Abschlussprojekt - WS2024/25

Julian Huber & Matthias Panny

Abschlussprojekt

- Gruppenarbeit mit bis zu 2 Personen
- Versionsverwaltung mit git
- Umsetzen des Projektes, welches wir in diesen Einheiten besprechen werden
- Abgabetermin: 27.02.2025 um 23:55 Uhr

Umfang des Abschlussprojektes

- 35h pro Person * 2 Personen pro Gruppe = 70h pro Gruppe
- 35h pro Person setzt sich aus folgenden Annahmen zusammen:
  - gemeinsame LV: 60UE á 45min = 45h
  - Vor- & Nachbereitung inkl. 15 HÜ: 1,5h pro 2 UE = 45h
  - Kursumfang: 5 ECTS = 125h
  - 125h - 45h - 45h = 35h

Abschlussprojekt

- In den nächsten Folien wird ein Projekt inkl. minimaler Umsetzung/Anforderungen vorgestellt
- Diese minimale Umsetzung entspricht 60% der Gesamtpunkte
- Die restlichen 40% können durch Erweiterungen erreicht werden
- Von diesen max. 100% werden die Minuspunkte der Hausübungen abgezogen

Aufgabenstellung

- Eine Anwendung mit Web-UI (`streamlit`) soll entwickelt werden
- Die Umsetzung erfolgt in Python mit Objektorientierung
- In der Anwendung können beliebige ebene Mechanismen (mit gewissen Einschränkungen) definiert & deren Kinematik simuliert werden

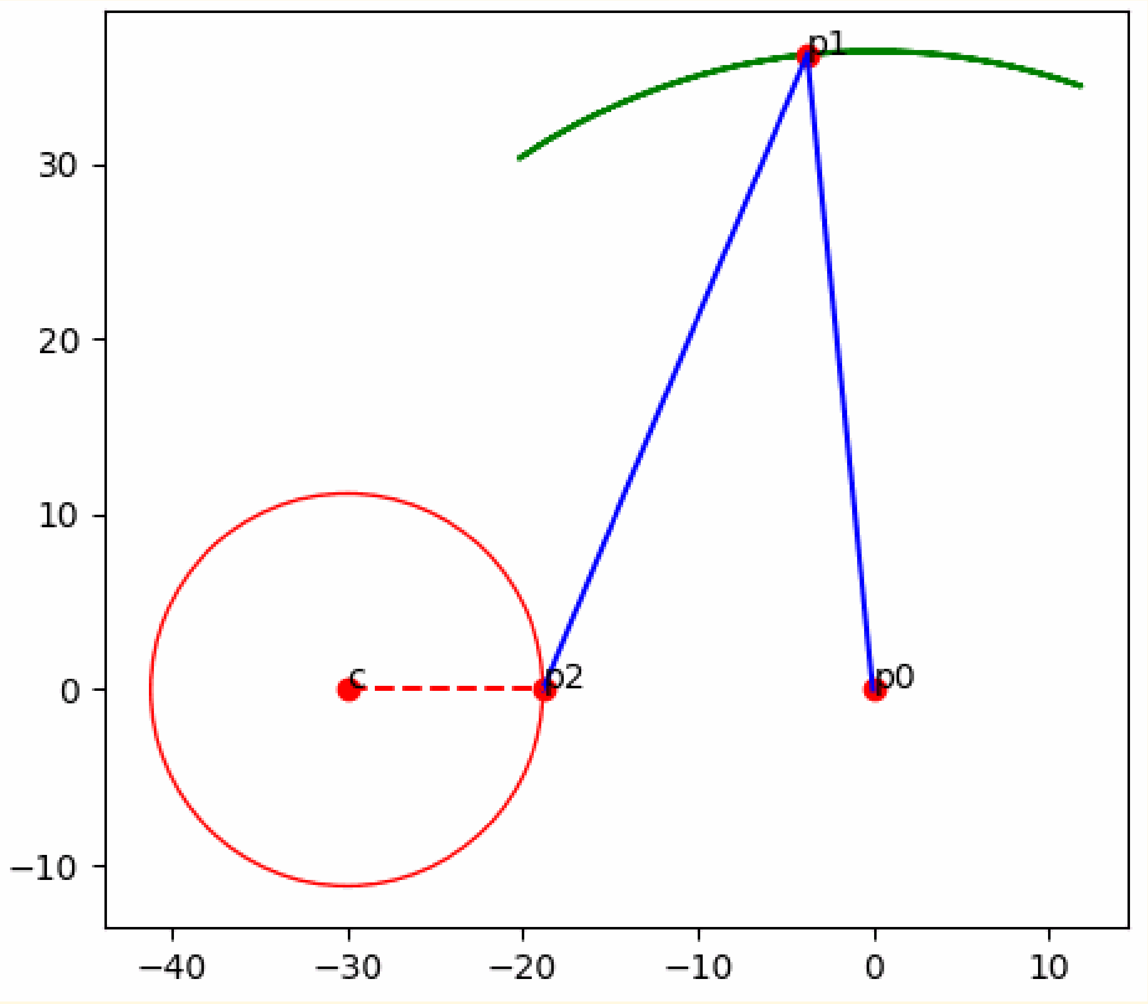
Alternativen zu diesem Projekt werden zum Schluss diskutiert

