Praktikum Teil 2/3

Mit ⋆ gekennzeichnete Aufgaben sind Vorbereitungsaufgaben. Diese sind vor dem Praktikum schriftlich zu erledigen und Voraussetzung für die Teilnahme. Nicht vorbereitete Gruppen, oder Teilnehmer, dürfen nicht am Versuch teilnehmen. Eine MATLAB-Lizenz können Sie kostenlos über Ihren URZ-Account beziehen.

1 Inverse Kinematik

1.1 Einführung

Inverse Kinematik (abgekürzt IK) beschreibt das Verfahren zur Bestimmung der Gelenkwinkel eines Roboters anhand der Pose $\boldsymbol{\rho}$ (Position und Orientierung) seines End-Effektors. Entgegengesetzt wird mit der Vorwärtskinematik die Pose des End-Effektors mittels der Gelenkpositionen des Roboters ermittelt.

Zur Lösung der IK gibt es unterschiedliche Möglichkeiten:

- Algebraische Methode
- Geometrische Methode
- Numerische Methode

Die algebraischen Methode versucht die Transformationsmatrix des End-Effektors ${}^{R}T_{TCP}$ nach den einzelnen Gelenkwinkeln q_i zu lösen.
Sind die Freiheitsgrade des Roboters gering, so lässt sich die geometrische Methode anwenden. Diese Methode berechnet die Gelenkwinkel über die Geometrie des Roboters. Industrieroboter mit sechs Freiheitsgraden sind in der Regel nach einem Arm-Handgelenk

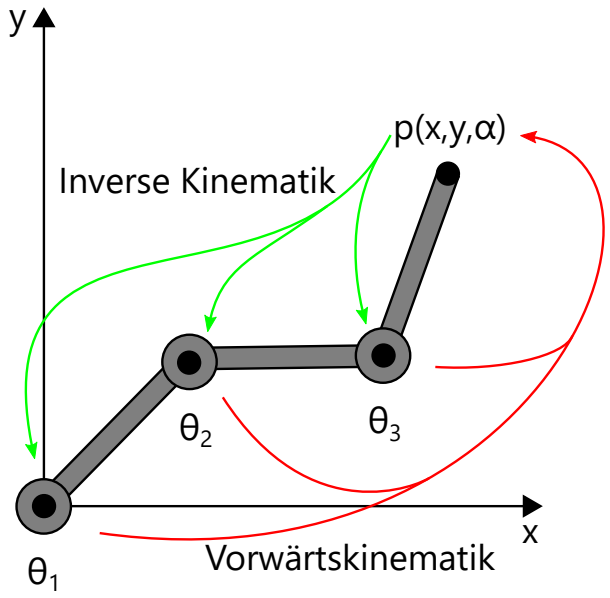

Abbildung 1: Inverse Kinematik und Vorwärtskinematik

Prinzip aufgebaut, sodass sich der Roboter konstruktionsbedingt in zwei Bereiche aufteilen lässt. Die ersten drei Achsen bilden den Arm des Manipulators und steuern die Position des End-Effektors im Arbeitsraum. Die letzten drei Achsen bilden eine Art Handgelenk, sodass deren Koordinatensysteme im selben Ursprung liegen und dadurch lediglich die Orientierung des End-Effektors ändern. Somit lässt sich die IK durch die einzelne Betrachtung der zwei Bereiche mit ihren drei Achsen relativ einfach lösen, sofern es die Konstruktion des Roboters zulässt.
Falls keine algebraische Lösung zu finden ist und ist die Kinematik des Roboters zu komplex, so lässt sich die IK berechnen, indem näherungsweise numerisch nach einer Lösung der IK gesucht wird.

