# Kollisionserkennung in einem mobilen Roboter mittels eines Neuronalen Netzes

Willy Bruhn
*Institut für Mathematik und Informatik*
*Universität Greifswald*
D-17487, Greifswald, Deutschland
willy.bruhn@gmx.de

***Zusammenfassung*—Selbstfahrende autonome Roboter, die mit Abstands-Sensoren zur Steuerung und Vermeidung von Kollisionen ausgestattet sind, arbeiten nicht immer fehlerfrei. Beispielsweise kann ein Roboter einem Hindernis in totem Winkel nicht ausweichen. Um Schäden an Roboter und Umgebung zu vermeiden, ist es hilfreich Kollisionen zuverlässig zu erkennen und dann entsprechend zu reagieren, sei es mit Abschalten des Roboters oder Änderung des Fahrverhaltens.**

**In dem vorliegenden Artikel stellen wir eine Methode vor, mithilfe eines Beschleunigungssensors und einem Neuronalen Netz Kollisionen in Echtzeit zu erkennen.**

***Index Terms*—selbstfahrender Roboter, Beschleunigungssensor, Kollisionserkennung, Künstliches Neuronales Netz**

## I. Einleitung

Kollisionserkennung findet in vielen Bereichen Anwendung, so zum Beispiel in Fahrzeugen [5], [7], [3], Bojen [9], selbstfahrenden Robotern [6] oder Roboterarmen [8], [15]. In dem vorliegenden Artikel untersuchen wir einen selbstfahrenden kleinen Roboter, der mit einem Beschleunigungssensor ausgestattet ist. Der Roboter fährt im Normalbetrieb mit Geschwindigkeiten um $1\,\mathrm{m\,s^{-1}}$. Da die auftretenden Geschwindigkeiten bei einem Kraftfahrzeug deutlich größer sind, sind auch die Beschleunigungs-Daten im Falle einer Kollision größer, was die Kollisionserkennung zu einer einfacheren Aufgabe macht. Bei dem Roboter sind die Beschleunigungsdaten beim Anfahren und einer Kollision sehr ähnlich. Kollisionen mit weichen Gegenständen sind besonders schwierig, da die resultierenden Beschleunigungswerte noch kleiner sind. Einfache Schwellwert-Tests sind hier oft nicht mehr ausreichend.

Als zusätzliche Schwierigkeit kommt hinzu, dass der Arduino-Mikroprozessor nur ein relativ kleinen Speicher und relativ geringe Rechenleistung aufweist. Neuronale Netze sind als universelle Klassifikator bekannt [12] und sind in diesem Artikel die Methode unserer Wahl.

Wir haben Trainings- und Test-Daten von verschieden schwierigen Kollisionen aufgezeichnet. Auf einer Workstation haben wir verschiedene Neuronale Netze trainiert und getestet. Bei der Implementierung und Wahl des Neuronalen Netzes auf dem Arduino haben wir einen Kompromiss zwischen Genauigkeit und Speicherbedarf bzw. Auswertungszeit gefunden. Der Speicherplatz des Arduino lässt zwar auch etwas tiefere Neuronale Netze zu, aber ein zufriedenstellendes Ergebnis haben wir mit logistischer Regression erhalten, welche ein Spezialfall eines Neuronalen Netzes ist.

## II. Aufbau und Sensorausstattung

Bei unserem selbstfahrenden Roboter handelt es sich um ein $29\,\mathrm{cm}$ x $36\,\mathrm{cm}$ x $27\,\mathrm{cm}$ großes Kettenfahrzeug der Marke T'Rex. Der Roboter ist mit einem Arduino-Mega 256 Board ausgestattet, welches die Rechenarbeit leistet. Zur Steuerung der beiden Motoren haben wir ein T'Rex-Mikrocontroller verbaut. An der Vorderseite sind drei Infrarot-Sensoren verbaut. Zwei davon vom Typ GP2D12 mit einer Reichweite von $80\,\mathrm{cm}$ in einem Winkel von $30\,\mathrm{cm}$ Grad und ein Sensor der nach vorne ausgerichtet ist, mit einer Reichweite von $120\,\mathrm{cm}$. Zentral auf dem Boden der Panzer-Chasis haben wir einen Arduino-Beschleunigungssensor vom Typ BNO055 verbaut.

