FH University of Applied Sciences
TECHNIKUM WIEN

Gruppenaufgabe: Problem 1A

MKS – Angewandte Mehrkörpersimulation im Maschinenbau

Buczolits Martin (mb23m001)
Gasser Philipp (mb23m002)
Kacarevic Nikola (mb23m016)
Krasniqi Kelvin (mb23m005)
Stojilkovic Emil (mb23m017)

FH University of Applied Sciences TECHNIKUM WIEN

Inhaltsverzeichnis

1. Angabe

Problem 1: Karussell

Ein Karussell soll vereinfacht modelliert werden. Das Koo befindet sich an der Befestigung der Drehachse am Boden und ist in der x-z-Ebene um $\psi_0 = \frac{\pi}{6}$ geneigt. Die Sitze, können über eine Schiene (grün) in radialer Richtung auf das Karussell aufgeschoben werden.
Sie werden mittels eines Federmechanismus daran gehindert im Betrieb herausgeschleudert zu werden. Die restlichen Sitze sollen über das Karussell verschmiert als Trägheit angenommen werden (angenähertes Trägheitsmoment gemäß Vorlesung)

Gruppe A + B: Modellieren sie das System entsprechend der nötigen Freiheitsgrade und deren zeitlicher Veränderung gemäß den recherchierten Daten (Abmessungen, Massen, Geschwindigkeiten)

Gruppe 1A:

Modellieren sie das System mit Verschlussmechanismus so, dass bei maximaler Geschwindigkeit ein Versagen vermieden wird. Das ist der Fall, wenn sich der Einrastbolzen nicht weiter als 5 mm radial bewegen kann.

Abbildung 1: Angabe Karussell

2. Recherche

Mithilfe von Google Search konnte der Typ des Karussells ausfindig gemacht werden. Er entspricht nämlich dem aus folgendem YouTube-Video:

https://www.youtube.com/watch?app=desktop&v=Vh6AVhZDfUc

Ein Blick in die Videobeschreibung offenbart nun den Namen des Karussells: „Hully Gully Disco Dancer“. Scheinbar dürfte es sich dabei, um ein vor allen in den Niederlanden (insbesondere der Kermis Weert) eingesetztes Fahrgeschäft handeln.

Wird nun mit den Schlagwörtern „Hully Gully Disco Dancer“ auf Google gesucht, offenbart sich folgende Webseite:

https://www.polypweb.eu/viewtopic.php?t=907

Obiger Link zeigt eine Sammlung verschiedenster Fotos des in der Angabe gegebenen Karussells. Es wurde im Jahre 1968 von dem deutschen Unternehmen MACK Rides GmbH & Co KG hergestellt. Der Fakt, dass MACK seit 1920 erst 170 Fahrgeschäfte erbaut hat, deutet darauf hin , dass es sich bei dem Karussell um ein Unikat handelt. Der Grund, warum zahlreiche Bilder an verschiedenen Standorten online abrufbar sind, liegt an der Historie der Vorbesitzer:

- 1968 – 1972: Distel (München, Deutschland)
- 1973 – 1989: Müller (Hannover, Deutschland)
- 1990 – 2000: Friedrich (Staakow, Deutschland)
- 2000 – 2005: Schwill (Görlitz, Deutschland)
- 2006: Van der Laan (Niederlande)
- 2007 – 2011: Greif /Castro (Belgien)
- 2012 – 11/2016: Boy Boers (Niederlande)
- 2017 – Now: Baghdad Island (Irak)

(Hinweis: Erst ab 1972 wurde der Name „Hully Gully“ eingebürgert, davor wurde es „Schabinchen“ genannt. Zu dem Zeitpunkt der Aufnahme dürfte es deshalb in Kermis Weert gestanden haben, welche von Boy Boers veranstaltet wurde.)

Mehr Informationen zum Karussell können unter folgendem Link abgerufen werden:

https://kulturgut-volksfest.de/enzyklopaedie/hully-gully/

Leider ist nun aber so, dass weder Online noch auf Nachfrage bei dem Hersteller und dem derzeitigen Inhaber, Eigenschaften, wie Fahrgeschwindigkeit oder Gewicht, ausfindig gemacht werden konnten. Deshalb wurde ein ähnliches Karussell, welches auch der Kategorie „trabant ride“ untergeordnet wird, das Dancing-Fly Karussell, gewählt.

https://www.emilianalunapark.com/ride/dancing-fly/

Nach Rücksprache mit EMILIANA LUNA PARK wurden zwei PDF-Dateien bereitgestellt, die Informationen zu Abmessungen, Massen und Geschwindigkeiten enthalten. Diese PDFs sind im Anhang verfügbar. [1]

2.1. Abmessungen

Durchmesser des Karussells [1]:

$$d_{Karussell} = 12m$$

$$r_{Karussell} = \frac{d_{Karussell}}{2} = 6m$$

Breite und Höhe der Gondel [Annahme]:

$$b_{\text{Gondel}} = 1{,}5m$$

$$h_{\text{Gondel}} = 1{,}5m$$

Neigungswinkel:

Laut Angabe ist das Karussell in der x-z-Ebene um 30° geneigt.

$$\varphi_0 = \frac{\pi}{6} = 30°$$

2.2. Massen

Das Gesamtgewicht des Karussells inklusive des Unterbaus beträgt laut Recherche 20.000kg. Um das reine Gewicht der Gondeln zu erhalten, wurden die Annahmen getroffen, dass der Unterbau 10.000kg, die Stange und Scheibe 4.000kg und die Gondeln im Betrieb, also bei voller Auslastung von 40 Personen, gesamt 6.000kg wiegen. [1]

Anzahl der Personen pro Gondel [1]:

$$n_{Personen} = 2$$

Anzahl der Gondeln [1]:

$$n_{Gondel} = 20$$

Masse einer Gondel im Betrieb [Annahme]:

$$m_{Gondel} = n_{Personen} \cdot m_{Personen} + m_{Rest} = 2 \cdot 100kg + 100kg = 300kg$$

Masse aller Gondeln im Betrieb [Annahme]:

$$m_{Gondel\_ges} = m_{Gondel} \cdot n_{Gondel} = 300kg \cdot 20 = 6.000kg$$

2.3. Geschwindigkeiten

Drehzahl des Karussells [1]:

Die Rotationsstruktur dreht sich mit 7U/min gegen den Uhrzeigersinn, wobei sich die Scheibe mit 21U/min im Uhrzeigersinn dreht. Daraus folgt eine gesamte Drehzahl des Karussells von 14U/min. Nur diese gesamte Drehzahl spielt in der nachfolgenden Berechnung eine Rolle. [1]

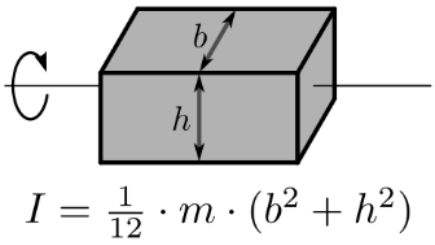
$$n_{Karussell} = 14\frac{U}{min} = 0{,}2333\frac{1}{s}$$

2.4. Trägheitsmoment

Anhand der durch Recherche ermittelten Abmessungen und der angenommenen Werte für die Massen kann nun das Trägheitsmoment der Gondel berechnet werden.