1.2 Newton-Verfahren

Die numerische Lösung der IK stellt ein Optimierungsproblem dar. Ein Optimierungsalgorithmus versucht eine Gütefunktion $f(\boldsymbol{x})$ anhand seiner Eingangsvariablen $\boldsymbol{x}$ zu minimieren.

$$f(\boldsymbol{x}_{\mathrm{Opt}}) = f_{\mathrm{min}} \tag{1}$$

Es stellt sich also die Frage, für welche $\boldsymbol{x}$ $f(\boldsymbol{x})$ minimiert wird. Eines der einfachsten Verfahren für den Fall der IK ist das Newton-Verfahren, welches mithilfe der Ableitung der Gütefunktion nach dem Minimum sucht. Die Rekursionsvorschrift, die das Newton-Verfahren beschreibt, ist wie folgt definiert:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \tag{2}$$

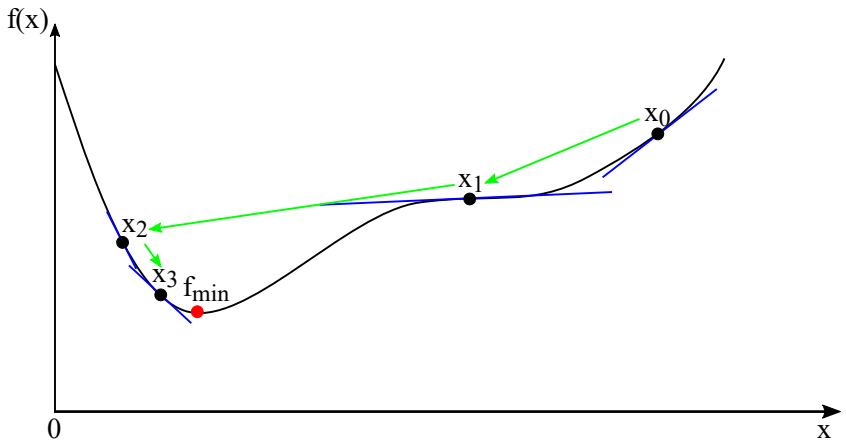

Abbildung 2: Newton-Verfahren: Suche nach Minimum f_{min}

1.3 Inverse Kinematik mit Newton-Verfahren

Für die Anwendung des Newton-Verfahrens in der IK, muss die Gütefunktion zunächst definiert werden, welche in diesem Fall die Abweichung $\boldsymbol{e}$ zwischen der Soll-Pose und Ist-Pose darstellt.

$$\boldsymbol{e} = \begin{pmatrix} \boldsymbol{e}_\mathrm{P} \\ \boldsymbol{e}_\mathrm{O} \end{pmatrix} \tag{3}$$

Für den Positionsfehler $\boldsymbol{e}_\mathrm{P}$ aus der Soll-Position $\boldsymbol{p}_\mathrm{d}$ und Ist-Position $\boldsymbol{p}_\mathrm{a}$:

$$\boldsymbol{e}_\mathrm{P} = \boldsymbol{p}_\mathrm{d} - \boldsymbol{p}_\mathrm{a} \tag{4}$$

Je nachdem, ob man Euler-Winkel oder Quaternionen zur Darstellung der Orientierung verwendet werden, gibt es unterschiedliche Beschreibungen der Abweichung in der Orientierung. Für Euler-Winkel gilt folgendes:

$$\boldsymbol{e}_\mathrm{O} = \begin{pmatrix} \phi_\mathrm{d} \\ \theta_\mathrm{d} \\ \psi_\mathrm{d} \end{pmatrix} - \begin{pmatrix} \phi_\mathrm{a} \\ \theta_\mathrm{a} \\ \psi_\mathrm{a} \end{pmatrix} \tag{5}$$

Da hier die Rotationsreihenfolge beachtet werden muss, ist kein linearer Zusammenhang gegeben. Das Annäherungsverfahren ist somit nur für kleine Iterationsschritte geeignet.

