# Abschlussbericht des Team $m^2$ zu dem Projektpraktikum Robotik und Automation: Künstliche Intelligenz

Marius Krusen und F. J. Michael Werner

***Abstract*—Brauchen wir ein Abstrakt?**

## I. Aufgabenstellung

ZIEL des Projektpraktikums ist es, eine funktionsfähige Künstliche Intelligenz (KI) für das Spiel rAIcer zu entwickeln. In dem Spiel können bis zu drei Spieler jeweils eine Figur über mehrere Runden auf einer Rundstrecke steuern. Die Figuren werden nur durch "Kraftimpulse" in den Richtungen oben, unten, links und rechts gesteuert. In einem abschließenden Turnier, soll die KI in der Lage sein, auf bekannten sowie unbekannten Strecken gegen KIs anderer Teams anzutreten.

## II. Lösungsansatz

Das Team $m^2$ verwendet für die Entwicklung der KI den NeuroEvolution of Augmenting Topologies (NEAT)-Algorithmus. Weiterhin wird das gegebene Problem darauf reduziert, dass die KI eine Folge von Kontrollpunkten auf der Strecke abfahren muss. Für die Berechnung der Kontrollpunkte ist eine Streckenerkennung notwendig. Für die Eingabe, der durch den NEAT-Algorithmus generierten neuronalen Netze, werden Merkmale berechnet, die unter anderem auf den Kontrollpunkten und der Position der Figur basieren. Die einzelnen Elemente des Lösungsansatzes werden nachfolgen im Detail vorgestellt.

### A. NEAT

In der Neuroevolution werden Neuronale Netze mit Hilfe von genetischen Algorithmen generiert. In [1] stellen Stanley et al. NEAT vor, der die Besonderheit hat, dass neben den Kantengewichten und Schwellenwerten, auch die Topologie des Netzes entwickelt wird. Die zentralen Bausteine des Algorithmus sind:

- Darstellung eines Netzes als Genom
- Verwendung von Historie-Markern
- Verwendung von Spezies
- Minimierung der Dimensionalität

In jeder Generation liegt eine Menge von Genomen (Population) vor. Durch eine Fitnessfunktion, wird die Performance der einzelnen Genome bestimmt. Anschließend wird basierend auf den stärkeren Genomen mithilfe von Mutation und Kreuzungen neue Genome für die nächste Generation erzeugt. Für die Implementierung des Projektes wurde das Python-Package NEAT-Python [2] verwendet.

*1) Genetische Darstellung und Mutation:* Jedes Genom besteht aus einer Liste von Kanten-Genen. Diese referieren jeweils auf zwei Knoten-Gene, die deren Eingangs- und Ausgangsknoten darstellen. Des Weiteren enthalten die Kanten-Gene Informationen über ihre Kantengewichte, ein Aktivierungsbit und eine Innovationsnummer.

Mutationen in NEAT können die Kantengewichte und die Netzstruktur verändern. Die Mutation der Kantengewichte erfolgt über Entscheidung, basierend auf Wahrscheinlichkeiten, ob ein Knoten mutiert oder nicht. Die Mutation der Struktur kann auf zwei Weisen erfolgen. Zum einen kann ein einzelnen Kanten-Gen zwischen zwei bisher nicht verbunden Knoten-Genen hinzugefügt werden. Zum anderen kann ein Kanten-Gen durch zwei neue Kanten-Gene und ein neues Knoten-Gen ersetzt werden. Das alte Kanten-Gen wird dabei deaktiviert. Dadurch wird ein neuer Knoten in die Netzstruktur eingefügt.

*2) Historie-Marker und Kreuzungen:* Für die Kreuzung zweier Genome muss überprüft werden, welche Gene übereinstimmen. Dafür wird bei jedem neuen Gen eine globale Innovationsnummer erhöht und diesem Gen zugewiesen. Zwei Gene die den gleichen historischen Ursprung haben, also eine gleiche Innovationsnummer besitzen, repräsentieren die gleiche Struktur. Während der Kreuzung kann so leicht bestimmt werden, welche Gene in beiden Elterngenomen vorliegen und übernommen werden können. Gene die nicht in beiden Elterngenomen vorkommen, werden von dem Elterngenome mit dem höheren Fitnesswert vererbt.