## III. Fahrverhalten

Mit den drei Infrarot-Sensoren haben wir einen einfachen Fahralgorithmus implementiert [2]. Der Roboter fährt mit einer konstanten Geschwindigkeit. Dabei wird die Differenz des linken und rechten Sensors berechnet. Diese Differenz wird dann auf den linken Motor addiert und vom rechten Motor subtrahiert. Das hat zur Folge, dass der Robotor Wänden, denen er zu nahe kommt, ausweicht. Eine interessante Untersuchung dieses Korridor-Fahrverhaltens am Beispiel der Honig-Biene lässt sich in [13] finden.

Der Infrarot-Sensor in der Mitte leitet eine Rückwärtsbewegung mit anschließender Drehung ein, falls der Abstand zur Wand nach vorne zu klein ist. Im Falle einer Kollision (deren Erkennung Gegenstand des restlichen Artikels sein wird) fährt der Roboter für $500\,\mathrm{ms}$ rückwärts und dreht sich danach für $500\,\mathrm{ms}$ in eine zufällig gewählte Richtung. Im Falle einer Kollision beim Auflösen einer vorherigen Kollision fährt der Roboter $500\,\mathrm{ms}$ vorwärts und dreht dann $500\,\mathrm{ms}$ in eine zufällige Richtung. Dieses Fahrverhalten im Falle einer Kollision liefert relativ gute Ergebnisse und reicht in den meisten Fällen schon aus, damit der Roboter nicht stecken bleibt. Für einen besseren Algorithmus der auch noch die Richtung der Kollision zur Kollisionsauflösung nutzt siehe [10] und [6].

## IV. Kollisions-Szenarien

Das oben beschriebene Fahrverhalten liefert gute Ergebnisse, wenn die Hindernisse zusammenhängende großflächige Objekte sind. Wenn das Hindernis allerdings zu niedrig ist (bspw. ein Holzbalken) oder das Hindernis zu schmal

(bspw. ein Stuhlbein) kann es zu einer Kollision kommen. Im schlimmsten Fall bleibt der Roboter dann komplett stecken, da weiterhin kein Hindernis registriert wird. In so einem Fall wollen wir eine Kollision mithilfe des Beschleunigungssensors erkennen. Dabei soll das normale Fahrverhalten allerdings nicht beeinträchtigt werden, durch ein übervorsichtiges Verfahren zur Kollisionserkennung. Das heißt Gegenstände die der Roboter überfahren kann, sollten nicht als Kollision eingestuft werden. Eine Kollision bedeutet in unserem Falle also, dass der Roboter feststeckt und sich nur durch ein Ändern der Fahrtrichtung befreien kann. Wir haben den Roboter in einem Zimmer voller Stühle und Tische und Gegenständen auf dem Boden fahren lassen.

## V. Ein Neuronales Netz zur Kollisionserkennung

Künstliche Neuronale Netze haben in den letzten Jahren in vielen Bereichen des Maschinellen Lernens bessere Ergebnisse erzielt als vergleichbare Methoden [12]. In [15] wurden mithilfe mehrerer Beschleunigungssensoren an einem Roboterarm verschiedene Materialien erkannt. In [16] wurde die Körperliche Aktivität von Menschen mithilfe eines Beschleunigungssensors gemessen und in verschiedene Aktivitäten, wie Laufen, Tanzen etc. klassifiziert.

Wir haben uns hier vor allem für Neuronale Netze entschieden, da die Auswertung eines fertig trainierten Netzes für relativ kleine Modelle auch in Echtzeit auf dem Arduino-Mega funktioniert. Bei alternativen Methoden bspw. K-nearest-neighbour [4] muss oft noch eine größere Menge an Speicherplatz bereit gestellt werden.