Trägheitsmoment Gondel (inkl. Satz von Steiner):

$$I_{Gondel} = \frac{1}{12} \cdot m_{Gondel} \cdot ({b_{\text{Gondel}}}^2 + {h_{\text{Gondel}}}^2)$$
$$= \frac{1}{12} \cdot 300kg \cdot (1.5m^2 + 1.5m^2)$$
$$= 112{,}5kg \cdot m^2$$



Zu dem berechneten Trägheitsmoment muss jetzt noch der Steiner'sche Anteil addiert werden:

$$I = I_{Gondel} + m_{Gondel} \cdot {r_{Karussell}}^2 = 112{,}5kg \cdot m^2 + 300kg \cdot (6m)^2$$
$$= 10.912{,}5kg \cdot m^2$$

Da laut Angabe alle Gondeln verschmiert als Trägheit angenommen werden müssen, wird das gesamte Trägheitsmoment aller 20 Gondeln berechnet.

$$I_{Gondel\_ges} = \sum I$$
$$= n_{Gondel} \cdot I = 20 \cdot 10.912{,}5kg \cdot m^2 = 218.250kg \cdot m^2$$

3. Skizze

In den folgenden beiden Abbildungen ist zum einen eine grobe Vereinfachung des Karussells und zum anderen eine detailliertere Skizze zu sehen, die für die Modellierung des mathematischen Modells verwendet wird. Die zwei generalisierten Koordinaten, welche zum Beschreiben des mathematischen Systems benötigt werden, sind in der groben Vereinfachung jeweils in grün gehalten.

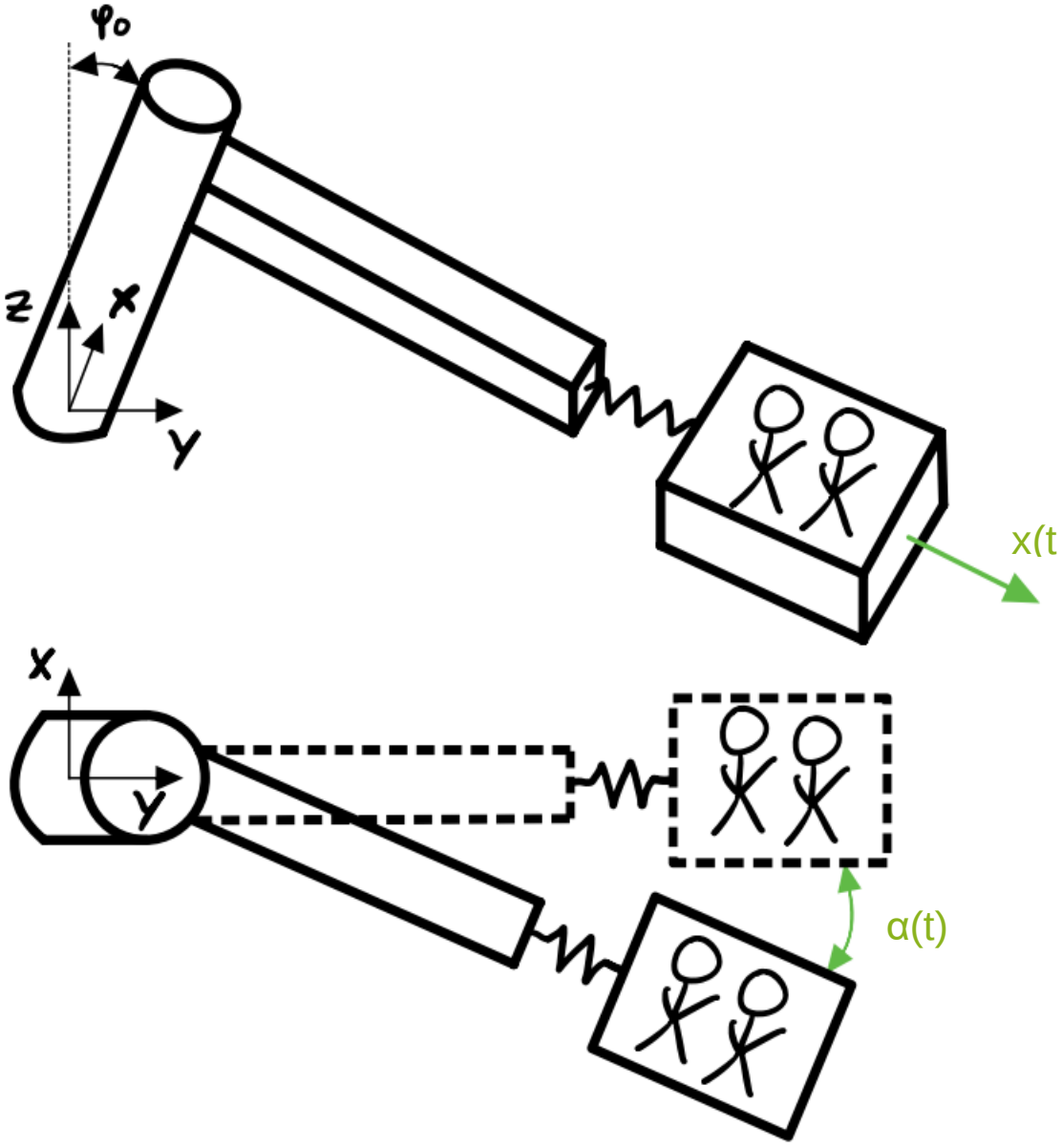


Abbildung 2: grobe Vereinfachung des Karussells

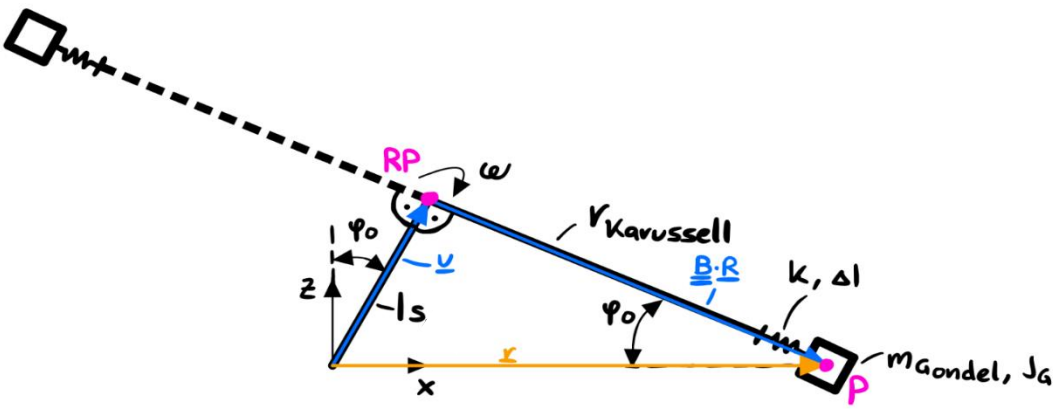


Abbildung 3: detaillierte Skizze für körperfestes Koordinatensystem

Der Ortsvektor, der die Kreisbahn der Gondel beschreibt, wird durch die Einführung eines körperfesten Koordinatensystems beschrieben. Die entsprechende Formel lautet:

$$\underline{r} = \underline{u} + \underline{\underline{B}} \cdot \underline{R}$$

Die einzelnen Parameter dieser Formel lassen sich wie folgt erläutern:

$$\underline{u} = l_{Stange} \cdot \begin{bmatrix} \sin\varphi_0 \\ 0 \\ \cos\varphi_0 \end{bmatrix}$$

$$\underline{\underline{B}} = \underline{\underline{B_y}} \cdot \underline{\underline{B_z}} = \begin{bmatrix} \cos\varphi_0 & 0 & \sin\varphi_0 \\ 0 & 1 & 0 \\ -\sin\varphi_0 & 0 & \cos\varphi_0 \end{bmatrix} \cdot \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha & -\cos\varphi_0 \cdot \sin\alpha & \sin\varphi_0 \\ \sin\alpha & \cos\alpha & 0 \\ -\sin\varphi_0 \cdot \cos\alpha & \sin\varphi_0 \cdot \sin\alpha & \cos\varphi_0 \end{bmatrix}$$

$$\underline{R} = \begin{bmatrix} x(t) \\ 0 \\ 0 \end{bmatrix}$$

Die Berechnung der Drehmatrix $\underline{\underline{B}}$ wurde dabei in folgenden drei Schritten durchgeführt:

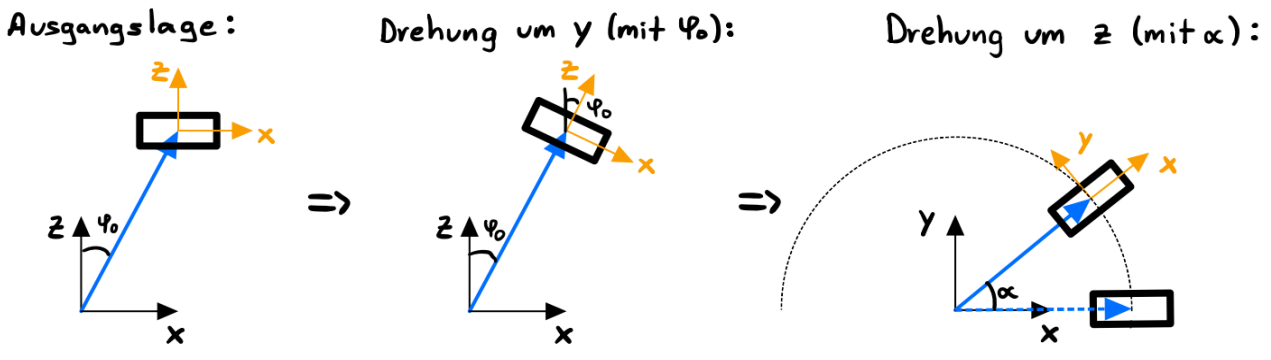


Abbildung 4: Skizze für die Berechnung der Drehmatrix

$$\underline{r} = l_{Stange} \cdot \begin{bmatrix} \sin\varphi_0 \\ 0 \\ \cos\varphi_0 \end{bmatrix} + \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) & -\cos\varphi_0 \cdot \sin\alpha(t) & \sin\varphi_0 \\ \sin\alpha(t) & \cos\alpha(t) & 0 \\ -\sin\varphi_0 \cdot \cos\alpha(t) & \sin\varphi_0 \cdot \sin\alpha(t) & \cos\varphi_0 \end{bmatrix} \cdot \begin{bmatrix} x(t) \\ 0 \\ 0 \end{bmatrix}$$

$$= l_{Stange} \cdot \begin{bmatrix} \sin\varphi_0 \\ 0 \\ \cos\varphi_0 \end{bmatrix} + \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \\ \sin\alpha(t) \cdot x(t) \\ -\sin\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \end{bmatrix}$$

$$= \begin{bmatrix} l_{Stange} \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \\ \sin\alpha(t) \cdot x(t) \\ l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \end{bmatrix}$$

Die Überprüfung mit GeoGebra 3D zeigt, dass der Ortsvektor $\underline{r}$ die Bewegung des Karussells genau widerspiegelt. In der x-y-Ebene ist dabei aufgrund der Höhenzunahme eine Ellipse ersichtlich.

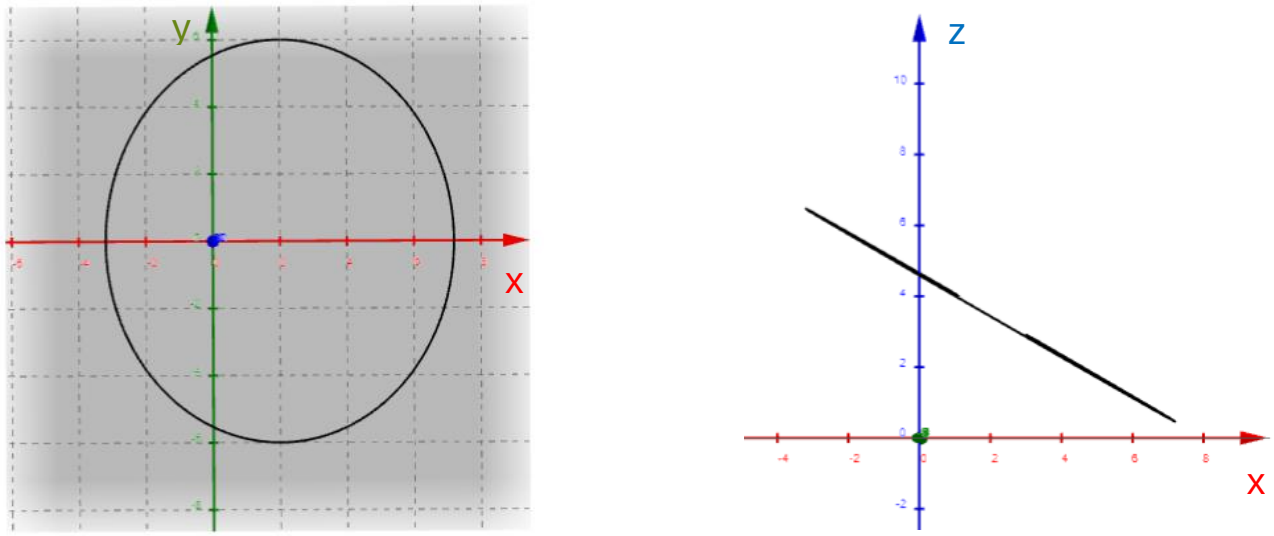


Abbildung 5: Kreisbahn des Ortsvektors in x-y-Ebene und y-z-Ebene nach [2]

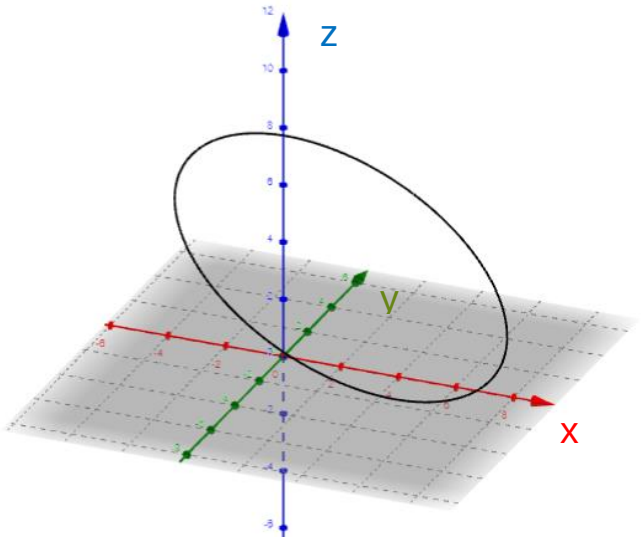


Abbildung 6: Kreisbahn der Ortsvektoren in 3D-Ansicht nach [2]

4. Bewegungsgleichungen

4.1. Allgemein

Um anhand des Ortsvektors die Bewegungsgleichungen aufstellen zu können, muss die Ableitung $\underline{\dot{r}}$, das Produkt $\underline{\dot{r}}^T \cdot \underline{\dot{r}}$, sowie das Produkt $\underline{r}^T \cdot \underline{r}$ berechnet werden. Die Handrechnung und Vereinfachungen können dabei dem Anhang entnommen werden.

