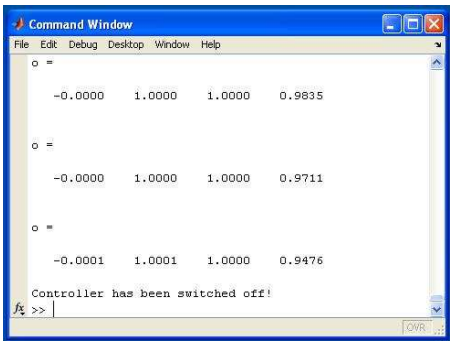
(a) Testaufbau



(b) Outputs des trainierten MLP



(c) Testaufbau



(d) Outputs des trainierten MLP

Abbildung 4.18: a) Ein Objekt nähert sich dem e-puck von hinten und zwei Hindernisse sind in Front b) Sektorenbelegung (o = Sektor 1 bis 4 von links nach rechts gelesen) zu dem Testaufbau in a) c) Der e-puck hat sich in den freien Sektor gedreht d) Sektorenbelegung (o = Sektor 1 bis 4 von links nach rechts gelesen) zu dem Testaufbau in a) nach dem Drehen

ist nicht so schlimm, da die ungenaue Fahrt ja keinen Einfluss auf das MLP hat. Der Einfluss äußert sich eher darin, dass die Fahrt des Roboters nicht immer der gewollten entspricht, die Ergebnisse der ausgeführten Programme aber meistens richtig sind. Ein weiteres, viel einflussreicheres Problem sind die vorhandenen Sensoren des e-puck. Die Infrarotsensoren liefern die Inputs für das MLP-Netz und sind deshalb von großer Wichtigkeit. Die Werte, die die Sensoren anzeigen differieren sehr stark von Sensor zu Sensor und von e-puck zu e-puck. Es kommt auch vor, dass die Sensoren gar nicht oder sehr ungenau reagieren. Gegen Ende der Bachelorarbeit hatte ich die Möglichkeit meine Programme auf zwei neuen e-pucks zu testen. Auf einem der neuen Roboter konnte keines der erlernten Verhalten ausgeführt werden, doch auf dem zweiten Roboter liefert das MLP eigentlich ganz gute Ergebnisse. Der ausschlaggebende Faktor, ob das MLP funktioniert oder nicht, sind die unterschiedlichen Werte, die die Sensoren liefern. Vergleicht man die drei zur Verfügung stehenden Roboter miteinander sieht man auf den ersten Blick wie groß die Unterschiede bei den gelieferten Werten der Sensoren sind. Wenn man die Sensorwerte eines einzelnen Roboters bei keinem Hindernis in Sicht vergleicht, erkennt man auch hier große Unterschiede. Die unterschiedlichen Sensorwerte bei einem einzelnen Roboter sind nicht so schlimm, weil die Trainingsdaten aus realen Szenarien gewonnen wurden. Will man aber die trainierten MLPs auf einen anderen Roboter adaptieren, muss man viel Zeit aufwenden. Jeder Sensor muss mit einem Faktor, der für jeden einzelnen Sensor bestimmt werden muss, angepasst werden. Die unterschiedlichen Sensorwerte kommen aber auch durch sich verändernde Lichtverhältnisse zu Stande. Damit eine gewisse Robustheit des MLP-Netzes geboten ist, kann man die Trainingsdaten an unterschiedlichen Tagen und zu unterschiedlichen Tageszeiten aufnehmen. Somit reagiert das MLP ohne größere Fehler auf veränderte Lichtverhältnisse. Ein allgemeiner Nachteil der beim Arbeiten mit optischen Abstandssensoren auftritt, ist die Anfälligkeit auf Verschmutzung. Die Aus- und Eingangsflächen des Lichts und der Lichtweg müssen „sauber“ gehalten werden. Wie sich das bei längerem Einsatz von einem e-puck auswirkt ist aber unklar. Die Einteilung der Sensoren zu den gewählten Sektoren ist auch ein Problem. Bei vier Sektoren den Himmelsrichtungen zugeordnet ist das Problem noch relativ gering. Man kann einfach definieren, wenn vorne ein Hindernis im Weg ist, ist der Sektor nach vorne belegt. Aber bei einer feineren Einteilung ist es schwieriger Szenarien zur Erfassung von Trainingsdaten aufzubauen. Eine Überlegung, die bei der Verwendung von Neuronalen Netzen immer für Schwierigkeiten sorgt ist die Frage, was das beste Neuronale Netz für eine gegebene Aufgabenstellung ist. Meine Architektur mit allen zu wählenden Funktionen liefert sehr gute Ergebnisse, aber muss nicht die allerbeste Lösung sein und ist es wahrscheinlich auch nicht.

