Formale semantische Repräsentation regelungstechnischer Methoden
Formal Semantic Representation of Methods in Automatic Control

Dipl.-Ing. Robert Heedt, Dr.-Ing. Carsten Knoll, Prof. Dr.-Ing. habil. Dipl.-Math. Klaus Röbenack
TU Dresden, Institut für Regelungs- und Steuerungstheorie, 01062 Dresden, Deutschland
E-Mail: {robert.heedt, carsten.knoll, klaus.roebenack}@tu-dresden.de

Kurzfassung

Regelungstechnisches Wissen in klassischer Textform ist schwer zugänglich, wenn man für die aktuelle Problemstellung nicht die exakten Begriffe kennt. Als Ergänzung zur klassischen Wissenrepräsentation schlagen wir daher ein „Methodennetz“ vor, bestehend aus Typen und Methoden angeordnet in einer Graphstruktur. Für eine konkrete Aufgabe kann dann mittels Tiefensuche ein prozeduraler Lösungsablauf generiert werden, dessen Nutzen für den Wissenstransfer wir am Beispiel der Trajektorienfolgeregelung für einen Dreifachpendelversuchsstand skizzieren.

Abstract

Written knowledge about automatic control theory is hard to access as it requires familiarity with the exact terminology. Therefore, we propose the “method net”, a supplement to classical knowledge representation, consisting of types and methods in a graph structure. From that, a schematic solution procedure can be generated for a specific problem. Trajectory tracking control for a triple pendulum is used to demonstrate how the proposed method supports knowledge transfer.

1 Einleitung und Motivation

In vielen mechatronischen Produkten sind regelungstechnische Komponenten unverzichtbar. Die zugrundeliegende Regelungstheorie deckt inzwischen ein sehr breites Methodenspektrum ab und unterliegt, wie auch andere Wissensgebiete, einem beständigen Wachstum.
Als domänenunabhängige bzw. -übergreifende Querschnittsdisziplin steht sie jedoch vor einer besonderen Herausforderung bezüglich des Wissenstransfers, sowohl innerhalb der Disziplin als auch und besonders in potenzielle Anwendungsfelder. Gleichzeitig weisen die regelungstheoretischen Konzepte und Verfahren eine starke Heterogenität bezüglich expliziter und impliziter Verständnis- und Anwendungsvoraussetzungen auf – man vergleiche etwa einen einschleifigen P-Regelkreis mit einer nichtlinearen Mehrgrößen-Trajektorienfolgeregelung.
Als Konsequenz können einerseits fortgeschrittene regelungstechnische Methoden ihr Potential nicht voll entfalten, weil potentielle Anwender nicht von deren Existenz wissen, oder mangelndes Verständnis die Akzeptanz verhindert. Andererseits behindert suboptimaler Wissenstransfer auch die wissenschaftliche Neu- und Weiterentwicklung von Methoden.
Der vorliegende Beitrag schlägt deshalb einen formalen Ansatz als experimentelle Ergänzung zur bisher dominierenden Wissensrepräsentation vor.
Üblicherweise wird Wissen über geschriebene natürliche Sprache vermittelt, angereichert mit mathematischer Symbolik und Visualisierungen. Dieser textbasierte Informationsvorrat ist semantisch bisher bestenfalls mit meist willkürlich gewählten Schlagwörtern erschlossen.
Für ein gegebenes konkretes Problem setzt das Auffinden und Evaluieren einsetzbarer Lösungsmethoden mithin eine profunde Kenntnis der üblichen Fachterminologie voraus und ist selbst unter Zuhilfenahme syntaxbasierter Volltextsuchwerkzeuge aufwendig.
Inzwischen existieren aber fortgeschrittene Konzepte zur maschinenlesbaren Repräsentation von Wissen auf semantischer Ebene, wie etwa sog. Beschreibungslogiken oder darauf aufbauende sog. *Ontologien* [11, 3, 9]. Eine solche Repräsentationsform ermöglicht eine zielgerichtete Suche mit hoher Spezifität. Weiterhin erlaubt die semantische Repräsentation auch die Anwendung eines automatischen Schließers (engl. „reasoner“) zum Aufdecken innerer Widersprüche.
Allerdings ist festzustellen, dass bisherige Bestrebungen regelungstechnisches Wissen formal abzubilden aus Perspektive der Autoren nicht die erforderliche Funktionaliät und/oder Reichweite erreicht haben, um über interessante „Nischenlösungen“ wie z. B. [2] hinaus eine Wirkung zu erzielen und an der oben beschriebenen Problemlage signifikant etwas zu verändern.