### A. Beschreibung der Modelle

Zu jedem Datenpunkt werden die drei Beschleunigungsdaten und drei Gyroskopdaten des Beschleunigungssensors ausgewertet. Zusätzlich wird die aktuelle Motorstärke des rechten und linken Motors notiert. Diese Acht Werte werden mit einer konstanten Abtastrate von $125\,\mathrm{Hz}$ abgefragt. In einem Fenster von fester Länge werden mehrere dieser Datenpunkte gleichzeitig betrachtet. Das bedeutet, dass die ersten acht Neuronen die Information des ersten Datenpunktes enthalten die nächsten acht Neuronen die Information des zweiten Datenpunktes und so weiter, siehe Abbildung 1. Diese Eingabeschicht ist für die Modelle, die wir betrachtet haben immer die gleiche.

### B. Training des Neuronalen Netzes

Die Implementation des Neuronalen Netzes auf einem leistungsstarken Computer haben wir mit tensorflow [1] durchgeführt. Experimente mit unterschiedlichen Architekturen Neuronaler Netze haben angedeutet, dass sich mit nur einer einzigen Ausgabeschicht bereits sehr gute Ergebnisse erzielen lassen. Dieses Neuronale Netze entspricht dem Modell der logistischen Regression.

### C. Implementation auf dem Arduino

Ein kritischer Punkt bei der Implementation ist die Einhaltung einer konstanten Abtastrate der Sensor-Daten. Eine

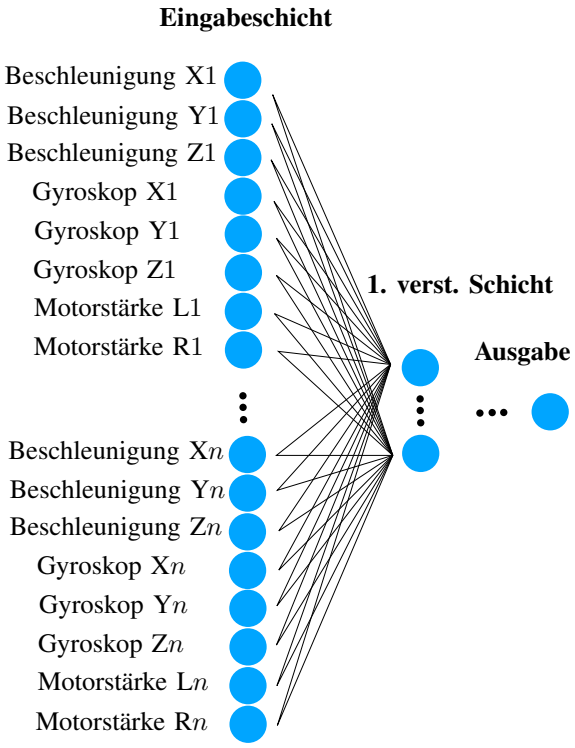


Abbildung 1: Eingabeschicht des Neuronalen Netzes

Auswertung der Daten durch das NN kann je nach Komplexität und gewählter Fensterbreite zwischen $20\,\mathrm{ms}$ und $40\,\mathrm{ms}$ benötigen. Bei schwankenden Zeitabständen der Sensor-Daten liefert das Neuronale Netz tatsächlich nicht so gute Ergebnisse. Wir haben hier eine Frequenz von $125\,\mathrm{Hz}$ verwirklicht, in dem auch während der Auswertung des Neuronale Netz Sensordaten abgefragt und in einem Puffer zwischengespeichert werden, bis das Neuronale Netz die Auswertung der nächsten Daten beginnt. Dabei werden also jedes mal einige Datenpunkte übersprungen. Eine zu lange Rechenzeit des Neuronalen Netzes beeinträchtigt auch das Fahrverhalten, da seltener Steuerungskommandos gesetzt werden können.

### D. Generierung der Trainingsdaten

Der Arduino ist über den Serialport mit einem Laptop verbunden und schickt durchgehend die acht Daten und einen boolschen Wert der Kollision oder Nicht-Kollision kodiert. Ein Kabel mit einem Knopf dient dabei dazu eine Kollision zu markieren. Bei Betätigung des Knopfes reagiert der Roboter zusätzlich mit einer Verzögerung von $500\,\mathrm{ms}$ um die Kollision aufzulösen. Dies dient dazu die Reaktion auf eine Kollision nicht mit zu trainieren.

