Dysfunktionale Methoden der Robotik

Frank Schröder

7. November 2017

Germany

Zusammenfassung

Bei der Realisierung von Robotik-Projekten kann man eine ganze Menge verkehrt machen. Damit sind nicht nur kalte Lötstellen oder abstürzende Software gemeint, sondern sehr viel grundsätzlichere Dinge spielen eine Rolle. Um Fehler zu vermeiden, muss man sich zunächst einmal mit den Failure-Patterns näher auseinandersetzen, also jenen Entwicklungsmethoden, nach denen man auf gar keinen Fall einen Roboter bauen und wie die Software möglichst nicht funktionieren sollte.

Stichworte: Antipattern, Robotics, Strong-AI, Failed software projects, Artificial Intelligence

1 Einleitung

Normalerweise ist es in den Wissenschaften üblich gesicherte Erkenntnisse und funktionierende Erfindungen zu publizieren.[8] Die dahinterstehende Idee lautet, gesichertes Wissen zu sammeln und zu mehren. Das Problem mit diesem Ansatz ist, dass sich innerhalb der Mathematik und der Informatik fast alles als wahr und korrekt beweisen lässt. Jede Formel (selbst wenn sie ausgedacht ist) ist in sich stimmig und bildet ein selbstbezügliches Weltbild. Nicht viel anders ist es mit Computerprogrammen wovon die meisten korrekt sind im Sinne eines Compilerdurchlaufs der keine Fehler ausgibt. Aber sind deswegen die mathematischen Konstrukten und die erstellten Programme bereits wertvoll im Sinne, dass sie die Wissenschaft voranbringen? Nicht automatisch, sondern nur manchmal. Worum es also geht ist einen Maßstab zu entwickeln der unterscheidet ob eine Formel wirklich korrekt ist oder nur so aussieht.[16]

Dieses Unterfangen mag zunächst sehr komplex klingen, weil es nichts anderes bedeutet als auf ein existierendes Bezugssystem ein zweites obendrauf zu legen um so das erste zu verifizieren. Und wer sagt uns, dass das neue Bezugssystem nicht ebenfalls auf unbewiesenen Annahmen basiert? Es gibt jedoch einen sehr viel leichteren Weg sich der Thematik zu nähern und zwar in dem man den Fokus auf all jene Dinge richtet, die nicht funktionieren. Wo es also in der Praxis immer wieder zu Bugs kommt und wodurch der Eindruck entsteht dass da ganz grundsätzlich etwas fehlerhaft sein könnte. Ein wenig umgangssprachlicher formuliert geht es um die Anti-Pattern und zwar speziell solche auf dem Gebiet der Robotik. Diese sollen im folgenden Text vorgestellt und diskutiert werden.

2 Neuronale Netze

2.1 Probleme lösen ohne zu programmieren

Neuronale Netze sind eine sehr alte Disziplin die sich zurückdatieren lässt bis in die 1940'er Jahre. Auch Alan Turing hat bereits ein Paper publiziert [1] in dem er sogenannte b-type neuronale Netze vorstellt. Die Idee dahinter lautet, die Turing-Maschine so zu erweitern, dass sie von alleine eine Aufgabe löst, einfach indem man Beispiel-Muster vorgibt und über einen Backpropagation Algorithmus dann die künstlichen Neuronen sich ähnlich wie Synapsen eines menschlichen Gehirns daran anpassen.

Eine schöne Idee nur leider hat sie einen Haken: sie funktioniert nicht. Rein formal sind neuronale Netze eine Art von Universal-Problem-Solver. Es gibt ein System worin man Parameter verändern kann und die Aufgabe besteht darin die richtigen Parameter zu ermitteln. Wer schonmal versucht hat eine Lösung zu finden wird entdecken, dass neuronale Netze weniger eine Antwort darstellen sondern ein Rätsel sind. In dem Sinne dass man einen Algorithmus sucht der die Gewichte der Neuronalen Netze von alleine bestimmt. Und genau hier liegt die Schwierigkeit. Bis heute gibt es keinen solchen Algorithmus.[2] Der bereits erwähnte Backpropagation Algorithmus ist zwar in der Lage die Schwellwerte für das XOR Problem zu ermitteln aber nicht für komplexere Aufgaben wie z.B. für die Primzahlsuche. Es gibt zwar relativ viel Veröffentlichungen in Bezug auf neuronale Netze zur Bilderkennung [10] sowie zur audiovisuellen Mustererkennung doch bisher ohne größere Erfolge. Offenbar ist es doch nicht so einfach eine Turing-Maschine zu konstruieren die nicht mehr programmiert werden muss.

Machen wir es etwas konkreter und untersuchen was passiert wenn man einen Roboter der einer Linie folgen soll über ein neuronales Netz steuert. Nun, der Roboter wird sich auf ähnliche Weise verhalten wie ein BEAM Roboter (also quasi-zufälliges Verhalten zeigen). Dieses Verhalten wird sich im Laufe der Zeit ändern und zwar wegen des Lernalgorithmus, es wird aber nicht dazu führen, dass der Roboter in einen stabilen Zustand kommt wo er am Ende die Aufgabe "Line following" bewältigt, sondern es wird auch nach vielen Durchläufen reines Glück sein ob der Roboter der Linie folgt oder nicht. Und falls nicht, hat der Programmierer keine Möglichkeit die Hintergrunde zu untersuchen weil neuronale

[1] http://www.alanturing.net/intelligent_machinery/

Netze die unangenehme Eigenschaft besitzen als Blackbox zu funktionieren.

