

# **Automatische Klassifizierung handgezeichneter Mechanismen durch maschinelles Lernen**

Stefan Gössner\*, Kai Lawrence\*\*

\* FH Dortmund, Professur für Dynamik, Mechanismentechnik und Webtechnologien  
stefan.gössner@fh-dortmund.de

\*\* FH Dortmund, Maschinenbaustudent  
kaihenning.lawrence002@stud.fh-dortmund.de

## **Kurzfassung**

Anwendungen für Mechanismentechnik leiden immer unter dem Problem, dass für eine schnelle Skizze eines Mechanismus die entsprechenden Benutzeroberflächen der bestehenden Mechanismeneditoren mit Aufwand verbunden sind. Es gibt zwar Anwendungen zur schnellen und intuitiven Erstellung von Mechanismen, aber für kurze Betrachtungen ist der Einsatz eines solchen Editors in vielen Fällen nicht der Mühe wert.

Um auch ad-hoc eine Möglichkeit zu haben, Mechanismen interaktiv oder animiert zu betrachten, wird nun ein Programm entwickelt, welches es ermöglichen soll, Handzeichnungen zu scannen und den respektiven Mechanismus digital zu ermitteln.

## **Abstract**

Applications for mechanism engineering always suffer from the problem that for a quick sketch of a mechanism the corresponding user interfaces of existing mechanism editors are connected with effort. Applications for a quick and intuitive

sketching of mechanisms exist, but for short considerations using a computer is in many cases not worth the effort.

In order to be able to view mechanisms interactively or animatedly on an ad-hoc basis, a program is now being developed which will allow hand drawings to be scanned and the respective mechanism to be determined digitally.

## 1 Einführung

Die Simulation, Analyse und Synthese von Mechanismen mittels Basiswebtechnologien stehen im Fokus der Arbeitsgruppe Mechanismentechnik und Webtechnologien im Fachbereich Maschinenbau der Fachhochschule Dortmund. Im Rahmen dieser Tätigkeiten sind die JavaScript-Bibliotheken `mec2.js`[7][4][5], einer Bibliothek zur Modellierung ebener Mechanismen auf der Grundlage einer partikelzentrierten Physik-Engine, und `g2.js`[6] als minimalistische Grafikbibliothek entstanden.

Anwendung finden diese bereits in einem speziellen `<mec-2>` HTML-Element und dem webbasierten Mechanismeneditor `mecEdit`[14][15], der bereits aktiv in der Lehre genutzt wird.

Webanwendungen stehen stets unter dem Vorsatz, eine möglichst reibungsfreie Nutzerinteraktion zu gewährleisten. `mecEdit` bietet hierfür dem Nutzer die Möglichkeit, per möglichst einfacher Eingabe der entsprechenden Elemente des Mechanismus diesen interaktiv und animiert auf den Bildschirm zu bringen.

Für eine schnelle Skizzierung ist jedoch bislang der schnellste Weg die Handzeichnung auf dem Papier, welche jedoch intrinsisch weder interaktiv noch animiert ist.

Die Vorzüge der analogen und digitalen Herangehensweise zusammenzuführen stellt nun eine weitere Schnittstelle in der Entwicklung dar. Sie soll handskizzierte Mechanismen durch Mustererkennung kinematischen Modellen zuordnen, indem entsprechend in das Programm eingeladene Bilder analysiert und das `mec-2` Modell ermittelt werden kann.

Hierfür sind mehrere statistische Modelle trainiert worden, welche in der Lage sind, sowohl die durch `mec2` beschriebenen Nodes und Constraints zu erkennen und daraus einen entsprechenden Mechanismus abzuleiten.

