

Automatische Klassifizierung handgezeichneter Mechanismen durch maschinelles Lernen

Stefan Gössner*, Kai Lawrence**

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

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

Kurzfassung

Für eine schnelle, digitale Skizze zweidimensionaler Mechanismen erlauben neue Technologien im Vergleich zu aktuellen Methoden komfortablere Formen der Benutzerinteraktion. Es gibt zwar bereits Anwendungen zur schnellen und intuitiven Erstellung von Mechanismen, aber für kurze Betrachtungen ist der Einsatz solcher Editoren in vielen Fällen nicht der Mühe wert.

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

Abstract

For fast, digital sketching of two-dimensional mechanisms, new technologies allow more comfortable forms of user interaction compared to current methods. Although there are already applications for the fast and intuitive creation of

mechanisms, for brief observations the use of such editors 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 in JavaScript implementierten Bibliotheken `mec2.js` [4, 5, 7], 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 animierbar auf den Bildschirm zu bringen.

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

Im Folgendem wird das Ziel der Zusammenführung der Vorzüge der analogen und digitalen Herangehensweise beschrieben. Der Prozess soll handskizzierte Mechanismen durch Mustererkennung kinematischen Modellen zuordnen. Bilder werden dafür in ein Programm eingeladen, durch Mustererkennung analysiert und das `mec2` Modell ermittelt.

Hierfür wurden im Sinne des maschinellen Lernens mehrere statistische Modelle trainiert, welche in der Lage sind, 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` [1, 2] genutzt.

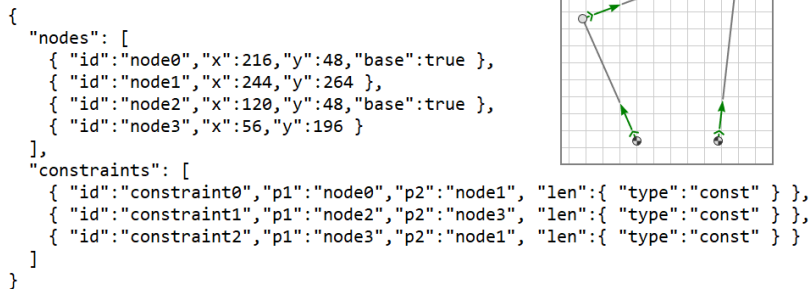


Abb. 1: Die JSON-Darstellung eines durch mec2 erstellten Vieregelenks mit dem entsprechend generierten Mechanismus.

2 Was erkannt werden soll

Im Gegensatz zu herkömmlichen Beschreibungen von Mechanismen durch Glieder und Gelenke modelliert mec2 Mechanismen durch Nodes und Constraints [4]. Beispielhaft zeigt Abbildung 1 ein Vieregelenk mit der zugehörigen Beschreibung des Mechanismus im JSON-Format.

Um solche Mechanismen durch Handzeichnungen zu erstellen, ist es erforderlich, einen Algorithmus zu trainieren, der mit einer entsprechenden Skizze als Input solchen JSON-Code als Output produziert.

Hierfür werden zunächst Trainingsdaten geschaffen, anhand derer ein trainierbarer Algorithmus Merkmale erlernen und so Eingangsbilder den entsprechenden Lösungen zuordnen kann. Für dieses Projekt wurden etwa 1200 Nodes, 1200 Base-Nodes und 1200 nicht zutreffende Bilder erstellt. Diese wurden vor dem Training durch Rotation und Spiegelung augmentiert, um die Varianz der Trainingsdaten zu erhöhen.

Dem zu trainierenden Algorithmus wurde zusätzlich beigebracht, die von mec2 genutzten Symbole den nicht zutreffenden Bildern zuzuordnen. Damit wird erreicht, dass Mechanismen erweitert werden können, ohne die bereits erkannten Elemente wiederholt zu erkennen.

Neben den Nodes wurden für die Constraints wieder 1200 rotatorische und 1200 translatorische Verbindungen gezeichnet. Sie orientieren sich in ihrer Gestaltung

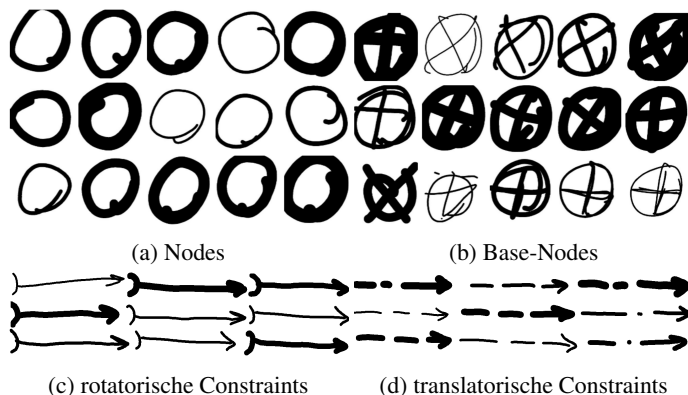


Abb. 2: Beispiele für handgezeichnete Symbole welche zum Trainieren der Algorithmen genutzt werden.