$$T = \frac{1}{2} m_{Gondel} \cdot \underline{\dot{r}}^T \cdot \underline{\dot{r}} + 20 \cdot \frac{1}{2} I_{Gondel} \cdot \dot{\alpha}(t)^2$$

$$\underline{r} = \begin{bmatrix} l_{Stange} \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \\ \sin\alpha(t) \cdot x(t) \\ l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha\,(t) \cdot x(t) \end{bmatrix}$$

$$\underline{\dot{r}} = \begin{bmatrix} -\cos\varphi_0 \cdot \sin\alpha(t) \cdot x(t) \\ \cos\alpha(t) \cdot x(t) \\ \sin\varphi_0 \cdot \sin\alpha\,(t) \cdot x(t) \end{bmatrix} \cdot \dot{\alpha}(t) + \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) \\ \sin\alpha(t) \\ -\sin\varphi_0 \cdot \cos\alpha\,(t) \end{bmatrix} \cdot \dot{x}(t)$$

$$\underline{\dot{r}}^T \cdot \underline{\dot{r}} = x(t)^2 \cdot \dot{\alpha}(t)^2 + \dot{x}(t)^2$$

Nun können einerseits die kinetische Energie T und andererseits die potentielle Energie V aufgestellt werden.

$$V = \frac{1}{2} \cdot c \cdot \left(\sqrt{\underline{r_F}^T \cdot \underline{r_F}} - \sqrt{\underline{r_0}^T \cdot \underline{r_0}} \right)^2 + m_{Gondel} \cdot g \cdot \underline{r_z}$$

$$\underline{r_1}(t) = \begin{bmatrix} l_{Stange} \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot r_{Karussell} \\ \sin\alpha(t) \cdot r_{Karussell} \\ l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha\,(t) \cdot r_{Karussell} \end{bmatrix}$$

$$\underline{r_2}(t) = \begin{bmatrix} l_{Stange} \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot (x(t) + r_{Karussell}) \\ \sin\alpha(t) \cdot (x(t) + r_{Karussell}) \\ l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha\,(t) \cdot (x(t) + r_{Karussell}) \end{bmatrix}$$

$$\underline{r_F}(t) = \underline{r_2}(t) - \underline{r_1}(t) = \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \\ \sin\alpha(t) \cdot x(t) \\ -\sin\varphi_0 \cdot \cos\alpha\,(t) \cdot x(t) \end{bmatrix}$$

$$\sqrt{\underline{r_F}^T \cdot \underline{r_F}} = \sqrt{x(t)^2} = x(t)$$

$$\underline{r_1}(0) = \begin{bmatrix} l_{Stange} \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot r_{Karussell} \\ \sin\alpha(t) \cdot r_{Karussell} \\ l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha\,(t) \cdot r_{Karussell} \end{bmatrix}$$

$$\underline{r_2}(0) = \begin{bmatrix} l_{Stange} \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot (x(0) + r_{Karussell}) \\ \sin\alpha(t) \cdot (x(0) + r_{Karussell}) \\ l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha\,(t) \cdot (x(0) + r_{Karussell}) \end{bmatrix}$$

$$x(0) = r_{Karussell}$$

$$\Rightarrow \underline{r_2}(0) = \begin{bmatrix} l_{Stange} \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot 2 \cdot r_{Karussell} \\ \sin\alpha(t) \cdot 2 \cdot r_{Karussell} \\ l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha\,(t) \cdot 2 \cdot r_{Karussell} \end{bmatrix}$$

$$\underline{r_0}(t) = \underline{r_2}(0) - \underline{r_1}(0) = \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) \cdot r_{Karussell} \\ \sin\alpha(t) \cdot r_{Karussell} \\ -\sin\varphi_0 \cdot \cos\alpha\,(t) \cdot r_{Karussell} \end{bmatrix}$$

$$\sqrt{\underline{r_0}^T \cdot \underline{r_0}} = \sqrt{r_{Karussell}^2} = r_{Karussell}$$

$$\left(\sqrt{\underline{r_F}^T \cdot \underline{r_F}} - \sqrt{\underline{r_0}^T \cdot \underline{r_0}}\right)^2 = x(t)^2 + r_{Karussell}^2 - 2 \cdot x(t) \cdot r_{Karussell}$$

$$\underline{r_z} = l_{Stange} \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha\,(t) \cdot x(t)$$

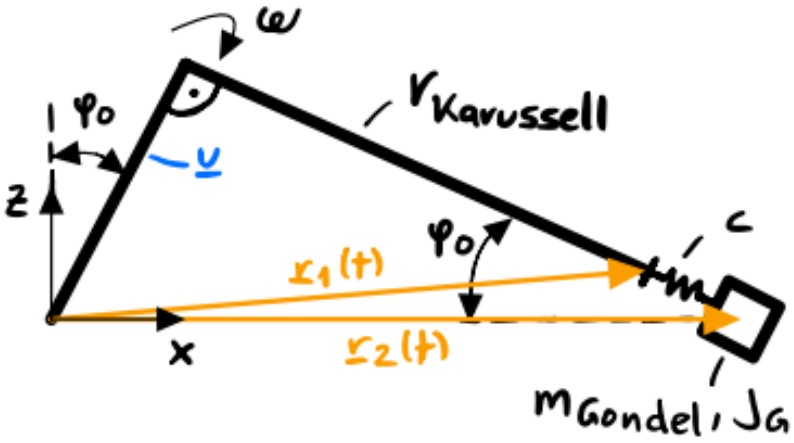


Abbildung 7: Skizze für die Berechnung des Federwegs

Da keine nicht-konservativen Kräfte beziehungsweise äußeren Kräfte vorhanden sind ($Q_i = 0$), kann mit T und V nun direkt die Lagrange-Funktion und in weiterer Folge die Euler-Lagrange-Gleichung für das gegebene System aufgestellt werden.

Die Lagrange-Funktion wird mit der nachfolgenden Formel berechnet und ist die kinetische Energie T minus der potentiellen Energie V.

$$L = T - V$$

$$L = \frac{1}{2} m_{Gondel} \cdot x(t)^2 \cdot \dot{\alpha}(t)^2 + \frac{1}{2} m_{Gondel} \cdot \dot{x}(t)^2 + 10 \cdot I_{Gondel} \cdot \dot{\alpha}(t)^2 - \frac{1}{2} \cdot c \cdot x(t)^2 - \dots$$

$$\dots - \frac{1}{2} \cdot c \cdot {r_{Karussell}}^2 + c \cdot x(t) \cdot r_{Karussell} - m_{Gondel} \cdot g \cdot l_{Stange} \cdot \cos \varphi_0 + \dots$$

$$\dots + m_{Gondel} \cdot g \cdot \sin \varphi_0 \cdot \cos \alpha\,(t) \cdot x(t)$$

In weiterer Folge kann nun mit der Euler-Lagrange-Gleichung und der vorhandenen Lagrange-Funktion, die Bewegungsgleichungen der beiden generalisierten Koordinaten $x(t)$ und $\propto (t)$ berechnet werden.

