Leichtgewichtige prototypische Mechanismensimulation im Web-Kontext

Lightweight prototypal mechanism simulation on the web

Pascal, Schnabel*; Stefan,Goessner**

***** TU Chemnitz, Professur Montage- und Handhabungstechnik
pascal.schnabel@mb.tu-chemnitz.de

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

Kurzfassung

Die Möglichkeiten zur Simulation und Analyse ebener Mechanismen mithilfe von Webanwendungen sind bislang sehr stark begrenzt.

Aus diesem Anlass wird ein neuer Ansatz zur Modellierung und Simulation planarer Mechanismen mithilfe von *Nodes* und *Constraints* vorgestellt. Im Anschluss folgt die Vorstellung der eigens entwickelten Javascript-Simulationsbibliotheken sowie eine Erklärung zu deren Anwendung an einer umfangreichen Anzahl von Beispielen. Abschließend wird durch einige komplexere Beispiele die Mächtigkeit des Verfahrens sowie der entwickelten Bibliothek demonstriert.

Abstract

1 Impulsbasierter Ansatz

2 Implementierung

Zur Implementierung des impulsbasierten Simulations- und Analysesystems werden ausschließlich die Standard-Webtechnologien HTML, CSS und Javascript sowie die erstellten Javascript-Bibliotheken *g2* und *cstr* verwendet. Am Beispiel einer Kurbelschwinge soll nun deren Verwendung exemplarisch erklärt werden.

Zunächst wird ein neues *HTML*-Dokument mit einem entsprechendem Dokumentkopf erstellt. Im Kopf des Dokuments können Informationen über den Autor, der Darstellung der Seite, der Titel der Seite und weitere Angaben definiert werden.

```
<!doctype html>
<html lang="en">

<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Titel des Dokuments</title>
</head>
```

Abb. 1: *HTML*-Dokumentkopf

Im Anschluss folgt die Definition der Zeichenfläche mithilfe des HTML-eigenen c*anvas*-Elements und zusätzliche Gestaltung der *HTML*-Seite durch weitere Elemente. Danach folgt die eigentliche Programmlogik in einem neuen Programmblock (*script*-Element). In diesem muss zunächst anhand der *id* des *canvas* auf die Zeichenfläche zugegriffen werden. Als Nächstes wird ein neues Interaktionsobjekt aus der *g2*-Bibliothek erstellt, dass später die grafische Animation und Interaktion erlaubt. Danach wird ein Selektionsobjekt erstellt, das das interaktive Ziehen und Bewegen („drag“) der *Nodes* ermöglicht.

Anschließend folgt die Definition der *Nodes*, die später die Rolle der Drehgelenke bzw. der Koppelpunkte des Koppelgetriebes. Für jeden *Node* kann durch die *x*- und *y*-Attribute die Ausgangsposition, die Masse mit dem Attribut *m* und eine Beschriftung der *Nodes* mit dem *label*-Attribut angegeben werden. Des Weiteren ist es möglich gestellfeste *Nodes* durch das *base*-Attribut zu kennzeichnen. In diesem Fall erhalten die *Nodes* eine unendliche Masse.

Danach folgt die Definition der *Constraints* zwischen den einzeln *Nodes*. Hierfür wird zunächst mit der Funktion *cstr()* ein neues

