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

Marius Krusen und F. J. Michael Werner

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.

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.

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.

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.

Minimierung der Dimensionalität: NEAT sieht vor, dass die anfängliche Population aus Netzen besteht, die keine verdeckten Knoten und keine Kanten 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

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 Strecke vom äußeren Rand und möglichen Hindernissen auf der Strecke zu unterscheiden. Der innere Rand wird

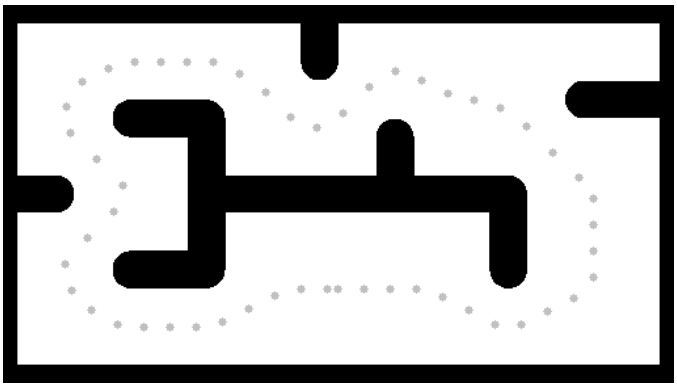

Fig. 1. Die resultierenden Kontrollpunkte nach Anwendung des Watershed-Algorithmus

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 schädlich ist, wird auf den Bildterm verzichtet. Außerdem hat sich gezeigt, das eine gleichmäßige Gewichtung

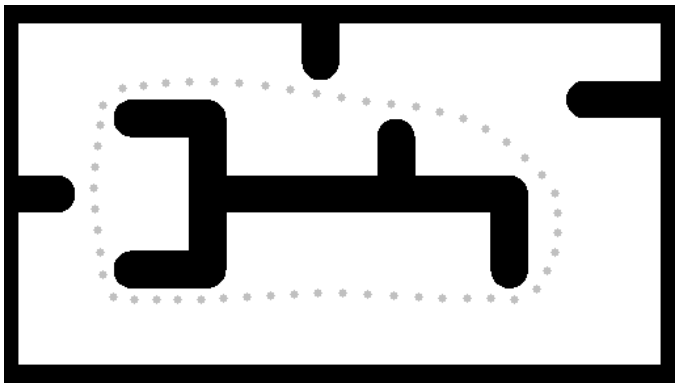

Fig. 2. Die resultierenden Kontrollpunkte nach Anwendung des Snake-Algorithmus

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, 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 wird ausgehend von der aktuellen Position berechnet, wie weit das nächste Hindernis in 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.

TABLE I
ÜBERSICHT DER VERWENDETEN NEAT-PARAMETER.

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

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 I zu finden. Zu dem wurden den initialen Population jeweils Kanten mit einer Wahrscheinlichkeit von 0,2 hinzugefügt.

A. Parameter

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.

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

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:

$$f_1 = \#KP, \tag{4}$$

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:

$$f_2 = \#KP + (180s - t) + (255 - s), \tag{5}$$

mit t als die benötigte Zeit in Sekunden und s als der erlittene Schaden. Dabei sind 180s und 255 die jeweils maximal möglichen Werte. Die Erweiterung wurde jedoch nur angewendet, wenn das Genome erfolgreich im Ziel angekommen ist, um das Erreichen des Zieles zu belohnen.

Kontrollpunkte: Als letzten Parameter haben wir angegeben, welche Kontrollpunkten als Eingabe verwendet werden. Die Liste $[2, 4]$ steht zum Beispiel dafür das die Differenzen in x- und y-Richtung von der aktuellen Position zu den übernächsten sowie dem viert nächsten Kontrollpunkt angegeben werden. Pro Kontrollpunkt entstehen also jeweils zwei weitere Eingaben

TABLE II
ÜBERSICHT ZU DEN EINZELNEN PARAMETERN DER JEWEILIGEN EXPERIMENTE.

Parameter	E1	E2	E3	E4	E5
# Ausgangsneurone	4	4	2	2	2
Strecke	1	3	3	3	3
Fittnessfunktion	f_1	f_1	f_1	f_2	f_2
Kontrollpunkte	$[1]$	$[2]$	$[2]$	$[2]$	$[2, 4]$

TABLE III
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.

