FACTA UNIVERSITATIS (NIŠ)
Series: Electronics and Energetics vol. 12, No. 3 (1999), 35-46

UDC 681.58

EIN SIMULATOR ZUR EINGABE UND VISUALISIERUNG VON TRAJEKTORIEN VON MANIPULATORARMEN

Matthias Ortmann

Kurzfassung. In der vorliegenden Arbeit soll ein Simulator für eine Roboterarbeitszelle vorgestellt werden, der in einer späteren Ausbaustufe die adaptive Trajektorienplanung für Manipulatorarme mittels genetischer Algorithmen erlaubt. Es sollen insbesondere das Modul zur Darstellung der Manipulatorarm und ein Compiler vorgestellt werden, der die Eingabe der Trajektorien der Roboterarme durch Angabe von Stützstellen erlaubt. Für den genetischen Algorithmus, der aus den Stützstellen die kompletten Bahnen berechnet, sollen noch einige grundlegende Vorüberlegungen angefügt werden.

1. Einleitung

Seit einigen Jahrzehnten halten Roboter immer mehr Einzug in die industrielle Fertigung. Angefangen bei einzelnen Manipulatorarmen existieren heute bereits z.B. in der Autoproduktion ganze Werkhallen, in denen nur noch Roboter arbeiten. Eine Roboterzelle mit mehreren Manupulatorarmen, in der z.B. Schweißarbeiten durchgeführt werden, soll hier nun näher betrachtet werden. In ihr verrichten alle Roboter auf vorgegebenen Bahnen ihre Arbeit. Die Trajektorien dieser Arme wurden dem System, in dem jeder Roboter über die Lage des anderen informiert ist, meist in mühevoller Kleinarbeit durch Teach-In eingegeben. In dieser Konfiguration arbeitet das System nach erfolgter Programmierung ohne Fehler. Was geschieht jedoch, wenn in den Arbeitsraum eines oder mehrerer Roboterarme plötzlich ein Hindernis, ein abgerissener Draht usw. hineinragt? Der Arm wird unweigerlich mit dem Hindernis kollidieren, da er keine Information über seine Umgebung besitzt. Würde der Arbeitsraum jedoch überwacht, hier sei z.B. die Überwachung mit einer CCD-Kamera erwähnt, so könnte er durch vorzeitiges Abschalten eine Kollision vermeiden. Würde er zusätzlich noch über

Manuscript eingegangen Sep. 18, 1999.

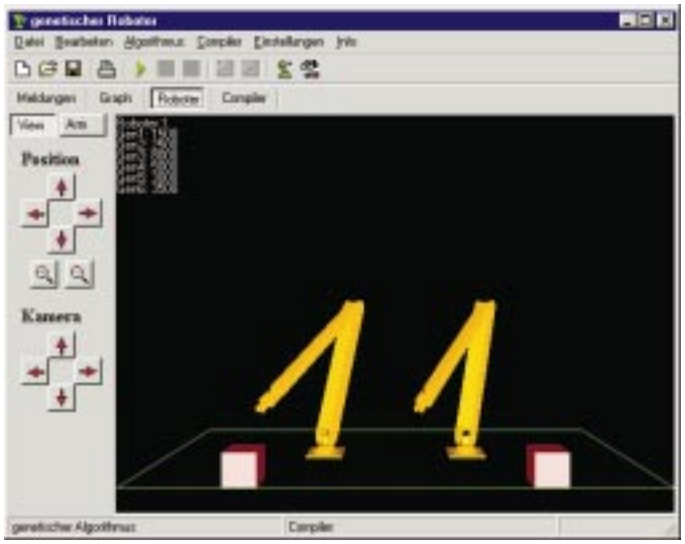
Dipl.-Ing. M. Ortmann ist wissenschaftlicher Mitarbeiter am Lehrstuhl für Datenverarbetung der Ruhr-Unversität Bochum, e-mail: ortmann@etdv.ruhr-uni-bochum.de .

einen Algorithmus verfügen, der eine adaptive Trajektorienplanung erlaubt, so könnte der Roboter seine Arbeit fortsetzen, sofern eine neue Bahn, die um das Hindernis herumführt, gefunden werden kann. Diese adaptive Bahnplanung könnte aber auch schon das am Anfang nötige Teach-In ersetzen, indem es aus vom Benutzer eingegebenen Stützpunkten, die auf jeden Fall angefahren werden müssen, die ganze Trajektorie berechnet. Hier ist dann auf Kollisionen mit dem Arbeitsraum und eventuell noch weiter vorhandener Manipulatorarme zu achten. In der vorliegenden Arbeit soll nun ein Simulator für eine Roboterarbeitszelle vorgestellt werden, der in einer späteren Ausbaustufe die adaptive Trajektorienplanung für Manipulatorarme mittels genetischer Algorithmen erlaubt.