Constraint-Objekt erstellt und mithilfe der Funktionen *n2()* eine Bindung zwischen zwischen zwei *Nodes* definiert. Jedem *Constraint* wird durch die Attribute *n1* und *n2* ein Start- und Endnode zugewiesen und durch die Attribute *len* (Länge) und/oder *ang (*Winkel*)* die Art der Bindung zwischen den einzelnen Nodes definiert. Durch zusätzliche Vergabe des *id*-Attributs ist es zu einem späteren Zeitpunkt möglich erneut auf den *Constraint* zuzugreifen. Ist es erforderlich einen weiteren *Node K* mit einem festen Abstand zu den *Nodes A* und *B* zu definieren, kann mithilfe der Funktion *n3()* ein *Constraint* zwischen drei *Nodes* oder zwei einzelne *n2-Constraints* erstellt werden. Hier ist darauf hinzuweisen, dass die Verwendung zwei einzelner *n2-Constraints* immer einem einzelnen *n3-Constraint* vorzuziehen ist und deutlich bessere Berechnungsergebnisse liefert.

```
<body>
    <canvas id="c" width="900" height="800"></canvas><!--Zeichenfläche-->
    <script src="./g2.js"></script>    <!--Verweis auf Grafikbibliothek-->
    <script src="./cstr.js"></script>    <!--Verweis auf Constraintbibliothek-->
    <script>
        //Zugriff auf canvas-Element
        const ctx = document.getElementById('c').getContext('2d');
        //Interaktionsobjekt erstellen
        const interactor = canvasInteractor.create(ctx, { x: 140, y: 140,
cartesian: true });
        // Selektionsobjekt
        const selector = g2.selectorHdl(interactor.evt);

        //Definition der Nodes
        const A0 = { x: 0, y: 0, base: true, label: 'A0' },
            A = { x: 0, y: 50, m: 2, label: 'A' },
            B = { x: 180, y: 110, label: 'B' },
            B0 = { x: 200, y: 10, base: true, label: 'B0' },
            K = { x: 90, y: 120, label: "K" };

        //Definition der Constraints
        const c = cstr().n2({ id: 'A0A', n1: A0, n2: A, len: 'const' })
            .n2({ n1: A, n2: B, len: 'const' })
            .n2({ n1: B0, n2: B, len: 'const' })
            .n2({ n1: A, n2: K, len: 'const' })
            .n2({ n1: B, n2: K, len: 'const' })
```

Abb. 2: Definition der *Nodes* und *Constraints*

Im Anschluss folgt das Zeichnen des Getriebes auf der Zeichenfläche. Hierfür werden die Funktionen der *g2*-Bibliothek, einer 2D-Grafik Bibliothek, basierend auf dem *Command-Pattern*, eingesetzt. Zuerst wird durch Aufruf der Funktion *g2()* ein neues *g2*-Objekt erstellt, die Zeichenfläche durch die Funktion *clr()* "gereinigt" und mithilfe der

Funktion *view()* eine Koordinatentransformation der Zeichenfläche definiert. Im Anschluss folgt mit den Funktionen *lin(), ply()* und *nod()* das Zeichnen des Getriebes durch Linien, Polgone und Kreise. *Nodes* die später mit der Maus interaktiv bewegbar sein sollen, werden durch die Funktion *hdl()* gezeichnet. Weitere verwendete Symbole bzw. Funktionen sind im Quelltext durch Kommentare erläutert. Intern erstellt das *g2*-Objekt nun eine Befehlswarteschlange (*Command-Queue*) mit einem Zeiger (*Pointer*) zu den auszuführenden Funktionen. Erst durch Aufruf der Funktion *exe(ctx)* erfolgt das eigentliche Zeichnen des Getriebes.