Es gibt aber noch einen weiteren Grund warum Neuronale Netze eine dysfunktionale Technologie darstellen, und zwar sind sie dogmatisch an biologischen Vorbildern des menschlichen Gehirns orientiert. Die Idee geht ungefähr so: das menschliche Gehirn ist in der Lage zu denken und zu lernen. Das menschliche Gehirn besteht aus Neuronen die untereinander vernetzt sind, also muss auch der Computer aus Neuronen bestehen damit er denken und lernen kann. Die Gleichsetzung von Computern mit biologischen Gehirnen besitzt den Fehler dass alternative Erklärungsmuster ausgeblendet werden. Ja mehr noch, neuronale Netze widersprechen sogar der abstrakten Turing-Machine. wie weiter oben schon erläutert wird eine Turing-Maschine durch ein Programm gesteuert, während neuronale Netze ohne Programmierung auskommen.

Ob nicht irgendwann in ferner Zukunft mit Hilfe neuronaler Netze nicht doch Roboter gesteuert werden können vermag derzeit niemand zu sagen. Fakt ist jedenfalls, dass bisher alle Roboter die in Wettbewerben konkrete Probleme gelöst haben ohne neuronale Netze ausgekommen sind und stattdessen klassisch in einer Hochsprache wie C programmiert wurden.

2.2 Blick in die Glaskugel

Es gab einmal eine Zeit wo Neuronale Netze sehr skeptisch betrachtet wurden. Damals wurden neuronale Netze eingesetzt um Zeitreihen vorherzusagen. Die Erwartung, welche formuliert wurde, lautete nicht nur, dass das Netz bitteschön intelligent sein soll, nein es sollte auch noch ermitteln können wie morgen das Wetter wird, oder noch besser auf welche Zahl die Roulette-Kugel beim nächsten Wurf gelangt. Wer sich ein wenig mit Wahrscheinlichkeitsrechnung auskennt wird ahnen, dass dies nicht möglich ist, weil die Roulettekugel zufällig fällt und man sie nicht vorhersagen kann. Nichts desto trotz wurden neuronale Netze für genau diese Aufgabe instrumentalisiert. Genauer gesagt wurde das Anliegen unter dem Stichwort “chaotic timeseries prediction” diskutiert und bedeutete, dass man sich auf wissenschaftlicher Ebene mit der Vorhersage von nicht-vorhersagbaren Ereignissen beschäftigte.

Später als der DeepLearning Hype losging:

> "At the same time, the hype surrounding deep learning has been exponentially growing and is at an all-time high." [15]

wurde die Vorhersage von Zufallsprozessen nicht länger diskutiert sondern nun hieß das Zauberwort: Lernen. Gemeint war Machine Learning, DeepLearning, Learning ganz allgemein. Die Erwartungshaltung war ähnlich gelagert wie beim Forcasting: es ging darum nicht einfach nur Systeme zu entwickeln, die intelligent sind, sondern sie sollte möglichst sehr viel mehr als das sein. Genau betrachtet ist Machine Learning und Time-Series-Prediction ein und dasselbe. Der Unterschied ist nur der, dass die Zielstellung einen Zufallsprozess vorhersagen zu wollen unter einigen Wissenschaftlern inzwischen als unseriös gilt, während Machine Learning akzeptiert ist.

Leider funktioniert beides nicht. Mit neuronalen Netzen kann man weder vorhersagen, wie morgen das Wetter wird noch kann ein Netz etwas lernen. Der Grund warum dennoch viele Forscher danach suchen hat etwas mit der Hoffnung zu tun, dass es doch irgendwie ginge. Es ist der alte Wunsch mit einem Computer weitaus mehr zu machen als stupide Berechnungen auszuführen, sondern Computer werden als Zauberkugel betrachtet durch die man die Welt auf eine neue Weise kennenlernt. Dazu gehört auch, dass man in Sphären schaut die normalerweise nicht sichtbar sind, wie die Zukunft. Auf diese Weise wird moderne Technologie mit traditionellem Aberglauben vermischt. Das führt dazu, dass neuronale Netze nicht so sehr als Problemlöser gesehen werden sondern dass sie zum Selbstzweck erhoben werden. Es also darum geht, neuronale Netze zu realisieren, damit andere Neuronale Netze davon profitieren.

Ein wenig in die Realität zurück findet man leicht dadurch, wenn man einmal versucht mit Robotern konkrete Aufgaben wie den Robocup Wettbewerb zu meistern. Dort kommt es nicht darauf an, wer das tiefste und schönste neuronale Netz implementiert hat was am meisten lernen kann, sondern es geht darum Tore zu schießen. Derzeit ist es so, dass keines der Teams neuronale Netze verwendet, was sehr viel über die tatsächliche Leistungsfähigkeit von DeepLearning aussagt. Wären neuronale Netze wirklich so toll würde sich das Konzept bei Robotik-Wettbewerben durchsetzen und andere Lösungsstrategien wie manuelles Erstellen von C++ Sourcecode verdrängen.

2.3 Reinforcement Learning

Warum Reinforcement Learning (als Teil des machinellen Learnings) zunächst attraktiv klingt hat damit zu tun, dass es als mögliche Alternative zu statischen Algorithmen gesehen wird. Ein statischer Algorithmus ist ein handkodierter Ablauf in Sourcecode, der gerade im Bereich der Robotik schwer bis unmöglich erstellt werden kann. Denn, wie soll der Algorithmus aussehen, der einen Roboter ein Tor schießen lässt oder über ein Hindernis klettern lässt? Und so liegt es auf der Hand in statischen Algorithmen die eigentliche Problematik auszumachen und nach vermeintlich besseren Alternativen Ausschau zu halten. Ganz vorne mit dabei ist Reinforcement Learning und Q-Learning. Beides sind dynamische Verfahren, bei denen es keine starren Algorithmen mehr gibt, sondern sich das Programm an die jeweilige Aufgabe anpasst. Das Problem hierbei ist, dass die Programme in sich immer richtig sind. Ein Reinforcement Learning Algorithmus kann nicht falsch sein, weil er sich laufend verändert, es gibt allenfalls Parameter, die falsch sein können und folglich geht es darum, diese Parameter zu finden.