2. Fenster des Programms

Bevor der Compiler und die graphische Darstellung der Manipulatorarme näher erläutert werden, soll an dieser Stelle erst einmal die Grundstruktur des Programms beschrieben werden. Es besteht aus einem Hauptfenster, das vier Unterfenstern enthält, die alle über TabControls erreichbar sind. Die Unterfenster füllen abzüglich des Bereiches für die TabControls den gesamten Clientbereich des Hauptfensters aus. Die vier Unterfenster verhalten sich also wie Registerkarten, die nach Bedarf an oberster Stelle angezeigt werden können. Es wurde dieses Fensterkonzept gewählt, damit sich auf dem Bildschirm nicht eine große Anzahl von Fenstern ansammelt, wie es bei MDI-Anwendungen normalerweise üblich ist, über die der Benutzer über kurz oder lang doch den Überblick verliert. Zusätzlich muß sich der Benutzer bei MDI-Anwendungen immer zum benötigten Fenster "durchhangeln", da meist das gesuchte Fenster durch andere teilweise oder ganz verdeckt wird. Hierzu müssen erst die verdeckenden Fenster verschoben oder minimiert werden.

Der Vorteil des MDI-Konzeptes besteht jedoch in der individuellen Größenanpassung eines Fensters. Bei dem Fensterkonzept des vorgestellten Simulators werden in den vier Fenstern immer nur die aktuellen und jeweils zusammengehörenden Daten in für alle Fenster gleicher Größe angezeigt. Zur Änderung der Fenstergröße muß immer die gesamte Applikation in der Größe verändert werden. Das Auswählen der Fenster geschieht einfach durch die Maus. Bild 1 a) zeigt nun das Programm nach dem Start. Es wird sofort das Roboterfenster angezeigt, in dem eine stilisierte Roboterzelle mit zwei Manipulatorarmen und Hindernissen zu sehen ist. Links neben dem Graphikbereich sind einige Buttons angeordnet, die die Bewegung der Kamera, also der Blickrichtung im Raum, erlauben. Es besteht die Möglichkeit, die Kamera nach rechts, links, oben und unten zu bewegen und zusätzlich in beliebige Richtung zu drehen. Das zur Verfügung gestellte Zoomen (vari-

a) 

b) 

Bild 1: a) Programm nach dem Start b) aktivierte Armsteuerung

abel nach vorne und hinten) erfolgt immer in die momentane Blickrichtung. Das Kameramodul teilt sich analog dem oben bereits beschriebenen Fensterkonzept den Platz mit dem Modul zur Steuerung der Roboterarme, das in Bild 1 b) zu sehen ist. Hierzu wird in der oberen DropDownComboBox des Steuerungsfensters der zu bewegende Roboter ausgewählt. Die einzelnen Gelenke des ausgewählten Roboters können dann jeweils mit den zur Verfügung gestellten Buttons auf und ab bewegt werden. Es werden maximal zehn Manipulatorarme vom Programm unterstützt.

Über die Registrierung "Compiler" des TabControls kann das Compilerfenster aufgerufen werden, das die Eingabe von Trajektorien in Schriftform erlaubt. Der Compiler und die graphische Ausgabe der Roboter wird weiter unten noch näher beschrieben.

Die beiden anderen Fenster dienen zur Ausgabe der Ergebnisse des genetischen Algorithmus. Hierzu steht auf der einen Seite unter dem Punkt Graph ein Fenster zur Verfügung, in das die im genetischen Algorithmus berechneten Fitneßwerte eingetragen werden können. Dies ist entweder als einzelner Punkt oder als Gerade, die vom vorherigen zum aktuellen Punkt verläuft, möglich. In das Meldungenfenster können Informationen zur Art und Einstellungen (Header) des genetischen Algorithmus und Informationen über die Populationsverläufe und das Ergebnisindividuum ausgegeben werden.