- **Bewegungsgleichung 1**

$$\frac{d}{dt}\left(\frac{\partial \mathrm{L}}{\partial\, \dot{x}(t)}\right) - \frac{\partial \mathrm{L}}{\partial\, x(t)} = 0$$

Aus dieser Formel folgt nun die finale Bewegungsgleichung für $x(t)$:

$$\boldsymbol{\ddot{x}(t) = x(t) \cdot \dot{\alpha}(t)^2 + g \cdot \sin \varphi_0 \cdot \cos \alpha(t) + \frac{c}{m_{Gondel}} \cdot \left(r_{Karussell} - x(t)\right)}$$

- **Bewegungsgleichung 2**

$$\frac{d}{dt}\left(\frac{\partial \mathrm{L}}{\partial\, \dot{\alpha}(t)}\right) - \frac{\partial \mathrm{L}}{\partial\, \alpha(t)} = 0$$

Aus dieser Formel folgt nun die finale Bewegungsgleichung für $\alpha(t)$:

$$\boldsymbol{\ddot{\alpha}(t) = -\frac{2 \cdot \dot{\alpha}(t) \cdot \dot{x}(t) \cdot x(t) + g \cdot \sin \varphi_0 \cdot \sin \alpha(t) \cdot x(t)}{x(t)^2 + \frac{5}{3} \cdot \left({b_{\mathrm{Gondel}}}^2 + {h_{\mathrm{Gondel}}}^2\right) + 20 \cdot {r_{Karussell}}^2}}$$

Die vollständigen manuellen Berechnungen für beide Bewegungsgleichungen können dem Anhang entnommen werden. Für die Zusatzaufgabe beziehungsweise die Simulation müssen Anfangsbedingungen definiert werden. Dabei wurden folgende Werte gewählt:

$$\propto (0) = 0\,;\ \dot{\propto}\,(0) = 2\pi \cdot n_{Karussell}$$

$$x(0) = r_{Karussell}\,;\ \dot{x}(0) = 0$$

4.2. Zusatzaufgabe

„Modellieren sie das System mit Verschlussmechanismus so, dass bei maximaler Geschwindigkeit ein Versagen vermieden wird. Das ist der Fall, wenn sich der Einrastbolzen nicht weiter als 5 mm radial bewegen kann.“ Diese Zusatzaufgabe wird mithilfe des in Kapitel 5 beschriebenen Python-Codes (Runge-Kutta-Verfahren) durchgeführt. Dabei wird die Federsteifigkeit schrittweise erhöht, bis jener Wert erreicht ist, bei dem sich der Einrastbolzen maximal 5 mm radial bewegen kann.

Bei einer Federsteifigkeit von $c = 300.000 \frac{N}{m}$ beträgt der maximale radiale Weg 5cm. Es muss die Federsteifigkeit weiter erhöht werden, bis der radiale Weg nur mehr 5mm beträgt. Dieser Wert der Federsteifigkeit ist: $\boldsymbol{c = 3.000.000 \frac{N}{m}}$

Ergebnisse:

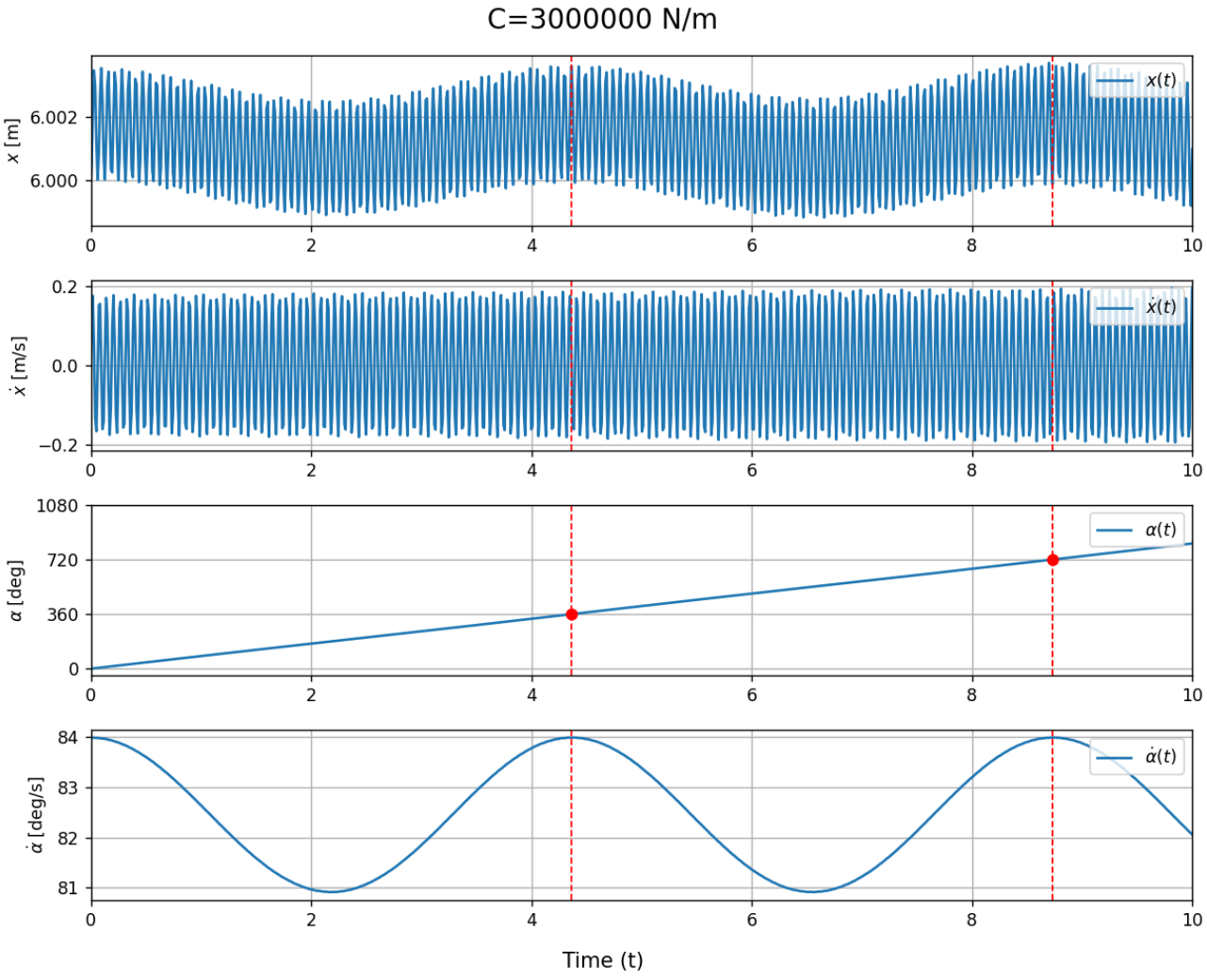


Abbildung 8: Ergebnisse der Zusatzaufgabe (Simulation-Time = 10s)

Um die Schwingung der Feder genauer untersuchen zu können, wird die Simulation-Time auf 5s reduziert.