Für Quaternionen $\boldsymbol{Q}_\mathrm{d} = \{\eta_\mathrm{d}, \boldsymbol{\epsilon}_\mathrm{d}\}$ und $\boldsymbol{Q}_\mathrm{a} = \{\eta_\mathrm{a}, \boldsymbol{\epsilon}_\mathrm{a}\}$ gilt für $\Delta\boldsymbol{Q} = \{\Delta\eta, \Delta\boldsymbol{\epsilon}\}$:

$$\Delta\boldsymbol{Q} = \boldsymbol{Q}_\mathrm{d} * \boldsymbol{Q}_\mathrm{a}^{-1} \tag{6}$$

Falls beide Orientierungen identisch sind $\boldsymbol{Q}_\mathrm{d} = \boldsymbol{Q}_\mathrm{a}$ gilt:

$$\Delta\boldsymbol{Q}_\mathrm{d} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{7}$$

Als Gütefunktion reicht es somit aus lediglich $\Delta\boldsymbol{\epsilon}$ zu betrachten:

$$\boldsymbol{e}_O = \Delta\boldsymbol{\epsilon} = \eta_\mathrm{a}(\boldsymbol{q})\boldsymbol{\epsilon}_\mathrm{d} - \eta_\mathrm{d}\boldsymbol{\epsilon}_\mathrm{a}(\boldsymbol{q}) - \boldsymbol{\epsilon}_\mathrm{d} \times \boldsymbol{\epsilon}_\mathrm{a}(\boldsymbol{q}) \tag{8}$$

Da wir nun mit $\boldsymbol{e}$ unsere Gütefunktion erstellt haben, können wir nun das Newton-Verfahren anwenden:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n - \frac{\boldsymbol{e}(\boldsymbol{q})}{\frac{d\boldsymbol{e}(\boldsymbol{q})}{d\boldsymbol{q}}} \tag{9}$$

Bei der Ableitung der Gütefunktion nach $\boldsymbol{q}$ fällt der Term der Soll-Pose weg, da dieser eine Konstante ist:

$$\frac{d\boldsymbol{e}(\boldsymbol{q})}{d\boldsymbol{q}} = \frac{d(\boldsymbol{\rho}_d - \boldsymbol{\rho}_a(\boldsymbol{q}))}{d\boldsymbol{q}} = -\frac{d\boldsymbol{\rho}_a(\boldsymbol{q})}{d\boldsymbol{q}} \tag{10}$$

Die Ableitung der Ist-Pose nach den Gelenkstellungen $\boldsymbol{q}$ ist die Jacobi-Matrix des Manipulators:

$$\boldsymbol{J}(q) = \frac{d\boldsymbol{\rho}_a(\boldsymbol{q})}{d\boldsymbol{q}} \tag{11}$$

Einsetzen von 10 und 11 in Gleichung 9 ergibt:

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + \boldsymbol{J}^{-1}\boldsymbol{e}(\boldsymbol{q}) \tag{12}$$

1.4 Anpassung des Newton-Verfahrens

Falls das Optimierungsproblem mehrere Minima besitzt, so kann es passieren, dass der Optimierungsalgorithmus gegen ein lokales Minimum konvergiert, jedoch nicht gegen das globale Minimum (Abb. 3). Daher ist es äußerst wichtig, die Anfangsposition so zu wählen, dass dieser sich schon im Tal des globalen Minimums befindet. Zudem ist eine Interpolation der Bewegung mithilfe von mehreren kleinen Zwischenschritten sinnvoll bevor die IK ausgeführt wird, wodurch sich die Anfangswerte des Optimierungsverfahrens schon in der Nähe des globalen Minimums befinden. Des Weiteren ist es möglich, dass der Algorithmus, obwohl sich die Startposition im Tal des globalen Minimums befindet, in das Tal des lokalen Minimums springt. Das kann passieren, falls aus 2 die Ableitung sehr klein wird und die Gütefunktion sehr groß, wodurch in sehr weiten Schritten gesprungen wird.Um sehr große Schritte zu vermeiden, kann die Schrittweite mit dem Faktor K skaliert werden.