Abschließend erfolgt die Definition der Interaktions- und Animationsfunktionen mithilfe des *Interactor*-Objektes. Dieses ermöglicht es, sobald bestimmte Ereignisse, sogenannte *Pointer*-Events auftreten eine grafische Änderung eines oder mehrerer *canvas*-Elemente durchzuführen. Im Codeblock *on('tick', (e)=>{...})* wird zunächst eine Funktion definiert, die bis zu 60x pro Millisekunde ausgeführt wird. Im Anschluss wird durch die Funktionen *on('drag', (e)=>{...})* das interaktive Verschieben der durch die *hdl()*-Funktion gezeichneten *Nodes* ermöglicht und abschließend durch Aufruf der Funktion *startAnimation()* die Animation gestartet. Genauere Erklärungen zur Funktionsweise der beiden Bibliotheken können den Dokumentationen entnommen werden [2],[3].

```
        //Zeichnen des Getriebe
        const g = g2().clr()
            .view(interactor.view)
            .lin({ p1: A0, p2: A, lw: 2 }) //Linie
            .lin({ p1: B0, p2: B, lw: 2 })
            .ply({ pts: [A, B, K], lw: 2, ls: 'darkslategray', closed: true })
//Polygon
            .gnd(A0) //Symbol für gestellfesten Node
            .gnd(B0)
            .hdl(A) //"Handle" - Symbol
            .nod(B)
            .nod(K) //"Node" Symbol
        g.exe(ctx);

        //Definition der Interaktions- und Animationsmethoden
        interactor
            .on('tick', (e) => {
                const itr = c.correct();
                g.exe(selector).exe(ctx);
            })
            .on('pan', (e) => { interactor.view.x += e.dx; interactor.view.y +=
e.dy; })
            .on('drag', (e) => {
                if (selector.selection && selector.selection.drag) {
                    selector.selection.drag({ x: e.xusr, y: e.yusr, dx: e.dxusr,
dy: e.dyusr, mode: 'drag' });
                }
            })
            .startTimer();
    </script>
</body>
</html>
```

Abb. 3: Zeichnen des Getriebes und Definition der Interaktions- und Animationsfunktionen

Das Ergebnis des beschriebenen Beispiels ist die in Abb. 4 a) dargestellte Kurbelschwinge. Wird eine andere bzw. zusätzliche Gestaltung der Zeichenfläche gewünscht, können weitere Funktionen der *g2*-Bibliothek genutzt und/ oder eigene Symbole erstellt werden. Hierfür sei auf die ausführliche Dokumentation verwiesen [2].

Gegenüber anderen Programmen zur Mechanismensimulation, die auf der Zeichenfolge-Rechenmethode basieren, wie Geogebra, begnügt sich das Vorgehen mit einer sehr geringen Dateigröße und ist äußerst perfomant. Zum Vergleich: Zur reinen Modellierung der gleichen Kurbelschwinge in Geogebra ist ein zusätzliches Programm und ebenfalls 10 Befehle notwendig, während die eigentliche Definition einer Kurbelschwinge mit dem impulsbasierten Verfahren nur

10 Befehle und kein zusätzliches Programm bedarf. Des Weiteren besitzt die *HTML*-Datei der definierten Kurbelschwinge mit 3*kB* eine deutlich geringere Dateigröße gegenüber der Geogebra-Datei mit 15*kB* und kann direkt auf Webseiten eingebunden werden.

3 Beispiele

Mit dem beschriebenen Verfahren lassen sich alle einschleifigen Koppelgetriebe (Abb. 4) problemlos und fehlerfrei simulieren. Des Weiteren ist die Animation deutlich komplexerer Getriebestrukturen, wie das der 14-gliedrigen-Tiefziehkurbelpresse (Abb. 7a) ebenfalls fehlerfrei möglich.

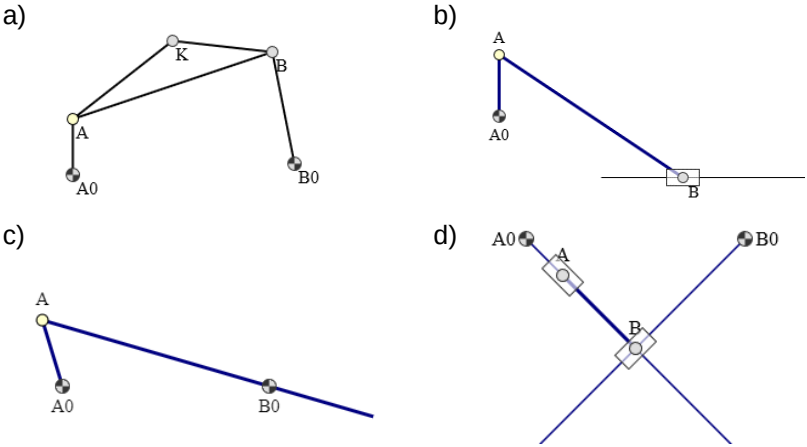
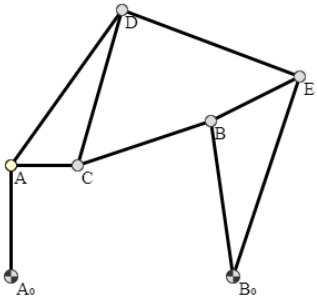