Leider bedeutet der Verzicht auf fehlerhaften Code automatisch, dass man Reinforcement Learning Algorithmen nicht länger iterativ verbessern kann. Während man bei statischen Algorithmen ein Problem erkennen und beseitigen kann ist dies bei Reinforcement Learning Verfahren nicht gegeben. Stattdessen dreht man sich im Kreis und lässt die Parameter solange fluktuieren bis der Roboter sich per Zufall richtig bewegt:

> "The system is destabilized until it relearns the correct action, which is reflected in the test phase.

These fluctuations are common in RL algorithms." [11]

2.4 Bezug zu den Neurowissenschaften

Ein ganz heißer Kandidat in Sachen Anti-Pattern innerhalb der Robotik ist die Fokussierung auf menschliche Intelligenz als Vorbild. Dies wurde vor allem in der Frühzeit der Kybernetik betrieben wo es darum ging, das menschliche Gehirn nachzubauen. Dabei ist so falsch der Ansatz gar nicht. Denn immerhin ist das menschliche Gehirn das beste Vorbild in Sachen künstlicher Intelligenz und die Psychologie ist eine Wissenschaft die sich sehr umfassend mit Denken und Emotionen auseinandergesetzt hat. Wenn also eine Theorie innerhalb der Neurowissenschaften und innerhalb der Psychologie als korrekt bezeichnet wird, dann ist sie auch im Bereich der Künstlichen Intelligenz ein wertvoller Beitrag – so zumindest die Hoffnung.

Das Problem mit dieser Sichtweise ist, dass die Forschung rein theoretisch bleibt. Es wird angenommen, dass es ausreicht ein neuronales Netz auf einem Computer nachzubauen oder eine kognitive Architektur zu realisieren und schon hätte man wissenschaftliche Robotik betrieben. Diese Herangehensweise kann sehr schnell abgleiten ins Formale, wo also eine Theorie als wahr angenommen wird weil sie in den Neurowissenschaften als wahr gilt ohne zu überprüfen ob auf realen Robotern damit sich auch nur die einfachsten Dinge lösen lassen.

Das beste Negativ-Beispiel ist wohl, wenn man auf einem Supercomputer ein riesiges mehrschichtiges Perceptron implementiert [13] und es wunderbar läuft, aber man mit diesem neuronalen Netz exakt null anfangen kann. Der Fehler besteht darin, keine Challanges oder Benchmarkts zu definieren anhand derer man sich abarbeitet und bei denen man scheitern kann. So ist die Forschung automatisch richtig und dreht sich im Kreis.

Warum die Künstliche Intelligenz sich an der Psychologie und der Biologie orientiert hat einen simplen Grund: auf diese Weise versucht man die Suche nach denkenden Maschinen auf eine wissenschaftliche Grundlage zu stellen. Man forscht nicht länger ins Blaue hinein sondern verfolgt als Zweck menschliches Denken besser zu verstehen und das geht am besten wenn man es in einem Computer simuliert. Leider sind die meisten neuronalen Netze und kognitive Modelle als Blackbox ausgerichtet, das heißt sie sind selbstbezüglich und folgen einer inneren Logik die sich von außen nicht verifizieren lässt. Damit ist gemeint, dass die so erstellten Roboter eben nicht eine konkrete Aufgabe wiederholt ausführen in einer bestimmten Zeitspanne sondern es gibt überhaupt keinen Roboter sondern die Theorie existiert nur auf dem Papier und muss später oder gar nicht konkret realisiert werden. Diese rein theoretische Robotik-Forschung führt dazu, dass zwar viel versprochen wird was alles möglich sei mit Künstlicher Intelligenz ohne dass davon etwas konkret realisiert wird.

> "Alchemists were so successfull in destilling quicksilver from what seemed to be dirt, that after several hundred years of fruitless effort to convert lead into gold they still refused to beliefe that on the chemical level one cannot transmute metals. [...] Does that mean that all the work and money put into artificial intelligence has been wasted?" [5]

3 Deklarative Programmierung

3.1 Rodney Brooks und sein “rant” gegen STRIPS

Rodney Brooks ist in die Geschichte der Robotik eingegangen als der Erfinder der Subsumption Architektur.[4] Dieses Framework ist entstanden als Reaktion auf die Robotik wie sie davor üblich war. Vor Brooks waren Topdown Systeme verbreitet. Das große Vorbild in Sachen Planning und deklarative Programmierung war über Jahrzehnte hinweg das “Shakey the robot” System. Gegen diese Art der Problemlösung hat Brooks in seinen Essays mehrfach Stellung bezogen um eine Alternative zu entwickeln. Aber was genau ist falsch an Shakey und dem STRIPS Planner? Die Idee hinter dem Paradigma der deklarativen Programmierung lautet, dass man Roboter nicht etwa programmiert wie eine normale Software, sondern stattdessen das zu lösende Problem beschreibt. Man versucht also auf einer höheren Abstraktionsebene zu agieren um so die Komplexität in Bezug auf Künstliche Intelligenz in den Griff zu bekommen. Konkretes Resultat dieser Bemühungen ist die Plansprache STRIPS.[6] Es handelt sich anders als bei C nicht um eine klassische Programmiersprache sondern STRIPS hat starke Ähnlichkeit zu einer natürlichen Sprache wie Englisch. Deshalb kann man Strips auch nicht direkt auf einem Computer ausführen (es gibt keine Strips to Assembly Konverter) sondern Strips ist das Eingabeformat für einen Solver. Und das was der Solver ausrechnet sind Aktionen des Roboters die wiederum in die Assembly Language übersetzt werden.