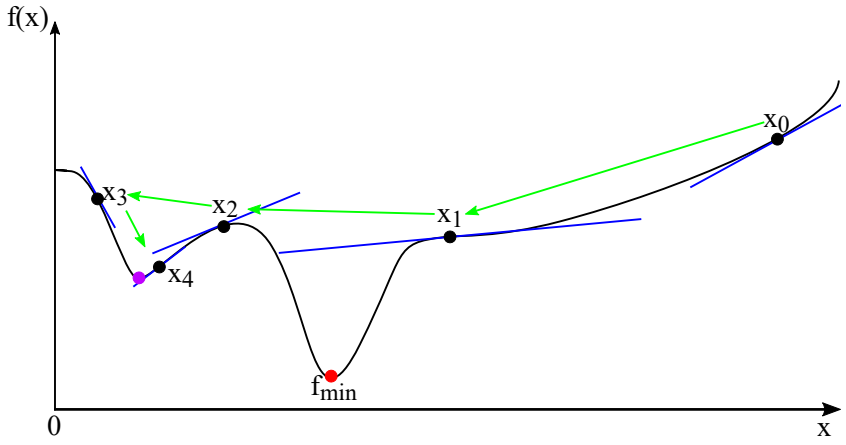

Abbildung 3: Verfahren konvergiert gegen falsches Minimum

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + K \cdot \boldsymbol{J}^{-1}\boldsymbol{e}(\boldsymbol{q}) \tag{13}$$

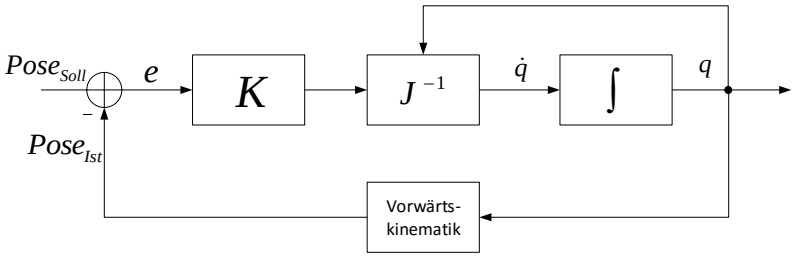

Abbildung 4: Inverse Kinematik mit Newton-Verfahren als Diagramm

Hierbei ist zu beachten, dass zwar ein sehr kleines K das Verfahren weniger wahrscheinlich in ein anderes Tal springen lässt, jedoch werden auch mehr Iterationsschritte benötigt bis das globale Minimum erreicht wird. Es muss daher ein Trade-off zwischen Rechenzeit und korrektes Konvergieren für K gewählt werden.

1.5 Inverse der Jacobi-Matrix

Für die Berechnung von 13 wird die Inverse $\boldsymbol{J}^{-1}$ der Jacobi-Matrix verwendet um den Iterationsschritt zu berechnen. In der Praxis ist die Bildung von $\boldsymbol{J}^{-1}$ nicht immer möglich, da die Determinante $Det(\boldsymbol{J})$ zu 0 werden kann. Im Falle einer Positionssteuerung ohne Beachtung der Orientierung wird die Jacobi-Matrix zu einer $3 \times n$-Matrix, wodurch die Inverse ebenfalls nicht gebildet werden kann.

1.6 Transponierte der Jacobi-Matrix

Anstatt der Inverse, kann die Transponierte der Jacobi-Matrix verwendet werden.

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_n + K \cdot \boldsymbol{J}^T \boldsymbol{e}(\boldsymbol{q}) \tag{14}$$

Natürlich ist die Transponierte nicht gleich der Inverse. Für kleine Abstände ist diese Methode aber hinreichend genau.