5 Analyse und Ausblick

Im Mittelpunkt dieser Arbeit stand die Frage, ob sich MLP-Netzwerke eignen um einem autonomen mobilen Roboter verschiedene Verhalten zu lernen. Meine Aufgabe war dabei im Speziellen die Verhalten Hindernisvermeidung, Flucht und das Erreichen eines Ziels mit Hindernisvermeidung einem Roboter beizubringen. Mit der Hilfe eines MLP-Netzes, welches für das Erkennen von Hindernissen in der näheren Umgebung des Roboters zuständig ist, konnten diese Verhalten leicht realisiert werden. Ein Zweck von Neuronalen Netzen ist die Generalisierungsfähigkeit. Bei den in dieser Arbeit verwendeten MLP-Netzen ist zu sehen, dass sich das Erlernte sehr gut generalisieren lässt. Schon bei relativ wenigen Trainingsdaten gibt das MLP auch für neue, unbekannte Eingabedaten die richtigen Ausgaben aus. Durch das Testen der Netze fallen die Schwächen des trainierten Netzes gleich auf und können durch entsprechende Erweiterungen der Trainingsdaten verbessert werden. Somit kann jederzeit ein Fehlverhalten des MLP durch Ergänzung der Trainingsdaten um Beispieldaten für das Szenario, in dem die Fehler in der Ausgabe aufgetreten sind, ausgebessert werden. In solch einem Fall fällt aber auch ein Nachteil von Neuronalen Netzen auf. Es ist leider nicht möglich einem schon trainierten Netz noch etwas dazu zu lernen. Man muss jedes mal die alten Trainingsdaten mit den zu Ergänzenden zusammenfassen und einem neuen Netz zum Lernen präsentieren. Dadurch kann es vorkommen, dass das neu erstellte Netz zwar die neu erlernten Daten gut generalisiert und in den problematischen Szenarien die richtigen Outputs ausgibt, aber dafür neue Probleme, die davor nicht bekannt waren, auftreten. In den meisten Fällen haben sich durch eine kleine Verbesserung der Netze, aber auch andere bestehende Fehlverhalten in Luft aufgelöst und es wurden keine neuen verursacht. Zu den Trainingsdaten an sich ist zu sagen, dass man zu viel besseren Ergebnissen kommt, wenn man reale Daten für das Training benutzt. Bei der Verwendung von selbst erstellten Daten hat sich gezeigt, dass die trainierten MLP-Netze bei Weitem nicht so gut zu generalisieren im Stande sind wie die, die auf realen Trainingsdaten basieren. Bei selbst erstellten Daten ist die Gefahr viel höher selbst Fehler beim Erstellen zu machen. Etwas, was noch für reale Trainingsdaten spricht ist meine Vermutung, dass sich die Fehlertoleranz des MLP erhöht, um so mehr Trainingsdaten aufgenommen werden. Das ist der Fall, weil eventuelle Fehler der Inputs schon in den Trainingsdaten vorkommen können. Die Wahrscheinlichkeit, dass bei der Aufnahme der Trainingsdaten Fehler vorgekommen sind, erhöht sich um so größer die Anzahl der Daten ist. Auf schon bekannte und ähnliche Fehler weiß das MLP somit wie es sich verhalten soll. Der Ansatz die Geschwindigkeit der Räder des Roboters durch ein MLP-Netzwerk zu steuern und dadurch Hindernisse zu vermeiden, hat sich als nicht so gut erwiesen. Bei diesem Ansatz kamen keine allzu guten Ergebnisse heraus. Bei sehr einfachen Szenarien ist die Geschwindigkeitssteuerung an Hand der Infrarotsensoren machbar, aber bei komplexen Szenarien und Aufgabenstellungen ist es wohl