Als potenzielle Alternative stellen wir deshalb mit dem *Methodennetz* und den daraus generierbaren *Aufgabenprozeduren* einen gleichzeitig sowohl anwendungsbezogenen als auch generalistischen Ansatz vor. Das Methodennetz formalisiert hierbei das Wissen darüber, wie typische regelungstechnische Aufgaben gelöst werden, d. h. mit welchen Teilschritten sich gegebene Informationen in gesuchte Informationen überführen lassen.

2 Das Methodennetz

2.1 Aufbau

Ziel des Methodennetzes ist es, Wissen über regelungstechnische Verfahren strukturiert menschen- und maschinenlesbar zu repräsentieren. Für eine konkrete regelungstechnische Aufgabe soll dann mithilfe des Methodennetzes auto-

matisch durch Verknüpfen hinterlegter Lösungsschritte eine Gesamtlösungsstrategie generiert werden. Diese Strategie wird nachfolgend als *Aufgabenprozedur* bezeichnet.
Da das Endergebnis auf eine prozedurale Beschreibung abgebildet wird, ist es naheliegend, zur Repräsentation des Wissens Konzepte der Programmierung zu übernehmen. Beim Entwurf von Software teilt man die Aufgabe typischerweise in Datentypen und Funktionen auf. Dementsprechend unterteilen sich auch die Bestandteile des Methodennetzes in zwei Kategorien: *Typen* und *Methoden*.

2.1.1 Typen

Ein *Typ* im Methodennetz entspricht einem Datentyp in einer konventionellen Programmiersprache, d. h. einer Beschreibung einer Teilmenge aller möglichen Daten, für die bestimmte Operationen möglich sind. Auf die Regelungstechnik bezogen, können dies grundlegende Rechentypen sein wie z. B. `Matrix` oder komplexere mathematische Objekte wie `DGLSystem`, `ZRLTI`[1], `Sprungantwort` oder `Trajektorienfolgeregler`.
Um später Methoden automatisiert miteinander zu verknüpfen, müssen diese ihre Ein- und Ausgänge in der Signatur hinreichend genau spezifizieren können. Somit werden unpraktikable Kombinationen möglichst ausgeschlossen – und zwar auf Typenebene, nicht erst zur Laufzeit. Für ein System in Zustandsraumdarstellung ist es beispielsweise entscheidend, ob dieses steuerbar und beobachtbar ist und wie viele Ein- und Ausgangsgrößen es besitzt. Man kann sich vorstellen, diese Informationen im Namen des Typs abzubilden (`ZRLTIControllableNotObservableSISO`); es wird aber offensichtlich, dass dieser Ansatz mit zunehmender Menge von Eigenschaften „kombinatorisch explodiert“.
Alternativ erlaubt das Methodennetz deshalb, *Parameter* als Teil des Typs zu spezifieren, auf deren Basis Methoden später Beschränkungen definieren können. Eine beispielhafte Typendefinition ist in Listing 1 dargestellt. Diese wird durch ein verschachteltes, assoziatives Array in YAML-Syntax[2] repräsentiert. Die Schlüssel sind durch Doppelpunkte von den Werten abgetrennt und die Verschachtelung durch Einrückungen realisiert. Der Parameter `Zustandsdimension` kann ganzzahlige Werte annehmen, `Normalform` und `Steuerbarkeit` hingegen nur Werte aus einer diskreten Menge von Optionen (sog. Enumeration)[3].
Diese parametrierten Typen sind mit sog. *dependent types* aus Sprachen wie Agda [12] vergleichbar, erlauben aber keine beliebigen Ausdrücke, sondern nur die angesprochenen Ganzzahlen und Enumerationen.