Für das Training des Modells wurden die Bibliotheken `Tensorflow`[3] und `Keras`[2][1] genutzt.

```

{
  "nodes": [
    { "id": "node0", "x": 216, "y": 48, "base": true },
    { "id": "node1", "x": 244, "y": 264 },
    { "id": "node2", "x": 120, "y": 48, "base": true },
    { "id": "node3", "x": 56, "y": 196 }
  ],
  "constraints": [
    { "id": "constraint0", "p1": "node0", "p2": "node1", "len": { "type": "const" } },
    { "id": "constraint1", "p1": "node2", "p2": "node3", "len": { "type": "const" } },
    { "id": "constraint2", "p1": "node3", "p2": "node1", "len": { "type": "const" } }
  ]
}

```

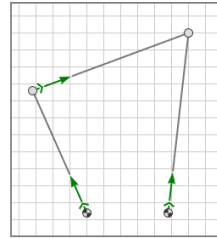


Abb. 1: Die JSON-Darstellung eines durch mec-2 erstellten Viergelenks mit dem entsprechend generierten Mechanismus.

## 2 Was erkannt werden soll

mec-2 modelliert Mechanismen durch sogenannte Nodes und Constraints. Die Gelenke sind entsprechend als Nodes und Glieder als Constraints zu betrachten. Beispielhaft für ein Viergelenk sähe das generierte JSON gemäß Abbildung 1 aus.

Um diesen Mechanismus durch eine Handzeichnung zu erstellen, ist es erforderlich, einen Algorithmus zu trainieren, der in der Lage ist, mit einer entsprechenden Skizze als Input solchen JSON-Code als Output zu produzieren.

Hierfür mussten zunächst Trainingsdaten geschaffen werden, anhand derer ein solcher Algorithmus Merkmale erlernen kann, die eine Zuordnung der Eingangsbilder zu den entsprechenden Lösungen bilden kann. Es wurden etwa 1200 Nodes, 1200 Base-Nodes und 1200 nicht zutreffende Bilder erstellt, welche vor dem Training durch Rotation und Spiegelung augmentiert wurden, um die Varianz der Trainingsdaten zu erhöhen.

Des Weiteren wurden von mec-2 genutzte Symbole dem Trainingsset hinzugefügt, sodass der Algorithmus diese nicht als Nodes erkennt, damit Mechanismen erweitert werden können, ohne die bereits erkannten Elemente nochmal zu erkennen.

Neben den Nodes wurden für die Constraints wieder 1200 rotatorische und 1200 translatorische Verbindungen gezeichnet, welche sich in ihrer Gestaltung an die

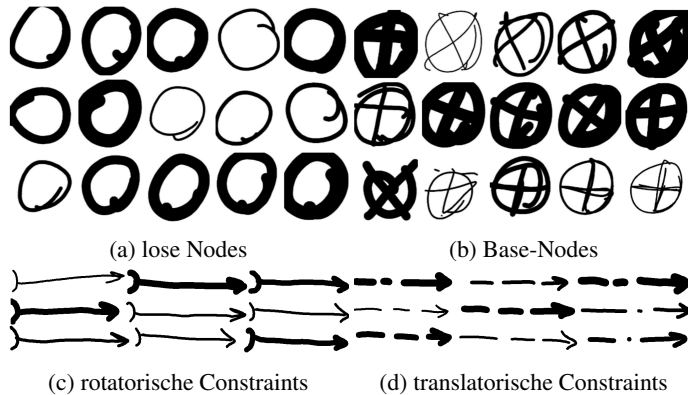


Abb. 2: Beispiele für handgezeichnete Symbole welche zum Trainieren der Algorithmen genutzt werden.

Länge	gebunden	frei	gebunden	frei
Winkel	gebunden	gebunden	frei	frei
DoF	0 ( <i>Fix</i> )	1 ( <i>Trans</i> )	1 ( <i>Rot</i> )	2 ( <i>Free</i> )

Abb. 3: Darstellung der Constraints in ihren unterschiedlichen Auslagemöglichkeiten. Auszug von Tabelle 1 des Beitrags von Prof. Gössner zum Sammelband des Getriebetechnischen Kolloquiums 2019 in Dortmund[4]

Darstellung von Constraints an Abbildung 3 orientieren. Für gebundene und freie Constraints gibt es noch keine Trainingsdaten, da diese für die momentan bestimmte Funktion zunächst irrelevant sind.

### 3 Datenpipeline

Als Konzeptnachweis ist ein neurales Netzwerk erstellt worden, welches die losen Nodes und Base-Nodes in den Abbildungen 2a und 2b von Bildern unterscheiden kann, in denen eben keine Node enthalten ist.