				
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 vier Ausprägungen. [4, Tab. 1]

an der Darstellung von Constraints in Abbildung 3. Zum aktuellen Stand des Projekts sind gebundene und freie Constraints nicht implementiert.

3 Datenpipeline

Für die durchgeführte Machbarkeitsstudie wurde ein neuronales Netzwerk erstellt. Dieses kann Nodes und Base-Nodes entsprechend den Abbildungen 2a und 2b erkennen oder als unzutreffend klassifizieren.

Das erstellte neuronale Netzwerk wurde dann in ein Fully Convolutional Neural Network [8] umgewandelt, um nicht nur die Klassen, sondern auch die Positionen der Nodes im Bild zu erhalten.

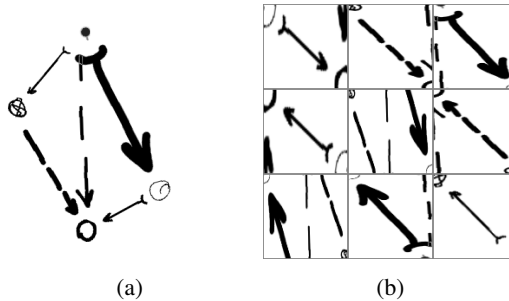


Abb. 4: Aus dem links zufällig generiertem Bild wurden die Bilder rechts generiert. Anzu-
merken ist, dass nur die Bilder der ersten Reihe als korrekt bezeichnet werden.

Ein weiteres neuronales Netzwerk wurde im weiteren Verlauf 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. Diese können wiederum genutzt werden, um das neurale Netzwerk zu trainieren. Hierfür wurden die vorher genutzten Nodes zufällig auf einem Bild verteilt und daraufhin zufällig durch Constraints verbunden (Abbildung 4a).

Die daraus entstandenen Schnitte (Abbildung 4b) werden verformt, um einheitliche Maße zu erhalten. Des Weiteren werden die Bilder so gespiegelt, dass das Anfangsnode stets oben links im Bild ist. Bilder werden demnach nur als korrekt erkannt, wenn ein enthaltenes Constraint von oben links nach unten rechts dargestellt wird. Das ist notwendig, um neben der Kategorisierung auch die Richtung der Constraints zu bestimmen.

Aufbauend auf diesen Algorithmen wird eine Datenpipeline programmiert, welche anhand eines Bildes zunächst die Nodes erkennt. Im nächsten Schritt werden Bildausschnitte generiert, die daraufhin untersucht werden können, ob zwischen zwei Nodes eine Constraint existiert.

Die daraus gewonnene Information kann dann in das von mec2 genutzte JSON-Format überführt und der Mechanismus dargestellt werden. Damit kann die mechanische Struktur im Sinne einer Bewegungssimulation und Kräfteanalyse weiter verwendet werden.

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. Mithilfe von CUDA [9] kann so direkt mit den Grafikprozessoren des PCs gearbeitet werden¹.

Das Erlernen von Merkmalen geschieht durch das Minimieren einer Verlustfunktion, welche die Differenz zwischen den vom Modell bestimmten Ergebnis und dem tatsächlichen Ergebnis misst [13, S.710]. Die Ableitung dieser Verlustfunktion nach dem Output des Modells wird dann auf das neuronale Netzwerk angewandt, um so mutmaßlich bei der nächsten Vorhersage einen niedrigeren Wert als Verlust zu bekommen [13, S.719].

Dieser Prozess wiederholt sich hinreichend oft, bis die Vorhersagen eine angestrebte Genauigkeit erreichen.

Ein neuronales Netzwerk enthält initial zufällig zugewiesene Werte für die einzelnen Zellen, sodass bei drei Kategorien mit einer Genauigkeit von etwa 33% auszugehen ist. Durch gewählte Trainingsparameter kann die Genauigkeit des Modells erhöht werden. So erreicht das neuronale Netzwerk zuständig für die Node-Erkennung eine Genauigkeit von 99,75% bei Daten, welche vor dem Training separiert wurden. Das Modell für die Erkennung von Constraints erreicht hier eine Genauigkeit von 97,68%.