Vom Prinzip her eine gute Idee, kann man darüber doch die Programmierung drastisch erleichtern. Und über Jahrzehnte hinweg wurde Strips und deklarative Programmierung als Alternativlos betrachtet. Es war ein Dogma innerhalb der Robotik, dass so und nicht anders eine Künstliche Intelligenz realisiert wird. Das Problem mit Strips ist folgendes. Damit ein Solver einen Plan berechnen kann muss die deklarative Beschreibung der Wirklichkeit sehr exakt sein. Man kann sich das ungefähr so vorstellen als wenn man mit einen Solver einen Stundenplan für eine Schule erstellen möchte und zuvor alle Nebenbedingungen in das System eingibt. In der Theorie gelangt man darüber ans Ziel. Man formuliert einfach welche Fächer aufeinander folgen dürfen, welche Räume wie belegt sein müssen und der Roboter berechnet dann den fertigen Plan. Wenn man die Nebenbedingungen ändert passt sich der Plan von alleine an. Das Problem ist, dass man bei den Randbedingungen sehr viele Details formalisieren muss. In der Domäne des Stundenplans mag es noch funktionieren, ebenfalls gute Erfahrungen hat die Informatik im Bereich von Zugfahrplänen gemacht, wo man so tatsächlich zu einem Ergebnis gelangt, als undurchführbar gilt das ganze jedoch im Bereich der Robotik.

Nehmen wir mal an, ein Roboter soll durch einen Raum

fahren. Man programmiert den Roboter aber nicht direkt sondern definiert den Task sehr allgemein. Man würde zunächst einmal die Hindernisse in die Beschreibung aufnehmen, dann würde man die verschiedenen Aktionen des Roboters benennen, dann auf die Wegroute eingehen usw. All diese Informationen würden in der Strips Spezifikation abgelegt werden. Das Problem ist folgendes: bisher ist kein praktisches Beispiel bekannt wo das jemand für einen echten Roboter tatsächlich fertiggebracht hat. Was im Shakey Projekt und bei anderen STRIPS-affinen Projekten stattdessen gemacht wird, ist immer nur mit vereinfachten Beispielen zu rechnen. Man nimmt an, dass sich diese Beispiele auf echte / größere Probleme übertragen lassen, doch das tun sie nicht.

Rodney Brooks war einer der ersten der das erkannt hat. Er hat sich die Frage gestellt wie es sein kann, dass die Künstliche Intelligenz einerseits über hochentwickelte deklarative Beschreibungssysteme verfügt mit denen sich extrem komplizierte Probleme formulieren lassen auf der anderen Seite aber reale Roboter nicht in der Lage sind die einfachsten Dinge zu tun. Brooks wollte keineswegs hochkomplexe Roboter bauen, sondern er wollte nur Roboter haben die in einem kleinen Teilbereich funktionieren. Und dieser Mismatch war es, der dazu geführt hat dass deklarative Programmierung insgesamt in Frage gestellt wurde.

Rückblickend ist heute bekannt wo der Fehler war. Einen Solver zu haben mit dem man Roboteraufgaben löst ist ein Ziel wo die Informatik hin-möchte, aber es ist nichts, was sie derzeit schon vorweisen kann. Die bisherigen sogenannten Solver aus dem Strips Umfeld sind keine richtigen Solver sondern man kann mit ihnen nur Toy-Probleme lösen wie Towers of Hanoi. Also Aufgaben die aus sehr wenigen States bestehen und wo man die Möglichkeiten ähnlich wie bei einem TicTacToe Solver der Reihe nach durchprobieren kann. Versucht man jedoch echte Probleme zu lösen, beispielsweise einen Gait-Pattern für einen Walking Robot zu finden wird man bemerken, dass erstens der Suchraum viel zu groß ist und zweitens das Aufstellen aller Nebenbedingungen zu lange dauert.

3.2 LISP

LISP ist eine der ältesten Programmiersprachen überhaupt. Sie wurde von Anfang an für Aufgaben der Künstlichen Intelligenz entwickelt und wird bis heute für diese Aufgabe verwendet. Auch im ROS Betriebssystem ist ein LISP Interpreter selbstverständlich enthalten. Der Aufbau von LISP ist spartanisch, damit ist gemeint, dass die Sprache vor allem darauf ausgelegt wurde, leicht zu parsen zu sein und Dinge wie selbstmodifzierenden Code out of the Box unterstützt. Ebenfalls hervorragend ist LISP als Entwicklungsumgebung für komplett neue Programmiersprachen geeignet. Dies wird als Bootstrapping bezeichnet: man programmiert in LISP einen Parser, der wiederum eine neue Sprache parst und mit dieser erstellt man dann das Robot-Control-System.

Leider überwiegen die Nachteile von LISP, so dass die Sprache ab ungefähr den 1990'er Jahren nur noch selten eingesetzt wird. Das Hauptproblem von LISP ist, dass es deutlich schwerer zu programmieren ist als eine moderne objektorientierte Hochsprache. Ferner sind die LISP Compiler nicht besonders effizient. Und zu guter Letzt benötigt man komischerweise in der Künstlichen Intelligenz nur selten eine Meta-Programmiersprache. Generell kann man natürlich Roboter auch in LISP programmieren, besonders in älteren Büchern wird das sogar noch gelehrt. Nur, LISP kann man durch jede andere Hochsprache wie Java oder C++ bedenkenlos ersetzen. Diese Sprachen können all das was auch LISP kann und noch einiges mehr.

Gehen wir vielleicht auf einige konkrete Anwendungen ein aus dem Bereich der Robotik. Üblicherweise nimmt man an, dass eine Robot-Control-Software komplett aus künstlicher Intelligenz besteht, denn immerhin soll der Roboter Entscheidungen treffen, seine Umwelt wahrnehmen und Pläne entwickeln. Nur, in der Praxis ist der Bereich wo die eigentliche Künstliche Intelligenz zum Einsatz kommt ein sehr geringer Anteil des Gesamtprogramms. Die meisten Roboter die heutzutage in Betrieb sind, bestehen ähnlich wie ein Desktop Computer aus einem Betriebssystem, aus Modulen um die Hardware anzusteuern und aus Anwenderprogrammen. Auch eine Software wie OpenCV kann man getrost dem Bereich Betriebssystem / Anwenderprogramm zuordnen, OpenCV ist nicht zwangsläufig bereits als Künstliche Intelligenz zu bezeichnen. Und derartige Standardprogramme werden üblicherweise nicht in LISP programmiert sondern mit den Methoden welche in der angewandten Informatik üblich sind. Der Anteil eines Roboters wo man prinzipiell LISP mit Gewinn einsetzen könnten, also für den Behavior Planner oder für das Logical Reasoning ließe sich mit anderen Sprachen wie Lua und ähnliches häufig sehr viel leichter implementieren, so dass es heutzutage keine Notwendigkeit mehr gibt für ein Programm was komplett in LISP erstellt wurde.