Diese so gesammelten Trainingsdaten haben wir mit einem R-Skript [11] bearbeitet. Da es nicht möglich ist den Knopf auf Millisekunden-Genauigkeit zu betätigen, haben wir in einem Fenster von $100\,\mathrm{ms}$ den Punkt der Kollision auf den Punkt verschoben in dem das Maximum der Beträge der Beschleunigungen der X-,Y-,Z-Richtung liegt. Da die Implementierung des Neuronalen Netzes auf dem Arduino in etwa $25\,\mathrm{ms}$ für eine Auswertung benötigt, haben wir die zehn Punkte

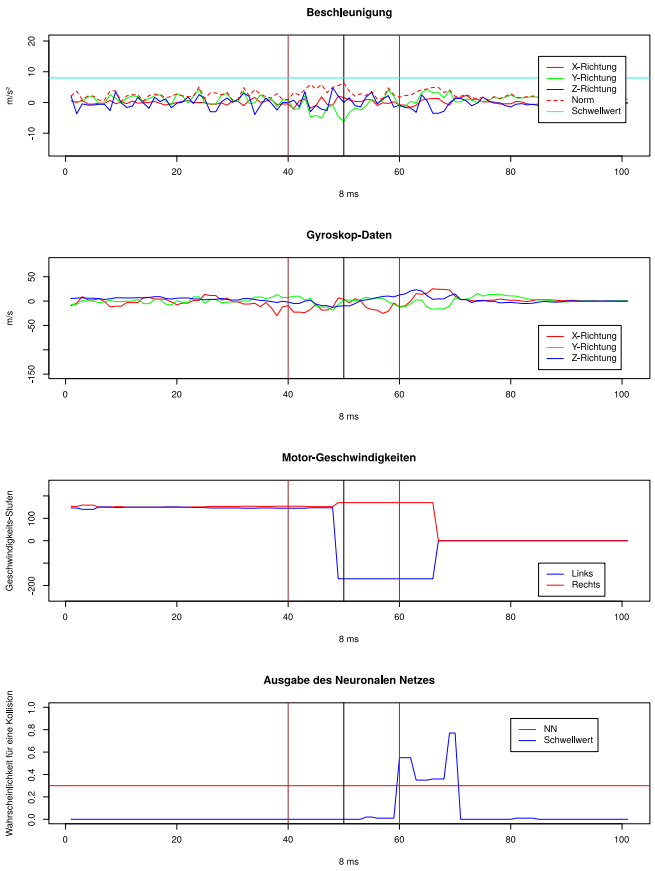


Abbildung 2: True-Positive-Beispiel einer Kollision, die bei einer Drehung stattfindet, was an den Motorstärken zu erkennen ist. Der Schwellwert erkennt die Kollision nicht, das Neuronale Netz erkennt die Kollision.

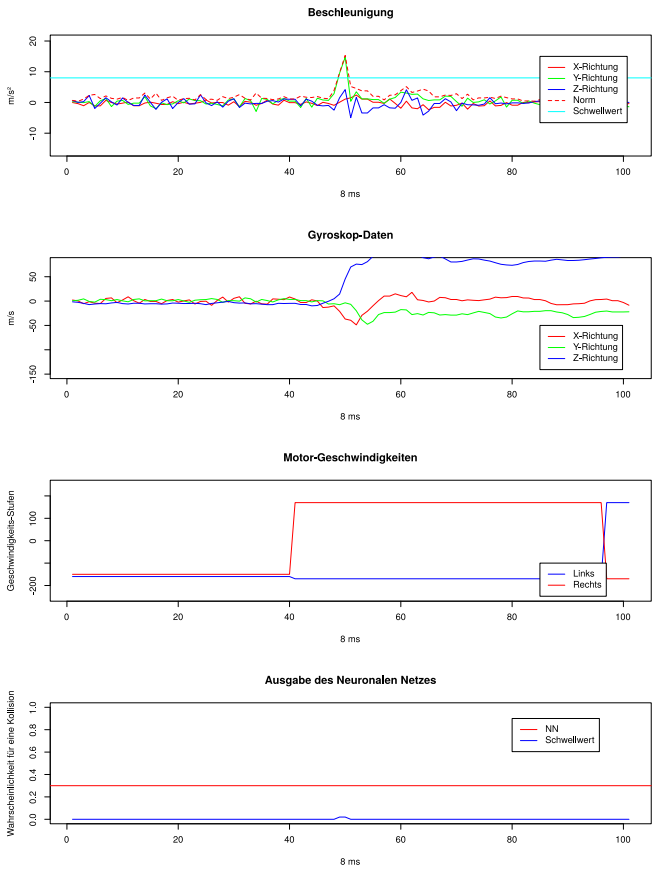


Abbildung 3: True-Negative-Beispiel einer Nicht-Kollision. Hier meldet der Schwellwert eine Kollision, das Neuronale Netz erkennt, dass es keine Kollision ist.