1.7 Pseudo-Inverse der Jacobi-Matrix

Eine weitere Möglichkeit bietet die Bildung der Pseudo-Inverse (auch bekannt als Moore-Penrose Inverse). Sie ist auch für $n \times m$-Matrizen gültig und bietet die beste Näherung für J^{-1} bezüglich des Least-Square-Fehlers.

$$\boldsymbol{J}^{+} = \boldsymbol{J}^{T}(\boldsymbol{J}\boldsymbol{J}^{T})^{-1} \tag{15}$$

$$\boldsymbol{q}_{n+1} = \boldsymbol{q}_{n} + K \cdot \boldsymbol{J}^{+}\boldsymbol{e}(\boldsymbol{q}) \tag{16}$$

Allerdings ist die Pseudo-Inverse in der Nähe von Singularitäten (Positionen, die der Roboter nicht annehmen kann) aufgrund des Terms $(\boldsymbol{J}\boldsymbol{J}^{T})^{-1}$ instabil, sodass sehr große Sprünge im Gelenkraum stattfinden, obwohl sich der Manipulator minimal im kartesischen Raum bewegt. Bei der Berechnung der IK mit der Pseudo-Inverse der Jacobi-Matrix sollte auf große Sprünge im Gelenkraum geprüft werden.

1.8 Kombination Transponierte und Pseudo-Inverse der Jacobi-Matrix

Für die Lösung der IK können die Transponierte und Pseudo-Inverse Jacobi-Matrix kombiniert werden. Beide Verfahren werden zur Berechnung des Iterationsschritts verwendet, wodurch man zwei Ergebnisse erhält. Für den nächsten Iterationsschritt wird der genauere Wert verwendet. Da die Transponierte für die Berechnung der Pseudo-Inverse sowieso bestimmt wird, erhöht sich der Berechnungsaufwand pro Iterationsschritt nur gering. Andererseits steigt die Wahrscheinlichkeit, dass weniger Iterationsschritte benötigt werden. Die Instabilitäten der Pseudo-Inverse in der Nähe von Singularitäten werden unterdrückt.

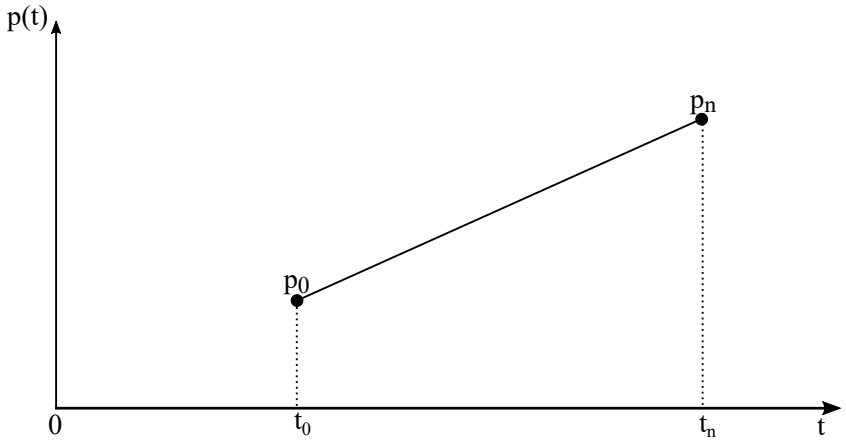

Abbildung 5: 1-Dimensionale lineare Interpolation von p_0 nach p_n

2 Interpolation

2.1 1-D Interpolation

Eine lineare Interpolation zwischen zwei Punkten wie in Abb.: 5 kann Anhand mit folgender Gleichung beschrieben werden:

$$p(t) = p_0 + t \cdot \frac{p_1 - p_0}{t_n - t_0} \tag{17}$$

Wobei der Term $\frac{p_1 - p_0}{t_n - t_0}$ die Geschwindigkeit darstellt:

$$p(t) = p_0 + v \cdot t \tag{18}$$

Wird die Beschleunigung mit berücksichtigt, ist es hilfreich die Trajektorie in drei Bereiche mit Beschleunigungsphase (t_0 bis t_1), Nullbeschleunigungsphase (t_1 bis t_2) und Abbremsphase (t_2 bis t_n) zu unterteilen.