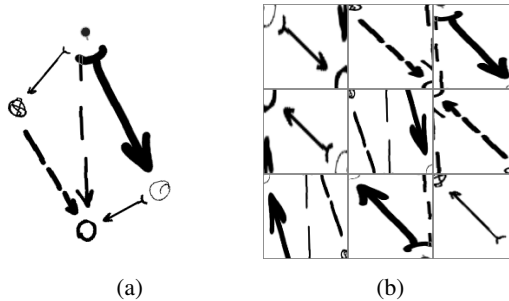


Abb. 4: Aus dem links zufällig generiertem Bild wurden die Bilder rechts generiert. Anzu-  
merken ist, dass nur die Bilder der ersten Reihe als korrekt bezeichnet werden.

Dieses wird dann in ein Fully Convolutional Neural Network[8] umgewandelt, um diese Überprüfung auf einem Bild zusätzlich lokalisieren zu können.

Ein weiteres neurales Netzwerk wird nun erstellt, um die Constraints zu erkennen. Hierfür wurden die Symbole, wie sie in den Abbildungen 2c und 2d zu sehen sind, benutzt, um Trainingsdaten zu generieren, die wiederum genutzt werden können, um das neurale Netzwerk zu trainieren. Hierfür wurden die vorher genutzten Nodes zufällig auf einem Bild verteilt und daraufhin zufällig durch Constraints verbunden.

Die daraus entstandenen Schnitte werden verformt, um einheitliche Maße zu erhalten. Des Weiteren werden die Bilder so gespiegelt, dass die erste Node stets oben links im Bild ist. So sind Bilder nur als korrekt anzusehen, wenn die Verbindung von oben links nach unten rechts dargestellt wird. Das ist notwendig, um neben der Kategorisierung auch die Richtung der Verbindung zu bestimmen.

Aufbauend auf diesen Algorithmen kann nun eine Datenpipeline programmiert werden, welche anhand eines Bildes zunächst die Nodes erkennt und daraufhin die Bildausschnitte generiert, welche dahingehend untersucht werden können, ob zwischen zwei Nodes eine Constraint existiert.

Die daraus gewonnene Information kann dann in das von mec-2 genutzte JSON Format überführt und der Mechanismus dargestellt werden. Ein vollständig definierter Mechanismus (alle Gelenke und Glieder, die Anzahl der Freiheitsgrade etc. seien dadurch bekannt) lässt sich so direkt der entsprechenden kinematischen Kette zuordnen.

## 4 Training

Während die Anwendung des Programms vollständig in JavaScript erfolgt und so leicht in eine Webapplikation einzubetten ist, geschieht das Training der neuronalen Netzwerke aus Gründen der Performanz in Python, welches durch Nvidia's CUDA[9] direkten Zugang auf die Grafikprozessoren des PC's hat<sup>1</sup>.

Das Erlernen von Merkmalen geschieht durch das Minimieren einer Verlustfunktion, welche die Differenz zwischen den vom Modell bestimmten Ergebnis und dem tatsächlichen Ergebnis misst. Die Ableitung dieser Verlustfunktion wird dann auf das neurale Netzwerk angewandt, um so mutmaßlich bei der nächsten Vorhersage einen niedrigeren Wert als Verlust zu bekommen.

Dieser Prozess wiederholt sich beliebig oft, bis die Vorhersagen eine ausreichende Genauigkeit erreichen.

Ein neuronales Netzwerk enthält initial zufällig zugewiesene Werte für die einzelnen Zellen, sodass bei drei Kategorien von einer Genauigkeit von etwa 33% zu rechnen ist. Durch entsprechend gewählte Trainingsparameter kann jedoch die Genauigkeit des Modells erhöht werden, sodass das neuronale Netzwerk zuständig für die Node-Erkennung eine Genauigkeit von 99,75% auf nie gesehene Daten erreicht, während mithilfe des Modells für die Erkennung von Constraints eine Genauigkeit von 97,68% erreicht werden konnte.