vor und nach der Kollision liegen, ebenfalls als Kollision markiert. Ansonsten könnten relevante Eingaben durch die Auswertungszeit übersprungen werden.

### *E. Vergleichsmodelle*

Zur Bewertung der Performanz des Neuronalen Netzes haben wir zwei weitere Verfahren getestet.

*1) Schwellwert der Norm der Beschleunigung:* Eine naheliegende und leicht zu implementierende Methode ist einen Schwellwert für die Beschleunigung zu wählen. Bei Überschreitung dieses Wertes, meldet dieses Verfahren eine Kollision.

*2) Entscheidungsbaum:* Als weiteres Verfahren haben wir einen Entscheidungsbaum mit dem R-Packet rpart [14] trainiert. Der entstandene Baum hat eine relativ geringe Tiefe von 19 Knoten. Die Implementation eines Baumes mit geringer Tiefe auf dem Arduino erscheint vielversprechend, da die Auswertungszeiten einer Handvoll if-Abfragen vermutlich wesentlich geringer sind, als die Auswertungszeiten eines mehrschichtigen Neuronalen Netzes, bei dem mehrere Matrixmultiplikationen durchgeführt werden müssen.

## VI. Ergebnisse

Die Trainings-Daten wurden in mehreren Fahrten in einem Büroraum mit einer Gesamtdauer von ca. $30\,\mathrm{min}$ aufgenommen. Dabei traten unter anderem Kollisionen mit Stuhlbeinen, Tischbeinen, Wänden, Pappkartons, Kleidungsstücken und Rucksäcken auf.

Nach dem Training des Neuronalen Netzes haben wir nach kurzen Test-Fahrten einen Schwellwert von $0.3$ eingestellt. Bei dem einfachen Schwellwert-Test haben wir den Wert bestimmt, der die Summe der True-Positives und True-Negatives maximiert. Den Entscheidungsbaum haben wir ebenfalls auf diesen Daten trainiert.

Die Test-Daten wurden in mehreren Fahrten mit einer Gesamtdauer von ca. $10\,\mathrm{min}$ aufgezeichnet.

### *A. Auswertung der Test-Daten*

Die Auswertung haben wir mit einem R-Skript durchgeführt. Dabei werden alle markierten Kollisionen der Reihe nach durchgegangen. Wird in einem Vorgegeben Bereich $W$ von $400ms$ ein Punkt gefunden, an dem der Wert des jeweiligen Verfahrens eine Kollision meldet, so wird diese Kollision als richtig erkannt (True-Positive) gezählt, andernfalls wurde die Kollision nicht erkannt (False-Negative). Danach werden die Bereiche durchgegangen, in denen das Verfahren