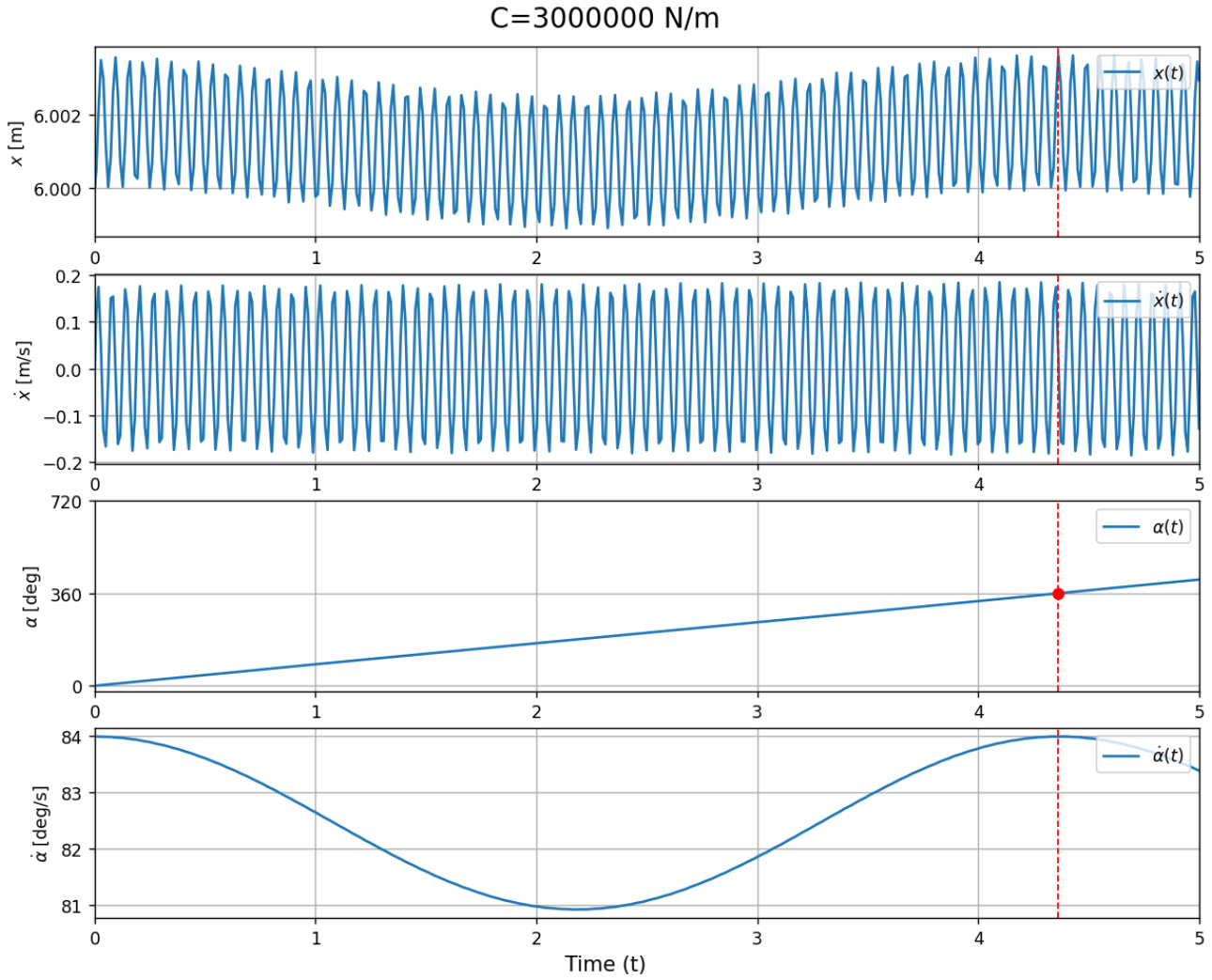


Abbildung 9: Ergebnisse der Zusatzaufgabe (Simulation-Time = 5s)

Werden nun beide Federn verglichen, zeigt sich, dass die Feder für den Karussellsitz einerseits einen deutlich geringeren Weg zurücklegt und andererseits deutlich steifer ist als eine typische Autofeder. Das liegt daran, dass Autofedern für dynamische Fahrsituationen und Komfort ausgelegt sind, während die Feder für den Karussellsitz extrem steif sein muss, um die hohen Belastungen mit minimalem Federweg zu bewältigen.

Federtyp	Federweg x [mm]	Federsteifigkeit c [N/m]
Autofeder	100 – 200	10.000 – 50.000
Feder Karussell	5	3.000.000

Tabelle 1: Feder des Karussells im Vergleich zu einer Autofeder nach [3]

5. Simulation

Folgendes Kapitel beschreibt die Erstellung zweier Python-Programme, einerseits mit dem explizitem 1-Schritt-Eulerverfahren und andererseits mit dem Runge-Kutta-Verfahren (ODE45-Verfahren), auf Basis der aufgestellten Bewegungsgleichungen. Beide Codes können dem Anhang entnommen werden.

Explizites 1-Schritt-Eulerverfahren:

Beim nachfolgendem Teil des Codes handelt es sich um das explizite 1-Schritt-Eulerverfahren, welches ein explizites numerisches Zeitintegrationsverfahren darstellt. Damit der Computer die Rechenschritte ausführen kann, ist es notwendig, dass die Bewegungsgleichungen auf ein Gleichungssystem 1. Ordnung gebracht werden.

```
def run_simulation():
    for i in range(0, number_of_steps-1):
        x[i + 1] = x_dot[i] * STEP_TIME + x[i]
        x_dot[i + 1] = (x[i] * (alpha_dot[i]**2) + G * np.sin(PHI_0_RAD) *
                        np.cos(alpha[i]) + ((C / M_GONDEL) *
                            (R_KARUSELL - x[i]))) * STEP_TIME + x_dot[i]

        alpha[i + 1] = alpha_dot[i] * STEP_TIME + alpha[i]
        alpha_dot[i + 1] = (-(2 * alpha_dot[i] * x_dot[i] * x[i] + G *
                              np.sin(PHI_0_RAD) * np.sin(alpha[i]) * x[i]) /
                                (x[i]**2 + (5/3) * (B_GONDEL**2 + H_GONDEL**2) +
                                    20 * R_KARUSELL**2)) * STEP_TIME + alpha_dot[i]

        time_array[i + 1] = (i + 1) * STEP_TIME
```

Ergebnisse:

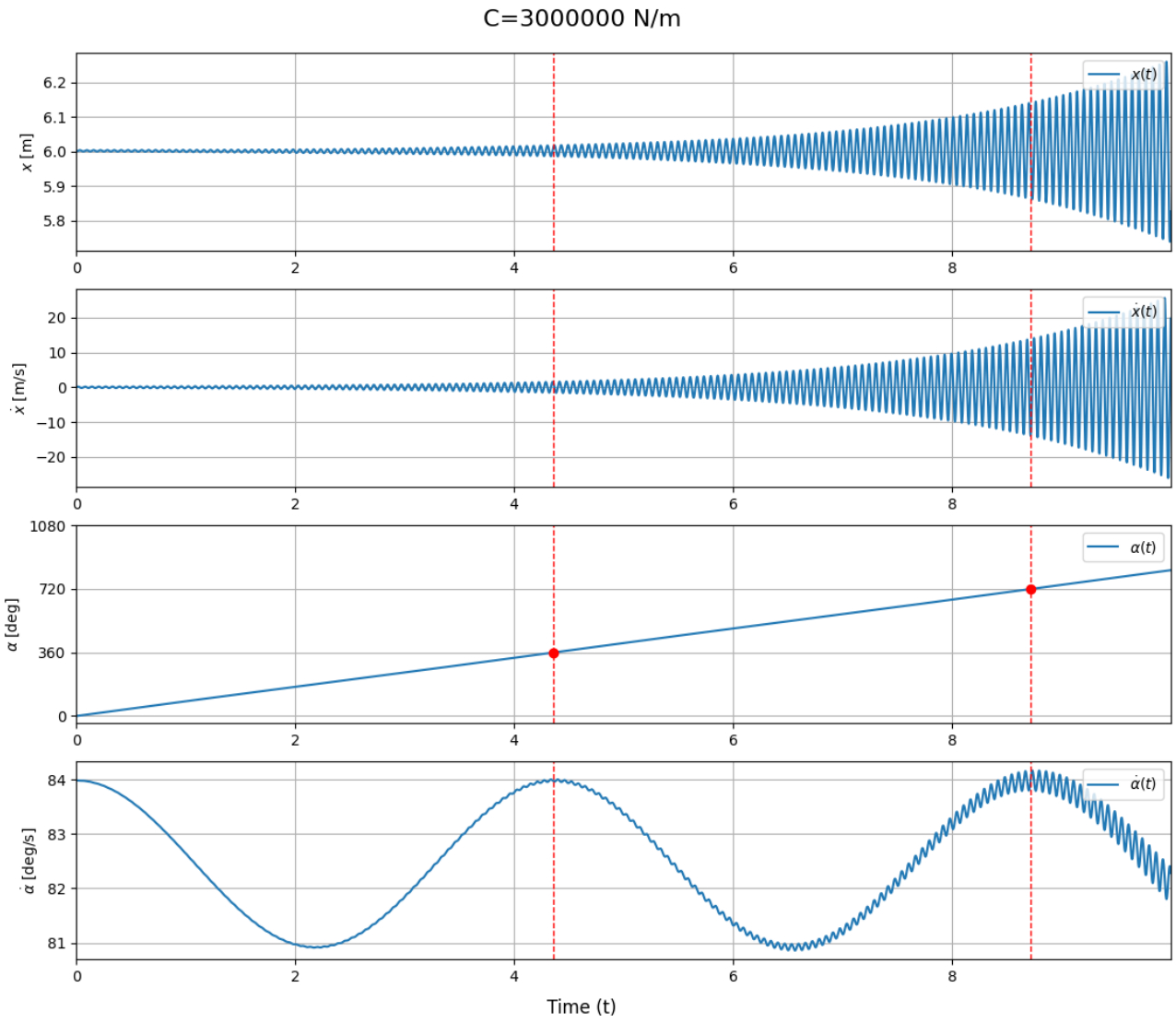


Abbildung 10: Drift von expliziten 1-Schritt-Eulerverfahren bei Step-Time =0.0001 (Simulation-Time = 10s)

Wie bereits aus der Vorlesung bekannt, sind explizite Verfahren anfällig für einen hohen Drift, welcher nichts anderes als eine mit der Simulationsdauer steigende Abweichung zur analytischen Lösung ist. Um dem Drift entgegenzuwirken, ist es deshalb erforderlich eine geringe Schrittweite zu implementieren. Dies hat zur Folge, dass bei gleichbleibender Simulationsdauer die Anzahl an Schritten und letztlich auch die Rechendauer erheblich zunimmt.

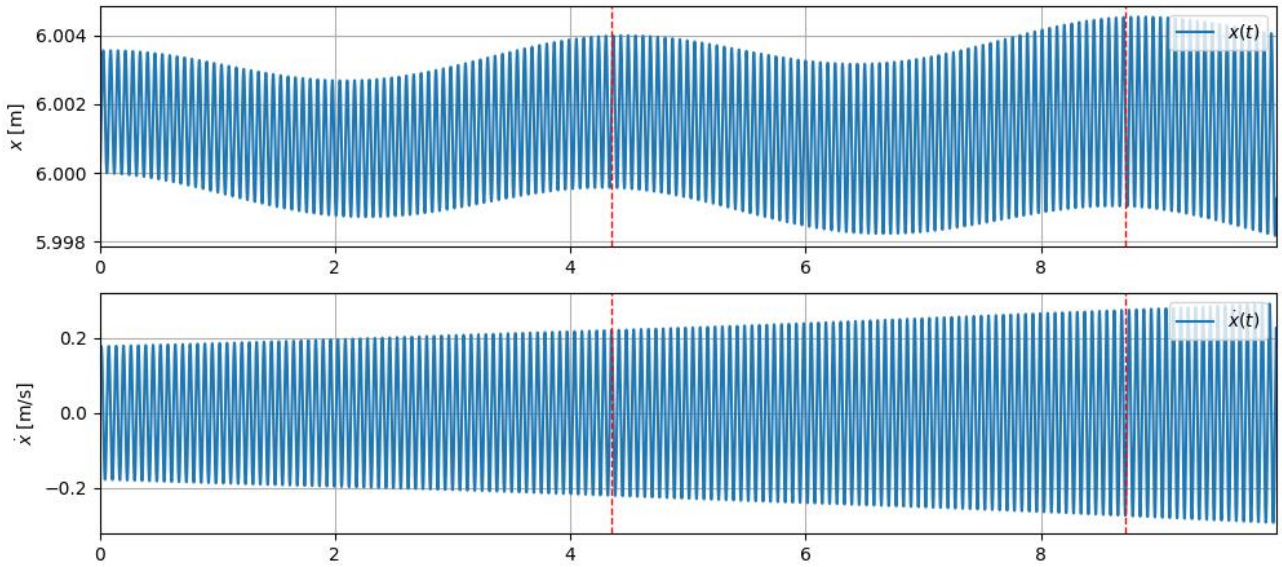


Abbildung 11: Drift von expliziten 1-Schritt-Eulerverfahren bei Step-Time = 0.00001 (Simulation-Time = 10s)

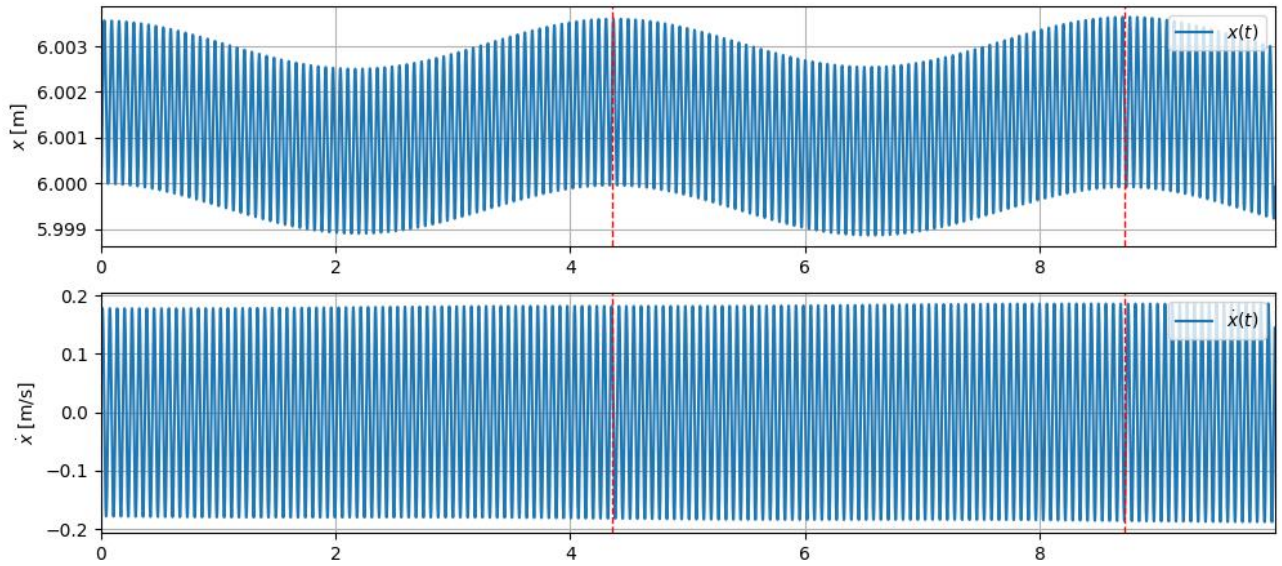


Abbildung 12: Drift von expliziten 1-Schritt-Eulerverfahren bei Step-Time = 0.000001 (Simulation-Time = 10s)

	Setup #1	Setup #2	Setup #3
SIMULATION_TIME [s]	10	10	10
STEP_TIME [s]	0.0001	0.00001	0.000001
Amount of Steps	100 000	1 000 000	10 000 000
COMPUTATION_TIME	2.846	29.566	292.707

Tabelle 2: Explizites 1-Schritt-Euler-Verfahren - Unterschiede in den Einstellungen

Die Ergebnisse bestätigen, dass die Vermeidung des Drifts eine erhebliche Verlängerung der Rechendauer mit sich zieht. Es ist anzumerken, dass bei dem derzeitigen expliziten 1-Schritt-Eulerverfahren die Schrittweite experimentell angepasst wurde, bis das Ergebnis zufriedenstellend war. Diese manuelle Anpassung der Schrittweite soll nun durch die Nutzung des ODE45-Solver optimiert werden.