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

Die Trainingsdaten wurden bewusst generiert, um einen Kompromiss zwischen ausreichendem Material für das Erlernen der notwendigen Merkmale in den Bildern zu finden, allerdings ohne tausende Mechanismen zeichnen und notwendigerweise beschriften zu müssen.

Die aktuell genutzten Modelle für das Erkennen von Nodes und Constraints verwenden zum Training ausschließlich weiße Symbole auf schwarzem Hintergrund². Merkmale können ausschließlich in dieser Gestalt erkannt werden.

¹Diese Zugriffe werden für den Entwickler durch die genutzten Bibliotheken Tensorflow und Keras abstrahiert.

²Die Abbildungen in dieser Ausarbeitung sind invertiert, um die Sichtbarkeit zu verbessern.

Beispielsweise würde ein schwarzer Kreis auf weißem Hintergrund nicht als Node erkannt werden, da der Algorithmus diesem die zur korrekten Zuordnung erforderlichen Merkmale nicht zuordnen kann.

5 Beispiele

Diese Beispiele wurden in einer dafür konzipierten Erweiterung für das mec2 HTML Element gezeichnet. Komplexere Skizzen von Mechanismen in einem Durchgang zu erkennen, ist mit den aktuellen Modellen noch mühselig. Die Modelle müssten vorab für den genutzten Anwendungsbereich trainiert werden, um die korrekte Funktionsweise zu gewährleisten.

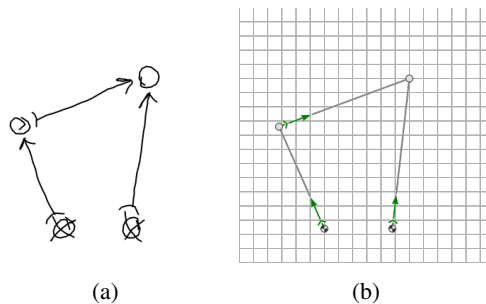


Abb. 5: Ein Viergelenk

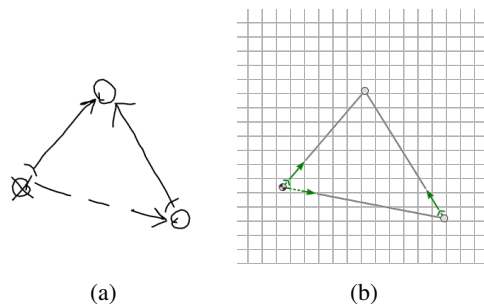


Abb. 6: Mechanismus mit translatorischem Glied

6 Zusammenfassung und Ausblick

Die Machbarkeitsstudie kam zu dem Ergebnis, 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 als Konzeptnachweis anzusehen. Trotz guter Genauigkeit auf dem Trainingsset werden nicht die Ergebnisse produziert, die für eine reibungslose Nutzung erforderlich sind. Bilder, die sich hinsichtlich der Summe der Merkmale vom Trainingsset unterscheiden, werden entgegen der Absicht des Zeichners teils nicht als Nodes beziehungsweise Constraints erkannt. Das macht es schwierig, komplexere Modelle in einer Zeichnung zu erstellen.

Des Weiteren benötigt der aktuelle Ansatz, unserer Meinung nach, noch zu lange, um ein Ergebnis liefern zu können. Hier eröffnet sich die Möglichkeit der Betrachtung alternativer Ansätze zur Erstellung neuronaler Netzwerke, deren Anwendung jedoch erst im zukünftigen Verlauf dieses Projektes betrachtet werden sollen [10–12].

Anzumerken ist weiterhin, dass sich die Anzahl der erkennbaren Kategorien, welche erkannt werden können³, beliebig erweitern lässt. Möchte man beispielsweise Linearfedern erkennen lassen, die zwischen zwei Nodes wirken, genügt das Hinzufügen eines entsprechenden Trainingssets.

Aktuell ist das Programm in der Lage, Bilder in einer dafür vorgesehenen Umgebung innerhalb des Canvas des `<mec-2>` HTML-Elements manuell zu zeichnen. Außerdem ist es möglich, Bilder in dieses Canvas hochzuladen. Die Implementierung der Erkennung analog gezeichneter Mechanismen in Fotografien stellt das Ziel dieses Projektes dar.

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

Das vollständige Projekt ist frei verfügbar und lässt sich verfolgen auf: <https://github.com/klawr/deepmech>

³Bisher sind jeweils drei Klassen pro neuronalem Netzwerk implementiert.

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).