

Doc of PathPlanning Approaches for GeckoBot

Lars Schiller

18. Juni 2019

Inhaltsverzeichnis

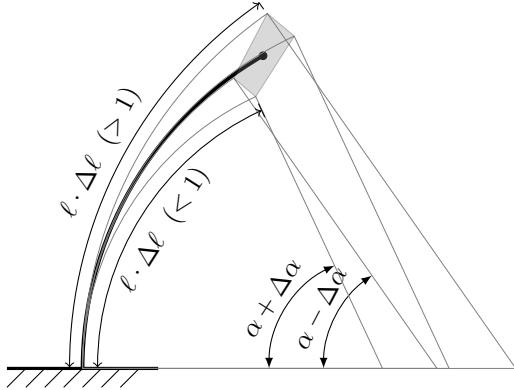
1	Predicting the next pose of the robot	1
1.1	Modeling of Soft Bending Actuator	1
1.2	Modeling of the robot	2
2	Path Planning with Search Tree	3
2.1	Different Gait Patterns for a curve	3
2.2	Search Tree	4
2.2.1	Search Tree with weights	6
2.3	Simulation Results	6
2.3.1	Simulation Results Curve	6
2.3.2	Simulation Results Straight	7
2.4	What happens if Process Noise occurs?	8
2.4.1	Curve Noise	9
2.4.2	Straight Noise	9
2.5	Conclusion	9
3	Finding a Analytic Model for describing the General Gait	10
3.1	Problem Statement	10
3.2	Approach: Guess structure for a analytic model for walking curves	11
3.2.1	Simulation Results	11
3.2.2	Observations	12
3.3	Approach: Find a reasonable structure	12
3.3.1	Simulation Results	13
3.4	Approach: Optimize Extra leg bending Angle for given extra torso bending	15
3.4.1	Optimization Results	16
3.4.2	Function Fitting	16
3.4.3	Simulation Results	17
3.5	Compare model 2 and model 3	18
3.5.1	Conclusion	18
4	Optimal PathPlanner based on General Gait Model	19
4.1	Idea	19
4.2	Coordinate Transformation in order to reduce complexity	20
4.2.1	Explizite Formulierung für mehrer Zyklen	21
4.3	Simplified Optimization Problem	22
4.3.1	Calculation of Jacobian	22

1 Predicting the next pose of the robot

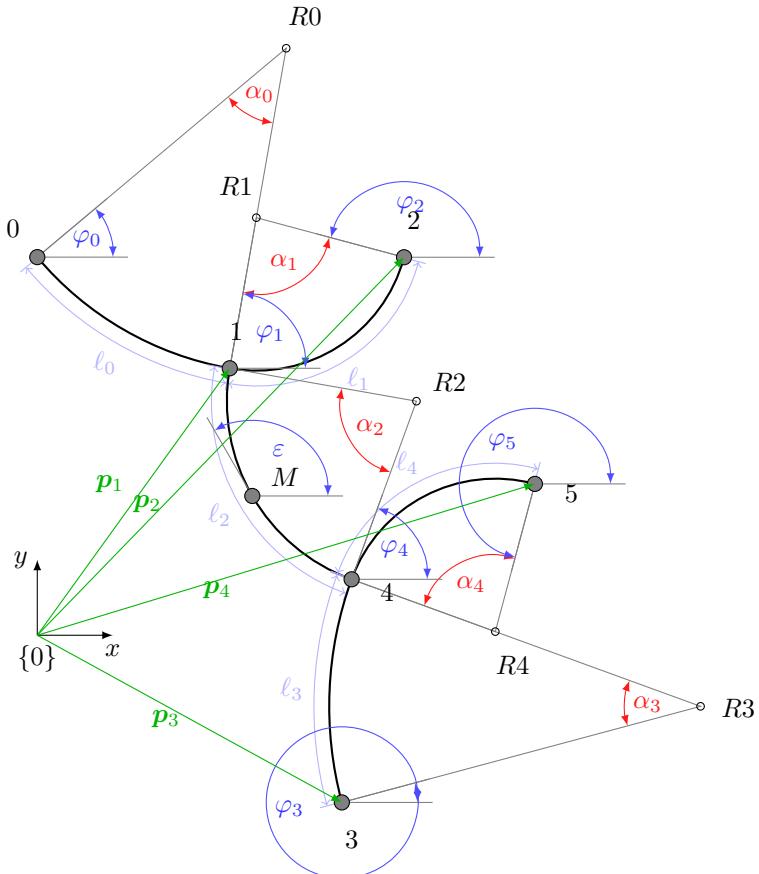
1.1 Modeling of Soft Bending Actuator

- Einführung virtueller Längen, um größeren Bereich erreichen zu können, und dennoch die Annahme von *Constant curvature* nutzen zu können.

- Weil Sehr effektiv zu rechnen.



1.2 Modeling of the robot



- Zustands- und Eingangsgrößen:

$$\mathbf{x} = [\alpha \ell \varepsilon], \quad \mathbf{r} = [\alpha_{\text{ref}} \mathbf{f}]$$

- Innere Spannung:

$$\begin{aligned} \sigma(\mathbf{x}_k) &= w_\ell |\ell_k - \ell_n|_2 \\ &\quad + w_\alpha |\alpha_k - \alpha_{\text{ref},k}|_2 \\ &\quad + w_\varphi |\text{diag}(\mathbf{f}_k)(\varphi_k - \varphi_{k-1})|_2 \end{aligned}$$

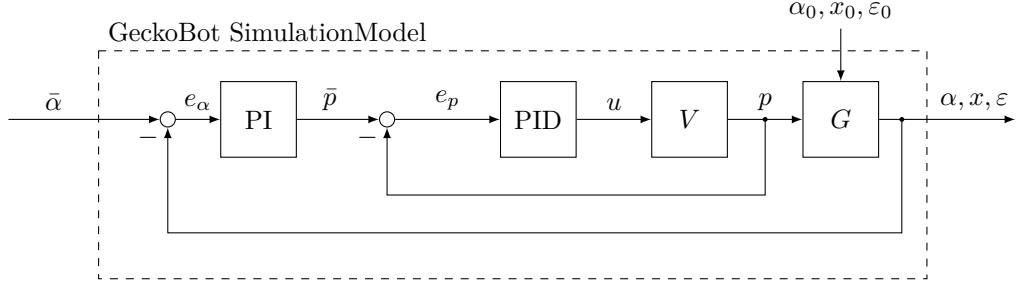
- Minimale Spannung:

$$\begin{aligned} &\min_{\mathbf{x}_k \in \mathcal{X}} \sigma(\mathbf{x}_k) \\ \text{s. t. } &|\text{diag}(\mathbf{f}_k)(\mathbf{P}_k - \mathbf{P}_{k-1})|_2 = 0 \end{aligned}$$

- Folgepose:

$$\rho_k = [\mathbf{x}_k \ \mathbf{P}_k \ \mathbf{f}_k] = \text{fun}_{\mathcal{P}}(\mathbf{r}_k, \rho_{k-1})$$

- Das Modell liefert dann eine quasi statische Vorhersage der neuen Ruhelage zu gegebenen Eingangsgrößen:



2 Path Planning with Search Tree

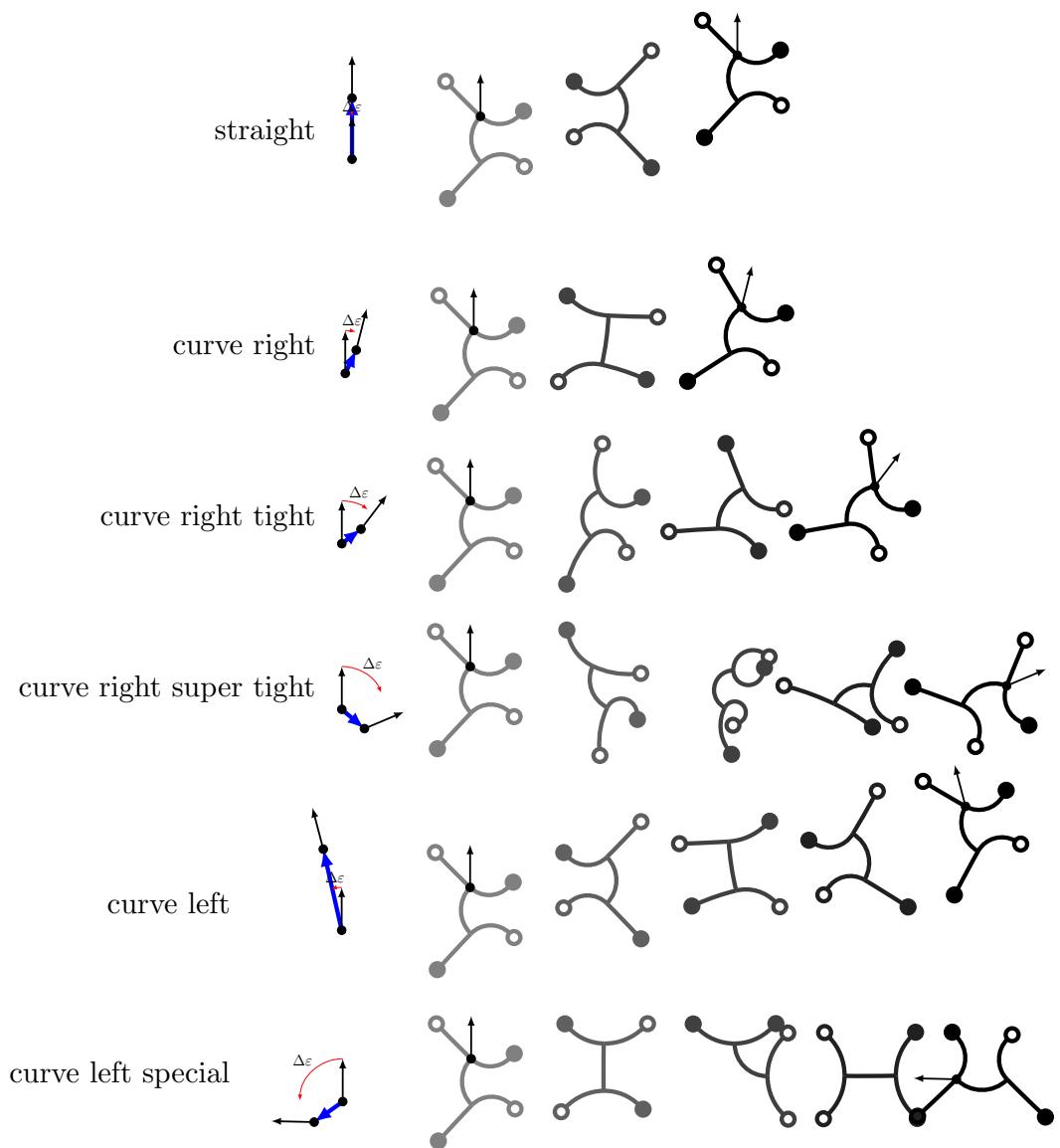
2.1 Different Gait Patterns for a curve

- Vom Kinematic Paper sind schon verschieden Laufmuster für Kurven bekannt, basierend auf dem Minimierungsproblem:

$$\min_{\alpha \in \mathcal{A}} \varepsilon(\alpha) \quad (1)$$

wobei α die Referenzwinkel von zwei Posen, also einem Zyklus enthält.