*3) Spezies:* Durch das Hinzufügen einer neuen Struktur wird die Fitness anfänglich reduziert. Um eine neue

Innovation zu schützen und ihr Zeit zur Optimierung zu geben, unterteilt NEAT die Population in mehrere Spezies. Eine neue Innovation konkurriert erst mit den Genomen in der jeweiligen Spezies. Erst nach eine gewissen Zeit, wird eine Auslöschung durch einen Vergleich mit der restlichen Population möglich. Die Einteilung der Population in Spezies durch die Berechnung einer Distanz zwischen zwei Genomen $\delta$:

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \cdot \overline{W}, \tag{1}$$

mit der Anzahl überflüssiger Gene $E$, die Anzahl verschiedener Gene $D$ und der durchschnittlichen Differenz der Kantengewichte übereinstimmender Gene $\overline{W}$. Die Koeffizienten $c_1, c_2, c_3$ dienen zur Gewichtung der Faktoren und $N$, die Anzahl an Genen in dem größeren Genom, dient zu Normalisierung der Distanz. Jedes Genome wird mit einem zufällig ausgewählten Genome einer Spezies verglichen. Wird ein Schwellenwert $\delta_t$ unterschritten, wird das Genome der Spezies hinzugefügt.

*4) Minimierung der Dimensionalität:* NEAT sieht vor, dass die anfängliche Population aus Netzen besteht, die keine verdeckten Knoten hat. Dadurch das Innovationen geschützt werden, können somit möglichst kleine Netze generiert werden, da nur neue Strukturen hinzugefügt werden, wenn sie im Sinne der Fitnessfunktion Verbesserungen einbringen. Allerdings ermöglicht die NEAT-Implementierung aus [2], dass auch mit nicht leeren Neuronalen Netzen begonnen werden kann. Diese Funktion wurde auch in unseren Experimenten verwendet.

### B. Streckenerkennung und Kontrollpunkte

### C. Merkmalsberechnung

## III. Experimente und Ergebnisse

Für die Entwicklung unserer KI haben wir verschiedene Parameter unseres Aufbau ausprobiert. Diese Parameter werden nachfolgend vorgestellt. Anschließend werden die ausgewählte Versuche mit ihren Ergebnissen vorgestellt, die zu guten Ergebnissen geführt haben. Für alle Experimente gilt, das zum Abfahren der geforderten Runden jeweils 3 Minuten Zeit gewesen sind. Die verwendeten NEAT-Parameter sind in der Tabelle III zu finden. Zu dem wurden den initialen Population jeweils Kanten mit einer Wahrscheinlichkeit von 0,2 hinzugefügt.

### A. Parameter

*1) Anzahl Ausgangsknoten:* Wir haben die NEAT-Algorithmus auf zwei verschiedene Ansätze angewendet, wie die Ausgangsneuronen für die Steuerung der Figur interpretiert werden können. In dem ersten Ansatz gibt es 4 Ausgangsneuronen, wobei je ein Neuron für eine der Richtungen oben, unten, links oder rechts steht. Da für die Aktivierungsfunktion der Neurone die Sigmoid-Funktion verwendet wurde, kann für ein Ausgangsneuron $o_r$ folgende Interpretation angewendet werden:

- $o_r < 0.5$ : Taste $r$ nicht gedrückt
- $o_r \geq 0.5$ : Taste $r$ gedrückt.

Dabei stet $r$ für die vier verschiedenen möglichen Richtungen.

Für den zweiten Ansatz werden nur zwei Ausgangsneurone $o_h$ und $o_v$ benötigt. Für $o_h$ gilt dabei folgende Interpretation:

- $o_h \leq 0.25$: Taste rechts gedrückt
- $0.25 < o_h < 0.75$: weder Taste links noch Taste rechts gedrückt
- $o_h \geq 0.75$: Taste links gedrückt.