In der Beschleunigungsphase erhöht sich die Geschwindigkeit mit a_0 solange, bis die

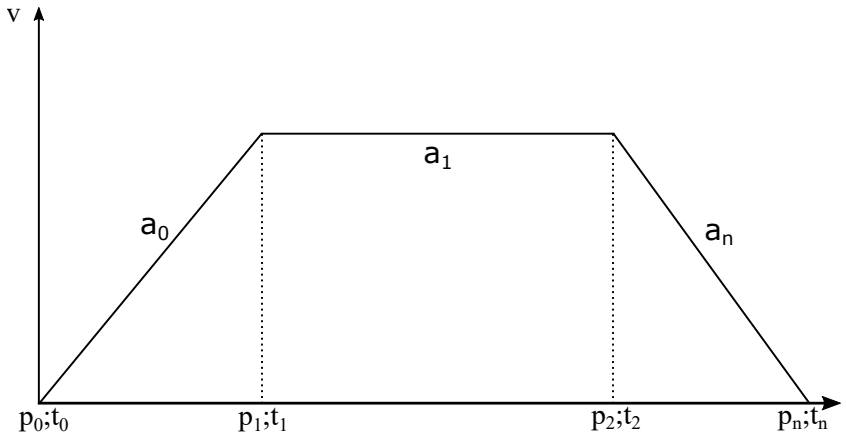

Abbildung 6: 1-Dimensionale lineare Interpolation von p_0 nach p_n mit Berücksichtigung von Beschleunigung

maximale Geschwindigkeit v_{max} bei t_1 erreicht wird. Für die Beschleunigungsphase gilt:

$$a_0 = const \tag{19}$$

$$v_0(t) = a_0 t \tag{20}$$

$$p_0(t) = \frac{1}{2} a_0 t^2 \tag{21}$$

Für die Nullbeschleunigungsphase gilt:

$$a_1 = 0 \tag{22}$$

$$v_1(t) = v_0(t1) = v_{max} \tag{23}$$

$$p_1(t) = p_0(t1) + v_0(t_1) \cdot (t - t_1) \tag{24}$$

In der Abbremsphase wird die Geschwindigkeit mit a_2 wieder reduziert, sodass die

Endposition erreicht und die Geschwindigkeit gleich 0 ist. Für die Abbremsphase gilt:

$$a_n = const \tag{25}$$

$$v_n(t) = v_1(t2) + a_n(t - t_2) \tag{26}$$

$$p_n(t) = p_1(t2) + v_1(t2) \cdot (t - t_2) + \frac{1}{2}a_2(t - t_2)^2 \tag{27}$$

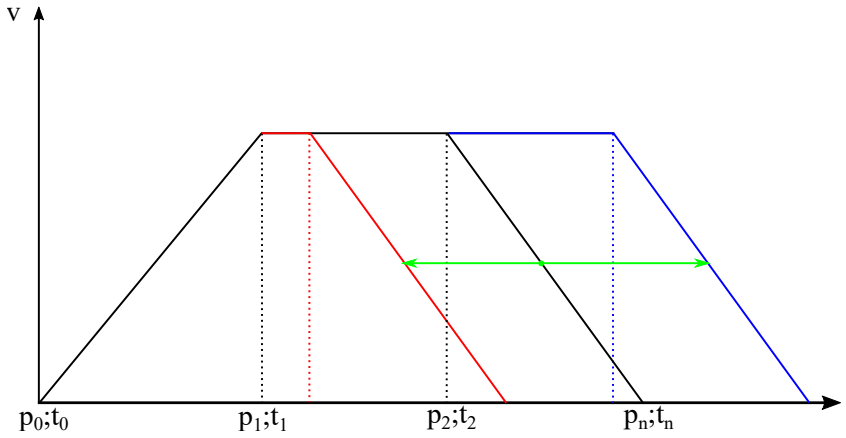

Abbildung 7: Unterschiedliche Verweildauer in der Nullbeschleunigungsphase

Die Verweildauer $t_2 - t_1$ in der Nullbeschleunigungsphase hängt wie in Abb. 7 zu sehen von der Beschleunigungsphase und der Abbremsphase ab.