nicht umzusetzen. Für komplexe Szenarien müssten meinen Erfahrungen nach sehr viele Trainingsdaten aufgenommen werden und auch dann wäre es wohl nicht möglich ein Ziel mit gleichzeitiger Hindernisvermeidung zu erreichen. Außerdem ist es wahrscheinlich besser, wenn die Ausgabe des Neuronalen Netz angibt wo sich Hindernisse relativ zum Roboter befinden und dann daraufhin entschieden wird, was zu tun ist. Das wurde beim zweiten Lösungsansatz auch noch nicht beachtet. Hier wurde ausgegeben welcher Sektor der am Besten zu wählende ist. Die Inputs in das Netz sind die Daten der Infrarotsensoren. Zur Hindernisvermeidung ist das durchaus sinnvoll, aber nicht um ein Ziel aufzufinden, da hier die Richtung des Ziels nicht beachtet werden kann. Um alle der genannten Roboterverhalten umzusetzen, hat sich das Einteilen der Umgebung des Roboters in Sektoren als sehr guter Ansatz herausgestellt. Die Wahl von vier Sektoren und das Wissen, wie sie durch Hindernisse belegt sind, ermöglicht dem Roboter durch eine ihm unbekannte Umgebung zu fahren, Hindernissen auszuweichen und ein gewünschtes Ziel zu erreichen. Auch das Fluchtverhalten ist mit dem Sektorenansatz lösbar. Das Wissen über die Belegung der Sektoren in Form von einem Belegungsgrad der im Bereich von 0 bis 1 angegeben wird, hat sich auch bewährt im Gegensatz zum Runden der Outputs. Wenn nur 0 oder 1 ausgegeben wird, weiß man zwar ob ein Sektor frei ist oder nicht, man kann aber nicht erkennen, ob ein Hindernis in unmittelbarer Nähe ist oder ob es sich noch in weiter Ferne befindet. Mit den ungerundeten Werten kann man beispielsweise auch eine Auswahl treffen welcher Sektor freier ist oder man kann definieren bis zu welchem Wert ein Sektor befahrbar ist bevor eine Auswahl eines anderen Sektors durchgeführt werden muss. Die Tests, der von mir geschriebenen Programme, in denen die fertig trainierten MLP-Netze Bestandteil waren, haben meist gute Ergebnisse geliefert. Bei der Umsetzung ein Ziel mit gleichzeitiger Hindernisvermeidung zu erreichen konnte ein, den Erwartungen entsprechendes Ergebnis erzielt werden. Wie in Abbildung 4.15 zu sehen ist, findet der Roboter für verschiedenste Szenarien auf relativ schnellem Weg zu dem vorgegebenen Ziel. Für das Fluchtverhalten konnte ich keine geeigneten Graphen aufnehmen, aber die Ergebnisse sind auch wie zu erwarten. Der e-puck flüchtet vor einem sich nähernden Objekt in die entgegengesetzte Richtung falls der Sektor frei ist. Sollte der Fluchtsektor belegt sein, wird ein passender freier Sektor ausgewählt. Ein Nachteil beim Verwenden von nur vier Sektoren ist, dass die Fahrt sehr eckig und unrund verläuft. Ist es nötig den Sektor zu wechseln, muss der Roboter jedes mal eine 90°-Drehung machen. Dieser Fakt verlängert die Suche des Ziels und die Fahrt sieht nicht so elegant aus.
Meine Lösungsansätze haben zu einer guten Realisierung der gestellten Aufgaben geführt. Trotzdem gibt es Möglichkeiten das Verhalten des Robters zu verbessern und auch andere Aufgabenstellungen könnten mit einer geeigneten MLP sinnvoll gelöst werden. Eine mögliche Verbesserung der Hindernisvermeidung und somit auch um ein Ziel zu erreichen und der Flucht könnte die feinere Einteilung der Umgebung in mehr als vier Sektoren sein. Eine Einteilung in sechs Sektoren ist meiner Meinung nach besser. Dabei könnte man die Sektoren den Himmelsrichtungen entsprechend in Nord, Nordwesten, Westen, Nordosten, Osten und einen kleinen Sektor für Süden einteilen. Bei einer noch feineren Einteilung der Sektoren sehe ich ein Problem bei der Beschaffung von Trainingsdaten. Es wird um so schwieriger Trainingsszenarien aufzubauen, um so größer die Anzahl der Sektoren ist. Aber mit sechs Sektoren sollte das noch kein