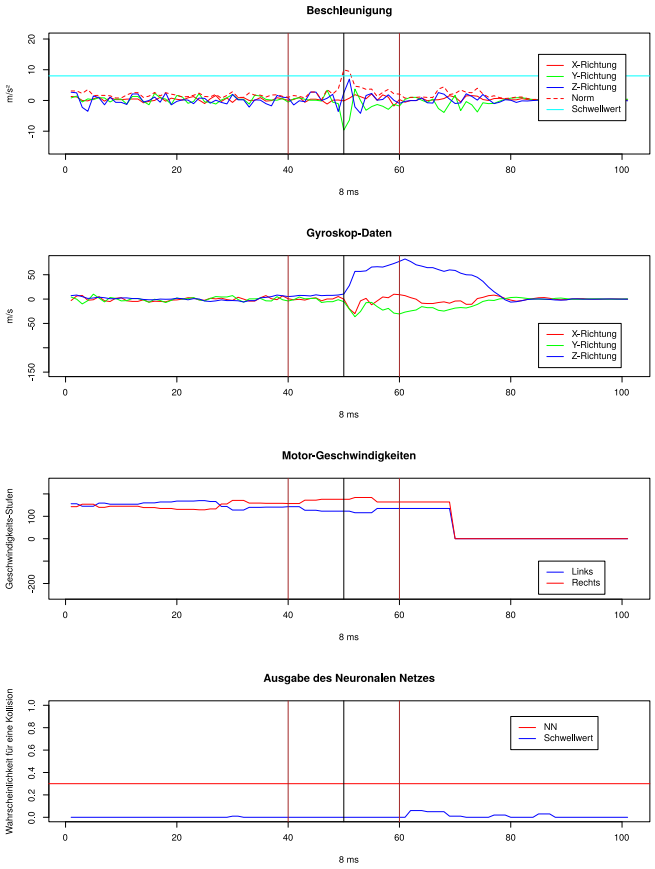


Abbildung 4: False-Negative-Beispiel einer Kollision. Der Schwellwert erkennt die Kollision, das Neuronale Netz erkennt die Kollision nicht.

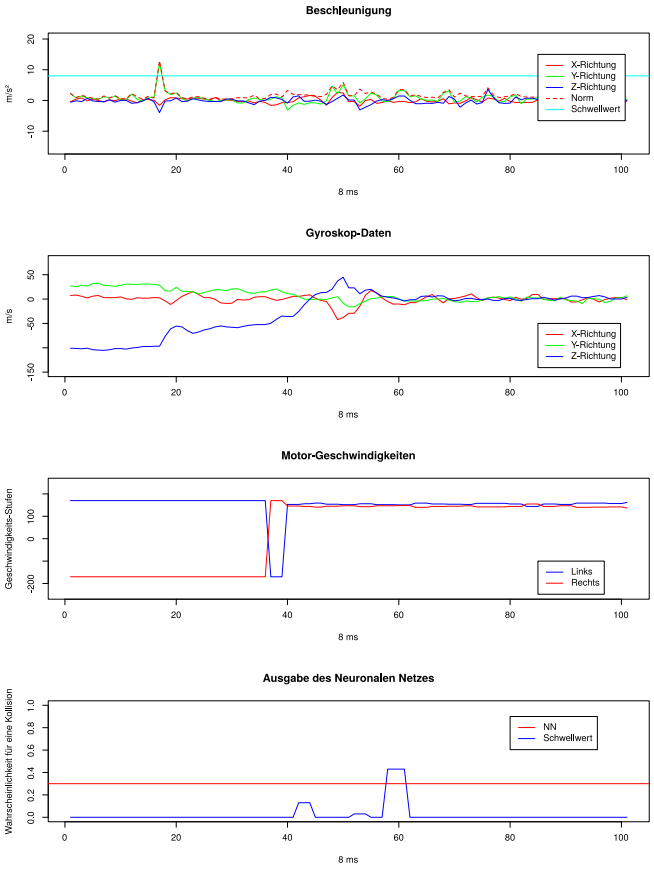


Abbildung 5: False-Positive-Beispiel einer Nicht-Kollision

eine Kollision meldet. Wird in $W$ um so einen Punkt keine markierte Kollision gefunden, so handelt hat das Verfahren fälschlicherweise eine Kollision gemeldet (False-Positive). Die Länge der restlichen Daten geteilt durch $W$ ergibt die Anzahl der richtig als Nicht-Kollision eingestuften Bereiche (True-Negative).

In den Tabellen I, II und III sind die Wahrheitsmatrizen der drei Verfahren dargestellt. Der Schwellwert-Test und Entscheidungsbaum liefern hier sehr ähnliche Ergebnisse. Bei gedämpften Kollisionen, beispielsweise mit einem weichen Gegenstand, bei denen keine starken Beschleunigungen auftreten, werden die Kollisionen nicht erkannt. Das Neuronale Netz erkennt mehr Kollisionen und scheint das bessere Verfahren zu sein.