Funktionsweise

(Ebene Mechanismen & deren Simulation)

Ebene Mechanismen

- Ebene Mechanismen (oft auch ebene Koppelgetriebe genannt) sind Mechanismen aus mehreren Gliedern die mit Dreh- und Schubgelenken verbunden sind
- Erzeugen Bewegungsabläufe in einer Ebenen → hängt von der Anzahl/Konfiguration der Gelenke & den kinematisch relevanten Abmessungen ab
- Ein Viergelenkgetriebe ist ein einfaches Beispiel für einen ebenen Mechanismus



Ebene Mechanismen

- Wir wollen nun solch einen ebenen Mechanismus simulieren
- In unserem Modell besteht ein Mechanismus aus n Gelenken ("Punkten") und m Gliedern ("Stangen")
- Wir wollen unseren Mechanismus immer für einen bestimmten/konstanten Drehwinkel θ betrachten

⚠️ Einschränkungen

- Es werden nur ebene Mechanismen betrachtet
- Nur Drehgelenke verbunden mit starren Gliedern
- Nur Drehgelenk (1 rot. Freiheitsgrad) als Antrieb $\rightarrow$ ein Gelenk des Mechanismus bewegt sich auf einer Kreisbahn
- Ein Gelenk des Mechanismus ist statisch/fest verankert