Das war nicht immer so: in den 1980'er Jahren ist man einen anderen Weg gegangen. Damals hieß die Devise dass man selbst normale Betriebssysteme in LISP programmiert hat. Gemeint sind natürlich die dezidierten Workstations aus dieser Zeit welche von der Firma Symbolics entwickelt wurden.[3]

4 Vollautonome Systeme

Künstliche Intelligenz wird normalerweise mit Autonomie gleichgesetzt. Das Ziel lautet, dass Systeme ohne Hilfe von außen eine Aufgabe ausführen. Sie sollen möglichst eigenständig Entscheidungen treffen und sich selber verbessern. Vorbild für diese Vision einer Maschine sind mechanische Automaten, die einmal eingeschaltet unermüdlich ihren Dienst verrichten. Bis heute grenzt sich die Robotik strikt ab von nicht-autonomen Systemen wie ferngesteuerten Autos. Diese werden nicht als Roboter sondern als Spielzeug bezeichnet während die richtige Robotik ohne Fernsteuerung funktioniert. Ja mehr noch, eine Fernsteuerung gilt sogar als Betrug, wo also jemand vorgibt einen autonomen Roboter gebaut zu haben, der in Wahrheit jedoch von einem Menschen heimlich kontrolliert wird.

Leider gibt es mit diesem Ideal ein Problem: es ist nicht realisierbar. Die meisten autonomen Roboter zeichnen sich dadurch aus, dass die Maschine eben nicht in der Lage ist, die gestellte Aufgabe eigenständig zu lösen. Auch die eingebaute Lernkomponente funktioniert nicht. Der Grund dafür lautet,

dass zwar das Ziel vorhanden ist, derartige Systeme zu bauen, es aber keine Technologie gibt mit der man es realisieren könnte. Denn wie bitteschön soll eine Turing-Maschine aussehen, die eigenständige Entscheidungen trifft und sich noch dazu bei auftretenden Schwierigkeiten selber verbessern kann?

Die Vorstellung man könnte autonome Systeme bauen hat weniger etwas mit einem tieferen Verständnis der Robotik zu tun, sondern ist Ausdruck eines kulturellen Setting mit dem Robotik-Ingenieure ihre eigene Aufgabe definieren. Üblicherweise gehen sie davon aus, dass sie zu einem vorhandenen Roboter gerufen werden, dort dann kleinere Modifikationen ausführen, den Ort des Geschehens verlassen und der Roboter dann eigenständig weiterläuft. Der Robotik-Bauer möchte also einmalig Zeit und Energie investieren und irgendwann sich dann von dem Roboter wieder entfernen weil er erfolgreich war. Wer hingegen neben dem Roboter steht oder ihn sogar permanent fernsteuert wird nicht als Robotik-Ingenieur sondern als Anfänger wahrgenommen, als jemand der den "Kniff" noch nicht herausgefunden hat und der Zielstellung "autonomes System" nicht gewachsen ist. Das Problem mit diesem Idealbild ist, dass es faktisch nicht erreichbar ist. Jedenfalls nicht wenn der Roboter auf einer halbwegs realistischen Aufgabe eingesetzt werden soll. Bei sehr einfachen Aufgaben wie Computerschach oder "Towers of Hanoi" ist es wohl möglich echte autonome Systeme zu konstruieren, die einmal eingeschaltet perfekt funktionieren. Sobald man sich jedoch halbwegs komplexe Aufgaben heraussucht wie selbstfahrende Autos oder Dexterous Grasping sind derart robuste Systeme die fehlerfrei und ohne Eingriff von Außerhalb laufen ein Ding der Unmöglichkeit.

Dass dennoch an diesem Ziel festgehalten wird hat etwas mit dem grenzenlosen Optimismus innerhalb der Künstlichen Intelligenz zu tun. Man glaubt, dass so schwer die Aufgabe nicht sein kann, echte Roboter zu konstruieren und das man irgendwie die Schwierigkeiten schon überwinden könne. Schaut man sich jedoch einmal die sehr lange Geschichte der Automaten und Androiden an wird man sehen, dass genau an dieser Frage die Grenze verläuft zwischen Wissenschaft und Märchen. Nur im Märchen funktionieren Roboter vollautonom und ohne Eingriff von Außen. In der Realität hingegen gibt es Bugs, falsch programmierte Software und teilautonome Systeme.

> "Die Unterscheidung in teilautonome und (voll)autonome Roboter betrifft den Grad an Autonomie des Roboters. Ein teilautonomer Roboter kann nicht alle seine Aufgaben gänzlich unabhängig lösen, sondern ist auf externe Unterstützung angewiesen." [9]

4.1 Selbstlernende Systeme

Eine besondere Spielart vollautonomer Systeme sind lernfähige Maschinen. Diese sind nicht nur in der Lage, eine Aufgabe eigenständig auszuführen sondern können sich zusätzlich an neue / bisher unbekannte Aufgaben anpassen. Konkret realisiert werden solche Systeme mit Hilfe von stochastischen Gleichungen wie Gausschen Kernel, Support Vector Machines und neuronalen Netzen. Aber auch Artificial Life bzw. Künstliches Leben kann in diese Kategorie eingeordnet werden. Viele sagen, dass selbstlernende Systeme den eigentlichen Kern der künstlichen Intelligenz ausmachen. Es geht nicht nur darum, ein System zu programmieren sondern es geht um Meta-Algorithmen, mit deren Hilfe die Systeme sich von allein programmieren.