3. Aufbau eines Individuums

Bevor nun der Aufbau eines kompletten Individuums beschrieben wird, soll zuerst ein Koordinatensystem festgelegt werden.

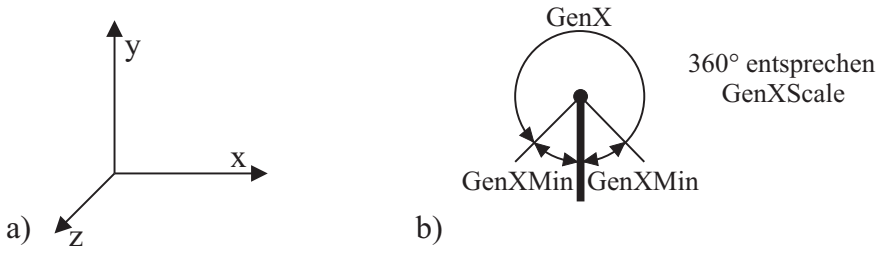


Bild 2: a) rechtshändiges Koordinatensystem b) Gelenkkoordinaten

Für die Berechnung und für die graphische Darstellung aus Benutzersicht wird ein übliches rechtshändiges Koordinatensystem gemäß Bild 2 a) verwendet. Dies bedeutet, daß zur Darstellung alle Koordinatenwerte umgerechnet werden müssen, da DirectX ein linkshändiges Koordinatensystem verwendet[1].

Bild 3 zeigt ein Beispiel für einen verwendeten Roboter. Es wird ohne Beschränkung der Allgemeinheit ein Roboter mit fünf Freiheitsgeraden - die Möglichkeit der Handöffnung wird also nicht als Freiheitsgrad gewertet - angenommen. Dieser Roboter wurde im Hinblick auf den genetischen Algorithmus gewählt, wobei dies auch hier keine Beschränkung der Allgemeinheit darstellt. Der Roboter verfügt also über fünf Achsen bzw. Gelenke (Joints) und drei Gelenkkörper (Links). Die Aufteilung der Gelenke gestaltet sich wie folgt:

Auf einer festen Basis befindet sich ein Doppelgelenk, das sowohl die Drehung (Gen1[2]) des gesamten Roboters und das Anheben (Gen2) des Armes erlaubt. Zwischen Unter- und Oberarm befindet sich ein Knickgelenk (Gen3). Als Handgelenk wird wiederum ein Doppelgelenk eingesetzt, das die Drehung (Gen4) und die Neigung (Gen5) der Hand erlaubt. Gen6 beschreibt die Öffnung der Hand bzw. der als Finger dienenden Greifbacken. Alle Rotationsgelenke benötigen Max-/Min-Anschläge, da die Gelenke aufgrund von

[1]Bei diesem Koordinatensystem bleiben die Richtungen der $x-$ und $y-$Achse identisch. Die Richtung für die $z-$Achse ist jedoch ivertiert.

[2]Die Winkel werden mit Rücksicht auf den genetischen Algorithmus als Gene bezeichnet, wie weiter unten noch näher zum Ausdruck gebracht wird.