Für $o_v$ ist die Interpretation analog mit den Richtungen Unten und Oben.

*2) Strecke:* Die KIs wurden entweder ausschließlich auf der ersten Strecke des Spiels oder auf der dritten trainiert.

*3) Fitnessfunktion:* Wir haben zwei Arten von Fitnessfunktionen entwickelt. Beide basieren auf den Kontrollpunkten die bei der Streckenerkennung berechnet werden. Die erste Fitnessfunktion $f_1$ wird über die Anzahl der abgefahrenen Kontrollpunkte definiert:

$$f_1 = \#KP, \tag{2}$$

mit $\#KP$ als die Anzahl der abgefahrenen Kontrollpunkte. Die verwendeten Strecken 1 und 3 bestehen aus 48 beziehungsweise 52 Kontrollpunkten. Die zweite Fitnessfunktion $f_2$ erweitert $f_1$ wie folgt:

$$f_2 = \#KP + (180s - t) + (255 - s), \tag{3}$$

mit $t$ als die benötigte Zeit in Sekunden und $s$ als der erlittene Schaden. Dabei sind 180s und 255 die jeweils maximal möglichen Werte.

| Parameter | Wert |
|---|---|
| Populationsgröße | 45 |
| $P$ (Knoten hinzufügen) | 0,3 |
| $P$ (Knoten entfernen) | 0,1 |
| $P$ (Schwellwert-Mutation) | 0,8 |
| $P$ (Kante hinzufügen) | 0,75 |
| $P$ (Kante entfernen) | 0,3 |
| $P$ (Kantengewicht-Mutation) | 0,8 |
| Minimale Anzahl an Spezies | 2 |

TABLE I

Übersicht der verwendeten NEAT-Parameter.

| | 0 | 1 | 2 | 3 | 4 | Durchschnitt |
|---|---|---|---|---|---|---|
| 0 | 3:17 | 3:24 | 3:29 | 2:47 | 3:05 | - |
| | 255 | 255 | 255 | 255 | 255 | 255 |
| 1 | 2:02 | 2:23 | 2:31 | 2:17 | 2:18 | - |
| | 255 | 255 | 255 | 255 | 255 | 255 |
| 2 | 3:23 | 3:22 | 3:10 | 3:20 | 3:30 | 3:24 |
| | 160 | 76 | 56 | 146 | 162 | 136 |
| 3 | 3:11 | 3:20 | 3:19 | 3:13 | 3:16 | 3:17 |
| | 92 | 120 | 129 | 70 | 102 | 110,75 |
| 4 | 3:16 | 3:19 | 3:18 | 3:19 | 3:17 | 3:18 |
| | 116 | 131 | 144 | 174 | 117 | 141,25 |

TABLE II

Vergleich der Performanz der durch die verschiedenen Experimente generierten Netze. Die Zelle in Zeile $n$ und Spalte $m$ enthält die gefahrene Zeit und den erhaltenen Schaden des Netzes $n$ aus dem direkten Vergleich zu Netz $m$. Die letzte Spalte enthält die Durchschnittswerte aus den Duell mit den anderen Netzen, sofern das Duell überlebt wurde. Mit rot, grün und grau, sind jeweils die Niederlagen, Siege oder Unentschieden gekennzeichnet.

### *B. Experimente*

## IV. Zusammenfassung

## References


[1] K. O. Stanley and R. Miikkulainen, "Efficient evolution of neural network topologies," in *Proceedings of the Genetic and Evolutionary Computation Conference*, W. B. Langdon, E. Cantu-Paz, K. E. Mathias, R. Roy, D. Davis, R. Poli, K. Balakrishnan, V. Honavar, G. Rudolph, J. Wegener, L. Bull, M. A. Potter, and A. C. Schultz, Eds. Piscataway, NJ: San Francisco, CA: Morgan Kaufmann, 2002, pp. 1757–1762. [Online]. Available: http://nn.cs.utexas.edu/?stanley:cec02

[2] "Welcome to neat-python's documentation!" aufgerufen: 2018-09-25. [Online]. Available: https://neat-python.readthedocs.io/en/latest/