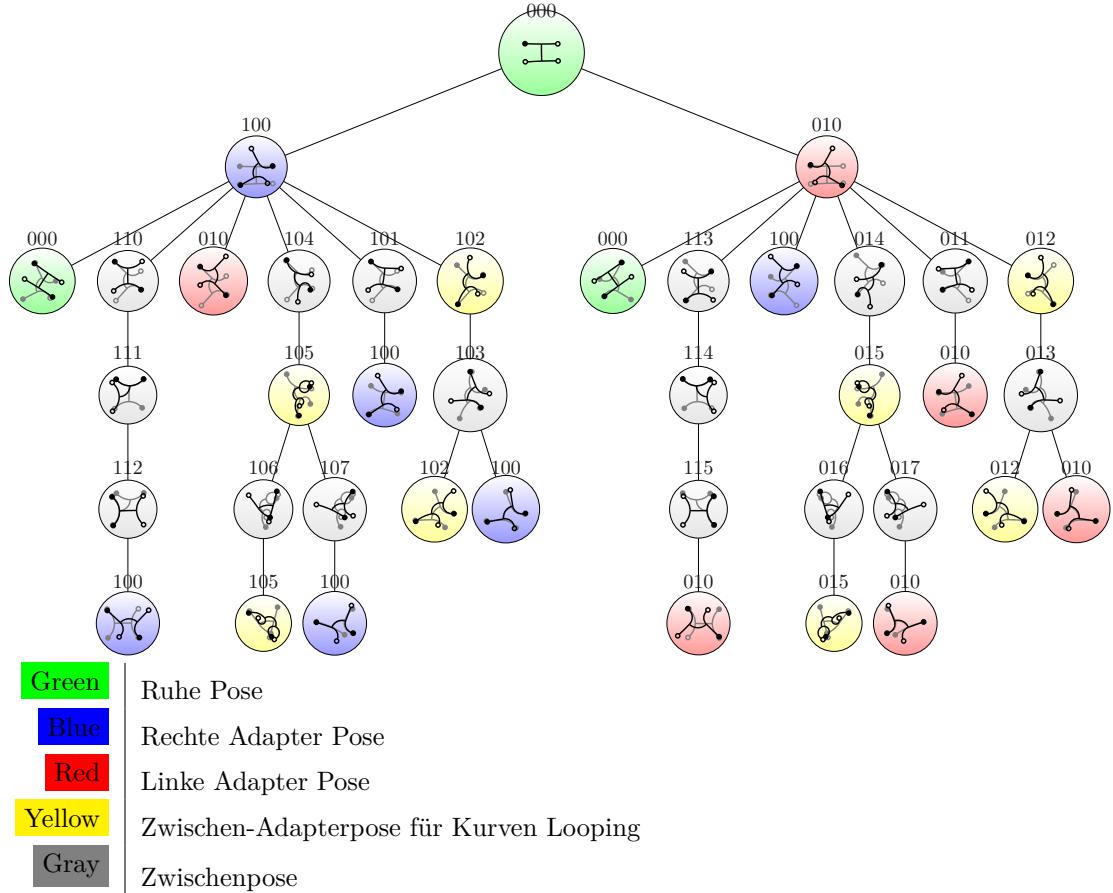
- Beispiele:



- Idee: Eine ausgewählte Anzahl an Posen, als Grundbausteine für einen beliebigen Gang.
- Diese dann wie Legosteine aufeinander setzen, um von A nach B zu gelangen.

2.2 Search Tree

- Folgender Suchbaum wurde implementiert.



- Dabei hat jede Kante des Baums eine Richtung und eine Gewichtung w .
- Die Gewichtung $w = ((\delta x, \delta y), \delta \varepsilon)$ gibt an, inwieweit das entsprechende Kind (Folgepose repräsentiert durch den Knoten, der mit der gewichteten Kante mit dem momentanen Knoten k verbunden ist) den Roboter relativ zur momentanen Orientierung bewegt: $(\delta x, \delta y)$ und wie weit diese Pose ihn drehen wird: $\delta \varepsilon$.
- Für eine gegebene, momentane Pose ρ_k wird für alle Kandidaten $j \in [0, \dots, J - 1]$ ausge-rechnet, wieweit der Abstand d_j der potentiell neuen Pose ρ_j zum Ziel \bar{x} ist:

$$d(\rho_k, w_j, \bar{x}) = \left| \bar{x} - \left(\mathbf{p}_{1,k} + \mathbf{R}(\varepsilon_k) \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} \right) \right|_2 \quad (2)$$

- Außerdem wird die Richtungsabweichung $\Delta \varepsilon_j$ aller potentiell neuen Pose j berechnet:

$$\Delta \varepsilon(\rho_k, w_j, \bar{x}) = \angle \left(\bar{x} - \left(\mathbf{p}_{1,k} + \mathbf{R}(\varepsilon_k) \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} \right), \mathbf{R}(\varepsilon_k + \delta \varepsilon) \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right) \quad (3)$$

- Die Folgepose ρ_{k+1} ergibt sich dann aus dem Minimum der mit $a = .5$ gewichteten Summe von Abstand und Orientierungsabweichung für alle Möglichkeiten ρ_j :

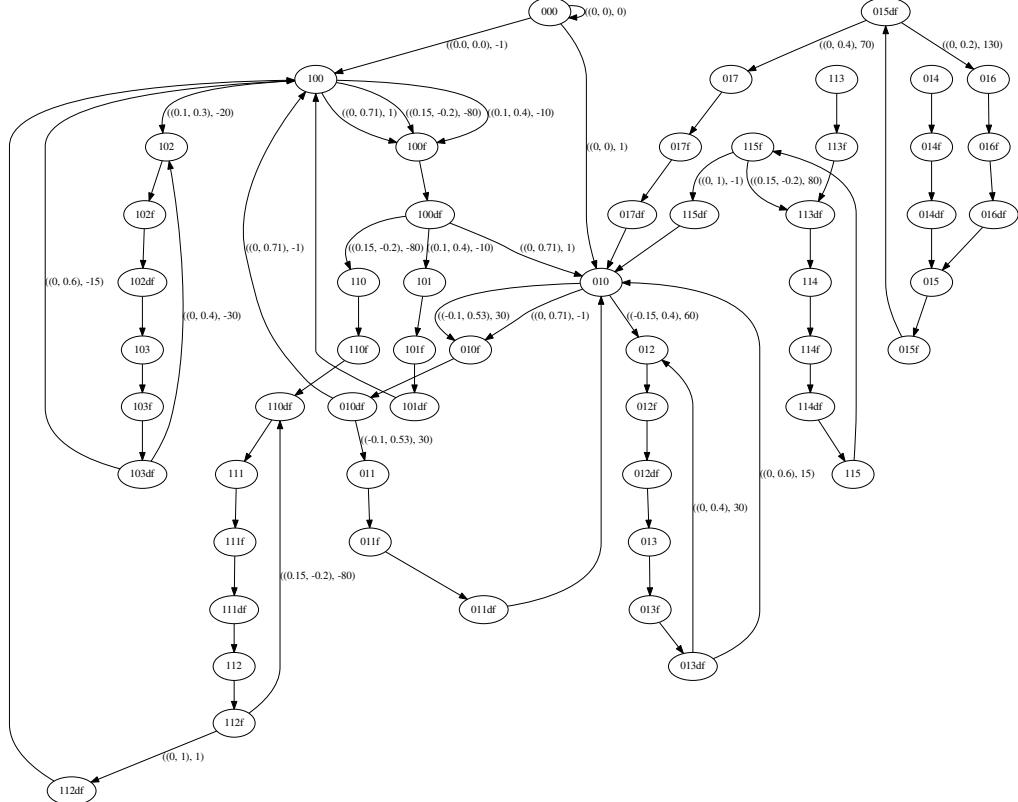
$$\rho_{k+1} = \min_j \left(a \frac{d_j}{d_{\min}} + (a - 1) \frac{\Delta \varepsilon_j}{\Delta \varepsilon_{\max}} \right) \quad (4)$$

- wobei $\Delta \varepsilon_{\max}$ die maximale Orientierungsabweichung aller Möglichkeiten ist; und d_{\min} der minimale Abstand aller Möglichkeiten ist.

2.2.1 Search Tree with weights

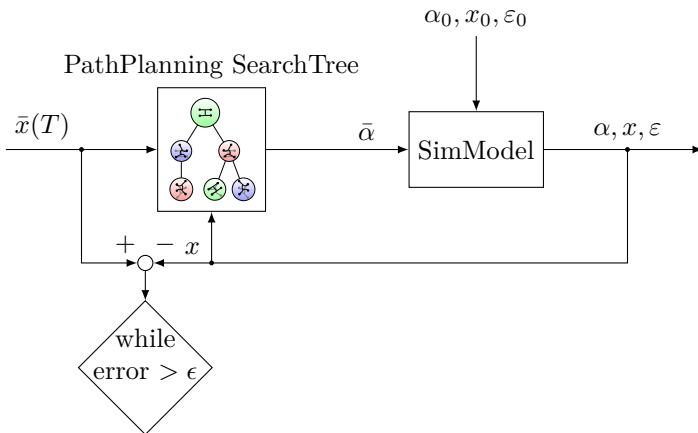
Complete SearchTree contains:

- all vertices (i.e. poses) including fix and defix poses
- weights of egdes: $w = ((\delta x, \delta y), \delta \varepsilon)$