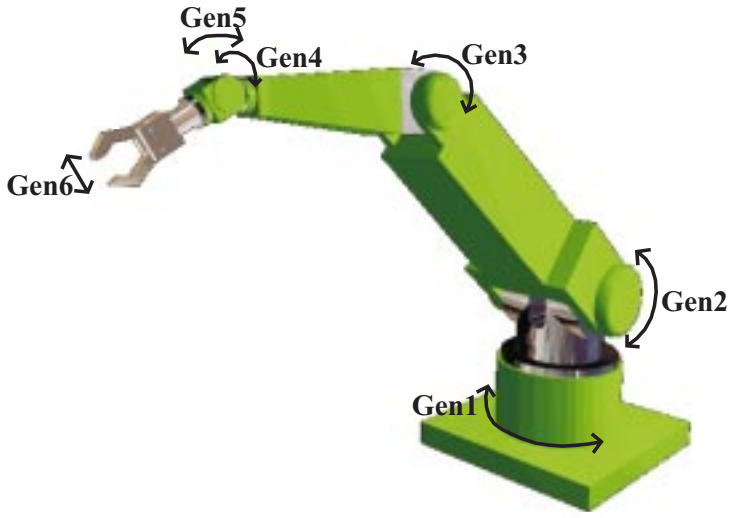


Bild 3: Koordinatenlage des verwendeten Roboters

mechanischen Widerständen und spätestens aufgrund von Kabeln, die zur Versorgung der Motoren dienen, nicht um einen beliebigen Winkel gedreht werden können. Bild 2 b) gibt Aufschluß über entsprechende Werte. Es wird ohne Beschränkung der Allgemeinheit angenommen, daß sich jedes Gelenk maximal um 360° drehen kann. Dieser Bereich wird durch die Variable GenXScale[3] in äquidistante Schritte unterteilt. Der Minimumanschlag wird durch die Angabe der Variablen GenXMin und der Maximalanschlag durch die Stellung GenXScale-2*GenXMin bestimmt. Der eigentliche Winkelwert liegt nun zwischen diesen Werten und beginnt bei GenXMin mit dem Wert Null. Das angesprochene Prinzip wird für jedes Gelenk eingesetzt, wobei für jedes Gelenk andere Scale- und Min-Werte angesetzt werden können.

Aus der Beschreibung der Koordinatenlage wird ersichtlich, daß die Stellung des Roboters zu jedem Zeitpunkt durch die Werte Gen1, Gen2, Gen3, Gen4, Gen5 und Gen6, die die Winkel angeben, eindeutig beschrieben werden kann. Um auch die Zeit, zu der sich der Arm an einer bestimmten Stelle befinden soll, zu berücksichtigen, wird diesen sechs Werten noch die Angabe der Zeit zugefügt. Diese sieben Werte werden mit Blick auf den genetischen Algorithmus Chromosom genannt. Um eine ganze Trajektorie, die wiederum

[3]Das X steht für eines der fünf rotatorischen Gelenke, da das beschriebene Prinzip auf alle Gelenke in gleicher Form engewandt wurde.

Chromosom1	Chromosom2	Chromosom3	...	ChomosomN
Gen1	Gen1	Gen1		Gen1
Gen2	Gen2	Gen2		Gen2
Gen3	Gen3	Gen3		Gen3
Gen4	Gen4	Gen4		Gen4
Gen5	Gen5	Gen5		Gen5
Gen6	Gen5	Gen6		Gen6
Zeit	Zeit	Zeit		Zeit

Bild 4: Zu einem Individuum zusammengefügte Chromosomen

mit Blick auf den genetischen Algorithmus Individuum genannt werden soll, zu erhalten, müssen die einzelnen Chromosomen nur noch hintereinander gereiht werden. Bild 4 veranschaulicht ein Individuum. In diesem Bild bezeichnet der Index N die maximale Chromosomenanzahl eines Individuums, die im Gegensatz zu realen Lebewesen variabel ist. Es gilt $N \in \mathcal{N}$, wobei immer $N \geq 2$ gelten muß, da sonst kein Weg vorhanden wäre.

4. Aufbau der graphischen Darstellung

Die Darstellung der Roboterzelle und der in ihr enthaltenen Roboter und Hindernisse erfolgt mittels DirectX [1]. Das hierbei eingesetzte DelphiX [2] entspricht dem üblichen Delphi-Komponentenkonzept und erlaubt ein relativ einfaches Ansprechen der DirectX-Funktionalität. Hierzu werden die beiden Komponenten TDXDraw und TDXTimer verwendet.

Die zuerst genannte Komponente stellt die eigentliche Zeichenfläche dar. Da die Darstellung im Fenstermodus erfolgt, erlaubt DirectX nur eine Bildschirmauflösung von 640×480 Bildpunkten in der Zeichenfläche. Der Vollbildmodus, der höhere Auflösungen erlaubt, wurde nicht gewählt, da die Buttons zur Bewegung der Kamera und der Roboter dann verdeckt wären.