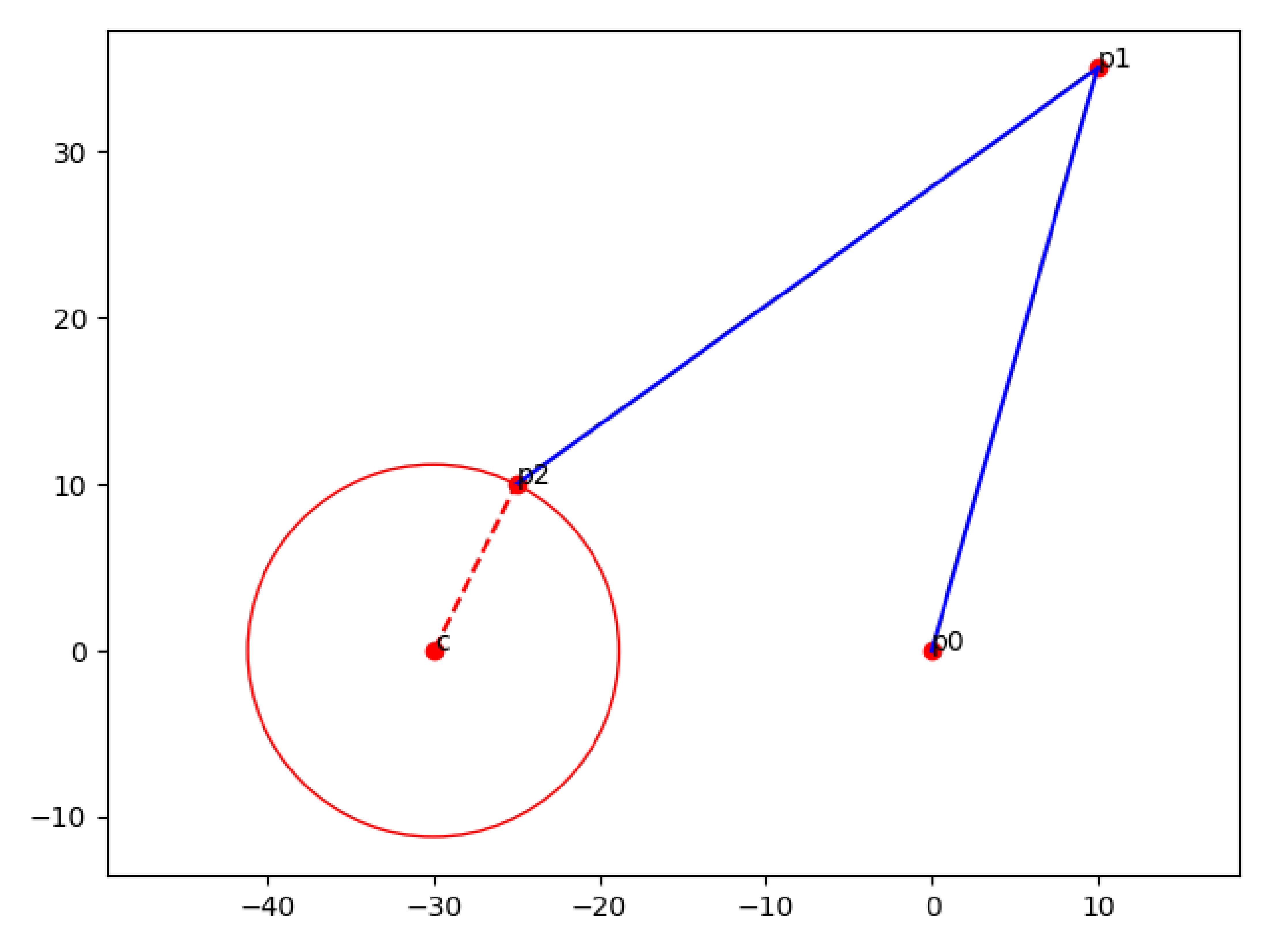
Mathematische Modellierung

- Zur Simulation muss unser Mechanismus in ein mathematisches Modell überführt werden
- Im allgemeinen hat ein Punkt in der Ebene 2 Freiheitsgrade (x- und y-Koordinate) → unser Modell soll beschreiben wie diese Freiheitsgrade entzogen werden