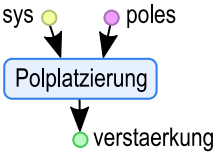
2.1.2 Methoden

Jede *Methode* beschreibt ein regelungstechnisches Verfahren oder einen Teilschritt eines solchen. Neben ihrem Namen spezifiziert eine Methode, welchen Typ die Objekte an den Eingängen besitzen dürfen, und welche Typen sich dafür an den Ausgängen ergeben.
Am Beispiel in Listing 2 wird das Prinzip deutlich. Die Methode besitzt zwei Eingänge mit den Namen `sys` und `poles`[4]. Für diese werden jeweils der erwartete Typ sowie Anforderungen an die Parameter festgelegt. Für die Zustandsdimension und die Anzahl der vorgegebenen Eigenwerte sind keine konkreten Werte vorgegeben, sondern es

```
types:
  ...
  ZRLTI:
    params:
      Zustandsdimension:
        type: Int
      Normalform:
        type: Normalform
      Steuerbarkeit:
        type: Steuerbarkeit
      ...
```

Listing 1 Ausschnitt aus der Definition des Typs `ZRLTI`.

```
methods:
  ...
  Polplatzierung:
    inputs:
      sys:
        type: ZRLTI
        params:
          Zustandsdimension: n
          Steuerbarkeit: <Steuerbar>
      poles:
        type: ListeEigenwerte
        params:
          Laenge: n
        tune: true
    outputs:
      erfolg:
        verstaerkung:
          type: Matrix
          params:
            Zeilen: 1
            Spalten: n
            Rolle: <Zustandsrueckfuehrung>
```



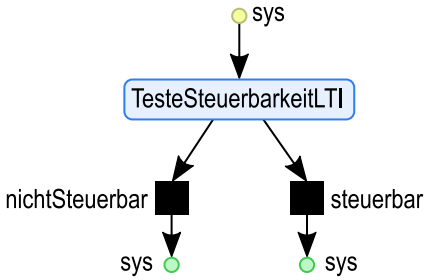
Listing 2 Definition der Methode `Polplatzierung` (YAML-Quelltext und schematische Visualisierung). Diese Methode hat nur eine Ausgangsoption (`erfolg`), vgl. Listing 3.

[1] Lineares zeitinvariantes System in Zustandsraumdarstellung
[2] YAML („Yet Another Markup Language“) ist eine Auszeichnungssprache, zur menschen- und maschinenlesbaren Datenserialisierung.
[3] Erlaubte Werte für diese Enumerationen sind separat definiert.
[4] Dieser Eingang ist als Designparameter (`tune`) markiert, es muss also kein Wert berechnet werden, da dieser vom Nutzer gegeben wird.

Preprint: Only for personal use.

wird ein Platzhalter `n` definiert, auf den sich später bezogen werden kann. Dadurch, dass für beide Werte der gleiche Platzhalter verwendet wird, stellt die Methode außerdem sicher, dass Zustandsdimension und Anzahl der Eigenwerte gleich sind. Über den Typ des Ausgangs `verstaerkung` können Aussagen zur Dimension getroffen werden, wobei hier der Wert von `n` wieder verwendet wird.
Eine andere Besonderheit sind Fallunterscheidungen (nicht zu verwechseln mit mehreren simultanen Ausgängen). Wenn erst zur Laufzeit klar werden wird, welchen Typ die Ausgangsobjekte annehmen, kann das über *Ausgangsoptionen* kenntlich gemacht werden. Die Eigenschaft der Steuerbarkeit beispielsweise existiert – wie oben erläutert – auf Typenebene. Eine Methode, die eine Systemdarstellung auf diese Eigenschaft prüft, kann also nicht im Vorhinein wissen, wie das Ergebnis lautet. Listing 3 stellt dar, wie diese Fallunterscheidung abgebildet wird.

```
methods:
  ...
  TesteSteuerbarkeitLTI:
    inputs:
      sys:
        type: ZRLTI
        params:
          Steuerbarkeit: unset
    outputs:
      steuerbar:
        sys:
          type: ZRLTI
          params:
            Steuerbarkeit: <Steuerbar>
      nichtSteuerbar:
        sys:
          type: ZRLTI
          params:
            Steuerbarkeit: <NichtSteuerbar>
```



Listing 3 Definition der Methode `TesteSteuerbarkeitLTI` (YAML-Quelltext und Visualisierung).

Hier werden zwei Ausgangsoptionen definiert: `steuerbar` und `nichtSteuerbar`. Außerdem wird am Eingang mit dem Schlüsselwort `unset` gefordert, dass dieser Parameter noch nicht gesetzt ist. Dies verhindert später in den Lösungsprozeduren, dass die Prüfung unnötigerweise mehrmals durchgeführt wird, obwohl das System sich nicht verändert hat.