Es ist für die trainierten Algorithmen anzumerken, dass obwohl die Genauigkeit durch Daten ermittelt wird, welche vom Modell nie gesehen wurden, entsprechend also kein Auswendiglernen der Zuordnungen[13, p.705] für die Genauigkeit verantwortlich sein kann, sie nur in der Lage sind, Daten korrekt zuzuordnen, welche dem Format der Trainingsdaten entsprechen.

Die Trainingsdaten sind bewusst generiert worden, um einen Kompromiss zwischen ausreichendem Material für das Erlernen der notwendigen Merkmale in den Bildern zu finden, allerdings ohne tausende von Mechanismen zeichnen und entsprechend beschriften zu müssen, um ein Training möglich zu machen.

Die aktuell genutzten Modelle verwenden zum Training ausschließlich weiße Symbole auf schwarzem Hintergrund (die Abbildungen in dieser Ausarbeitung sind invertiert, um die Sichtbarkeit zu verbessern), sodass Merkmale, welche erkannt werden, ausschließlich in dieser Darstellung erkannt werden können. Beispielsweise würde ein schwarzer Kreis auf weißem Hintergrund nicht als

---

<sup>1</sup>Diese Zugriffe wird für den Entwickler durch die genutzten Bibliotheken TensorFlow und Keras abstrahiert.

Node erkannt werden, da der Algorithmus ihr die zur korrekten Zuordnung erforderlichen Merkmale nicht zuordnet. Eine entsprechende Vorbearbeitung von Bildern ist entsprechend notwendig, um die gewünschten Ergebnisse zu erhalten.

## 5 Beispiele

Diese Beispiele wurden in einer dafür konzipierten Erweiterung für das mec-2 HTML Element gezeichnet. Komplexere Modelle in einem Durchgang zu zeichnen, ist mit den aktuellen Modellen noch mühselig, was zum einem daran liegt, dass die Trainingsdaten nicht mit demselben Zeichenwerkzeug generiert wurden und zum anderen, dass die Modelle nicht robust genug trainiert sind, als dass dies ein Problem darstellt.

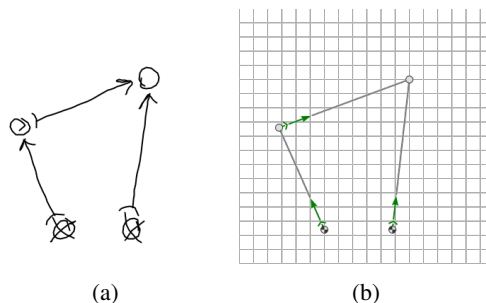


Abb. 5: Ein Viergelenk

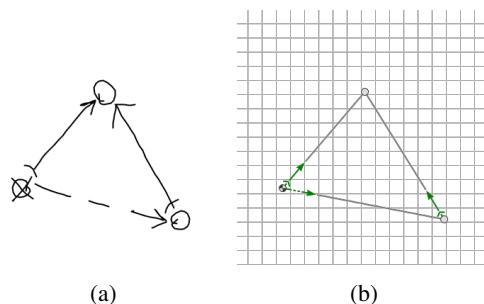


Abb. 6: Mechanismus mit translatorischem Glied

## 6 Aussicht

Der aktuelle Prototyp zeigt, dass durch das Anwenden von entsprechend trainierten Algorithmen durchaus Programme geschrieben werden können, welche in der Lage sind, von Hand gezeichnete Mechanismen digital verfügbar zu machen, um weitere Analysen an ihnen durchzuführen.

Der aktuelle Algorithmus ist jedoch eher als Konzeptnachweis anzusehen, da er trotz guter Genauigkeit auf dem Trainingsset nicht die Ergebnisse produziert, die für eine reibungslose Nutzung erforderlich wären. Eingangsbilder, die von den Trainingsdaten in Bezug auf Beschaffenheit abweichen aber noch als Nodes, beziehungsweise Constraints erkannt werden sollten, aber nicht werden, machen es schwierig, komplexere Modelle in einer Zeichnung zu erkennen.

Des Weiteren benötigt der aktuelle Ansatz einige Sekunden, um eine Prognose zu erstellen. Methoden, welche Ergebnisse in Echtzeit liefern, sind jedoch existent[10][11][12], jedoch sind für die korrekte Anwendung dieser Methoden weitere Anpassungen notwendig.