Zur Darstellung der Roboterarme und der Umgebung wird das $3D-$Interface von DirectX genutzt. Es erlaubt die Erzeugung von dreidimensionalen Objekten, die dann nach dem Rendern, es stehen verschiedene Rendermöglichkeiten zur Verfügung, im Fenster dargestellt werden. Die Manipulatorarme werden nicht durch Angabe von Polygonen erzeugt, sondern aus Dateien geladen. Dies hat den Vorteil, daß sie mit $3D-$Zeichenprogrammen einfach erzeugt werden können. Hierbei ist jedoch meist noch eine Konvertierung des Dateiformates in das $*.x$ Format von DirectX nötig. Der Manipulatorarm wird nun linkweise gezeichnet und jeweils in einer Datei gespeichert. Diese Dateien werden beim Generieren des Roboters in die einzelnen Frames, die von DirectX zur Verfügung gestellt werden, geladen. Es existieren insgesamt pro Roboter sechs Frames (Base, Unterarm, Ober-

arm, Hand, Finger1 und Finger2). Diese Frames werden unter Zuhilfenahme der Konstanten für die Base-, Unterarm-, Oberarm- und Handlänge geeignet positioniert, so daß sich der Roboter ergibt. Um den Roboter in jedem Gelenk bewegen zu können, müßte jedes Frame einzeln positioniert und gedreht werden. Hierzu sind Translations- und Rotationsmatrizen nötig, deren Anwendung relativ viel Rechenzeit des Hauptprozessors in Anspruch nehmen würde. DirectX stellt jedoch Funktionen und Konzepte zur Verfügung, die diese Operationen einfach durchführen. Die Rechenarbeit wird hierbei meist noch in der Graphikkarte durchgeführt. Hierzu wird eine Abhängigkeitsstruktur der Frames angegeben. Der Unterarm wird als Kind der Basis angegeben, der Oberarm als Kind des Unterarms und so fort, bis die beiden Finger als Kinder der Hand deklariert sind. Diese Abhängigkeitsstruktur hat zur Folge, daß sich die Operation, die z.B. auf das Unterarmframe angewendet wird, auch mit allen Kind- und Kindeskindframes durchgeführt wird. Es wird also für die Rotation eines Frames dessen Position ermittelt. Dann wird das Frame als Kind eines anderen Frames deklariert, wodurch es dessen Rotation erhält. Danach wird die eigene Rotation durchgeführt und die am Anfang gespeicherte Position gesetzt. Werden diese Operationen für alle Gelenke durchgeführt, so kann der Manipulatorarm durch Drücken der entsprechenden Tasten bewegt werden.

An dieser Stelle stellt sich die Frage, wie das entsprechende Zeitraster für die Bilddarstellung und die Aktualisierung der Roboterposition erzeugt werden kann. Hierzu dient die bereits oben erwähnte Timerkomponente des Typs TDXTimer. Durch sie werden beide Operationen realisiert. Dies ist möglich, wenn die Bildwiederholrate auf 25 Frames pro Sekunde eingestellt wird, was für den Benutzer eine flimmerfreie Darstellung erlaubt und zusätzlich gerade dem Robotertakt von 40ms entspricht. Durch den Abschluß der Timer-Interrupt-Routine durch die Funktion Flip wird eine Synchronisation auf den Elektronenstrahl der Bildröhre durchgeführt, so daß eine flickerfreie Darstellung der Bilder möglich ist. Um den Verlauf der Roboterarme im verlangsamten Modus betrachten zu können, besteht die Möglichkeit der Zeitlupe. Hierbei wird die Bildwiederholfrequenz und somit die Einheit zur Aktualisierung der Roboterposition auf ein Bild pro Sekunde eingestellt.

Zur Simulation einer Roboterarbeitszelle fehlt nun nur noch die Umgebung der Roboter. Hier wird ohne Beschränkung der Allgemeinheit für jedes Stück der Umgebung bzw. jedes Hindernis eine rechtwinklige Box angenommen. Aus diesen Boxen lassen sich dann die Hindernisse zusammenfügen. Hierzu steht im Programm ein Editor zur Verfügung, der in Bild 5 a) gezeigt

a) b)