2.2 Inhaltlicher Rahmen

Das bis jetzt im Methodennetz dargestellte Wissen erhebt keinen Anspruch auf Vollständigkeit, sondern soll nur einen groben Rahmen vorgeben, an dem sich die zukünftige Erweiterung orientiert. Deshalb beschränkt sich der Inhalt thematisch hauptsächlich auf endlichdimensionale SISO-Systeme und deren Folgeregelung.
Ein Teil der Typen sind verschiedene Systemdarstellungen: `ZRLTI`, `ZRLTV`, `ZRNichtlinear` und `ÜTF`[5]. Eigenschaften wie Steuerbarkeit, mögliche Normalformdarstellungen oder die Zustandsdimension sind dabei schon im Typ als Parameter abgebildet.
Eine weitere Kategorie besteht aus mathematischen Grundtypen wie `Matrix` und `Zeitfunktion`. Um eine sinnvolle, automatische Verkettung von Methoden zu ermöglichen, erweist es sich als hilfreich, deren semantische Bedeutung auch mit im Typ abzubilden. Dazu kann der Parameter `ZeitfunktionRolle` beispielsweise die Werte `Eingangstrajektorie` oder `Zustandsrückführung` (für eine zeitabhängige Reglerverstärkung) annehmen.
Methoden teilen sich im Wesentlichen auf vier Kategorien auf:

1. **Konvertieren**: Methoden zum „verlustfreien“ Überführen zwischen verschiedenen Darstellungen wie `ÜTFzuZR` sowie Approximationen wie `Linearisieren`.
2. **Prüfen**: Tests auf Systemeigenschaften, die dann im Typ abgebildet werden, z. B. `TesteSteuerbarkeitLTI`, `TesteTeilerfremdheitÜTF`.
3. **Entwerfen**: Verfahren wie `Polplatzierung`, `ParametriereZieglerNichols` oder `EKFEntwurf`.
4. **Zusammensetzen**: Methoden, die Objekte zu einem funktionsfähigen Ganzen zusammenfügen, z. B. `BaueTrajektorienfolgeregler`, `BaueLuenbergerBeobachter`.

3 Aufgabenprozeduren

Die zweite Art von Graph – neben dem Methodennetz – ist die *Aufgabenprozedur*. Diese wird für eine konkrete Aufgabe aus dem Methodennetz generiert und ähnelt konzeptionell einem Programmablaufplan der herkömmlichen Programmierung. Sie beschreibt, welche Methoden in welcher Reihenfolge welche Objektinstanzen verarbeiten und welche Zwischenergebnisse damit entstehen.

3.1 Aufbau

Ausgangspunkt jeder Aufgabenprozedur sind

1. Eingangsspezifikation: Beschreibung von *Objekten*, die die gegebenen Informationen repräsentieren, sowie

[5] `ZRLTV` bezeichnet eine lineare zeitvariante Zustandsraumdarstellung. `ZRNichtlinear` meint hier „nicht zwangsläufig linear“; damit ist es die allgemeinste Systemdarstellung. `ÜTF` steht für Übertragungsfunktion.

Preprint: Only for personal use.

2. Ausgangsspezifikation: eine Beschreibung der gesuchten Informationen in Form eines Zielobjektes.

Objekte besitzen jeweils einen eindeutigen Bezeichner und einen konkreten Typ, dessen Parameter konkrete Werte enthalten (siehe Abschnitt 2.1.1).
Eine gültige Aufgabenprozedur ist ein gerichteter Graph in dem sich Methodenaufrufe mit (Zwischenergebnis-)Objekten abwechseln. Den Abschluss bildet (mindestens) ein Objekt, was der Zielspezifikation entspricht.
Eine exemplarische Aufgabenprozedur, die die Methoden und Typen des letzten Abschnitts verknüpft, ist in Abbildung 1 dargestellt.