Obwohl das zunächst sehr neu und aufregend klingt ist der Ansatz sehr alt. Ray Solomonoff hat schon in den 1950'er Jahren dazu publiziert und schließt nahtlos an den klassischen Kybernetik-Begriff von Norbert Wiener an. Solomonoff hat Feedback Systeme konstruiert, also universelle Problemlöser, die in Richtung Artificial General Intelligence gehen.[12] Warum selbstlernende Systeme nicht funktionieren lässt sich anhand von Gödelnummern und Turing-Maschinen erläutern. Eine Gödelnummer bezeichnet ein konkretes ausführbares Computerprogramm. Wenn man die Gödelnummer hochzählt erhält man nacheinander alle möglichen Programme, die ein Computer verarbeiten kann. So ähnlich wie das beim Busy-Beaver Wettbewerb angestrebt wird. Selbstlernende Systeme versuchen nun, diesen Raum der möglichen Programme durch weitere Algorithmen einzugrenzen. Also den Raum der Gödelnummern auf ähnliche Weise zu betrachten als wenn man den kürzesten Weg durch ein Labyrinth plant. Üblicherweise werden dafür Heuristiken eingesetzt. Die Schwierigkeit und der Grund warum derartige Verfahren scheitern hat damit zu tun, dass der Raum aller möglichen Programme extrem umfangreich ist. In der Theorie kann man diesen absuchen in der Praxis findet die Suche auf konkreter Hardware statt, die im Zweifel sehr langsam ist.

Das hat dazu geführt, dass seit den 1950'er automatisches Programmieren und selbstlernende Systeme im Bereich der theoretischen Informatik ungeheuer populär wurden, aber ihr praktischer Erfolg bisher ausblieb. Insbesondere in der Robotik ist diese Entwicklung vorhanden. Auf dem Papier kann man mindestens seit 50 Jahren selbstlernende vollautonome Roboter programmieren, die "mit geometrischer Geschwindigkeit" lernen. In der Realität sind solche Roboter bei Wettbewerben wie Robocup, Micromouse oder Starcraft AI bisher noch nicht gesichtet worden. Insofern sind selbstlernende Systeme ein klassisches Antipattern. Es ist wichtig sich damit auseinanderzusetzen um dieses Pattern bei der Realisierung eigener Projekte zu vermeiden.

5 Hardware in-the-loop

Bis heute versteht sich die Robotik als Hardware-zentriertes Fachgebiet. Damit ist gemeint, dass man nicht konventionelle Apparate und Anlagen konstruiert sondern eben Roboter. Also eine CNC Maschine die um einen Microcontroller ergänzt wurde zu einem intelligentem System. Infolge dessen geht es bei der Robotik vorwiegend um Servo-Motoren, 110 Volt Systeme, raffinierte Sensoren und Akkus mit langer Laufzeit – so die These. Genau diese Hardware-zentrierte Sichtweise hat dafür gesorgt hat, dass die Robotik seit Jahrzehnten auf der Stelle steht. Weil Roboter eben keine normale Maschinen sind, die mit ein wenig Software erweitert wurden, sondern Roboter sind überwiegend Software-zentrierte Systeme, bei dem die Hardware ein Add-on ist.

Der Grund, warum Hardware in der Robotik noch immer so wichtig ist hat damit zu tun, dass man Hardware sehr leicht verstehen kann und Probleme dort keine wirklichen Probleme sind. Nehmen wir mal an, bei einem Roboter ist ein Radlager gebrochen und muss ersetzt werden. Wie anspruchsvoll ist diese Aufgabe? Sehr gering, im Grunde handelt es sich dabei um ein Mechanik Problem bei dem es nicht nur unzählige Ersatzteile gibt sondern auch sehr viel Literatur wo drinsteht wie man das Rad erneuert. Genau genommen ist es kein Problem im Sinne einer wissenschaftlichen Betrachtung. Dennoch finden sich in vielen schlechten Roboterbüchern seitenlangen Ausführungen über die mechanischen Komponenten eines Roboters, so als wäre es bedeutsam welche Elektronik verbaut ist, wie ein Elektromotor funktioniert oder wie ein Getriebe funktioniert.

Wenn man jedoch umfangreich die Hardware erläutert bleibt keine Zeit mehr sich der Software zuzuwenden. Die wird dann vernachlässigt und entweder gar nicht thematisiert oder im Anhang mit einigen Programmbeispielen erläutert. Die logische Folge von dieser Prioritätensetzung ist, dass der Roboter am Ende eher einem RC Car gleicht, was perfekt auf der Straße liegt aber über keinerlei Software verfügt. Genau genommen ist es also kein Roboter sondern ein Auto, ein Bagger oder was auch immer.

Nun mag es sicherlich spannend sein, elektrische Autos zu konstruieren oder servogesteuerte Gliedmaßen, nur hat das leider mit Robotik nicht das geringste zu tun. Robotik ist eine Disziplin die vorwiegend in der Simulation stattfindet und wo bevor die physische Hardware aufgebaut wird, der Algorithmus getestet sein will. Funktioniert der Roboter nochnichtmal in der virtuellen Realität wird er ganz sicher nicht in echt funktionieren.