$$t_1 - t_0 = \frac{v_{max}}{a_0} \tag{28}$$

$$p_1 - p_0 = \frac{1}{2}a_2 t_1^2 \tag{29}$$

Für die Zeit und Position in der Abbremsphase gilt:

$$t_n - t_2 = \frac{-v_{max}}{a_2} \tag{30}$$

$$p_n - p_2 = -\frac{1}{2}a_2(t_n - t_2)^2 \tag{31}$$

Für die Nullbeschleunigungsphase gilt mit 29 und 31:

$$p_2 - p_1 = v_{max} \cdot (t_2 - t_1) = (p_n - p_0) - (p_1 - p_0) - (p_n - p_2) \tag{32}$$

$$t_2 - t_1 = \frac{p_2 - p_1}{v_{max}} \tag{33}$$

2.2 Sonderfall: Kleine Distanzen

Bei kleinen Distanzen wie in Abb.: 8 kann unter Umständen die Endgeschwindigkeit v_{max} gar nicht erreicht werden, da schon vorher der Abbremsvorgang eingeleitet werden muss, damit der Manipulator an der Endposition zum stehen kommt.

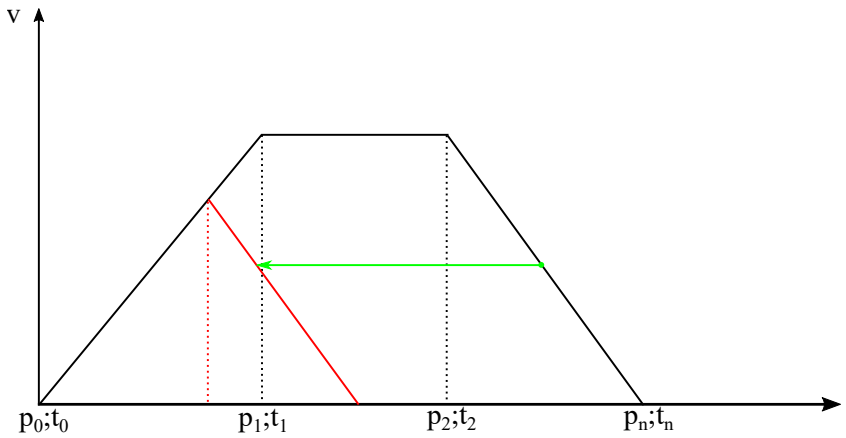

Abbildung 8: Trajektorie bei kleinen Distanzen

Für diesen Fall ist die Verweildauer in der Nullbeschleunigungsphase $(t_2 - t_1) < 0$,

wodurch sich die Beschleunigungsphase und die Abbremsphase verkürzt:

$$t_{1,kurz} - t_0 = t1 + \frac{abs(a_0)}{abs(a_0) + abs(a_n)} \cdot (t_2 - t_1) \tag{34}$$

$$t_{n,kurz} - t_2 = t1 + \frac{abs(a_n)}{abs(a_0) + abs(a_n)} \cdot (t_2 - t_1) \tag{35}$$

$$v_{max,kurz} = a_0 \cdot (t_{1,kurz} - t_0) = a_n \cdot (t_{1,kurz} - t_0) \tag{36}$$

3 Aufgaben

1. Inverse Kinematik.
   Für diese Aufgabe stehen MATLAB-Vorlagen bereit, die Sie bitte entsprechend vervollständigen. Tipp: Informationen zu den Berechnungen finden Sie im ersten Übungsblatt.

   a) ⋆ Programmieren Sie eine MATLAB-Funktion, die aus DH-Parametern eine Transformationsmatrix erzeugt.