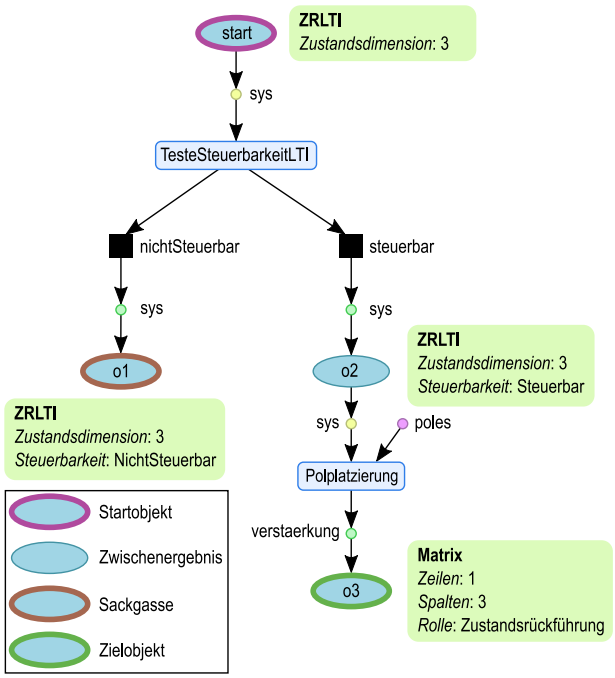


Abbildung 1 Eine beispielhafte Aufgabenprozedur, bestehend aus zwei Methodenaufrufen. Ein LTI-System in Zustandsraumdarstellung wird erst auf Steuerbarkeit überprüft, dann wird eine Rückführung entworfen, die die Pole wie gewünscht platziert. Ellipsen stehen für Objekte, abgerundete Rechtecke für Methodenaufrufe, Kreise für Ein- und Ausgänge, Quadrate für Ausgangsoptionen.

Hervorzuheben ist die zusätzliche Komplexität bei Methoden mit mehreren Ausgangsoptionen. Beim Abarbeiten der Aufgabenprozedur zur Laufzeit wird in diesen Methoden eine Entscheidung getroffen und nur der gewählte *Entscheidungsast*[6] weiter verfolgt. Deshalb dürfen bei einem Methodenaufruf immer nur Objekte verarbeitet werden, die auf dem gleichen Ast von Entscheidungen liegen; siehe Abb. 2.

3.2 Generieren der Aufgabenprozedur

In der aktuellen Implementierung [6] wird eine Aufgabenprozedur aus dem Methodennetz in zwei Schritten erzeugt:

1. Mittels Tiefensuche werden alle möglichen zu den Startobjekten passenden Methodenaufrufe angelegt, die neue Objekte erzeugen. Für neu geöffnete Entscheidungsäste wird diese Suche rekursiv ausgeführt. Die Suche bricht ab, wenn keine neuen Objekte mehr erzeugt werden können.
2. Alle Methodenaufrufe, die nicht zu einem Zielobjekt führen, werden entfernt.

[6] Dieser kann sich durchaus weiter verzweigen.

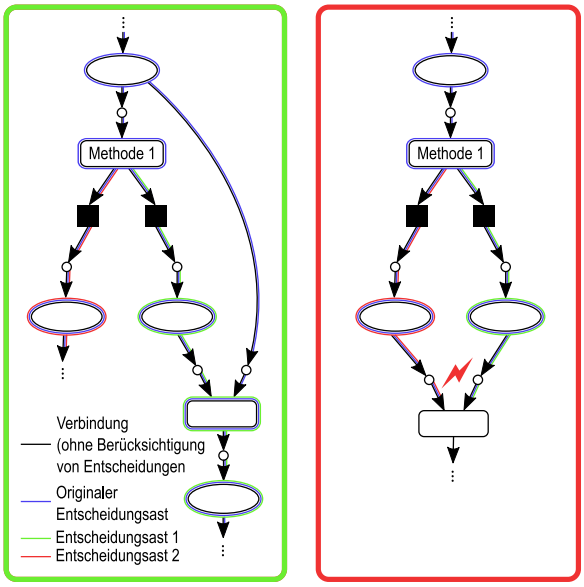


Abbildung 2 Erlaubte und nicht erlaubte Verbindungen. *Alle* Objekte und Methoden liegen auf dem ursprünglichen Entscheidungsast (blau). Methode 1 trifft Entscheidung zwischen zwei Ausgangsoptionen. Dadurch entstehen zwei neue Entscheidungsäste (rot und grün). Diese dürfen nachfolgend nicht kombiniert werden (rechts), Elemente des blauen Asts dürfen in beiden verwendet werden (links).