2.3 Simulation Results

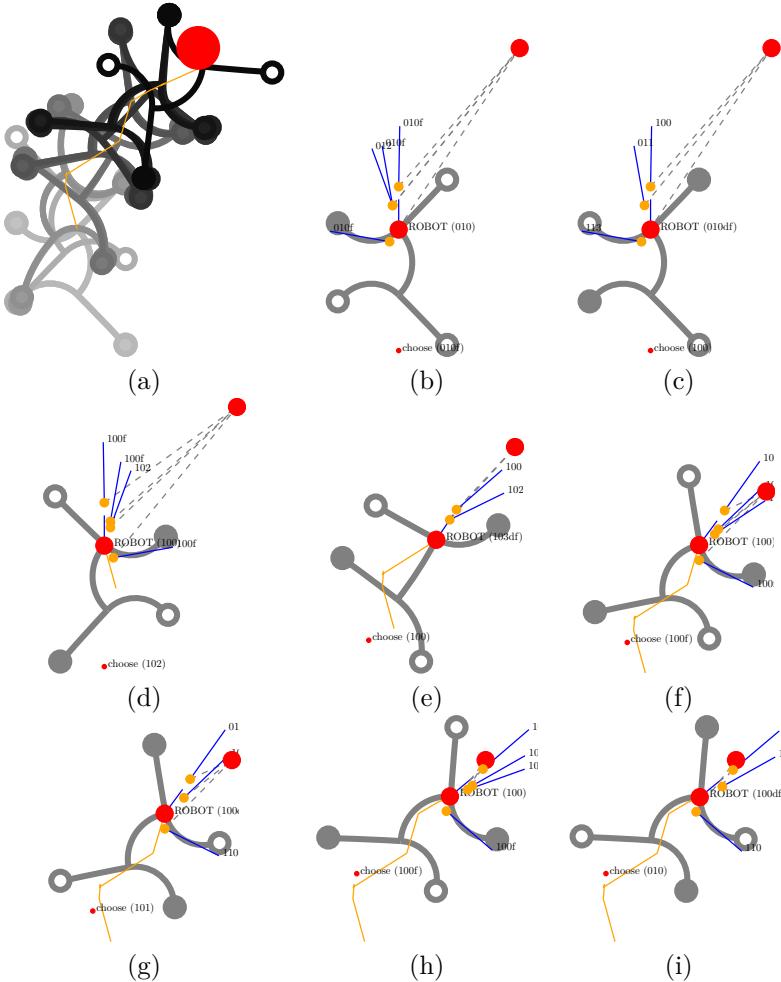
Block Diagram of Simulation:



2.3.1 Simulation Results Curve

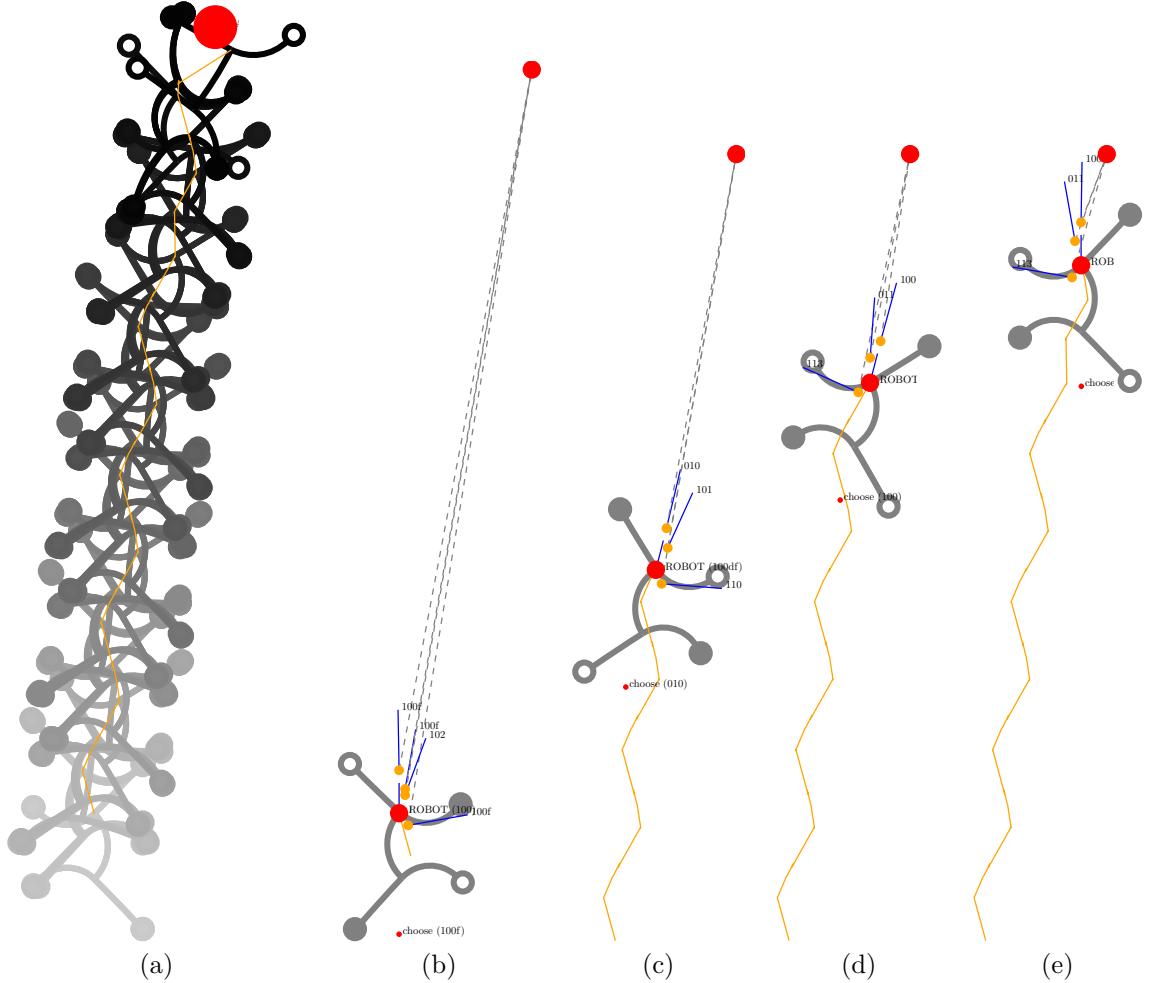
- Src: path_planner.py

- Startwerte: $p_{1,0} = (0, 0)$, $\bar{x} = (2, 3)$, $\varepsilon_0 = 0$, $\alpha_0 = [90, 0, -90, 90, 0]$
- Zu sehen sind:
 - Der gesamte Gang
 - Und alle Schritte, in denen eine Entscheidung getroffen werden musste.
 - Die Position des Roboters und die Zielposition sind als rote Punkte markiert
 - Das möglichen Folgeposen sind als orange Punkte dargestellt.
 - Alle Orientierungen sind als blaue Linien dargestellt



2.3.2 Simulation Results Straight

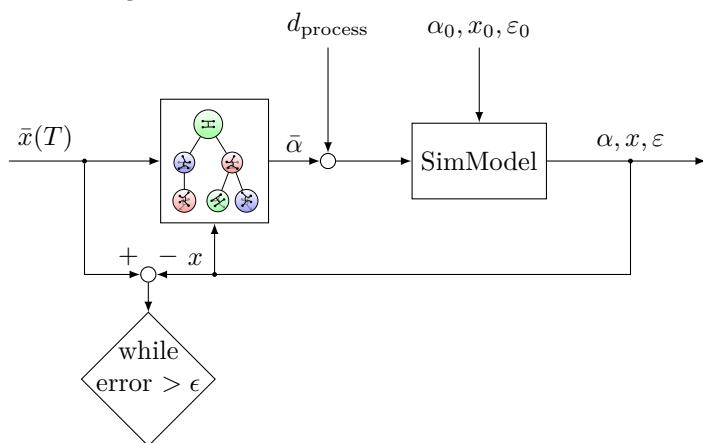
- Src: `path_planner.py`
- Startwerte: $p_{1,0} = (0, 0)$, $\bar{x} = (2, 13)$, $\varepsilon_0 = 0$, $\alpha_0 = [90, 0, -90, 90, 0]$



2.4 What happens if Process Noise occurs?

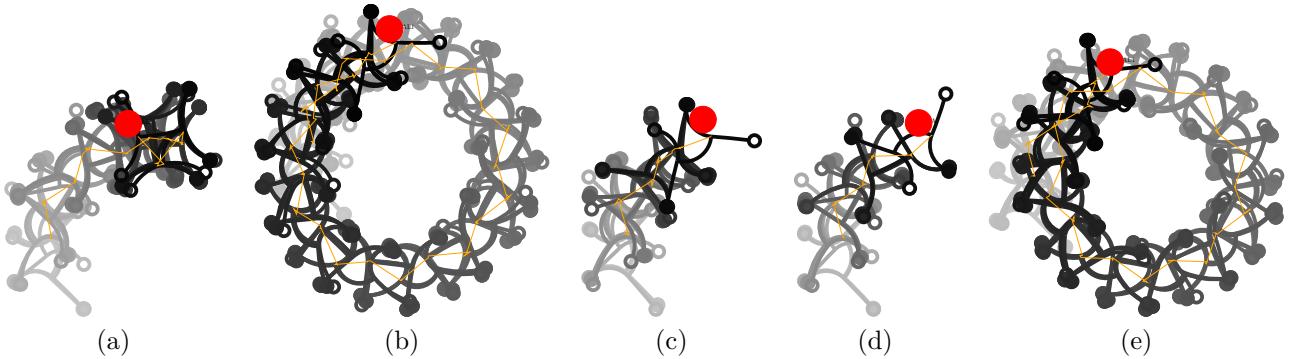
- Es kann nicht davon ausgegangen werden, dass der Roboter stets exakt die ReferenzWinkel einnimmt.
- Diese Abweichung soll nun modelliert werden
- Add process noise to Simulation input (0 mean, 5 standard deviation)
- Implementierung: `alp = alp + np.random.normal(0, 5, 5)`

Block Diagram of Simulation with noise:



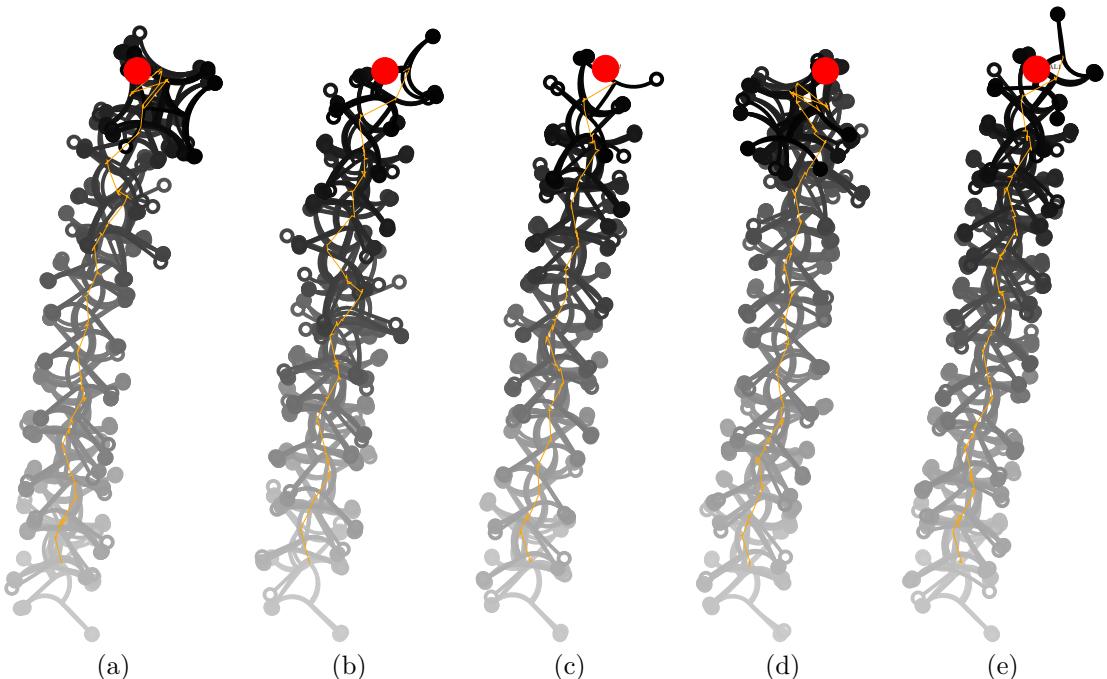
2.4.1 Curve Noise

- Src: `path_planner.py`
- Startwerte: $p_{1,0} = (0, 0)$, $\bar{x} = (2, 3)$, $\varepsilon_0 = 0$, $\alpha_0 = [90, 0, -90, 90, 0]$
- Die Simulation wurde 5 mal wiederholt.
- Gezeigt ist jeweils nur der gesamte Gang.