Anzumerken ist weiterhin, dass sich die Anzahl der Klassen, welche erkannt werden können (in diesem Fall jeweils drei Klassen pro neuralem Netzwerk), beliebig erweitern lassen. Durch das Hinzufügen eines entsprechenden Trainingssets für das Erkennen von Federn, welche zwischen zwei Gelenken eingesetzt sind, lässt sich dies beispielsweise erkennen und entsprechend in das mec-2 Modell einbringen. Das Gleiche gilt auch für weitere Darstellungsarten von Gelenken.

Aktuell ist das Programm lediglich in der Lage, Bilder in einer dafür vorgesehenen Umgebung innerhalb des von mec-2 genutzten Canvas zu zeichnen und Bilder hochzuladen. In Aussicht steht die Aufnahme von analog gezeichneten Mechanismen auf Papier oder anderen analogen Methoden, was den Nutzen des Projektes im Kern darstellt.

Schließlich lässt sich das Programm in eine sogenannte progressive Webapplikation umwandeln, um sie so auf dem Smartphone direkt mit Zugang auf die Kamera verwendbar zu machen, was in Bezug auf die Anwendbarkeit den größten Nutzen bringt.

Das gesamte Projekt ist offen zugänglich und lässt sich auf <https://github.com/klawr/deepmech> verfolgen.



## Literatur

- [1] Chollet, F.: *Deep Learning with Python*. Manning Publications, 28. Okt. 2017. 384 S. isbn: 1617294438.
- [2] Chollet, F.: *Keras*. 2019. url: <https://keras.io/> (besucht am 09. 10. 2019).
- [3] Google: *Tensorflow*. 2019. url: <https://www.tensorflow.org/> (besucht am 09. 10. 2019).
- [4] Gössner, S.: *Ebene Mechanismenmodelle als Partikelsysteme - ein neuer Ansatz. Tagungsband 13. Kolloquium Getriebetechnik, Fachhochschule Dortmund, 18. - 20. September 2019*. Logos Berlin, 2019, S. 169–180. isbn: 978-3-8325-4979-4.
- [5] Gössner, S.: *Fundamentals for Web-Based Analysis and Simulation of Planar Mechanisms, EuCoMeS 2018, Proceedings of the 7th European Conference of Mechanism Science*. 2018.
- [6] Gössner, S.: *g2*. 2019. url: <https://github.com/goessner/g2>.
- [7] Gössner, S.: *mec2*. 2019. url: <https://github.com/goessner/mec2>.
- [8] Long, J., E. Shelhamer und T. Darrell: „Fully Convolutional Networks for Semantic Segmentation“. In: (14. Nov. 2014). arXiv: <http://arxiv.org/abs/1411.4038v2> [cs.CV].
- [9] nvidia: *CUDA*. 2019. url: <https://www.geforce.com/hardware/technology/cuda> (besucht am 09. 10. 2019).
- [10] Redmon, J., S. Divvala, R. Girshick und A. Farhadi: „You Only Look Once: Unified, Real-Time Object Detection“. In: (8. Juni 2015). arXiv: <http://arxiv.org/abs/1506.02640v5> [cs.CV].
- [11] Redmon, J. und A. Farhadi: „YOLO9000: Better, Faster, Stronger“. In: (25. Dez. 2016). arXiv: <http://arxiv.org/abs/1612.08242v1> [cs.CV].
- [12] Redmon, J. und A. Farhadi: „YOLOv3: An Incremental Improvement“. In: (8. Apr. 2018). arXiv: <http://arxiv.org/abs/1804.02767v1> [cs.CV].
- [13] Stuart Russell, P. N.: *Artificial Intelligence: A Modern Approach, Global Edition*. Addison Wesley, 28. Nov. 2018. isbn: 1292153962.
- [14] Uhlig, J.: *Entwicklung einer modularen Web-App zur interaktiven Modellierung und impulsbasierten Analyse beliebiger planarer Koppelmechanismen*. Masterarbeit. Fachhochschule Dortmund, 2019.
- [15] Uhlig, J.: *mecEdit*. 2019. url: <https://mecedit.com> (besucht am 11. 12. 2019).