3.3 Beispiel: Trajektorienfolgeregelung für Dreifachpendel

In [7] wird ein Versuchsstand betrachtet, der aus einem Schlitten, der auf einer Schiene horizontal angetrieben wird und drei passiven Pendelgliedern, die über Lager am Schlitten befestigt sind, besteht, siehe Abb. 4. Weiterhin ist in [8] ein Video zu sehen, in dem das Dreifachpendel vom energieärmsten Zustand (alle Pendelglieder hängen nach unten) in die vollständig aufgerichtete Konfiguration (alle Pendelglieder stehen aufrecht) überführt wird.
Das Anwendungsszenario für das Methodennetz sei nun wie folgt: Ein grundsätzlich mit der Regelungstechnik vertrauter Nutzer möchte die gesehenen Ergebnisse reproduzieren und interessiert sich deshalb für die detaillierten Lösungsschritte. Die Bewegungsgleichungen des Systems werden als gegeben angenommen. Als Suchanfrage an das Methodennetz gibt er also folgende Start- und Zielspezifikation an:

```
start:
  type: DGLSystem
  params:
    Linear: <NichtLinear>
target:
  type: Trajektorienfolgeregler
```


Preprint: Only for personal use.

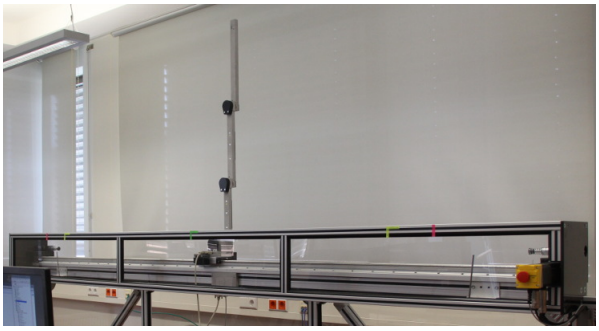


Abbildung 4 Dreifachpendelversuchsstand, bestehend aus aktuiertem Schlitten und drei passiven Pendelgliedern. Das Foto zeigt das System während die obere Ruhelage stabilisiert wird.

Damit wird automatisch eine Aufgabenprozedur generiert, die im rechten Teil von Abb. 3 dargestellt ist.
Es wurde also korrekt geschlossen, dass das System steuerbar[7] und beobachtbar sein muss, eine Trajektorie geplant werden muss, man das System entlang dieser Trajektorie linearisieren kann um einen Folgeregler zu entwerfen, und schlussendlich ein Beobachter benötigt wird. Dass man die Prüfung auf Steuerbarkeit und Beobachtbarkeit vertauschen kann, ist im Suchalgorithmus noch nicht abgebildet, weshalb dieser in der jetzigen Version zwei separate Pfade generiert.
Im Beispielszenario geht für den Nutzer aus dem Ergebnis hervor, dass die meisten Methoden prinzipiell bekannt sind, aber bezüglich der optimierungsbasierten Trajektorienplanung noch Recherchebedarf besteht.

[7] Die Verallgemeinerung von Steuerbarkeit auf Stabilisierbarkeit ist noch nicht im Methodennetz hinterlegt, siehe Abschnitt 2.2.

4 Softwareumsetzung

Zum Erstellen und Erkunden des Methodennetzes, sowie zum Generieren von Aufgabenprozeduren wurde eine prototypische Webanwendung implementiert [5, 6]. Diese basiert auf dem Python-Framework *Django*, sowie der Javascript-Bibliothek *vis.js*. Die Elemente des Methodennetzes sind in YAML spezifiziert. Der Nutzer kann diese im Webbrowser als Graph betrachten und Anfragen wie in Abschnitt 3.3 stellen, woraufhin die entsprechende Aufgabenprozedur generiert wird. Ein Screenshot der Webanwendung ist in Abb. 3 zu sehen.

5 Semantische Interpretation

Das beschriebene Methodennetz repräsentiert regelungstheoretisches Wissen in einer auf einen konkreten Anwendungsfall zugeschnittenen Weise: Für gegebene Ein- und Ausgangsspezifikationen kann eine zielführende Abfolge von Zwischenschritten (die Aufgabenprozedur) bestimmt werden. Um die in der Einleitung skizzierten grundsätzlichen Wissenstransferprobleme mit Hilfe (teil-)automatisierter Assistenzsysteme zu überwinden, ist dies aber noch nicht ausreichend. Beispielsweise ist im Methodennetz die *Bedeutung* der einzelnen Begriffe (z. B. „steuerbar“, „Zustand“) nicht abgebildet. Es ist daher erstrebenswert, das Methodennetz mit einer standardisierten semantischen Wissensrepräsentation zu verknüpfen.
Die gegenwärtig zur formalen Wissenrepräsentation verbreitetste Methode sind sog. semantische Tripel (Subjekt-Prädikat-Objekt) und daraus zusammengesetzte *Ontologi-*

