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 nachfolgend 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 werden 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 einzelnes 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 erfolgt durch die Berechnung einer Distanz zwischen zwei Genomen δ:

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

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 Genom wird mit einem zufällig ausgewählten Genom einer Spezies verglichen. Wird ein Schwellenwert δ_t unterschritten, wird das Genom der Spezies hinzugefügt.

4) Minimierung der Dimensionalität: NEAT sieht vor, dass die anfängliche Population aus Netzen besteht, die keine verdeckten Knoten hat. Dadurch dass Innovationen geschützt werden, können somit möglichst kleine Netze generiert werden, da nur neue Strukturen hinzugefügt werden, wenn sie im Sinne der Fitnessfunktion Verbesserungen einbringen.

A

B. Streckenerkennung und Kontrollpunkte

Die Streckenerkennung und die Berechnung einer Art Ideallinie oder Leitlinie, die durch eine endliche Anzahl an Kontrollpunkten repräsentiert wird, beruht im Wesentlichen auf zwei Algorithmen. Mit Hilfe der Watershed-Algorithmus wird zunächst eine grobe Leitlinie erzeugt, die einen "sicheren" Verlauf entlang der Mitte der Strecke liefert. Anschließend wird diese Leitlinie durch Verwendung eines Snake-Algorithmus, der überflüssige Richtungsänderungen eliminiert und insgesamt einen kürzeren, glatteren Kurvenverlauf erzeugt, verbessert.

1) Watershed-Algorithmus: Um zu Beginn überhaupt ein eindeutiges, klares Bild der Strecke zu bekommen, wird ein einfaches Schwellenwertverfahren eingesetzt, dass alle Pixel mit einem Grauwert unter 10 als schwarz und alle anderen als weiß interpretiert. Um kleine Artefakte, wie den schwarzen Rand der Figur, zu entfernen wird das resultierende Binärbild durch ein *Opening* (eine Erosion gefolgt von einer Dilatation) gefiltert. Anschließend werden mit Hilfe von OpenCV [3] die Konturen der Strecke gefunden, um den inneren Rand der

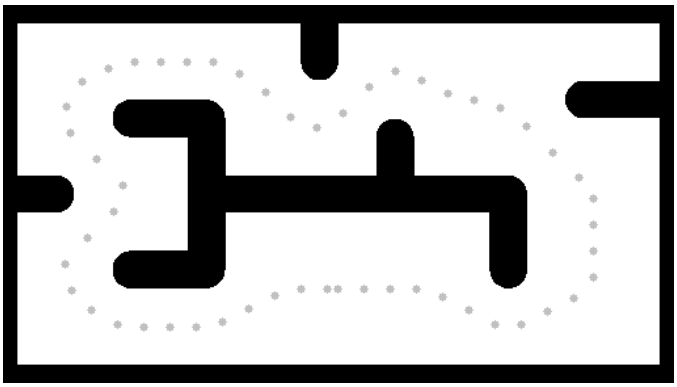

Fig. 1. Die resultierenden Kontrollpunkte nach Anwendung des Watershed-Algorithmus

Strecke vom äußeren Rand und möglichen Hindernissen auf der Strecke zu unterscheiden. Der innere Rand wird mit Label 1 markiert, der äußere Rand und Hindernisse mit Label 2. Die eigentliche Strecke erhält kein Label. Auf Basis dieser Labels wird der Watershed-Algorithmus auf einem leeren, einfarbigen Bild ausgeführt. Der Algorithmus wird nicht auf das Bild der Strecke angewendet, da ja nicht die Ränder der Strecke, sondern die Mitte der Strecke gefunden werden soll. Durch das *Fluten* des Bildes ausgehend von den Labels entsteht genau in der Mitte der Strecke eine Linie, an der Label 1 und Label 2 aufeinander treffen. Dies ist die erste grobe Leitlinie, von der in regelmäßigen Abständen ein Punkt als Kontrollpunkt für die weitere Berechnung ausgewählt wird (Bild 1).

2) Snake-Algorithmus: Bevor der Snake-Algorithmus die Leitlinie glättet, wird der Rand der Strecke (innen, außen und Hindernisse) um den Radius der Figur dilatiert, um zu verhindern, dass der Algorithmus Kontrollpunkte liefert, die für die Figur gar nicht erreichbar sind. Der Snake-Algorithmus (auch *Aktive Kontur* genannt) beruht auf einem zu minimierendem Energieterm, der sich normalerweise aus drei einzelnen Energietermen zusammensetzt:

$$E = \int \alpha(s) E_{cont}(s) + \beta(s) E_{curv}(s) + \gamma(s) E_{image} ds \tag{2}$$

Dabei entspricht s im diskreten Fall den Punkten der Snake und α, β und γ dienen der Gewichtung der drei Terme. Der Kontinuitätsterm E_{cont} dient hauptsächlich dazu, dass die Punkte der Snake einen gleichen und möglichst kurzen Abstand zueinander halten. Der Krümmungsterm E_{curv} sorgt für einen glatten statt eines zackigen Kurvenverlaufs. Der Bildterm E_{image} hingegen zieht die Snake zu Kanten auf dem Bild hin. Da in diesem Fall das Hauptziel eine kurze, glatte Leitlinie ist und eine zu starke Tendenz zum Rand der Strecke