Abb. 4: Kurbelschwinge, exzentrische Schubkurbel, Kurbelschleife, Kreuzschubkurbel

Der Stephenson II Mechanismus (Abb. 5a) ist bekanntlich analytisch nicht lösbar. Das beschriebene iterative Lösungsverfahren hingegen führt auch in solchen Fällen zuverlässig zu einer Lösung. Auch die Veranschaulichung wichtiger Lehrsätze aus dem Bereich der Mechanismentechnik ist möglich. Der Satz von Bobillier [5] ermöglicht die Ermittlung des Wendekreises und wurde als grafische interaktive Simulation mit dem Verfahren aufbereitet (Abb. 5b).

a) b)



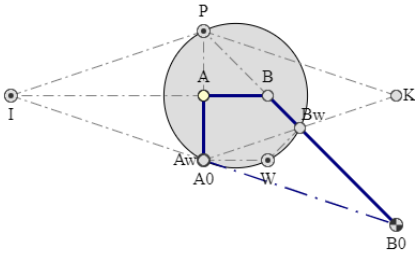
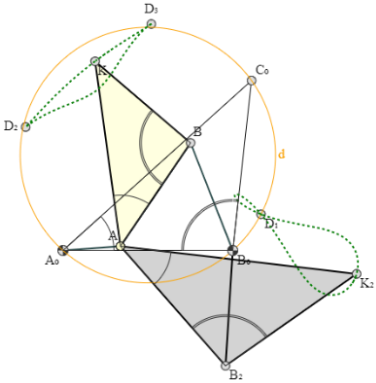


Abb. 5: Stephenson II Mechanismus; Satz von Bobillier

Wird das Interaktionsobjekt um einige Funktionen erweitert können die Bahnkurven einzelner *Nodes* gezeichnet und zusätzliche Informationen wie der Drehwinkel der Kurbel oder dem Winkel zwischen Koppel und Schwinge angezeigt werden. In Abb. 6a) wurde das Verfahren genutzt, um die zweiteilige Koppelkurve einer Kurbelschwinge zu veranschaulichen und in Abb. 6b) den Übertragungswinkel zwischen Koppel und Schwinge darzustellen.

a) b)



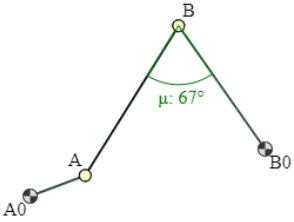


Abb. 6: zweiteilige Koppelkurve, Übertragungswinkel

Werden zusätzliche Grafiksymbole benötigt, kann die *g2*-Bibliothek um einige benutzerdefinierte Symbole und Befehle erweitert werden. Zur grafischen Gestaltung der Getriebe in Abb. 7 wurde die Bibliothek um einige zusätzliche mit den an der Professur Montage und Handhabungstechnik häufig verwendeten Getriebesymbolen erweitert. In Abb. 7a) ist der Mechanismus einer Tiefziehkurbelpresse und in Abb. 7b) ein Textilmaschinenmechanismus mit langer Rast dargestellt.