so großes Problem sein. Die zu erwartenden Ergebnisse beim Auffinden eines Ziels sollten um einiges besser sein als meine erzielten Ergebnisse, da eine genauere Navigation möglich sein sollte. Um zu einem Ziel zu fahren könnte man eine andere Variante als die von mir gewählte verwenden. Ein anderer Ansatz wäre eine zweite MLP zu designen, die die Aufgabe, wie das Ziel am Besten erreicht werden kann übernimmt. Dazu könnte man dem zweiten MLP als Inputs die Outputs des ersten MLP zur Hindernisvermeidung, also die Sektorbelegung übergeben und als zusätzlichen Input das Ziel. Einer der Kernpunkte der Navigation eines Roboters ist die Lokalisation. Die Lokalisation ist wichtig zur Bahnplanung. Es gibt zwar die Möglichkeit über die Odometrie herauszufinden wo sich der Roboter relativ zu seiner Umgebung befindet, aber diese Variante ist sehr fehleranfällig. Eine andere Variante zu Lösung des Lokalisationsproblems könnte ein MLP Netzwerk bieten.
Als Fazit bleibt zu sagen, diese Bachelorarbeit hat gezeigt, dass MLP Netzwerke für einen autonomen mobilen Roboter ein sehr gutes Werkzeug darstellen um ihm gewünschte Verhalten zu erlernen.

Literaturverzeichnis

[1] Yves Chauvin, David E. Rumelhart *Backpropagation: theory, architectures, and applications Developments in connectionist theory* Routledge, 1995

[2] Howard Demuth, Mark Beale, Martin Hagan *Neural Network Toolbox^{TM}6 User's Guide*. The MathWorks, Inc.,2009

[3] Russell C. Eberhart, Roy W. Dobbins *Neural Network PC Tools: A Practical Guide* Academic Press, 1990

[4] Sven David Krause *Verfahren zur Merkmalsselektion für die Klassifikation mit Künstlichen Neuronalen Netzen* Shaker Verlag, Aachen 2003.

[5] David Kriesel *Ein kleiner Überblick über Neuronale Netze* URL: http://www.dkriesel.com, 2007

[6] Mark H. Lee *Inelligente Roboter* VCH Verlagsgesellschaft mbH, Weinheim 1991

[7] *Lexikon der Naturwissenschaften* MEYERS LEXIKONVERLAG, Bibliographisches Institut & F.A.Brockhaus AG, Mannheim 2008, Seite 791

[8] D. McCloy, D.M.J. Harris *Robotertechnik - Einführung* VCH Verlagsgesellschaft mbH, Weinheim 1989

[9] Francesco Mondada, Michael Bonani, Xavier Raemy, James Pugh, Christopher Cianci, Adam Klaptocz, Stèphane Magnenat, Jean-Christophe Zufferey, Dario Floreano, Alcherio Martinoli *The e-puck, a Robot Designed for Education in Engineering* Proceedings of the 9th Conference on Autonomous Robot Systems and Competitions, 1(1) pp. 59-65 ,2009

[10] Jorge Nocedal, Stephen J. Wright *Numerical Optimization* Springer, 1999

[11] Mohamed Oubbati *Neuronale Netze in der Robotik* unveröff. Folien zur Vorlesung, Ulm 2008.

[12] Mohamed Oubbati *Einführung in die Robotik* unveröff. Skriptum, Ulm 2008.

[13] Catarina Silva, Manuel Crisóstomo, Bernardete Ribeirot *MONODA: A Neural Modular Architecture for Obstacle Avoidance Without Knowledge of the Environment* ijcnn, vol.6, pp.6334, IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)-Volume 6, 2000

[14] *Offizielle e-puck Internetseite* http://www.e-puck.org