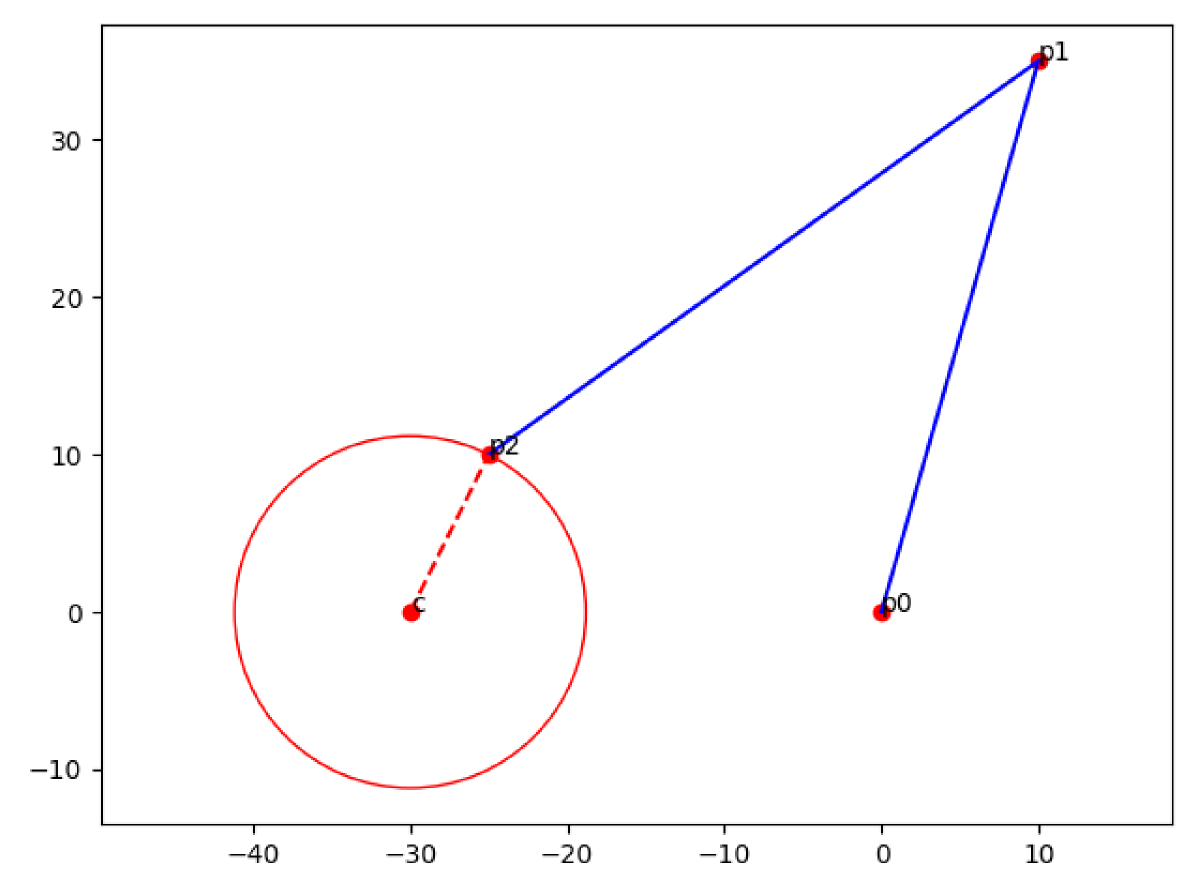
Freiheitsgrade

- Anzahl der Gelenkspunkte n
  - je 2 Freiheitsgrade pro Punkt $\vec{p}_i = [x_i, y_i]^{\mathrm{T}}$ für $i = 0 \ldots n-1$
  - → $2n$ Freiheitsgrade im ganzen Mechanismus
- Anzahl der Glieder m
  - verbinden je zwei Gelenkspunkte $\vec{p}_i$ und $\vec{p}_j$ → fixe Länge/fixer Abstand l_{ij}
  - → reduziert die Freiheitsgrade
- Randbedingungen (engl. boundary conditions) m_{BC}
  - reduzieren die Freiheitsgrade weiter
  - $m_{BC_{stat}}$ statische Randbedingungen → fixe Punkte
  - $m_{BC_{dyn}}$ dynamische Randbedingungen → bewegte Punkte

Beispiel der Viergelenkkette



Beispiel der Viergelenkkette:



- Besteht aus $n = 4$ Gelenken und 4 Gliedern $\rightarrow$ je 2 davon sind besonders und müssen speziell behandelt werden:
  - Gestell: Ist jenes Glied das fest ist $\rightarrow$ verbindet gedanklich $\vec{p}_0$ mit $\vec{c}$ $\rightarrow$ diese beiden Glieder sind statisch
  - Kurbel: Ist am Gestell angeschlossen und bewegt sich auf einer Kreisbahn $\rightarrow$ verbindet $\vec{c}$ mit $\vec{p}_2$, es bleibt aber weiterhin $\vec{c}$ statisch, und $\vec{p}_2$ dynamisch $\rightarrow$ 1 Freiheitsgrad (Rotation)
  - Alle anderen Glieder: Verbinden zwei (dynamische) Punkte und halten deren Abstand konstant $\rightarrow$ $\vec{p}_0$ und zu $\vec{p}_1$ und $\vec{p}_2$ zu $\vec{p}_1$

Beispiel der Viergelenkkette:

- Daraus folgt in unserer Art der Modellierung:
  - $n = 3 \rightarrow \vec{p}_0, \vec{p}_1, \vec{p}_2$ ($\rightarrow$ $\vec{c}$ ist nur für die Kreisbahn notwendig)
  - $m_{BC_{stat}} = 2 \rightarrow \vec{p}_0$ ist statisch
  - $m_{BC_{dyn}} = 2 \rightarrow \vec{p}_2$ ist auf einer Kreisbahn $\rightarrow$ bei konstantem θ wird $\vec{p}_2$ vollständig bestimmt
  - $m_{BC_{stat,dyn}}$ bilden zusammen das Gestell & die Kurbel ab

Übrige Freiheitsgrade

- $f = 2n - m_{BC_{stat}} - m_{BC_{dyn}} = 2 \rightarrow$ "normale" Glieder notwendig
- Glieder zw. $\vec{p}_0$ und $\vec{p}_1$ sowie $\vec{p}_1$ und $\vec{p}_2$ berücksichtigen $\rightarrow m = 2$
- $f = 2n - m_{BC_{stat}} - m_{BC_{dyn}} - m = 0 \rightarrow$ ✓

Beispiel der Viergelenkkette:

- Wie kann über dieses Modell die Kinematik bestimmt werden?