2.4.2 Straight Noise

- Src: `path_planner.py`
- Startwerte: $p_{1,0} = (0, 0)$, $\bar{x} = (2, 13)$, $\varepsilon_0 = 0$, $\alpha_0 = [90, 0, -90, 90, 0]$
- Die Simulation wurde 5 mal wiederholt.
- Gezeigt ist jeweils nur der gesamte Gang.



2.5 Conclusion

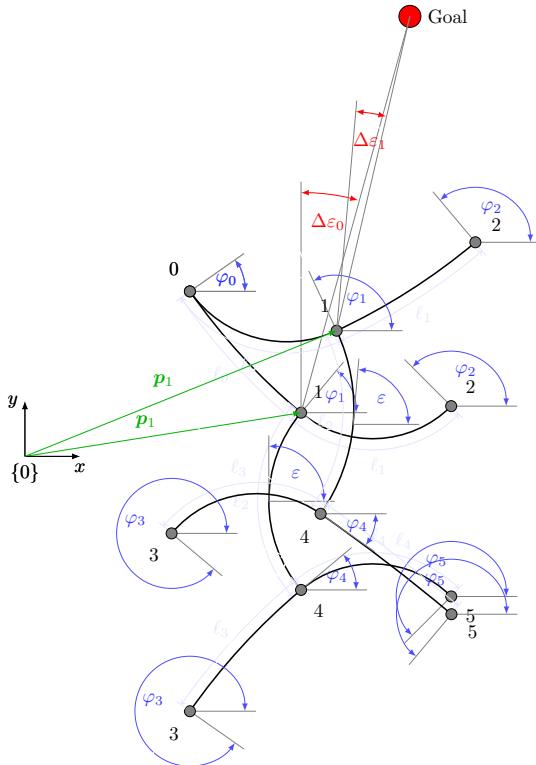
- SearchTree Pathplanner funktioniert. Der Roboter kommt ans Ziel! Sowohl in der Simulation als auch im Experiment.

- Durch die finite Anzahl an möglichen Folgeposen, läuft der Roboter allerdings im ZickZack auf das Ziel zu.
- Wird Prozess Rauschen in die Simulation eingebaut, wird der Prozess instabil.
- Teilweise läuft der Roboter knapp am Ziel vorbei und ist in einem Kurven-Loop gefangen.
 - Diese Phänomen konnte auch schon im Experiment gesichtet werden.
- Fazit:
 - PathPlanner SearchTree ist nicht wirklich flexibel. Kann zwar erweitert werden, aber umständlich.
 - instabil bei ProzessRauschen.

3 Finding a Analytic Model for describing the General Gait

3.1 Problem Statement

- Angenommen die Konfiguration / Pose des Roboters $\rho = [\alpha, p_1, \varepsilon]$ ist vollständig bekannt, wobei α die Gelenkkoordinaten / Biegewinkel der einzelnen Glieder sind, p_1 die Position des vorderen Torsoendes und ε die Orientierung des Roboters. Siehe Bild:



- Für die Pfadplanung, wäre eine Funktion hilfreich, die zu einer gegebenen Wunschdrehung $\Delta\varepsilon$, eine entsprechende Abfolge von Roboter-Konfigurationen / Posen ausgibt, sodass sich der Roboter entsprechend dreht.
- So könnte zB die Richtung des Roboters so justiert werden, dass er sich auf ein gegebenes Ziel zu bewegt.
- Für den geraden Gang ist eine analytische Funktion bekannt, die die Geschwindigkeit des Roboters einstellt. Geschwindigkeit im Sinne von Schrittweite, bzw. Vorschub pro Zyklus:

$$\boldsymbol{\alpha} = \begin{bmatrix} 45 - \frac{x_1}{2} \\ 45 + \frac{x_1}{2} \\ x_1 \\ 45 - \frac{x_1}{2} \\ 45 + \frac{x_1}{2} \end{bmatrix} \quad (5)$$

Die Schrittweite ist hier als x_1 beschrieben.

3.2 Approach: Guess structure for a analytic model for walking curves

- Src can be found: `analytic_model.py`

- Model:

x_1 beschreibt hier die Schrittweite

x_2 das Maß der Drehung.

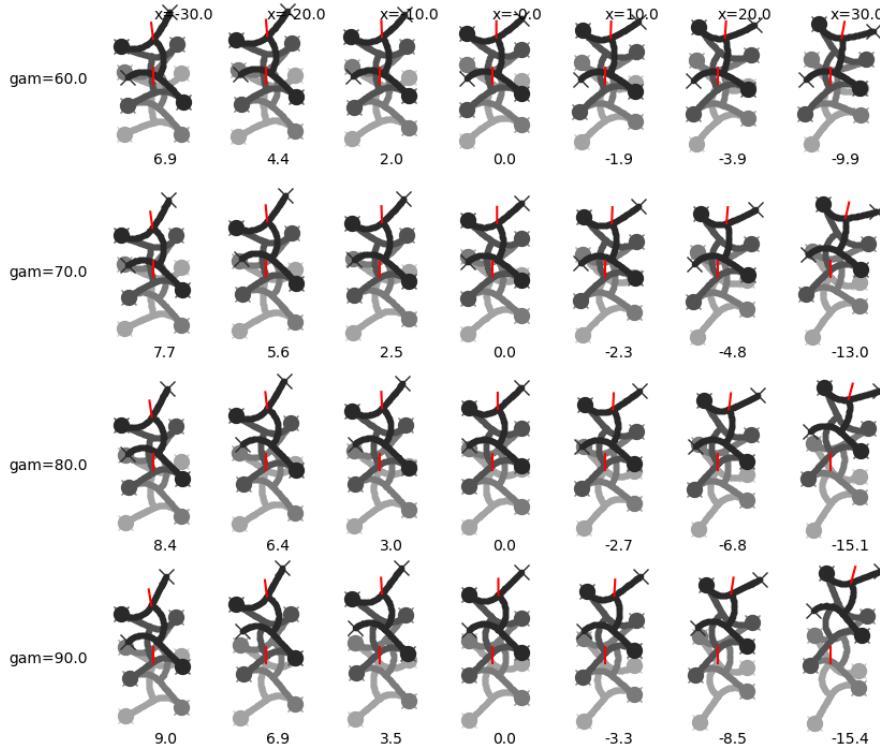
$$\boldsymbol{\alpha} = \begin{bmatrix} 45 - \frac{x_1}{2} \\ 45 + \frac{x_1}{2} \\ x_1 + x_2 \\ 45 - \frac{x_1}{2} \\ 45 + \frac{x_1}{2} \end{bmatrix} \quad (6)$$

- Method:

Simulate for different x_1 and x_2 (in der Abbildung unten ist $x_1 = \text{gam}$ und $x_2 = \text{x}$)

3.2.1 Simulation Results

Results für 2 Zyklen:



3.2.2 Observations

- Es funktioniert. Der Roboter läuft eine Kurve.
- Kurve ist unsymmetrisch. Rechts klappt besser als links.
- Startpose ist besser für Rechtskurve geeignet.
- Noch nichts über die innere Spannung des Roboters herausgefunden

3.3 Approach: Find a reasonable structure

- Src can be found:

analytic_model_2.py	First Tries
analytic_model_2_adv.py	Final Results

- Orientierung der Füße soll konstant bleiben:

$$\varphi_0 = \varepsilon + \frac{\alpha_2}{2} - \alpha_0 \quad (7)$$

Da im vorhergegangenem Versuch die asymmetrischen Aktuierung des Torsos schon zu guten Ergebnissen geführt hat, soll dieses Modell beibehalten werden. Allerdings in einer leicht variierten Form. x_2 ist nun ein relatives Maß für die Drehung:

$$\alpha_2 = x_1 + x_2|x_1| \quad (8)$$

Es muss also α_0 so gewählt werden, dass φ_0 möglichst unabhängig von x_i wird. Deshalb wird ein noch unbekannter Term x_3 hinzugefügt. Damit ergibt sich der Biegewinkel des Beines:

$$\alpha_0 = 45 + \frac{x_1}{2} + x_3. \quad (9)$$

Für die Orientierung des Fußes bedeutet das:

$$\varphi_0 = \varepsilon + \frac{x_1 + x_2|x_1|}{2} - \left(45 + \frac{x_1}{2} + x_3\right) \quad (10)$$

$$= \varepsilon - 45 + \frac{x_2|x_1|}{2} - x_3 \quad (11)$$

$$(12)$$

Es wird **angenommen**, dass die Orientierung des Roboters mit der Schrittweite linear wächst (i.e. Der Roboter dreht sich ein wenig zwischen seinen Extremposen):

$$\varepsilon = c_1 x_1 + \varepsilon_0 \quad (13)$$

Mit konstantem Orientierungswinkel $\varphi = \varphi_0$ ergibt sich somit:

$$\varphi_0 = c_1 x_1 + \varepsilon_0 - 45 + \frac{x_2|x_1|}{2} - x_3 \quad (14)$$

$$x_3 = c_1 x_1 + \frac{x_2|x_1|}{2} + c \quad (15)$$

Unter der **Annahme**, dass $\varphi_0 \approx \varepsilon_0 - 45$ ist, ergibt sich $c \approx 0$. Das meint, die Orientierung ändert sich nur minimal. entspricht also im Wesentlichen der Ausgangskonfiguration. Weiterhin wird **angenommen**, dass für einen fixierter Fuß der Term $c_1 x_1 \approx 0$ vernachlässigbar ist. Somit ergibt sich für den Biegewinkel des fixierten vorderen linken Beins:

$$\alpha_{0,f} = 45 - \frac{x_1}{2} + \frac{1}{2}x_2|x_1| \quad (16)$$

Wenn das Bein nicht fixiert ist, kann es beliebige Orientierung annehmen. Hierfür wird **angenommen**, dass sich die Drehung des Körpers erst in der nicht fixierten Phase eines Beines in dessen Orientierung auswirkt. Deshalb, wird der Term c_1x_1 in dieser Phase aktiv. Weiterhin wird **angenommen**, dass $c_1 = x_2$. Damit ergibt sich für einen nicht fixierten Fuß:

$$\alpha_{0,\bar{f}} = 45 - \frac{x_1}{2} + x_2x_1 \quad (17)$$

- Das resultierende Modell sieht so aus:

$$\alpha = \begin{bmatrix} 45 - \frac{x_1}{2} + f_0 \frac{1}{2}|x_1|x_2 + \bar{f}_0 x_1 x_2 \\ 45 + \frac{x_1}{2} + f_1 \frac{1}{2}|x_1|x_2 + \bar{f}_1 x_1 x_2 \\ x_1 + |x_1|x_2 \\ 45 - \frac{x_1}{2} + f_2 \frac{1}{2}|x_1|x_2 + \bar{f}_2 x_1 x_2 \\ 45 + \frac{x_1}{2} + f_3 \frac{1}{2}|x_1|x_2 + \bar{f}_3 x_1 x_2 \end{bmatrix} \quad (18)$$