	E1	E2	E3	E4	E5	Durchschnitt
E1	3:17 255	3:24 255	3:29 255	2:47 255	3:05 255	- 255
E2	2:02 255	2:23 255	2:31 255	2:17 255	2:18 255	- 255
E3	3:23 160	3:22 76	3:10 56	3:20 146	3:30 162	3:24 136
E4	3:11 92	3:20 120	3:19 129	3:13 70	3:16 102	3:17 110,75
E5	3:16 116	3:19 131	3:18 144	3:19 174	3:17 117	3:18 141,25

B. Experimente

In Tabelle II sind die jeweiligen Parameter der Experimente dargestellt. Neben den Distanzen zu den jeweiligen Kontrollpunkten wurde die aktuelle Geschwindigkeit in x- und y-Richtung, sowie der Abstand zu dem nächsten Hindernis in der aktuellen Bewegungsrichtung als Eingabe verwendet. Der NEAT-Algorithmus wurde auf einer Population der Größe 45 über 200 Generationen angewendet. Um die Laufzeit zu verringern wurde von drei Genomen zur gleichen Zeit der jeweilige Wert der Fitnessfunktion berechnet, indem sie gleichzeitig ein Rennen gefahren sind. Das Genome, das in der 200 Generation den höchsten Wert hat, repräsentiert nachfolgend das jeweilige Experiment.

Nach Anwendung des NEAT-Algorithmus, haben wir die Repräsentanten der Experimente in einem Turnier in Zweierduellen gegen einander antreten lassen. In Tabelle III sind die jeweils benötige Zeit und der erlittene Schaden aus den Duellen zu finden. Dabei wird deutlich, dass das Netz aus Experiment 4 sich gegen alle anderen, im Bezug auf Zeit und Schaden durchsetzen konnte. Daher wird dieses auch in dem Turnier am Ende Projektpraktikums eingesetzt.

In Abbildung 3 ist das neuronal Netz aus Experiment

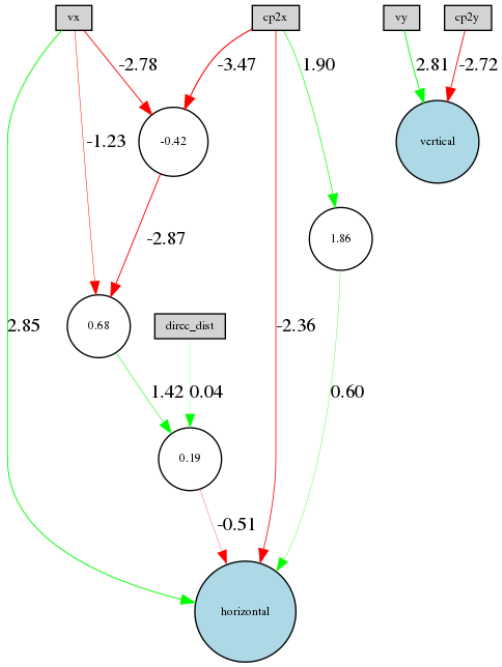

Fig. 3. Das neuronale Netz des Repräsentanten des Experiment 4. Aus Gründen der Übersichtlichkeit wurden deaktivierte Kanten, sowie Knoten und Kanten, von den man die Ausgangsknoten nicht erreichen kann, nicht dargestellt

4 dargestellt. Dabei ist auffällig, dass für die vertikale Steuerung ein lineares Modell, basierend auf der vertikalen Geschwindigkeit und der vertikalen Abweichung zum Kontorotrollpunkt vorliegt. Für den horizontalen Fall ist das nicht ganz gegeben, wobei die zusätzlichen Kanten und Knoten nur einen vergleichsweise geringen Einfluss auf die Ausgabe habe. Auch in anderen Experimenten konnte die Bildung eines linearen Modells beobachtet werden.

Aus dem Vergleich den Experimente lassen sich die Vermutungen aufstellen, dass zum einen die Wahl von zwei Ausgangsknoten zu besseren Ergebnissen führt, da ein einfacheres Modell gelernt werden muss. Zum anderen kann die erweiterte Fitnessfunktion f_2 dazu führen, dass die Genome ein etwas schnelleres und sicheres Fahren lernen. Die Wahl von mehreren Kontrollpunkten erschwert das Problem. Allerdings könnten über mehr Generationen auch bessere Ergebnisse erzielt werden. Diese Vermutungen müssten noch durch intensivere Untersuchengen bestätigt werden.

IV. Zusammenfassung

Wir haben im Rahmen des Praktikums mithilfe von Streckenerkennung, Merkmalsberechnung und dem NEAT-Algorithmus mehrere KIs für das Spiel rAIcer entwickelt. Diese haben wir mit einander verglichen. Das beste Netz wurde mit 2 Ausgangsneuronen, der erweiterten Fitnessfunktion und mit einem Kontrollpunkt entwickelt. Dieses Netz ist in der Lage auf den bereitgestellten Strecken das Ziel zu erreichen. Überraschen ist, das durch NEAT stellen weise sehr einfache Modelle für die Steuerung gelernt werden.

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/

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