Tabelle I: Neuronales Netz

| | keine Koll. vorhergesagt | Koll. vorhergesagt |
|---|---|---|
| keine Koll. | 591 | 14 |
| Koll. | 26 | 34 |

Tabelle II: Schwellwert

| | keine Koll. vorhergesagt | Koll. vorhergesagt |
|---|---|---|
| keine Koll. | 594 | 22 |
| Koll. | 44 | 19 |

Tabelle III: Entscheidungsbaum

| | keine Koll. vorhergesagt | Koll. vorhergesagt |
|---|---|---|
| keine Koll. | 589 | 25 |
| Koll. | 43 | 18 |

## VII. Zusammenfassung

Wir haben einen kleinen selbstfahrenden Roboter mit einem Neuronalen Netz zur Kollisionsdetektion ausgestattet. Im Vergleich zu einem einfachen Schwellwert-Test schneidet das Neuronale Netz deutlich besser ab, wenn verschiedene Arten von Kollisionen auftreten können.

Die Software, die wir für diesen Artikel entwickelt haben, sollte es möglich machen den Roboter relativ schnell auch in neuen Szenarien zu trainieren.

Sämtlicher Quellcode, Trainings- und Test-Daten sind unter **https://github.com/WillyBruhn/CollisionDetection** zu finden.

## Literatur

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang

Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Valentino Braitenberg. *Vehicles: Experiments in synthetic psychology*. MIT press, 1986.

[3] Javier Cervantes-Villanueva, Daniel Carrillo-Zapata, Fernando Terroso-Saenz, Mercedes Valdes-Vela, and Antonio F Skarmeta. Vehicle maneuver detection with accelerometer-based classification. *Sensors*, 16(10):1618, 2016.

[4] Padraig Cunningham and Sarah Jane Delany. k-nearest neighbour classifiers. *Multiple Classifier Systems*, 34(8):1–17, 2007.

[5] Ignacio García, José D Martín-Guerrero, Emilio Soria-Olivas, Rafael J Martínez, Silvia Rueda, and Rafael Magdalena. A neural network approach for real-time collision detection. In *Systems, Man and Cybernetics, 2002 IEEE International Conference on*, volume 5, pages 5–pp. IEEE, 2002.

[6] F He, Z Du, L Sun, and R Lin. Collision signal processing and response in an autonomous mobile robot. *NEURAL PARALLEL AND SCIENTIFIC COMPUTATIIONS*, 15(3):319, 2007.

[7] Tomasz Kamiński, Michał Niezgoda, Mikołaj Kruszewski, Rafał Grzeszczyk, Tomasz Drop, and Przemysław Filipek. Collision detection algorithms in the ecall system. *Journal of KONES*, 19:267–274, 2012.

[8] William McMahan, Joseph M Romano, and Katherine J Kuchenbecker. Using accelerometers to localize tactile contact events on a robot arm. In *Proc. Workshop on Advances in Tactile Sensing and Touch-Based Human-Robot Interaction, ACM/IEEE International Conference on Human-Robot Interaction*. Citeseer, 2012.

[9] Erkki Moorits and Aivar Usk. Buoy collision detection. In *ELMAR, 2012 Proceedings*, pages 109–112. IEEE, 2012.

[10] Seung Y Na, Daejung Shin, Jin Y Kim, and Su-Il Choi. Collision recognition and direction changes using fuzzy logic for small scale fish robots by acceleration sensor data. In *International Conference on Fuzzy Systems and Knowledge Discovery*, pages 329–338. Springer, 2005.

[11] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[12] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[13] Julien R Serres, Guillaume P Masson, Franck Ruffier, and Nicolas Franceschini. A bee in the corridor: centering and wall-following. *Naturwissenschaften*, 95(12):1181, 2008.

[14] Terry Therneau and Beth Atkinson. *rpart: Recursive Partitioning and Regression Trees*, 2018. R package version 4.1-13.

[15] Piyamate Wisanuvej, Jindong Liu, Ching-Mei Chen, and Guang-Zhong Yang. Blind collision detection and obstacle characterisation using a compliant robotic arm, 05 2014.

[16] Yonglei Zheng, Weng-Keen Wong, Xinze Guan, and Stewart Trost. Physical activity recognition from accelerometer data using a multi-scale ensemble method. In *IAAI*, 2013.