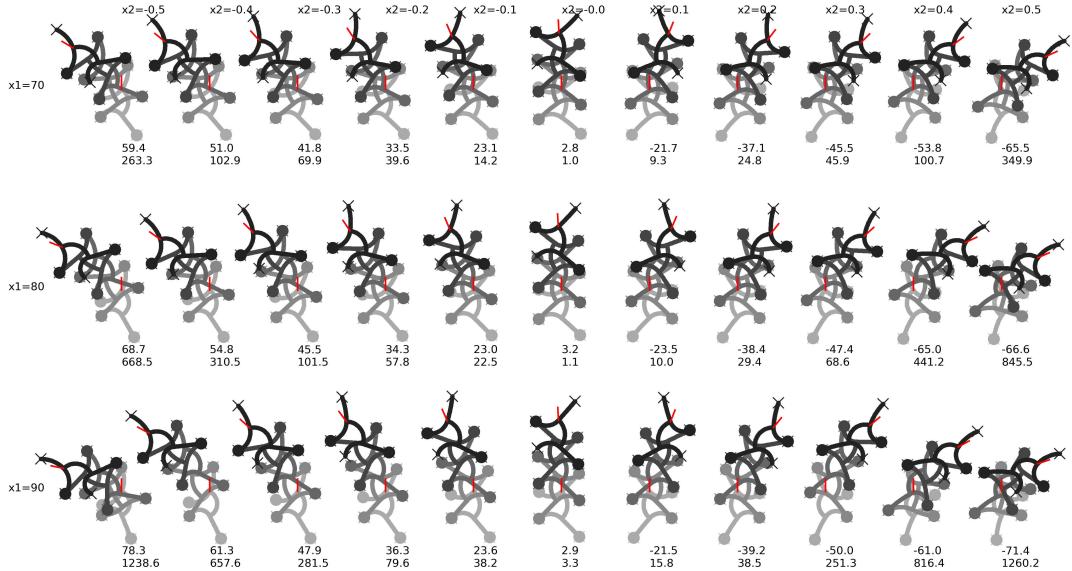
Wobei f_i den Zustand des Fußes beschreibt:

$$f_i = \begin{cases} 1 & \text{if} & \text{foot fixed} \\ 0 & \text{else} & \end{cases} \quad (19)$$

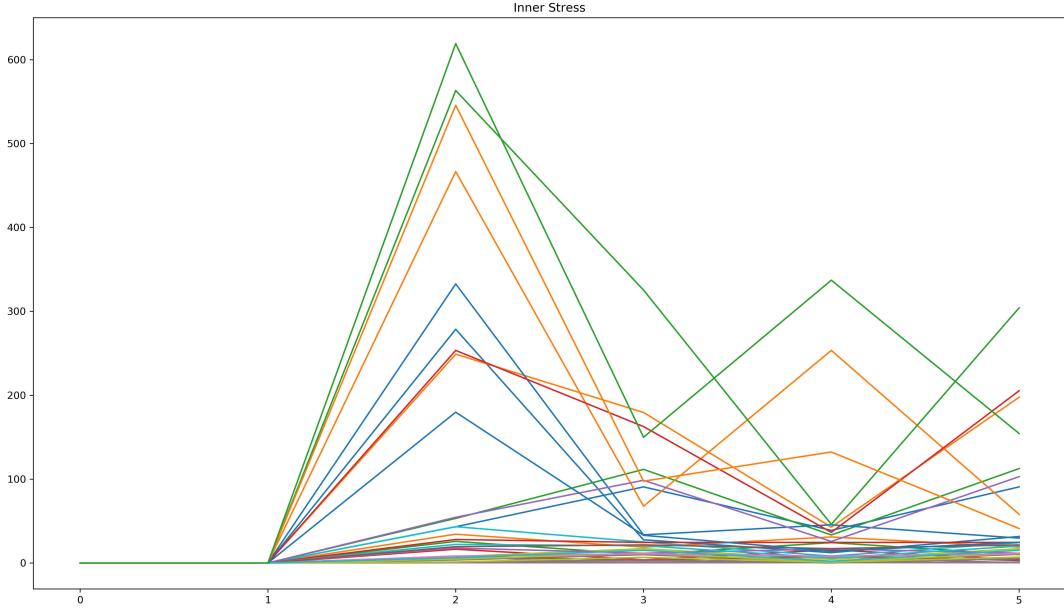
$$\bar{f}_i = \begin{cases} 0 & \text{if} & \text{foot fixed} \\ 1 & \text{else} & \end{cases} \quad (20)$$

3.3.1 Simulation Results

- In der abgebildeten Matrix sind in den Reihen konstante Schrittweite x_1
- Spalten: konstantes Kurvenmaß x_2
- Jedes Einzelbild zeigt die Simulation von zwei Zyklen mit dem zu (x_1, x_2) entsprechenden Laufmuster.
- Unter jedem Einzelbild ist die Drehung in Grad $\Delta\epsilon$ und die Summe der inneren Spannung aller Posen innerhalb des Laufes $\sum_i \sigma(\rho_i)$.



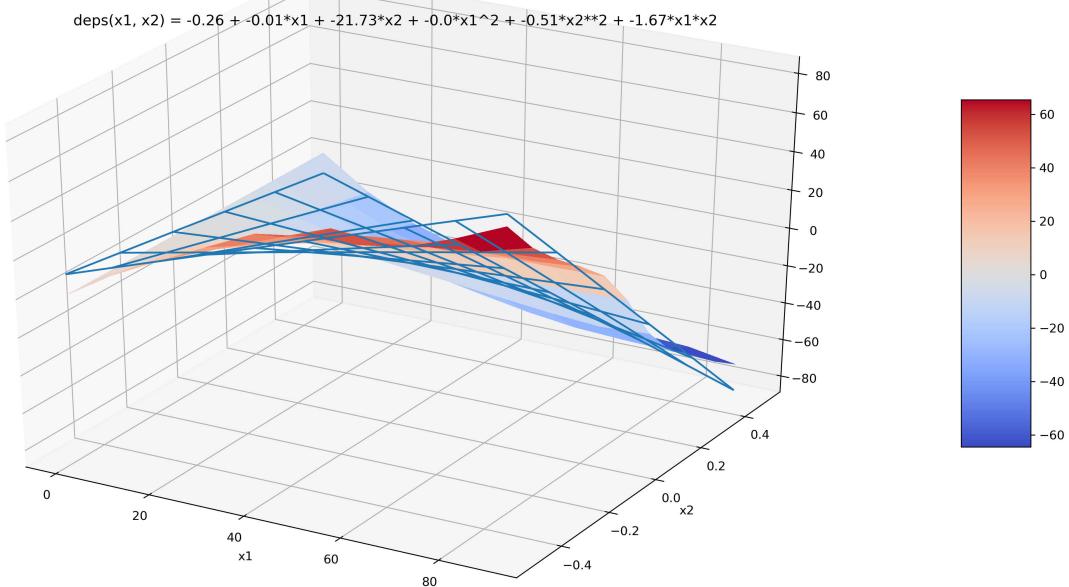
- Folgende Abbildung zeigt den Stressverlauf $\sigma(\rho_i)$
- Bis auf ein paar Ausreißer ist die innere Spannung über den Lauf moderat.



- Delta Epsilon kann jetzt gefittet werden:

$$\frac{\Delta \varepsilon}{2\text{cycle}}(x_1, x_2) = -21.7x_2 - .51x_2^2 - 1.67x_1x_2 \quad (21)$$

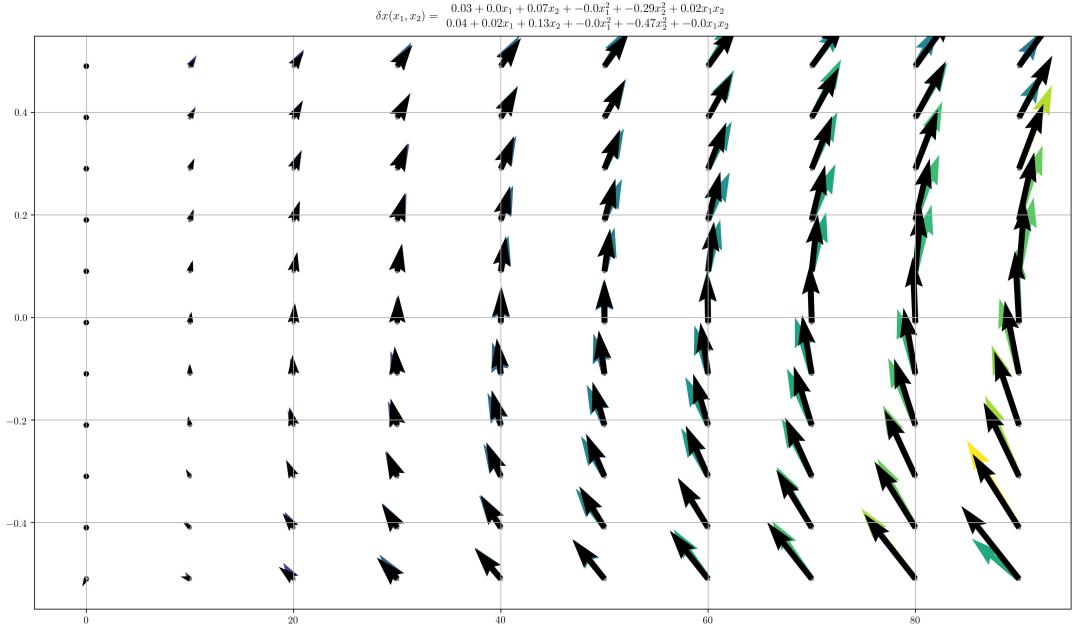
- SurfacePlot ist Simulations Data und WirframPlot ist der Fit



- Ebenso kann die Translation in Bewegungsrichtung gefittet werden:

$$\frac{\begin{bmatrix} \delta x \\ \delta y \end{bmatrix}(x_1, x_2)}{1\text{cycle}} = \begin{bmatrix} .07x_2 - .29x_2^2 + .02x_1x_2 \\ .02x_1 + .13x_2 - .47x_2^2 \end{bmatrix} \quad (22)$$

- Schwarze Pfeile sind Fit. Bunte Pfeile Simulations Ergebnisse.