Warum Hardware-gestützte Robotik so beliebt ist hat einen simplen Grund: unter einem Roboter stellen sich die meisten eine Maschine und keine Software vor. Ein Roboter ist etwas das man anfassen kann, und folglich wird er so gebaut, als wenn man bei einem Seifenkisten-Rennen teilnimmt, man nimmt vier Räder, einige Sensoren und fertig ist der Roboter. Wie jedoch reale Wettbewerbe zeigen, ist das nicht ausreichend, sondern im Grunde wird exakt so kein Roboter gebaut. Jedenfalls nicht wenn mit einer wissenschaftlichen Fragestellung an das Thema herangeht. Und zwar deshalb nicht, weil die Hardware Fehler enthalten kann. Selbst eine Aufteilung von 50% Hardware zu 50% Software ist eine falsche Prioritätensetzung, auch dort hätte die Hardware viel zu viel Gewicht. Wer in einem Roboterprojekt beteiligt ist und dort die meiste Zeit mit Motoren, elektrischen Widerständen und 3D Druck von Komponenten beschäftigt ist kann getrost davon ausgehen, dass es um alles mögliche geht nur nicht um Robotik.

Die Definition von Robotik ist unmissverständlich die Erforschung von Künstlicher Intelligenz und diese kann nur mit Hilfe von Software realisiert werden. Intelligenz aus der Hardware hingegen ist nicht möglich. Mechanische Maschinen führen mechanische Arbeit aus, während Turing-Maschine Software ausführen. Ein Roboter im engeren Sinne ist ein Avatar in einem Computerspiel, also eine körperlose Simulation.

5.1 Overengineering

Das Paradebeispiel in Sachen gescheitertes Robotik-Projekt ist zweifellos die Halle 54 von VW wo Mitte der 1980'er eine hochautomatisierte Fertigungsstraße in Betrieb genommen wurde die mit den allerneuesten Robotern der damaligen Zeit versehen war. [7] Die Idee war simpel: durch Roboter sollte menschliche Arbeit ersetzt werden, die Kosten sollten sinken und die Produktivität erhöht werden. Zunächst schien dieser Plan aufzugehen. Die neuen Anlagen wurden installiert, die Systeme von Grund auf neu programmiert. Als es jedoch daran ging, die Fertigungsstraße in Betrieb zu nehmen stellte man Fest dass es zu Störungen kam. Offenbar waren die Ingenieure mit der Technik komplett überfordert, die Roboter halfen nicht etwa die Produktion zu verbessern sondern erwiesen sich als heimliche Saboteure. Die vollautomatisiere Anlage verursachte höhere Kosten als angenommen, es kam zu hohen Stillstandzeiten. Gleichzeitig stieg die Arbeitsbelastung für die Mitarbeiter. Sie mussten jetzt neben ihrer eigentlichen Arbeit noch die neu installierten Roboter beaufsichtigen die zwar in der Theorie komplett logisch funktionierten weil sie programmgesteuert waren, faktisch jedoch unberechenbar waren. Man konnte nie genau abschätzen ob der Roboterarm als nächstes etwas hervorragend erledigt oder komplett versagt.

6 Brute-Force Solver

Funktionsfähige Künstliche Intelligenz aus dem Bereich der Spieltheorie funktioniert mit Hilfe von Brute-Force-Solvern. Eine Schachengine erstellt zuerst den Game-Tree und durchsucht diesen dann nach einer optimalen Lösung. Wenn der Computer ausreichend schnell ist lassen sich damit menschliche Spieler besiegen:

> "Brute-force programs win because they have a good mix of depth of search, and knowledge applied at leaf nodes." [1, page 6]

Mit geschickter hardware-naher Programmierung lassen sich Chess-Engines konstruieren die eine höhere ELO Punktzahl erzielen als Menschen und weil das so gut funktioniert, gilt Schach als Paradebeispiel für Künstliche Intelligenz. Nur leider hat das dazu geführt, dass Brute-Force manchmal als universale Problemlösungsstrategie verstanden wird und die Versuchung groß ist, auch Probleme der Robotik darüber zu lösen. Ein Beispiel aus dem Bereich Control-Theory könnte lauten, einen Roboter zu steuern, der einen Anhänger rückwärts einparken soll. Man betrachtet das ganze als Optimierungsproblem, definiert Randbedingungen und sucht dann im Gametree nach einer Lösung.[17] Nur, anders als beim Schach ist die praktische Realisierung schwieriger. Der Suchraum ist größer, die Nebenbedingungen lassen sich nicht so exakt definieren und wenn man das Problem nur geringfügig erweitert explodiert die Anzahl der Möglichkeiten geradezu.

Auf dem Papier sind Aufgaben der Control-Theory ähnlich gelagert wie die Mattsuche beim Schach: man hat ein konkretes Ziel was es zu erreichen gilt, es gibt Regeln die

zu beachten sind und die Lösung besteht darin, im Gametree nach dem Optimum zu suchen. Der Unterschied ist jedoch, dass sich Schachprobleme mit Computern der Gegenwart lösen lassen, Optimal Control Probleme der Robotik hingegen nicht. Man bräuchte dafür Computer die sehr viel schneller wären als heutige Modelle. Es ist also nicht möglich, das Optimierungsproblem in echt zu lösen, sondern man kann es nur theoretisch formulieren.

Anders formuliert, Brute-Force funktioniert nur bei sehr einfachen Problemen mit einem überschaubaren Suchraum. Aufgaben der Robotik zählen nicht dazu. Man kann zwar auch dort versuchen Brute-Force anzuwenden wird jedoch scheitern. Selbst mit besseren Heuristiken und Samplingbasierenden RRT Solvern ist der Suchraum viel zu groß. Damit ist gemeint, dass ein humanoider Roboter mit nur 20 DOF sehr viel mehr Aktionsmöglichkeiten bereitstellt als es noch bei Computerschach waren. Der Versuch, dennoch über Brute-Force Probleme der Robotik zu lösen ist dennoch weit verbreitet weil wie eingangs erwähnte diese Lösungsstrategie als zuverlässig gilt und oftmals die einzig bekannte ist.