Idee

- Die genaue Konfiguration des Mechanismus kann für einen konstanten Drehwinkel θ bestimmt werden
- θ wird nun variiert (z.B. in 1° Schritte) $\rightarrow$ die Konfiguration des Mechanismus ändert sich mit einem Freiheitsgrad (θ) $\rightarrow$ Kinematik des Systems

Wie kann die Konfiguration bestimmt werden?

- Wir können die Lage von allen Punkten $\vec{p}_i(\theta)$ bestimmen die mit Randbedingungen versehen sind
- Für diese neuen Lagen können die Längen der Glieder bestimmt werden $\rightarrow$ da sich die Punkte bewegt haben sind die Längen der Glieder nicht mehr korrekt
- Aus den aktuellen Ist-Längen und den Soll-Längen der Glieder kann ein Fehler bestimmt werden
- $\rightarrow$ Optimierung soll diesen Fehler minimieren

Viergelenk - Modell in Ausgangskonfiguration

- Wir sehen uns an wie dies mathematisch für das Viergelenk funktioniert
- Die Ausgangslage ist jene aus der Abbildung oben → in diese wird der Mechanismus modelliert:
  - $\vec{p}_0 = \begin{bmatrix} 0 & 0 \end{bmatrix}^{\mathrm{T}}$ → statisch BC → unbeweglich
  - $\vec{p}_1 = \begin{bmatrix} 10 & 35 \end{bmatrix}^{\mathrm{T}}$
  - $\vec{p}_2 = \begin{bmatrix} -25 & 10 \end{bmatrix}^{\mathrm{T}}$ → dynamisch BC → Kreisbewegung um $\vec{c}$
  - $\vec{c} = \begin{bmatrix} -30 & 0 \end{bmatrix}^{\mathrm{T}}$ → Mittelpunkt der Kreisbahn
- Der Drehwinkel θ wird als $\theta = \arctan\left(\frac{10}{5}\right)$ initialisiert

Viergelenk - Bestimmen der Längen

- ⚠️ Für jeden Schritt der Berechnung wird θ (temporär) konstant gehalten
- Der Vektor $\vec{x}$ beinhaltet alle Punkte des Systems:

$$\vec{x} = \begin{bmatrix} \vec{p}_0 & \vec{p}_1 & \vec{p}_2 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} p_{0_x} & p_{0_y} & p_{1_x} & p_{1_y} & p_{2_x} & p_{2_y} \end{bmatrix}^{\mathrm{T}}$$

- Aus den Gliedern des Systems lässt sich die Matrix $\boldsymbol{A}$ ableiten, die nur die Elemente -1, 0 und 1 enthält und damit die Verbindungen der Punkte (= Glieder) beschreibt:

$$\boldsymbol{A} = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}_{2m \times 2n}$$

- Das Produkt $\boldsymbol{A}\vec{x}$ ergibt den Vektor $\hat{\vec{l}}$ der Längen der Glieder für den aktuellen, konstanten Winkel θ enthält
- Diese sind in $\hat{\vec{l}}$ komponentenweise angeordnet

Viergelenk - Bestimmen der Längen

- Um aus $\hat{\vec{l}}$ die tatsächlichen Längen der Glieder zu erhalten, wird $\hat{\vec{l}}$ umgeformt in $\boldsymbol{L}$ und über die zeilenweise angewendete euklidische Norm der Vektor $\vec{l}$ berechnet:

$$\hat{\vec{l}} = \boldsymbol{A}\vec{x} = \underbrace{\begin{bmatrix} p_{0_x} - p_{1_x} \\ p_{0_y} - p_{1_y} \\ p_{1_x} - p_{2_x} \\ p_{1_y} - p_{2_y} \end{bmatrix}}_{2m \times 1} \Rightarrow \boldsymbol{L} = \underbrace{\begin{bmatrix} p_{0_x} - p_{1_x} & p_{0_y} - p_{1_y} \\ p_{1_x} - p_{2_x} & p_{1_y} - p_{2_y} \end{bmatrix}}_{m \times 2}$$