3.4 Approach: Optimize Extra leg bending Angle for given extra torso bending

- Src can be found:

<code>analytic_model_3.py</code>	Optimizing x_3 und x_4
<code>eval_analytic_model_3.py</code>	Evaluate Simulation and Function Fitting
<code>eval2_analytic_model_3.py</code>	Simulate Gait with obtained gait law
- Nun soll untersucht werden, welche Extra Leg Bending Angle die innere Spannung des Roboters minimiert.
- Model:

$$\boldsymbol{\alpha} = \begin{bmatrix} 45 - \frac{x_1}{2} + \bar{f}_0 x_3 + f_0 x_4 \\ 45 + \frac{x_1}{2} + \bar{f}_1 x_3 + f_1 x_4 \\ x_1 |x_2| \\ 45 - \frac{x_1}{2} + \bar{f}_2 x_4 + f_3 x_3 \\ 45 + \frac{x_1}{2} + f_3 x_4 + f_4 x_3 \end{bmatrix} \quad (23)$$

- Annahme:
Die Extra Biegung x_3 für freie Beine und die Extra Biegung x_4 für fixierte Beine sind abhängig von der Extra Biegung x_2 für den Torso.

Hinter- und Vorderbeine sind nicht symmetrisch, aber kreuzweise symmetrisch: Die Extra-biegung für ein **nicht fixiertes Vorderbein** entspricht der Extrabiegung eines **fixierten Hinterbeins** und andersherum.

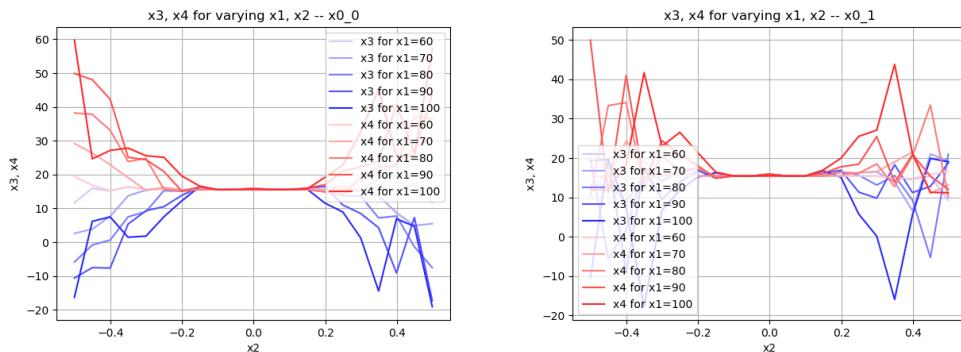
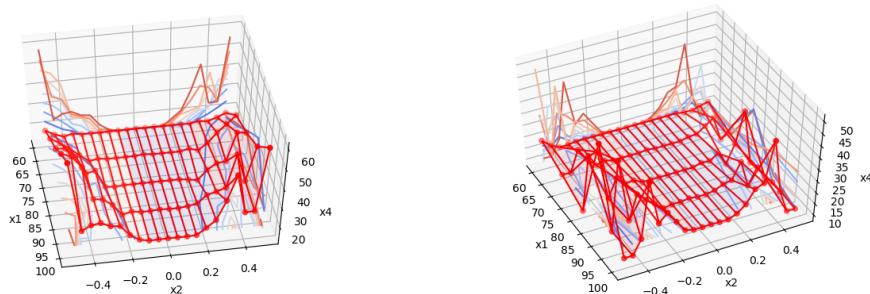
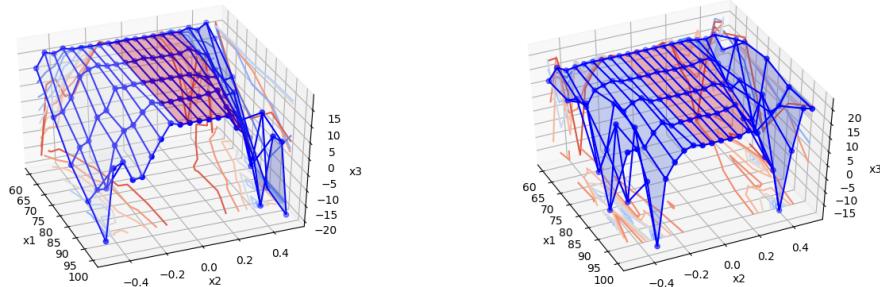
- Methode:
Für gegebenes Extra Torso Bending x_2 und gegebenen Schrittweite x_1 minimiere die Innere Spannung über den Gang mit n Zyklen aufsummiert:

Gegeben:	x_1	Schrittweite
	x_2	Extra Torsobiegung
Gesucht:	x_3	Extra Beinbiegung fixiert vorn
	x_4	Extra Beinbiegung fixiert hinten

$$(x_{3,\text{opt}}, x_{4,\text{opt}}) = \min_{x_3, x_4} \sum_k^{2n} \sigma(\rho_k(\alpha(x_1, x_2, x_3, x_4))) \quad (24)$$

3.4.1 Optimization Results

Results (links mit Startwert (10, 20) rechts mit Startwert (20, 10)):

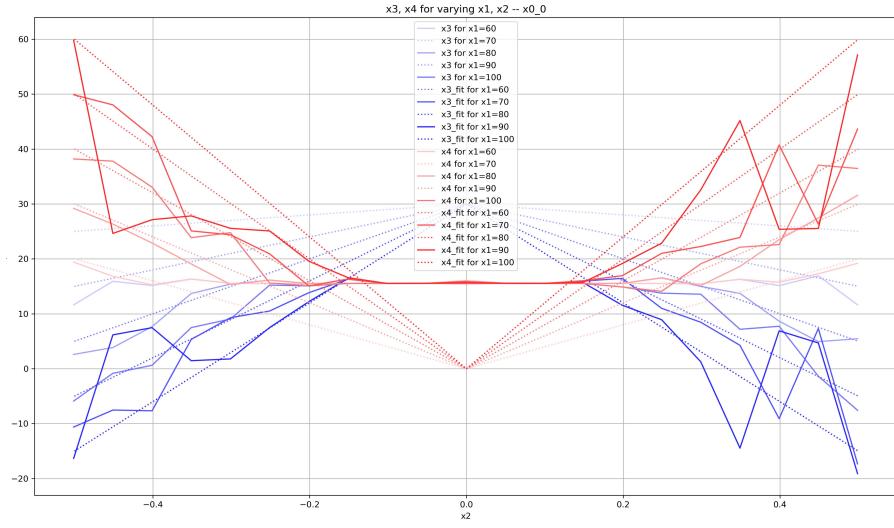


3.4.2 Function Fitting

- Es werden Ergebnisse des ersten Startwert genommen.
- Es ergibt sich:

$$x_3(x_1, x_2) = 30 - 2|x_1 x_2| + 110|x_2| \quad (25)$$

$$x_4(x_1, x_2) = 2|x_1 x_2| - 80|x_2| \quad (26)$$

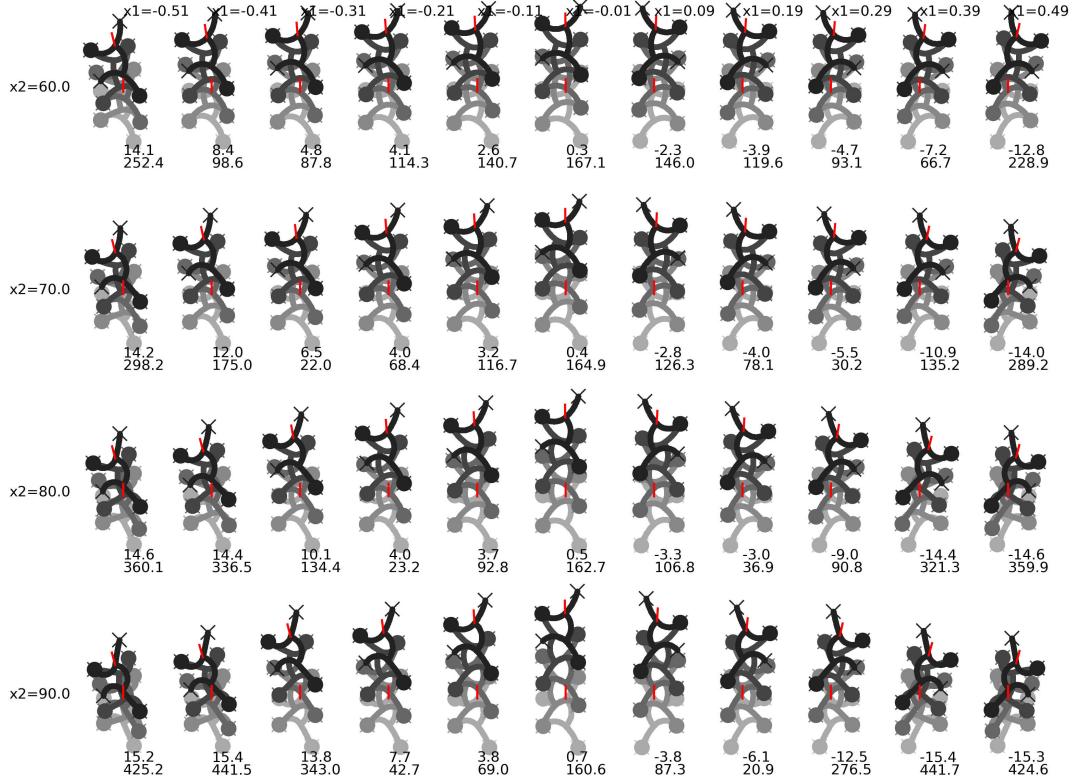


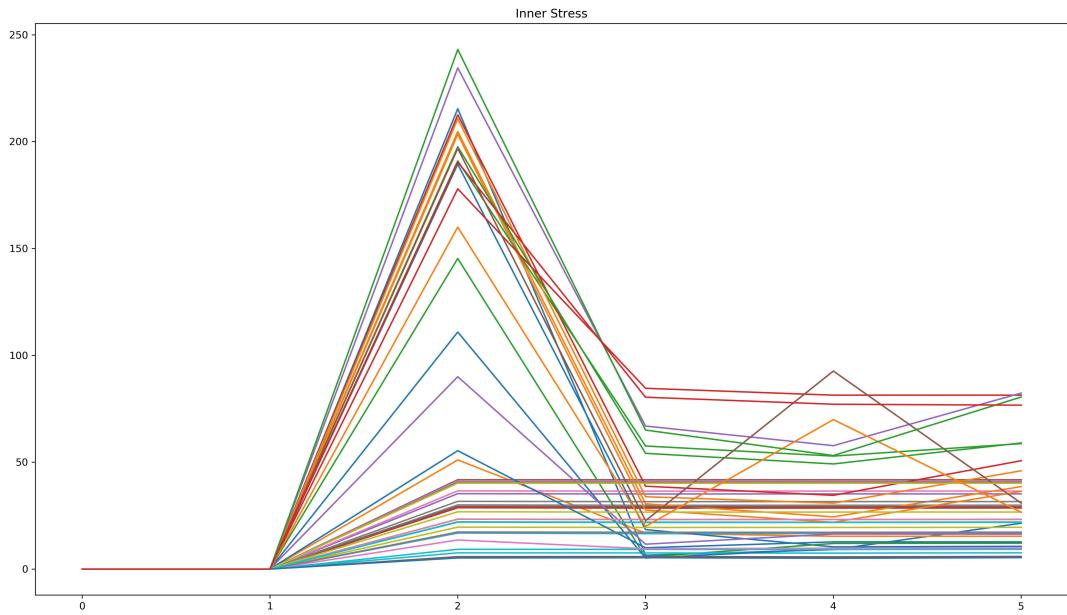
Und somit als Gait Law:

$$\alpha = \begin{cases} 45 - \frac{x_1}{2} + \bar{f}_0(30 - 2|x_1x_2| + 110|x_2|) + f_0(2|x_1x_2| - 80|x_2|) \\ 45 + \frac{x_1}{2} + f_1(30 - 2|x_1x_2| + 110|x_2|) + f_1(2|x_1x_2| - 80|x_2|) \\ x_1|x_2| \\ 45 - \frac{x_1}{2} + \bar{f}_2(2|x_1x_2| - 80|x_2|) + f_3(30 - 2|x_1x_2| + 110|x_2|) \\ 45 + \frac{x_1}{2} + f_3(2|x_1x_2| - 80|x_2|) + f_4(30 - 2|x_1x_2| + 110|x_2|) \end{cases} \quad (27)$$

3.4.3 Simulation Results

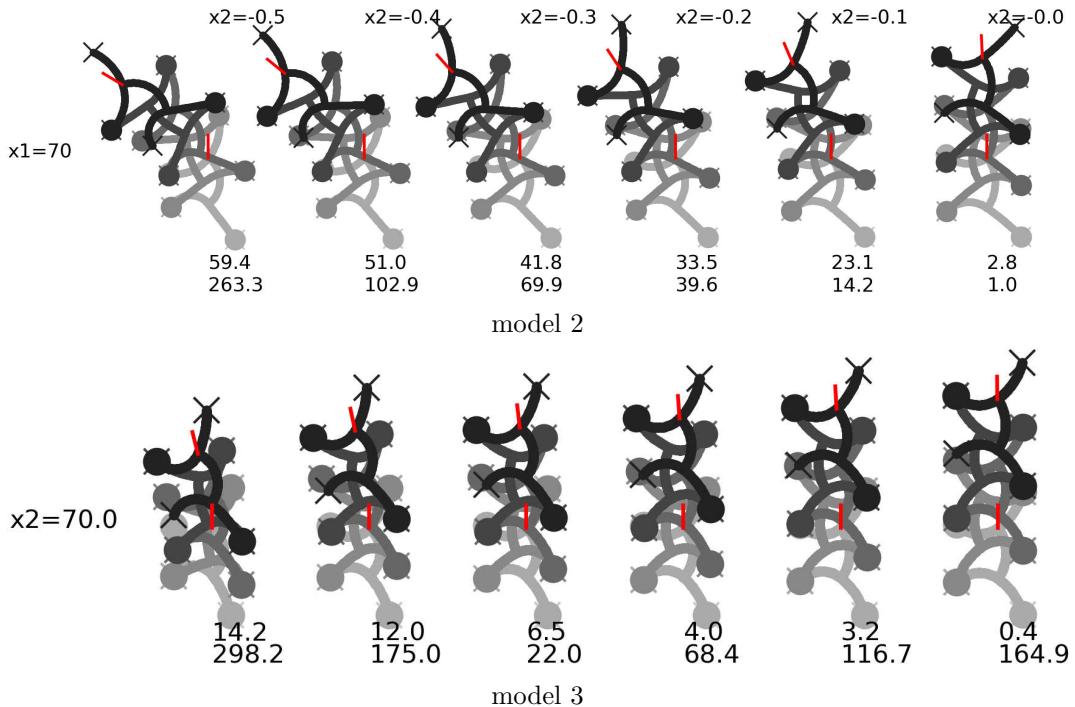
- Gleiche Struktur wie in model 2
- Stress ist deutlich kleiner. Drehung allerdings auch.





3.5 Compare model 2 and model 3

- Betrachtet man die beiden unteren Bilder, stellt man fest, dass der dritte Ansatz nicht wirklich erfolgreich war.
- Model 2 führt zu besseren Drehungen und kleinerer innerer Spannung. (Zumindest für die betrachtete Zeile ($x_1 = 70$))



3.5.1 Conclusion

- Es wird das Gait Law aus Model 2 für den Optimalen Pathplanner gewählt.

4 Optimal PathPlanner based on General Gait Model

4.1 Idea

- Wie schon beim SearchTreePathplanner soll die optimale Folgepose berechnet werden.
- Optimal in dem Sinne, dass sie den Roboter näher an das Ziel bringt und seine Orientierung so verändert, dass er in die Richtung des Ziels zeigt.
- Dieses Mal soll aber nicht aus einer endlichen Anzahl aus vorher definierten möglichen Folgeposen gewählt werden, sondern aus der unendlichen Anzahl aller möglichen Posen.
- Das Minimierungsproblem ist ähnlich:

$$\boldsymbol{\rho}_{k+1} = \min_j \left(a \frac{d(\boldsymbol{\rho}_j)}{d_k} + (1-a) \frac{\Delta\varepsilon(\boldsymbol{\rho}_j)}{\Delta\varepsilon_k} \right) \quad (28)$$

- Mithilfe des analytischen Modells lässt sich die Distanz zum Ziel d und die Orientierungsabweichung $\Delta\varepsilon$ in Abhängigkeit der Schrittweite x_1 und des Drehungsmaßes x_2 beschreiben.
- Zur Erinnerung:

$$\begin{bmatrix} \delta x \\ \delta y \end{bmatrix} (x_1, x_2) / \text{cycle} = \delta \mathbf{x}(x_1, x_2) = \begin{bmatrix} .07x_2 - .29x_2^2 + .02x_1x_2 \\ .02x_1 + .13x_2 - .47x_2^2 \end{bmatrix} \quad (29)$$

$$\delta\varepsilon(x_1, x_2) / \text{cycle} = -.005x_1 - 10.85x_2 - .255x_2^2 - .835x_1x_2 \quad (30)$$

- Der Abstand d ergibt sich zu:

$$d(\boldsymbol{\rho}_k, \bar{\mathbf{x}}, x_1, x_2) = \left| \bar{\mathbf{x}} - (\mathbf{x}_k + \mathbf{R}(\varepsilon_k) \delta \mathbf{x}(x_1, x_2)) \right|_2 \quad (31)$$

- Die Richtungsabweichung $\Delta\varepsilon'$ der Folgepose ergibt sich zu:

$$\Delta\varepsilon'(\boldsymbol{\rho}_k, \bar{\mathbf{x}}, x_1, x_2) = \angle(\bar{\mathbf{x}} - (\mathbf{x}_k + \mathbf{R}(\varepsilon_k) \delta \mathbf{x}(x_1, x_2)), \mathbf{R}(\varepsilon_k + \delta\varepsilon(x_1, x_2)) \mathbf{e}_x) \quad (32)$$

wobei \mathbf{e}_x der Einheitsvektor in x -Richtung ist.

- Das Minimierungsproblem lässt sich nun konkretisieren:

$$x_{1,\text{opt}}, x_{2,\text{opt}} = \min_{x_1, x_2} \left(a \frac{d(x_1, x_2)}{d_k} + (1-a) \frac{\Delta\varepsilon(x_1, x_2)}{\Delta\varepsilon_k} \right) \quad (33)$$

Der nächste Zyklus (i.e. die nächsten zwei Posen) ist damit definiert:

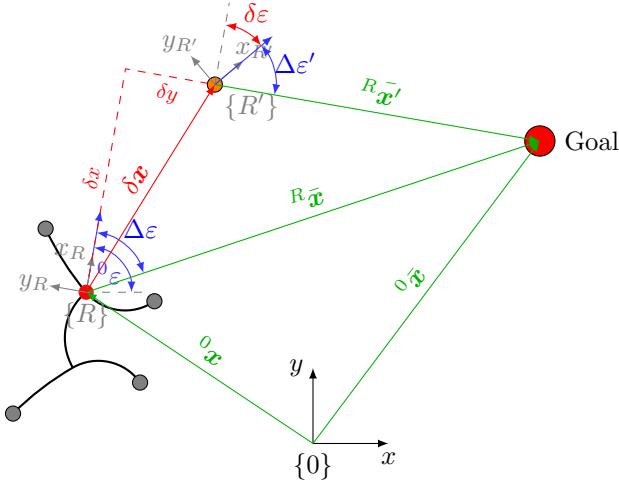
$$\boldsymbol{\rho}_{k+1} = [\boldsymbol{\alpha}(x_1, x_2), \bar{\mathbf{f}}_k], \quad (34)$$

$$\boldsymbol{\rho}_{k+2} = [\boldsymbol{\alpha}(-x_1, x_2), \mathbf{f}_k], \quad (35)$$

wobei $\bar{\mathbf{f}}$ die genau entgegen gesetzte Fixierung von \mathbf{f} ist.