Was ist Brute-Force? Bisher wurde der Begriff "Brute-force" zwar verwendet, allerdings wurde noch nicht erläutert was er bedeutet. Brute-Force, besser bekannt als rohe Gewalt, ist ein Lösungsverfahren durch Ausprobieren. Bekannt wurde es im Bereich der Kryptographie. Ein Beispiel: ein Zahlenschloss lässt sich durch die Ziffernfolge 243 öffnen. Wenn man diese Folge nicht kennt kann man das Schloss trotzdem öffnen wenn man aufsteigend mit 000, 001, 002 usw. alle Möglichkeiten der Reihe nach durchprobiert. Bei 3 Stellen gibt es: $10^3 = 1000$ Möglichkeiten. Die Geschwindigkeit mit der eine Lösung nach der Brute-Force-Methode erzielt wird hängt entscheidend von der Anzahl der Möglichkeiten ab. Bereits bei Computerschach ist diese Anzahl höher als 1000. Wenn der Spieler pro Zug die Auswahl aus einem von 12 Möglichkeiten hat gibt es nach 5 Halbzügen bereits $12^5 = 248832$ Möglichkeiten. Aber auch beim Travelling Salesman Problem spielt die Brute-Force Suche eine wichtige Rolle. Wenn man 10 Städte nacheinander besuchen will, kann man die Städte in unterschiedliche Reihenfolge absuchen, was mathematisch als Permutation bezeichnet wird.

Der entscheidende Nachteil von Brute-Force besteht darin, dass es für viele praktische Probleme nicht anwendbar ist. Fast alle Probleme die halbwegs kompliziert sind, besitzen einen großen Suchraum. Dieser ist umfasst deutlich mehr als 1 Million Möglichkeiten und überfordert die Rechenleistung heutiger Computer. Selbst der als effizient geltende RRT (Rapidly-exploring random tree) Algorithmus ist auf größeren Maps nicht mehr anwendbar. Mit Map ist eine Spielfläche gemeint die beispielsweise 2000x1000 Pixel beträgt. Wenn man auf dieser Kartengröße alle Möglichkeiten durchprobiert um von Punkt A nach Punkt B zu gelangen wird der Suchraum zu groß. Es führt konkret dazu, dass der Algorithmus zwar fehlerfrei die Lösung liefert, aber dafür mehrere Stunden benötigt.

Nichts desto trotz erfreut sich RRT gerade in der Robotik einer ungebrochenen Beliebtheit. RRT ist eines von diesen Antipattern was extrem häufig zum Einsatz kommt obwohl es damit nicht möglich ist, Greifplanung oder Wegeplanung auszuführen. Es ist deshalb so beliebt weil es naiv betrachtet eine ausgezeichnete Lösungsmethode darstellt. Wenn der Gripper eines Roboters einen Gegenstand greifen soll, probiert man einfach verschiedene Griffe der Reihe nach durch und löst das Problem als würde man die Ziffernkombination eines Zahlenschlosses herausfinden wollen. Leider wird dabei vergessen zu ermitteln dass es fast unendlich viele Möglichkeiten gibt, mit einem Gripper ein Objekt aufzunehmen und das das Durchprobieren (selbst wenn es über RRT-Bäume erfolgt) sehr lange dauern würde.

Um es genauer zu sagen ist nicht RRT der Flaschenhals sondern es sind die Probleme auf die der Algorithmus angewendet wird:

> "The search space of all possible manipulator configurations is growing exponentially with the degrees of freedom of the manipulator." [14, page 16]

Literatur

[1] Hans J Berliner. An examination of brute force intelligence. In *IJCAI*, pages 581–587, 1981.

[2] Avrim Blum and Ronald L Rivest. Training a 3-node neural network is np-complete. In *Advances in neural information processing systems*, pages 494–501, 1989.

[3] Hank Bromley, H Bromley, and Richard Lamson. *LISP Lore: A Guide to Programming the LISP Machine Second Edition*. Springer, 1987.

[4] Rodney A Brooks. Elephants don't play chess. *Robotics and autonomous systems*, 6(1):3–15, 1990.

[5] Hubert L Dreyfus. Alchemy and artificial intelligence, 1965.

[6] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[7] Martina Heßler. Die halle 54 bei volkswagen und die grenzen der automatisierung. *Zeithistorische Forschungen*, 11:56–76, 2014.

[8] John PA Ioannidis. Why most published research findings are false. *PLoS medicine*, 2(8):e124, 2005.

[9] Thomas Klute and B Reusch. *Konzeption und Aufbau eines teilautonomen Fußballroboters*. PhD thesis, Diplomarbeit, Lehrstuhl Informatik I, Universität Dortmund, 2001. http://www. robosoccer. de/fileadmin/dortmunddroids/documents/PG-Berichte/Diplomarbeiten/diplomarbeit_klute. pdf, 2001.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[11] David L Moreno, Carlos V Regueiro, Roberto Iglesias, and Senen Barro. Using prior knowledge to improve reinforcement learning in mobile robotics. *Proc. Towards Autonomous Robotics Systems. Univ. of Essex, UK*, 2004.

[12] Eray Özkural. Diverse consequences of algorithmic probability. In *Algorithmic Probability and Friends. Bayesian Prediction and Artificial Intelligence*, pages 285–298. Springer, 2013.

[13] A Pinti and J-C Grossetie. Comparing the performance of fifteen processors using the cps unit. *WIT Transactions on Information and Communication Technologies*, 11, 1970.

[14] Christian Potthast. Robot manipulation planning with rapidly exploring random trees in simulated environments. Diplomarbeit, Unpublished manuscript, Technische Universitaet Muenchen, http://robotics.usc.edu/ potthast/thesis.pdf, 2008.

[15] Rutger Ruizendaal. The potential of deep learning in marketing: insights from predicting conversion with deep learning. Master's thesis, University of Twente, 2017.

[16] John F Sowa. Why has artificial intelligence failed? and how can it succeed? *Computación y Sistemas*, 18(3):433–437, 2014.

[17] Petr Svestka and Jules Vleugels. Exact motion planning for tractor-trailer robots. In *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, volume 3, pages 2445–2450. IEEE, 1995.