- $\vec{l} \in \mathbb{R}^{m \times 1}$ enthält nun die Längen aller m Glieder:

$$\underbrace{\vec{l}}_{m \times 1} \Rightarrow l_i = \left(\sum_{j=1}^{2} (L_{ij})^2 \right)^{1/2} = \begin{bmatrix} \sqrt{(p_{0_x} - p_{1_x})^2 + (p_{0_y} - p_{1_y})^2} \\ \sqrt{(p_{1_x} - p_{2_x})^2 + (p_{1_y} - p_{2_y})^2} \end{bmatrix}$$

Viergelenk - Bestimmen der Längen

- mit den konkreten Zahlen der Anfangskonfiguration ergibt sich:

$$\vec{x} = \begin{bmatrix} 0 & 0 & 10 & 35 & -25 & 10 \end{bmatrix}^{\mathrm{T}}$$

$$\hat{\vec{l}} = \boldsymbol{A}\vec{x} = \underbrace{\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}}_{2m\times 2n} \underbrace{\begin{bmatrix} 0 \\ 0 \\ 10 \\ 35 \\ -25 \\ 10 \end{bmatrix}}_{2n\times 1} = \underbrace{\begin{bmatrix} -10 \\ -35 \\ 35 \\ 25 \end{bmatrix}}_{2m\times 1}$$

$$\boldsymbol{L} = \underbrace{\begin{bmatrix} -10 & -35 \\ 35 & 25 \end{bmatrix}}_{m\times 2} \Rightarrow \vec{l} = \underbrace{\begin{bmatrix} 36.4005 \\ 43.0116 \end{bmatrix}}_{m\times 1}$$

- Wird nun θ um 10° erhöht, so ändert sich die Konfiguration des Mechanismus
- Aufgrund der Kreisbewegung von $\vec{p}_2$ ändert sich dessen Position zu $\vec{p}_2' = \begin{bmatrix} -26.81 & 10.72 \end{bmatrix}^\mathrm{T}$, alle anderen Punkte bleiben unverändert

$$\vec{x}' = \begin{bmatrix} 0 & 0 & 10 & 35 & -26.81 & 10.72 \end{bmatrix}^\mathrm{T}$$

$$\hat{\vec{l}}' = \boldsymbol{A}\vec{x}' = \underset{2m\times 2n}{\begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 \end{bmatrix}} \underset{2n\times 1}{\begin{bmatrix} 0 \\ 0 \\ 10 \\ 35 \\ -26.81 \\ 10.72 \end{bmatrix}} = \underset{2m\times 1}{\begin{bmatrix} -10 \\ -35 \\ 36.81 \\ 24.28 \end{bmatrix}}$$

$$\boldsymbol{L}' = \underset{m\times 2}{\begin{bmatrix} -10 & -35 \\ 36.81 & 24.28 \end{bmatrix}} \Rightarrow \vec{l}' = \underset{m\times 1}{\begin{bmatrix} 36.4005 \\ 44.1005 \end{bmatrix}}$$

Viergelenk - Matrizen

- nun können die beiden Längenvektoren $\vec{l}$ und $\vec{l}'$ verglichen werden
- der Fehler $\vec{e}$ ergibt sich aus der Differenz der beiden Vektoren:

$$\vec{e} = \vec{l}' - \vec{l} = \begin{bmatrix} 0 \\ 1.0889 \end{bmatrix}$$

- dieser Fehler kann nun im Sinne der kleinsten Fehlerquadrate minimiert werden $\rightarrow$ z.B. mit Funktionen aus dem `scipy.optimize` -Modul

Anforderungen, Erweiterungen & Abgabe

Minimalanforderungen

- Softwareprojekt in Python mit Objektorientierung
- Versionskontrolle über git & GitHub
- Anwendung mit Web-UI:
  - Simulation von Mechanismen und deren Kinematik (Detail siehe unten)
- Software-Dokumentation:
  - `requirements.txt`-Datei mit allen packages
  - `README.md` im Repository mit Anleitung zur Installation und Ausführung
- Projekt-Dokumentation - Zwei Möglichkeiten
  - `README.md` ergänzen um umgesetzten Erweiterungen, UML-Diagrammen der Softwarestruktur, Quellen zu verwendeten Inhalten etc.
  - *ODER*
  - Kurzer Bericht als pdf-Dokument mit selbem Inhalt