4.2 Coordinate Transformation in order to reduce complexity

- Idee: Formulierung des Optimierungsproblems in Koordinaten System des Roboters, Um Komplexität zu verringern.
- Messung wird im globalen Kamera system erfolgen, d.h. folgende Größen werden zu verfügen stehen:
 - ${}^0\boldsymbol{x}$ Position des Roboters im globalen (Kamera-) COS
 - ${}^0\bar{\boldsymbol{x}}$ Soll Position im Globalen COS
 - ${}^0\varepsilon$ Orientierung des Roboters im Globalen COS



- Da das Problem nun im roboterfesten Koordinaten system beschrieben wird, ist der Ortsvektor des Roboters ${}^R\boldsymbol{x} = \mathbf{0}$.
- Den Ortsvector vom Roboter zum Ziel erhält man durch folgende Koordinatentransformation:

$${}^R\bar{\boldsymbol{x}} = \mathbf{R}({}^0\varepsilon)({}^0\bar{\boldsymbol{x}} - {}^0\boldsymbol{x}) \quad (36)$$

- Die Position der Folgepose ${}^R\boldsymbol{x}'$ beschrieben im Roboterfesten COS:

$${}^R\boldsymbol{x}' = \delta\boldsymbol{x}(x_1, x_2) \quad (37)$$

- Der Vektor der von der Folge Pose zum Ziel zeigt, ergibt sich:

$${}^R\bar{\boldsymbol{x}}' = {}^R\bar{\boldsymbol{x}} - {}^R\boldsymbol{x}' \quad (38)$$

- Der Abstand der Folgepose des Roboters beschrieben im Koordinatensystem des Roboters $\{R\}$ zum Ziel d vereinfacht sich nun zu:

$$d({}^R\bar{\boldsymbol{x}}, x_1, x_2) = |{}^R\bar{\boldsymbol{x}}'|_2 \quad (39)$$

$$= |{}^R\bar{\boldsymbol{x}} - \delta\boldsymbol{x}(x_1, x_2)|_2 \quad (40)$$

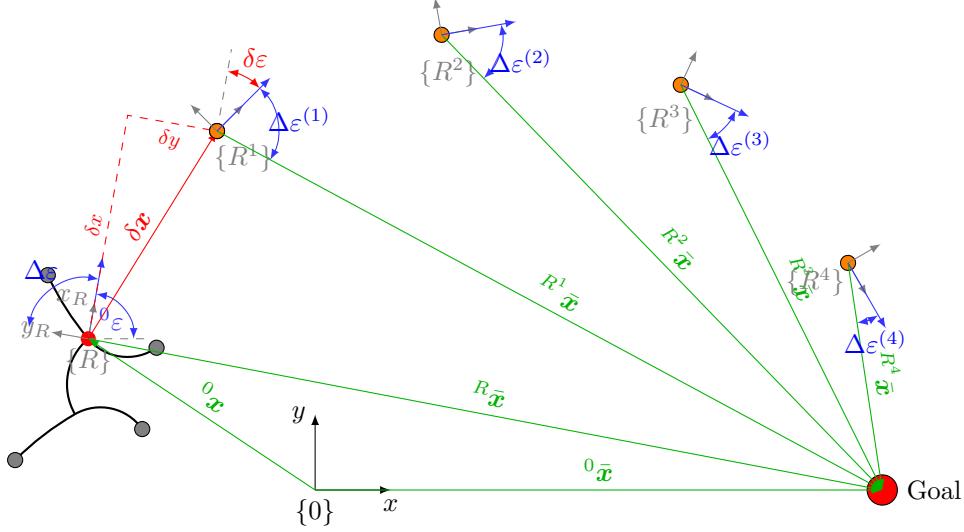
$$= \sqrt{({}^R\bar{\boldsymbol{x}}_x - \delta\boldsymbol{x}_x(x_1, x_2))^2 + ({}^R\bar{\boldsymbol{x}}_y - \delta\boldsymbol{x}_y(x_1, x_2))^2} \quad (41)$$

- Und die Richtungsabweichung $\Delta\varepsilon'$ der Folgepose vereinfacht sich nun zu:

$$\Delta\varepsilon'({}^R\bar{\boldsymbol{x}}, x_1, x_2) = \angle({}^R\bar{\boldsymbol{x}} - \delta\boldsymbol{x}(x_1, x_2), {}^R\boldsymbol{e}_x) - \delta\varepsilon(x_1, x_2) \quad (42)$$

$$= \text{atan}\left(\frac{{}^R\bar{\boldsymbol{x}}_y - \delta\boldsymbol{x}_y(x_1, x_2)}{{}^R\bar{\boldsymbol{x}}_x - \delta\boldsymbol{x}_x(x_1, x_2)}\right) - \delta\varepsilon(x_1, x_2) \quad (43)$$

4.2.1 Explizite Formulierung für mehrere Zyklen



- Mit jedem Zyklus dreht sich der Roboter um $\delta\varepsilon$ und verschiebt sich um $\delta\mathbf{x}$.
- Die Position der $(n+1)$ -ten Folgepose beschrieben im COS der (n) -ten Folgepose kann dann rekursiv beschrieben werden:

$${}^{R^{(n)}}\mathbf{x}^{(n+1)} = \delta\mathbf{x}(x_1, x_2) \quad (44)$$

- Den Ortsvector der (n) -ten Pose des Roboters zum Ziel (beschrieben im COS der (n) -ten Folgepose) erhält man durch folgende Koordinatentransformation der vorhergegangen Pose:

$${}^{R^{(n)}}\bar{\mathbf{x}}^{(n)} = \mathbf{R}(-\delta\varepsilon) \left({}^{R^{(n-1)}}\bar{\mathbf{x}}^{(n-1)} - {}^{R^{(n-1)}}\mathbf{x}^{(n)} \right) \quad (45)$$

- Explizit lässt sich das Formulieren zu:

$${}^{R^{(n)}}\bar{\mathbf{x}}^{(n)} = \mathbf{R}(-\delta\varepsilon) \left(\mathbf{R}(-\delta\varepsilon) \left({}^{R^{(n-2)}}\bar{\mathbf{x}}^{(n-2)} - \delta\mathbf{x} \right) - \delta\mathbf{x} \right) \quad (46)$$

$$= \mathbf{R}^n(-\delta\varepsilon)^R\bar{\mathbf{x}}^{(0)} - \sum_{i=0}^n \mathbf{R}^i(-\delta\varepsilon)\delta\mathbf{x} \quad (47)$$

$$= \mathbf{R}(-n\delta\varepsilon)^R\bar{\mathbf{x}}^{(0)} - \sum_{i=0}^n \mathbf{R}(-i\delta\varepsilon)\delta\mathbf{x} \quad (48)$$

da mehrfache Rotation um die gleiche Achse und somit $\mathbf{R}^n(x) = \mathbf{R}(nx)$

- Und die Richtungsabweichung $\Delta\varepsilon^{(n)}$ der (n) -ten Pose zu:

$$\Delta\varepsilon^{(n)} = \angle \left({}^{R^{(n)}}\bar{\mathbf{x}}^{(n)}, {}^{R^n}\mathbf{e}_x \right) \quad (49)$$

$$= \text{atan} \left(\frac{{}^{R^{(n)}}\bar{\mathbf{x}}_y^{(n)}}{{}^{R^{(n)}}\bar{\mathbf{x}}_x^{(n)}} \right) \quad (50)$$

- Nun kann der Abstand zum Ziel d_n nach n Zyklen mit dem zu $\delta\mathbf{x} = [\delta x, \delta y, \delta\varepsilon]^\top$ korrespondierenden Laufmuster formuliert werden:

$$d_n({}^R\bar{\mathbf{x}}, \delta\mathbf{x}) = \left| \mathbf{R}(-n\delta\varepsilon)^R\bar{\mathbf{x}} - \sum_{i=0}^n \mathbf{R}(-i\delta\varepsilon)\delta\mathbf{x} \right|_2 \quad (51)$$

4.3 Simplified Optimization Problem

- Da die Richtungsabweichung $\Delta\varepsilon$ im der Abstandsformel für mehrere Zyklen schon implizit enthalten ist, kann das Minimierungsproblem nun drastisch vereinfacht werden (die Funktion wird quadriert, um die Wurzel zu beseitigen und die Ableitung zu vereinfachen):

$$x_{1,\text{opt}}, x_{2,\text{opt}} = \min_{x_1, x_2} d_n^2({}^R\bar{\mathbf{x}}, \delta\mathbf{x}) \quad (52)$$

- Zur Lösung des Optimierungsproblems ist die Jacobi-Matrix sehr hilfreich:

$$\mathbf{J} = \begin{bmatrix} \frac{\delta f}{\delta x_1} & \frac{\delta f}{\delta x_2} \end{bmatrix} \quad (53)$$

- Möge die Suche nach der Jacobi Matrix beginnen

4.3.1 Calculation of Jacobian

Ableitung der Trigonometrischen Summe in einer Dimension Als Euler'sche Formulierung

$$\sum_{k=0}^n \sin(k\delta_\varepsilon) = \frac{i(-e^{i\delta_\varepsilon n} + 1)(-e^{i\delta_\varepsilon(n+1)} + 1)e^{-i\delta_\varepsilon n}}{2(-e^{i\delta_\varepsilon} + 1)} \quad (54)$$

Ableitung:

$$\frac{(-ne^{i\delta_\varepsilon} + ne^{2i\delta_\varepsilon(n+1)} - ne^{i\delta_\varepsilon(2n+1)} + n - e^{i\delta_\varepsilon} + 2e^{i\delta_\varepsilon(n+1)} - e^{i\delta_\varepsilon(2n+1)})e^{-i\delta_\varepsilon n}}{2(e^{2i\delta_\varepsilon} - 2e^{i\delta_\varepsilon} + 1)} \quad (55)$$

Laut Formelsammlung (BFS39):

$$\sum_{k=0}^n \sin(k\delta_\varepsilon) = \frac{\sin\left(\frac{\delta_\varepsilon n}{2}\right) \sin\left(\frac{\delta_\varepsilon(n+1)}{2}\right)}{\sin\left(\frac{\delta_\varepsilon}{2}\right)} \quad (56)$$

Ableitung

$$\frac{n \cos(\delta_\varepsilon n) - n \cos(\delta_\varepsilon(n+1)) + \cos(\delta_\varepsilon n) - 1}{2(-\cos(\delta_\varepsilon) + 1)} \quad (57)$$

Partielle Ableitung der Trigonometrischen Summe Euler:

$$\frac{i(-e^{in(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1)(-e^{i(n+1)(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1)e^{-in(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)}}{2(-e^{i(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1)} \quad (58)$$

Ableitung:

$$\frac{\delta}{\delta x_1} = -\frac{(c_0 + c_3x_2) \left(\left(n \left(e^{in(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1 \right) \left(-e^{i(n+1)(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1 \right) + (n+1) \left(-e^{in(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1 \right) \right) \right)}{2(-e^{i(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1)} \quad (59)$$

$$\frac{\delta}{\delta x_2} = -\frac{\left(\left(n \left(e^{in(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1 \right) \left(-e^{i(n+1)(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1 \right) + (n+1) \left(-e^{in(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1 \right) \right) \right)}{2(-e^{i(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)} + 1)} \quad (60)$$

Normal:

$$\frac{\sin\left(\frac{n(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)}{2}\right) \sin\left(\frac{(n+1)(c_0x_1+c_1x_2+c_2x_2^2+c_3x_1x_2)}{2}\right)}{\sin\left(\frac{c_0x_1}{2} + \frac{c_1x_2}{2} + \frac{c_2x_2^2}{2} + \frac{c_3x_1x_2}{2}\right)} \quad (61)$$