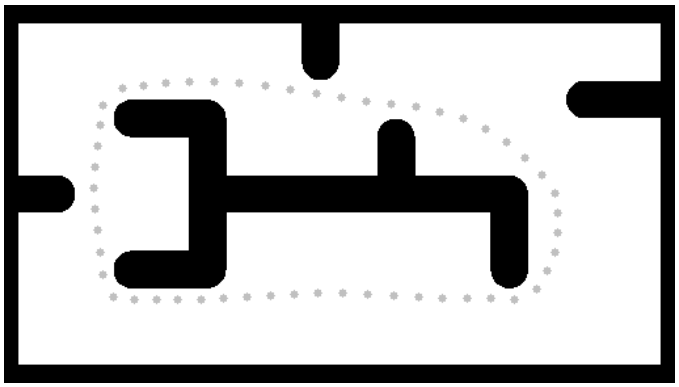

Fig. 2. Die resultierenden Kontrollpunkte nach Anwendung des Snake-Algorithmus

schädlich ist, wird auf den Bildterm verzichtet. Außerdem hat sich gezeigt, das eine gleichmäßige Gewichtung der ersten beiden Terme gute Ergebnisse liefert. Der zu minimierende Term vereinfacht sich also zu:

$$E = \int E_{cont}(s) + E_{curv}(s)ds \tag{3}$$

Um diese Energie zu minimieren wird iterativ vorgegangen. Jeder aus dem Watershed-Algorithmus hervorgegangene Kontrollpunkt wird nacheinander betrachtet und verbessert, indem der Punkt aus dessen Nachbarschaft gesucht wird, der die Gesamtenergie am stärksten verringert. Dabei werden nur Punkte in Betracht gezogen, die auch auf der Strecke liegen. Es hat sich gezeigt, dass nach etwa 10 Durchläufen eine zufriedenstellende Leitlinie resultiert (Bild 2).

C. Merkmalsberechnung

Um die Trainingsdauer kurz zu halten und eine möglichst streckenunabhängige KI zu erhalten, wird eine kleine Anzahl aussagekräftiger Merkmale verwendet. Dazu gehören die Geschwindigkeit der Figur, ihre Entfernung zum nächsten Hindernis in verschiedenen Richtungen, sowie ihre Entfernungen zu den nächsten Kontrollpunkten. Im Gegensatz zu der Streckenberechnung, die nur einmalig zu Beginn des Spiels durchgeführt werden muss, müssen diese Merkmale für jeden Frame neu berechnet werden, um sie dem neuronalen Netz als Eingabe zu übergeben und eine Ausgabe zu erhalten.

Da jedes Merkmal auf der aktuellen Position der Figur basiert, ist es wichtig zunächst diese möglichst genau und zuverlässig zu bestimmen. Dazu werden mit einem Schwellenwertverfahren alle Pixel bestimmt, die annähernd die Farbe der gesuchten Figur haben und der Mittelwert über die Positionen aller dieser Pixel gebildet.

Die aktuelle Geschwindigkeit der Figur lässt sich aus der Differenz der aktuellen und der letzten Position berechnen. Dabei wird die Geschwindigkeit in x- und y-Richtung aufgeteilt, stellt also zwei Merkmale dar. Um aber kleinen Störungen und starken Schwankungen der Geschwindigkeit entgegenzuwirken, werden die letzten sieben Positionen der Figur gemerkt und die durchschnittliche Geschwindigkeit in diesem Zeitraum gebildet.

Für die Entfernung zum nächsten Hindernis in verschiedenen Richtungen wird ausgehend von der aktuellen Position berechnet, wie weit das nächste Hindernis nach oben, unten, links, rechts und in den vier diagonalen Richtungen, sowie der aktuellen Bewegungsrichtung entfernt ist. Als Hindernisse gelten dabei sowohl die Wände der Strecke als auch gegnerische Figuren.

Um die Entfernungen zu den nächsten Kontrollpunkten zu berechnen, ist es zuerst nötig zu bestimmen, welchem Kontrollpunkt die Figur zur Zeit am nächsten ist. Um dies möglichst schnell herauszufinden, wird bereits zu Beginn des Spiels nach der Streckenerkennung eine Karte angelegt, die jedem Pixel der Strecke ihren nächsten Kontrollpunkt zuordnet. Ist der aktuelle Kontrollpunkt bestimmt, kann durch die Differenz der Position des nächsten Kontrollpunktes und der aktuellen Ballposition die Entfernung in jeweils x- und y-Richtung berechnet werden. Gleiches kann auch für den übernächsten oder weitere Kontrollpunkte unternommen werden, um der KI eine größere Vorausplanung zu ermöglichen.

III. Ergebnisse

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!” [Online]. Available: https://neat-python.readthedocs.io/en/latest/

[3] “Opencv: Opencv modules.” [Online]. Available: https://docs.opencv.org/3.4.3/