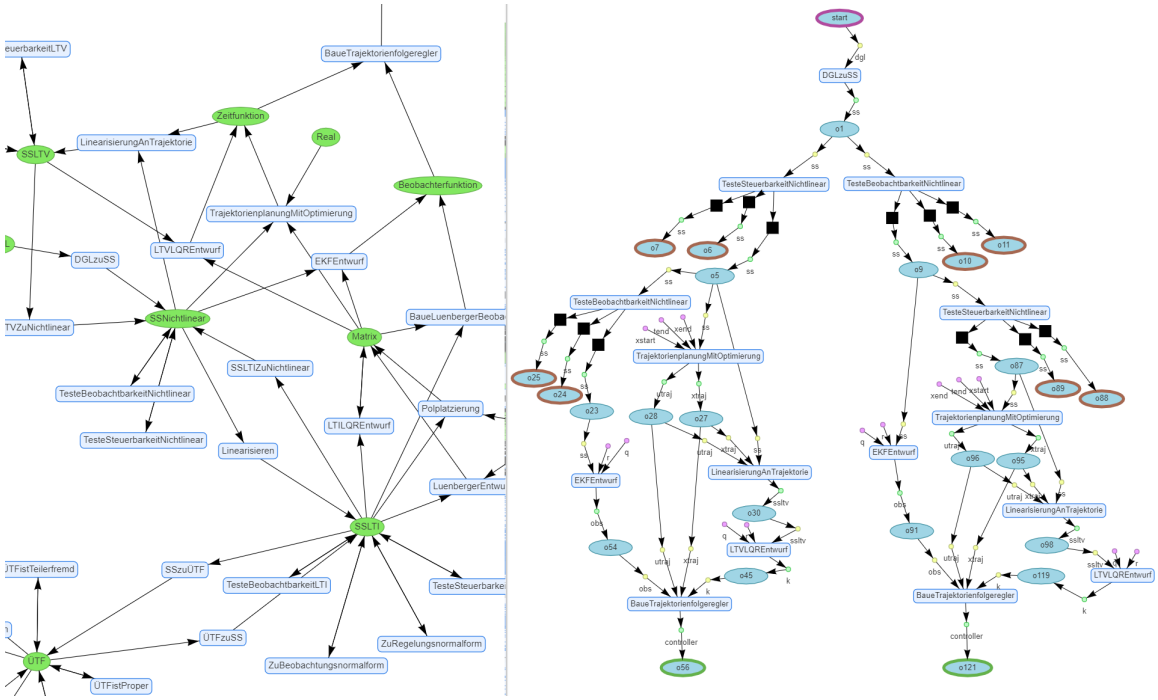


Abbildung 3 Screenshot der Webanwendung [5] (nur Ergebnisbereich). Links: Ausschnitt des Methodennetzes; Rechts: Aufgabenprozedur für Anwendungsbeispiel Dreifachpendelversuchsstand. Unter [6] ist die Abbildung in Originalauflösung verfügbar.

en. Darunter versteht man im informatischen Kontext eine *explizite formale* (d. h. maschinenlesbare) *Spezifikation einer geteilten* (d. h. konsensfähigen) *Konzeptualisierung* (begrifflichen Abdeckung) einer Wissensdomäne [4]. Im engeren Sinn wird damit ein in der *Web Ontology Language (OWL)* repräsentierter Datensatz bezeichnet.
Ontologische Wissenrepräsentation ist bisher insbesondere in der Medizin und Biologie weit verbreitet, siehe z. B. [3]. Für die Ingenieurwissenschaften und speziell für die Regelungstheorie sind den Autoren für den vorliegenden Anwendungsfall keine direkt nutzbaren Ontologien bekannt.