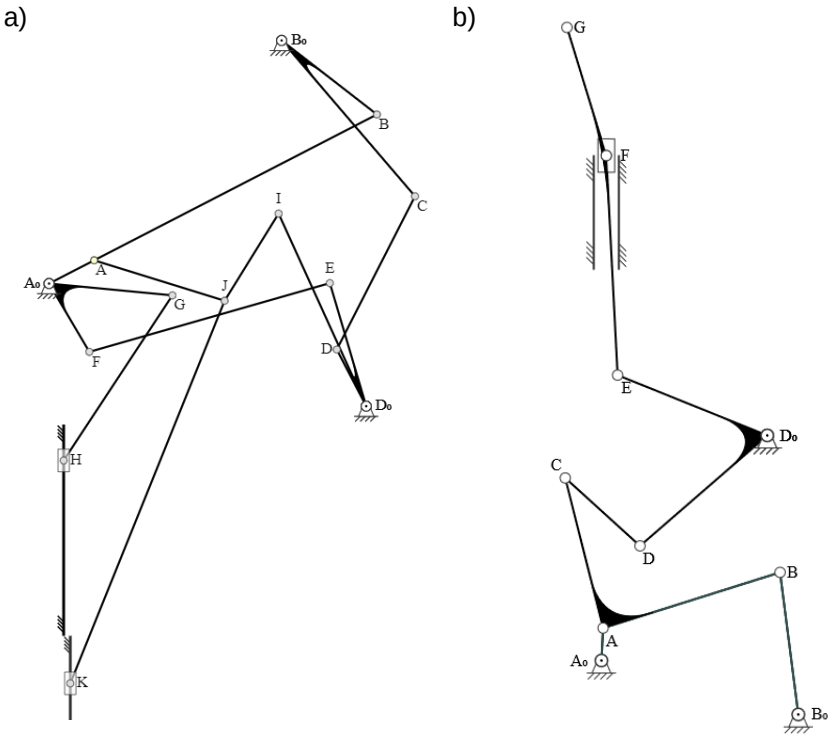


Abb. 7: 14-gliedrige-Tiefziehkurbelpresse, Textilmaschine

Alle hier vorgestellten Beispiele mit Quellcode können [4] entnommen werden.

4 Zusammenfassung und Ausblick

Das vorgestellte impulsbasiert Verfahren zur Modellierung von Mechanismen mittels massebehafteter Nodes (Partikel) und vektorieller Constraints zwischen ihnen, beweist sich als möglicher Weg und bietet einige Vorteile gegenüber dem klassischen Weg. Die vorgestellten Beispiele beweisen die sehr einfache Verwendung der zwei entwickelten Javascript-Bibliotheken, zur Simulation verschiedener Koppelmechanismen mithilfe des impulsbasierten Ansatzes. Gleichzeitig können die Simulationen direkt im Web-Umfeld eingesetzt werden.

Aktuell ist die Entwicklung an beiden Bibliotheken noch nicht komplett abgeschlossen und die Vervollständigung der Anwenderdokumentation ausstehend. Insgesamt beweisen die erstellten Beispiele eindrucksvoll die sehr einfache Anwendung und machen zuversichtlich, zukünftig Constrainttypen, wie Seil- und Kurvenbindung implementieren zu können. Des Weiteren sollen weitere Animationen, wie der Drahzahlplan nach Kutzbach oder der Satz von Roberts für die Verwendung in der Lehre erstellt werden.

Literatur

[1] Gössner, S., "Fundamentals for Web-Based Analysis and Simulation of Planar Mechanisms", EuCoMes 2018, Proceedings of the 7th European Conference on Mechanism Science, Springer (2018).

[2] Gössner, S., "g2", URL: http://goessner.github.io/g2/.

[3] Gössner, S., "Make your HTML canvas Interactive", URL: https://goessner.github.io/canvasInteractor/.

[4] Schnabel, P., “Beispiele Getriebetagung 2022”, URL: https://github.com/Pasquale19/Beispiele_GT2022.

[5] Gössner, S. Mechanismentechnik. Vektorielle Analyse ebener Mechanismen. Logos Verlag Berlin, 2017, ISBN 978-3-8325-4362-4.