Minimalanforderungen 1/2

- Eine `Python`-Anwendung mit Web-UI (`streamlit`) soll entwickelt werden
- Darin können beliebige ebene Mechanismen mit unseren Einschränkungen definiert werden
- Die Positions-Kinematik des Mechanismus soll für den Winkel θ im Bereich von 0° bis 360° berechnet werden
- Der Mechanismus und seine Kinematik bzw. die Bahnkurven der Punkte werden visualisiert
- Die Bahnkurve kann als $x(\theta) - y(\theta)$-Koordinaten im `csv`-Format (o.ä) abgespeichert werden
- Der Mechanismus selbst (und damit seine Kinematik) soll gespeichert und geladen werden können
- Die Kinematik soll als Optimierungsproblem gelöst werden, bei der die Länge-Fehler der Glieder minimiert werden $\rightarrow$ siehe Erklärung oben

Minimalanforderungen 2/2

- Es muss verifiziert werden, dass der Mechanismus valide ist
- Testen der Implementierung am Beispiel des "Strandbeest" bzw. an einem seiner Beine → in Dokumentation zeigen
- Die Anwendung soll mit `streamlit` deployed werden

Mögliche Erweiterungen

- Visualisierung der Längen-Fehler aller Glieder als Funktionen des Winkels θ
- Animation als Video/gif speichern
- Overlay z.B. von Winkeln oder Längen auf die Visualisierung
- Lösen von Kinematiken ermöglichen bei denen ein Punkt noch einen Freiheitsgrad hat $\rightarrow$ z.B. eine Schubkurbel
- Definition einer Auszeichnungssprache um Modelle der Mechanismen zu beschreiben $\rightarrow$ Modell kann heruntergeladen und ggf. hochgeladen werden
- Optimieren der Gliederlängen für eine bestimmte Bahnkurve bzw. einer Bahnkurve die gewissen Kriterien entspricht

Mögliche Erweiterungen

- Alternative Ansätze im UI z.B. Drag and Drop oder Sketches
- Import von Sketches mit Bildererkennung (kariertes Papier, etc.)
- Erstellen einer Stückliste für ausgewählte Gestänge, Antriebe und Gelenke
- Berechnung der maximalen Vorwärts-Geschwindigkeit (z.B. eines festgelegten Punktes in X) eines Strandbeests in Abhängigkeit von der Drehgeschwindigkeit der Kurbel, Schrittlänge und maximale Schritthöhe → das könne eine Metrik für die Optimierung der Gliederlängen sein
- 3D-Volumenmodell des Mechanismus mittels `OpenSCAD` erstellen
- ⚠️ Sehr große Erweiterung: Kräfte/Momente berücksichtigen (Kinetik) und dadurch die Positions-, Geschwindigkeits- und Beschleunigungskinematik bestimmen

Abgabe

- Abgabetermin: 27.02.2025 um 23:55 Uhr
- Alle Gruppenmitglieder müssen den Link zum Repository auf Sakai abgeben
- Der Beitrag der einzelnen Mitglieder wird anhand ihrer Commits beurteilt → jedes Mitglied soll aktiv und inhaltvoll am Projekt mitarbeiten
- Es wird der letzte Commit im `main` bzw. `master`-Branch bewertet

Alternatives Abschlussprojekt

- Falls die Aufgabenstellung einer Gruppe nicht zusagt, kann ein alternatives Abschlussprojekt mit dem jeweiligen Vortragenden vereinbart werden
- In der ersten Hälfte des letzten Termins der LV muss diese Aufgabenstellung mit dem Vortragenden vereinbart werden → anschließend wird sie mit dem Vortragenden der anderen Gruppe abgestimmt

Was muss vereinbart werden?

- Idee des eigenen Projektes
- Minimalanforderungen → Programmiersprache, Versionsverwaltung, Dokumentation, etc. bleibt immer gleich
- Mögliche Erweiterungen
- Diese sind dann verbindlich für die jeweilige Gruppe → es kann aber immer auf das allgemeine Projekt zurück gewechselt werden