Als Ausgangspunkt für weitere Entwicklungen wird daher eine aus dem Methodennetz automatisch generierte und mit zusätzlichen Informationen angereicherte „Ontologie Regelungstechnischer Methoden“ (ORM) vorgeschlagen. Diese spezialisiert Konzepte aus der Basic Formal Ontology (BFO) [1], was eine zukünftige Interoperabilität mit anderen Ontologien erleichtert. Die ORM erlaubt den Einsatz der standardisierten Abfragesprache SPARQL um konkrete (abgeleitete) Fakten aus der Wissensbasis zu extrahieren, die nur implizit im Methodennetz repräsentiert sind. Als einfaches Demonstrations-Beispiel ist in [6] die Abfrage implementiert, welche der Methoden eine LTI-Zustandsraumdarstellung als Eingang verarbeiten. Auf Basis einer Volltextsuche wäre selbst diese simple Frage nur mit erheblichem manuellem Zusatzaufwand zu beantworten.

6 Zusammenfassung und Ausblick

Das unter [5, 6] verfügbare Methodennetz formalisiert regelungstechnische Verfahren sowohl menschen- als auch maschinenlesbar. Für eine konkrete Problemstellung kann mittels Tiefensuche eine sog. Aufgabenprozedur generiert werden, die dann algorithmisch abarbeitbar ist. Zudem dient das Methodennetz als Basis der Ontologie Regelungstechnischer Methoden (ORM), welche die repräsentierten Verfahren auf semantischer Ebene ergänzt.
Diese formale Repräsentation bietet durch zusätzliche Recherchemöglichkeiten das Potenzial, den inner- unter interdisziplinären Wissenstransfer zu verbessern. Die hier vorgeschlagenen Ansätze sind dabei nicht als fertige Lösungen sondern als Diskussionsbeitrag aufzufassen.

Das Fernziel ist dabei ein Assistenzssystem, das eine Aufgabensituation inhaltlich erfassen, Fragen beantworten und bei Entwurfsentscheidungen und Rechnungen unterstützen kann. Ein plausibler Zwischenschritt dazu ist die Integration der abstrakt repräsentierten Methoden mit den in [10] hinterlegten Beispielimplementierungen.

Danksagung

Die Autoren bedanken sich bei Johannes Bender für die Unterstützung bei der Literaturrecherche.

Literatur

[1] Robert Arp, Barry Smith und Andrew D. Spear. *Building ontologies with basic formal ontology*. MIT Press, 2015.

[2] Carmen Benavides u. a. „An ontology-based approach to knowledge representation for Computer-Aided Control System Design“. In: *Data & Knowledge Engineering* 118 (2018), S. 107–125.

[3] Christophe Dessimoz und Nives Škunca, Hrsg. *The Gene Ontology Handbook*. Springer New York, 2017.

[4] Nicola Guarino, Daniel Oberle und Steffen Staab. „What is an ontology?“ In: *Handbook on ontologies*. Springer Berlin, 2009, S. 1–17.

[5] Robert Heedt und Carsten Knoll. *Demoinstanz des Methodennetzes*. 2021. URL: https://methodnet.ackrep.org/.

[6] Robert Heedt und Carsten Knoll. *Methodennetz-Quellcode-Repositorium*. 2021. URL: https://github.com/TUD-RST/methodnet.

[7] Institut für Regelungs- und Steuerungstheorie, TU Dresden. *Versuchsstände am Institut – Technischer Bericht*. 2020. URL: https://github.com/TUD-RST/technical-reports.

[8] Institut für Regelungs- und Steuerungstheorie, TU Dresden. *Video des Aufschwingvorgangs für den Dreifachpendelversuchsstand*. 2021. URL: https://videocampus.sachsen.de/video/Swing-Up-and-stabilization-of-a-triple-pendulum-on-a-movable-cart/a4b68ff99aed87c9a095f322002ac7b7.

[9] Maria Keet. *An introduction to ontology engineering, v1.5*. College Publications, 2020.

[10] Carsten Knoll und Robert Heedt. „‚Automatic Control Knowledge Repository‘ – A Computational Approach for Simpler and More Robust Reproducibility of Results in Control Theory“. In: *24th International Conference on System Theory, Control and Computing (ICSTCC)*. 2020, S. 130–136.

[11] Jan Lunze. *Künstliche Intelligenz für Ingenieure: Methoden zur Lösung ingenieurtechnischer Probleme mit Hilfe von Regeln, logischen Formeln und Bayesnetzen*. 3. Auflage. Walter de Gruyter Berlin, 2016.

[12] Ulf Norell. „Dependently Typed Programming in Agda“. In: *Proceedings of the 6th International Conference on Advanced Functional Programming*. AFP'08. Springer Berlin, Heidelberg, 2008, S. 230–266.

Preprint: Only for personal use.