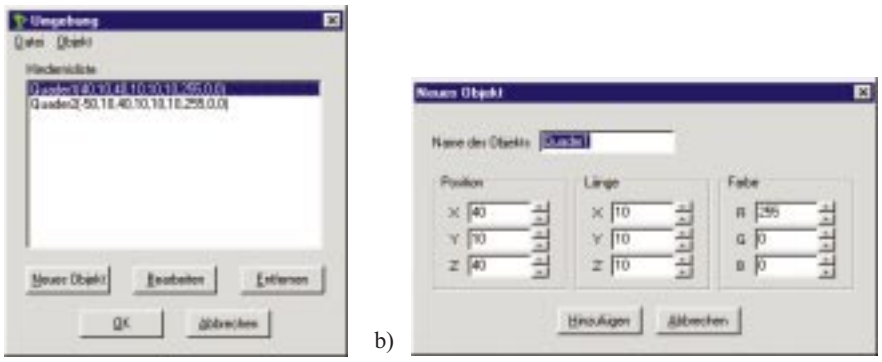


Bild 5: a) Editor zur Hinderniserzeugung b) Dialog zur Hinderniseingabe

ist. Von hier besteht die Möglichkeit, neue Objekte einzugeben oder alte zu bearbeiten bzw. zu entfernen. Sollen neue Objekte erzeugt bzw. alte bearbeitet werden, so wird ein Dialog gemäß Bild 5 b) aufgerufen, der im ersten Fall leer oder im zweiten Fall die zu editierenden Werte enthält. Die gesamte Scenerie kann auch abgespeichert werden, damit sie nicht immer neu eingegeben werden muß. Die Angabe über Lage und Größe der Objekte wird in einer entsprechenden Struktur gespeichert und steht dem genetischen Algorithmus zur Kollisionsdetektion zur Verfügung.

5. Der Compiler

Bisher ließen sich die Position und die Winkel der Manipulatorarme nur manuell durch Drücken der entsprechenden Taste verändern. Wünschenswert wäre es nun, dem Roboterarm mehrere Punkte mitteilen zu können, die er dann später in einer Simulation anfahren kann. Hierfür sind in der Industrie zwei Verfahren üblich. Auf der einen Seite können die Trajektorien dem Roboter durch Teach-In mitgeteilt werden. Hierbei wird der Manipulatorarm an eine Position gefahren, der Punkt gespeichert und dann der nächste Punkt angefahren und so fort. Bei diesem Verfahren dauert die Eingabe eines Weges jedoch relativ lange, und das Robotersystem muß schon real vorhanden sein. Auf der anderen Seite steht die Off-Line-Programmierung. In diesem Fall werden dem Robotersystem die anzufahrenden Punkte meist als Programmtext mitgeteilt. Die hierin angegebenen Werte bilden dann eine Trajektorie. Das zuletzt beschriebene Verfahren soll auch in dem vorliegenden Simulator zur Anwendung kommen. Es wurde hierzu ein Compiler implementiert, der die Eingabe von Trajektorien für bis zu zehn Roboter erlaubt. Die Grundstruktur für einen Roboter gestaltet

sich wie folgt:

Procedure *Prozedurname(Roboternummer, x-Position, y-Position, z-Position)*;
Interpolation:=an bzw. aus;
Begin
{Eingabe der anzufahrenden Punkte}
End;

Jede Roboter wird als Prozedur angegeben. Er erhält durch den Prozedurnamen einen eindeutigen Namen. In der Parameterliste muß nun der gewählte Robotertyp durch Angabe der Roboternummer und die Roboterposition durch die Werte x-Position, y-Position und z-Position bestimmt werden. Die anzufahrenden Punkte werden jeweils durch die Struktur

Punkt(*Gen1, Gen2, Gen3, Gen4, Gen5, Gen6, Zeit)*;

bestimmt. Es ist bei der Programmierung darauf zu achten, daß mindestens zwei Punkte angegeben werden, da sonst bei der Compilierung ein Fehler gemeldet wird, da keine Trajektorie berechnet werden kann. Die einzelnen Gene stehen für die Winkel des Roboterarms, wie sie schon weiter oben eingeführt worden sind. Der Parameter Zeit gibt bezogen auf den Startpunkt die Zeit an, zu der sich der Manipulatorarm bei der Simulation am angegebenen Punkt befindet.