   ```
   T = DH([theta d a alpha]);
   ```

   b) ⋆ Berechnen Sie aus den gegebenen DH-Parametern die Transformationsmatrizen für die einzelnen Gelenke und die Vorwärtskinematik ${}^{R}T_{Hand}$. Für Gelenkstellung $\boldsymbol{q}$ sollte folgende Transformationsmatrix ${}^{R}T_{Hand}$ gebildet werden:

$$\boldsymbol{q} = \begin{pmatrix} 180 \\ 270 \\ 90 \\ 180 \\ 180 \\ 0 \end{pmatrix} \qquad {}^{R}\boldsymbol{T}_{Hand}(\boldsymbol{q}) = \begin{pmatrix} 0 & 1 & 0 & 0,41 \\ -0,5 & 0 & 0,866 & 0,2595 \\ 0,866 & 0 & 0,5 & 0,084 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

c) ⋆ Warum benötigt man für die Berechnung der inversen Kinematik die Vorwärtskinematik ${}^{R}T_{Hand}$?

d) ⋆ Bestimmen Sie die Jacobi-Matrix für die Position (ohne Orienterierung) mit Hilfe der Transformationsmatrizen. Erstellen Sie dazu eine Funktion, die für jede Gelenktransformation die Ableitung bildet:

```
j_i = Jacobi(T_0_i-1, T_0_n);
J = [j_0 j_1 j_2 ...];
```

Zur Überprüfung: Da das sechste Gelenk nur die Orientierung ändert, ist die sechste Spalte der Jacobi-Matrix für die Position immer Null.

e) ⋆ Programmieren Sie die Inverse Kinematik für die Position jeweils mit der Transponierung J^T und der Pseudo-Inversen J^+. Die Anzahl der Iterationsschritte soll 128 betragen

```
q_soll = IK_Transp(q_ist, [x_soll y_soll z_soll]);
q_soll = IK_Pseudo(q_ist, [x_soll y_soll z_soll]);
```

f) ⋆ Vergleichen Sie beide Ergebnisse miteinander. Für den Vergleich sollten die Iterationsschritte verringert werden. Was fällt auf?

g) ⋆ Erweitern Sie IK_Pseudo() so, dass zu große Sprünge im Gelenkraum ($q\_soll - q\_ist$) erkannt werden . Zudem soll der Positionsfehler im kartesischen Raum nicht mehr als $abs(e) < 0,01m$ betragen. Eine Ausgabe der Fehlermeldung kann mit der MATLAB-Funktion `disp('message')` erfolgen.

h) ⋆ Nennen Sie mindestens drei Gründe, weshalb der Fehler e der IK nicht gegen 0 konvergieren kann. Nennen Sie mindestens zwei Lösungsansätze.

2. Trajektoriengenerierung

   a) Programmieren Sie eine lineare Interpolation der Position im kartesischen Raum.

      ```
      P = Interpolation(P_start, P_soll, Zwischenschritte);
      ```

      Wobei P eine $n \times 3$-Matrix darstellt mit n Interpolationspunkten für x, y und z.

   b) $\star$ Warum ist eine Interpolation sinnvoll, selbst wenn nur die End-Position von Interesse ist?

   c) Erweitern Sie die Interpolation so, dass auch Geschwindigkeit und Beschleunigung berücksichtigt werden.

      ```
      P = P_Interp(P_start, P_soll, v, a);
      ```

      Die Abbremsbeschleunigung ist gleich der negativen Beschleunigung. Die Abtastzeit beträgt $0,01s$.

   d) Berechnen Sie die IK für die einzelnen Interpolationspunkte und speichern Sie diese in eine TXT-Datei ab. Tipp: `fprintf()`

   e) Starten Sie ihr Eclipse Projekt aus dem ersten Teil des Praktikums und erweitern Sie ihren Regler so, dass alle 6 Achsen angesteuert werden können.

   f) Fügen Sie eine Positionsbegrenzung der Gelenke ein, sodass der Roboter nicht mit sich selbst kollidiert.

   g) Schreiben Sie eine weitere Funktion, die die vorher erstellte TXT-Datei einließt und anschließend den Roboter die eingelesene Trajektorie abfahren lässt. Achtung: Da hier alle sechs Achsen bewegt werden, können Fehler in der Programmierung zu Kollisionen mit dem Tisch oder dem Roboter an sich führen.

   h) Fügen Sie eine Vorsteuerung der Geschwindigkeit hinzu.