Wird nun der eingegebene Programmtext compiliert, so wird zunächst der gesamte Text zeilenweise auf Fehler überprüft. Hierbei werden die angegebenen Punkte in einem dynamischen Array gespeichert. Es wird sofort überprüft, ob die angegebenen Winkel die maximal zulässigen Werte für den angegebenen Roboter überschreiten, oder ob die Differenz in der Zeitangabe zwischen zwei Punkten nicht ein Vielfaches von 40 ms, was dem üblichen Robotertakt entspricht, beträgt. Ist der eingegebene Programmtext syntaktisch richtig, so werden das Genetic-Individuum und die Trajektorie für jeden Roboter berechnet. Im Fall der Trajektorie werden gemäß den angegebenen Zeitwerten zusätzliche Zwischenstellen generiert, die im Abstand von 40ms liegen. Danach werden die nötigen Winkelwerte durch lineare Interpolation in Gelenkkoordinaten aus dem im Compiler angegebenen Werten berechnet. Bei dem Genetic-Individuum gestaltet sich die Erzeugung etwas anders. Hier ist die Anzahl der Stellen, die zwischen zwei vorgegebenen Stützstellen liegen, immer gleich der Stützstellenanzahl, die im Dialog zur Einstellung des genetischen Algorithmus angegeben wurde. Die Zwischenwerte werden je nach Angabe der Interpolationsart für diesen Roboter entweder in Gelenkkoordinaten analog der Trajektorie interpoliert oder zufällig bestimmt. Hierbei liegen die Zufallswerte immer im Bereich zwischen den bei den Stützstellen angegebenen Werten. Diese Werte stehen nun für die Simulation zur Verfügung. Es ist zu erwähnen, daß der Simulator immer

an der ersten Stelle beginnt und nach abgefahrener Trajektorie wieder den ersten Punkt anfährt. Hierbei kann es zu Sprüngen in der Armbewegung kommen, die vom Programmierer gegebenenfalls zu beheben sind.

6. Der genetischen Algorithmus

Die Theorie der genetischen Algorithmen wurde Mitte der sechziger Jahre von zwei verschiedenen Gruppen unabhängig voneinander entwickelt. Auf der einen Seite standen Rechenberg und Schwefel für das europäische bzw. deutsche, auf der anderen Seite Holland und Goldberg für das amerikanische Lager. Die Intention war bei beiden Lagern gleich. Es sollte die genetische Entwicklung von Lebewesen im Rechner simuliert werden und hiermit wissenschaftliche Probleme gelöst werden. Doch schon hier zeigten sich Diskrepanzen, wie dies zu realisieren sei. Bei Holland und Goldberg und dem sich daraus entwickelnden Lager der genetischen Algorithmen steht die möglichst genaue Nachahmung der real vorkommenden Vererbungskonzepte und der Vorgänge in Zellen im Vordergrund. Bei Rechenberg und Schwefel war dies jedoch nicht der Fall. Hier ging es in erster Linie darum, ein vorhandenes Problem zu lösen und aus der Vererbungstheorie nur so viel zu übernehmen, wie es für die Problemstellung nötig war. Aus diesen Überlegungen haben sich dann die evolutionären Strategien entwickelt. Beide Lager existieren heute nebeneinander, wobei schon einige Annäherungen zwischen ihnen zu erkennen sind. Die Grundstruktur des Algorithmus ist jedoch bei beiden Lagern gleich. Folgende Algorithmusteile müssen immer durchlaufen werden:

Rekombination
Mutation
BerechneFitness
Selektion

Nachdem die Populationen erzeugt worden sind, kommt es zur Generierung von Kindern aus den Eltern. Hierbei können verschiedene Strategien verfolgt werden. Es kann z.B. eine eingeschlechtliche Vermehrung stattfinden, oder aus zwei Eltern z.B. durch Mittelwertbildung oder durch Crossing-Over ein Kind bzw. Kinder erzeugt werden. Dieser Vorgang wird als Rekombination bezeichnet. Die erzeugten Kinder werden nun mutiert. Dies bedeutet, daß auf jeden Wert einer zufällig ermittelten Genposition eines Chromosoms bzw. eines Individuums ein zufällig bestimmter Wert addiert oder von diesem subtrahiert wird. Von den so veränderten Individuen und von den Eltern wird nun die Fitneß berechnet. Dies bedeutet, daß berechnet wird, wie gut das Individuum eine spezielle Fehlerfunktion erfüllt.

Der sich ergebende Fitneßwert wird nun in der Selektion in Betracht gezogen. Zuerst wird geguckt, ob ein Individuum einen kleineren Fehlerwert als die vorgegebene Fehlerschranke besitzt. Ist dies der Fall, so ist der Algorithmus abgeschlossen, und ein Ergebnis ist gefunden. Ist diese Bedingung für kein Individuum erfüllt, so werden je nach Algorithmusart aus den Kindern oder aus den Eltern und den Kindern die besten Exemplare herausgesucht und hieraus eine neue Population generiert. Die erzeugte Population ist nun wieder Ausgangspunkt für einen Durchlauf der oben beschriebenen Struktur.

Obige Algorithmusstruktur wird auch für die Trajektorienberechnung im vorliegenden Simulator benutzt. Bild 6 zeigt den Verlauf einer Wegberechnung. Hierbei sind die über eine Population gemittelten Fitneßwerte über der laufenden Populationsnummer aufgetragen.

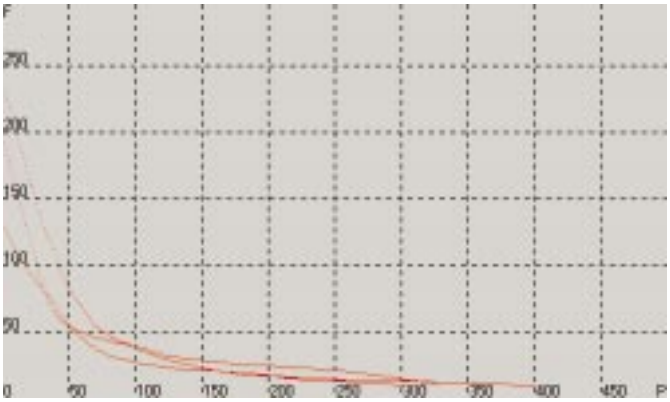


Bild 6: Verlauf der Fitneßwerte

Die Stützstellen sind im Punktmodus in das Diagramm eingetragen. Es sind drei Kurven zu erkennen, da im Compiler vier Stützstellen angegeben worden sind, wobei jeder der drei Optimierungsbereiche zehn Stützstellen umfaßt. Das Optimierungskriterium besteht aus der Länge der Teiltrajektorie, die die Handwurzel bei der Bewegung im $xyz-$Koordinatensystem zurücklegt, bezogen auf die minimal mögliche Länge, die sich aus den gegebenen Stützpunkten einfach berechnen läßt. Hierbei wird nicht berücksichtigt, ob die Abstände der einzelnen Stützpunkte auch äquidistant sind.

Es handelt sich um einen (5,25)-Algorithmus. Dies bedeutet, daß aus 5 Eltern ungeschlechtlich 25 Kinder gebildet werden. Aus diesen Kindern werden dann wieder 5 Eltern für die neue Population generiert. Die Mutationswahrscheinlichkeit beträgt für die Kinder 20 Prozent. Die Eltern werden nicht mutiert, da dies keinen Sinn hätte, da die neue Population nur aus

den Kindern und nicht aus den Eltern und den Kindern gebildet wird. Der Algorithmus wird beendet, wenn die Fitneß eines Individuums unter dem vorgegebenen Abbruchwert von 10 liegt.

Danksagungen

Ich danke Herrn Prof.-Dr.-Ing. Dr. E.h. W. Weber für seine Unterstützung und seine Anregungen bei der Realisierung des Simulators.

LITERATUR

1. DirectX-Onlinedokumentation: *Version 6.0.*
2. Herrn Hori: *DelphiX für Delphi 4.0.* `http://www.ingjapan.ne.jp/hori/programe.html`
3. Bargen, Bradley, Donnelly, Peter: *Inside DirectX.* Microsoft Press,1998, ISBN 3-86063436-4
4. Schöneburg, Eberhard, Heinzmann, Frank, Feddersen, Sven: *Genetische Algorithmen und Evolutionsstrategien - Eine Einführung in Theorie und Praxis der simulierten Evolution.* Addison-Wesley, 1996, ISBN 3-89319-493-2
5. Heistermann, Jochen: *Genetische Algorithmen - Theorie und Praxis evolutionärer Optimierung.* B. G. Teubner Verlagsgesellschaft, 1994, ISBN 3-8154-2057-1
6. Bäck, Thomas: *Evolutionary Algorithms in Theory and Practice - Evolution Strategies, Evolutionary Programming, Genetic Algorithms.* Oxford University Press, 1996, ISBN 0-19-509971-0