ODE45-Verfahren (bzw. RK45 Runge-Kutta):

Bei nachfolgendem Code handelt es sich um das ODE45-Verfahren (Ordinary Differential Equation), welches ebenfalls ein explizites numerisches Zeitintegrationsverfahren ist. Der Solver passt hier aber die Schrittweite automatisch an, um eine vorgegebene Toleranz zu erreichen. Die Differenz zwischen der Methode 4. Ordnung und 5. Ordnung gibt eine Abschätzung des Fehlers und ermöglicht es dem Algorithmus, die Schrittweite für die numerische Integration dynamisch anzupassen. Wenn der Fehler klein ist, kann die Schrittweite erhöht werden, um die Rechenzeit zu verringern. Ist der Fehler zu groß, wird die Schrittweite verkleinert, um die Genauigkeit zu erhöhen.

Deshalb ist der ODE45-Solver aufgrund seiner mittleren Genauigkeit, meistens der geeignetste Solver für den ersten Versuch, insofern es sich um eine nicht steife DGL handelt.

Solver	Anwendungsfeld	Genauigkeit	Nutzungshinweis
ode45	nicht steife DGL	mittel	meistens der geeignetste Solver für 1. Versuch
ode23	nicht steife DGL	niedrig	effizient bei geringen Genauigkeitsanforderungen
ode113	nicht steife DGL	niedrig bis hoch	effizient bei hohen Genauigkeitsanforderungen
ode15s	steife DGL	niedrig bis mittel	bei Verdacht auf steife Systeme
ode23s	steife DGL	niedrig	effizient bei geringen Genauigkeitsanforderungen
ode15i	implizite DGL	niedrig	einziger Solver für rein implizite DGL

Abbildung 13: Auswahl des Solvers nach [4]

Für steife Systeme sollten optimierte implizite Lösungsverfahren wie z.B. der modifizierte Rosenbrock-Algorithmus (ode23s) genutzt werden. Der Hauptunterschied besteht darin, dass ode23s in jedem Zeitschritt ein Gleichungssystem lösen muss. Zwar ist der Rechenaufwand pro Zeitschritt höher, jedoch ermöglicht der größere Stabilitätsbereich des Verfahrens die Nutzung größerer Zeitschritte. Das kann die Gesamtzeit für das Lösen der Gleichung sogar verringern.

Ob der passende Solver gewählt wurde, lässt sich auch anhand des Simulationsergebnisses interpretieren:

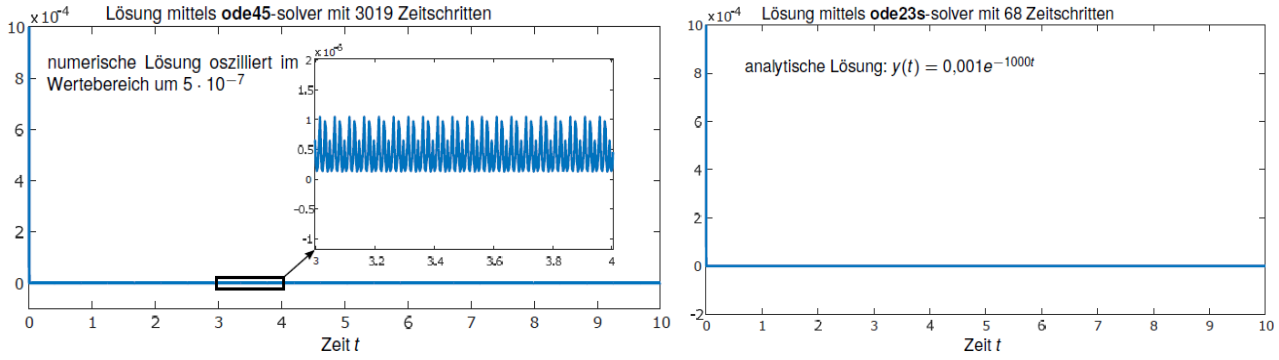


Abbildung 14: Vergleich von ODE45 (explizit) und ODE23S (implizit) bei steifer DGL nach [4]

Rauscht bzw. oszilliert das Signal und hat der Lösungsvektor demnach sehr viele Einträge, so handelt es sich vermutlich um eine steife DGL, weshalb der Wechsel von einem expliziten zu einem impliziten Lösungsverfahren empfehlenswert ist.

Im Folgenden kann ein Ausschnitt des Codes, welcher die Bewegungsgleichungen des ODE45-Verfahren umfasst, gesehen werden:

```
def system_dynamics(t, y):
    x, x_dot, alpha, alpha_dot = y
    dxdt = x_dot
    dx_dotdt = (x * (alpha_dot**2) + G * np.sin(PHI_0_RAD) *
                np.cos(alpha) + ((C / M_GONDEL) * (R_KARUSELL - x)))
    dalphadt = alpha_dot
    dalpha_dotdt = (-(2 * alpha_dot * x_dot * x + G * np.sin(PHI_0_RAD) *
                      np.sin(alpha) * x) / (x**2 + (5/3) *
                        (B_GONDEL**2 + H_GONDEL**2) + 20 * R_KARUSELL**2))
    return [dxdt, dx_dotdt, dalphadt, dalpha_dotdt]
```

Die Ergebnisse können dem Kapitel 4.2 entnommen werden.

Die Simulation kann entweder durch Kopieren des Codes aus dem Anhang oder durch das Klonen des Github Repositories https://github.com/phgas/fhtw-mmb2-mks-2024 erfolgen. Hinweis: Das Repository wird erst nach Abgabe sämtlicher Gruppen von Privat auf Öffentlich geschalten.

Eine kleine Anmerkung gilt es hier jedoch noch zu machen: Während der ODE45 mit automatischer Schrittweitenanpassung auf Basis der Standardwerte (Absolut- und Relativtoleranz) lediglich 0.396 Sekunden für ein Simulationsdauer von 10s gebraucht hat, benötigte ein ähnlich zufriedenstellendes Ergebnis des 1-Schritt-Eulerverfahrens 292.07 Sekunden. Dies hat natürlich auch mit der Optimierung der beiden Algorithmen zu tun, doch dies würde den Sinn dieser Arbeit maßgeblich überschreiten, weshalb hier nicht weiter darauf eingegangen wird.

6. Lessons Learned

Zu Beginn des Projektes, als die Einführung eines körperfesten Koordinatensystems in der Vorlesung noch nicht besprochen wurde, wurde der Ortsvektor in Abhängigkeit vieler Variablen aufgestellt und mittels GeoGebra überprüft. Dadurch entstehen beim Ableiten und späteren Quadrieren sehr lange Terme, welche händisch kaum lösbar sind. Die Einführung eines körperfesten Koordinatensystems in der Gondel vereinfacht dieses Problem erheblich. Durch die geeignete Drehung der Gondel mittels Aufstellung der Drehmatrix kann die Berechnung der Bewegung wesentlich vereinfacht werden.

Ein körperfestes Koordinatensystem ermöglicht es, die Bewegung der Gondel aus einer fest mit ihr verbundenen Perspektive zu betrachten. Dies reduziert die Komplexität der mathematischen Beschreibung und erlaubt eine präzisere Modellierung des Systems. Insbesondere vereinfacht es die Berechnung von Ableitungen, die für das Aufstellen der Bewegungsgleichungen erforderlich sind. Die Nutzung der Drehmatrix $\underline{\underline{B}}$ erleichtert es, die Transformation zwischen dem globalen Koordinatensystem und dem körperfesten Koordinatensystem der Gondel zu beschreiben. Dadurch können die Bewegungen und Rotationen der Gondel relativ zu ihrer eigenen Achse besser verstanden und modelliert werden.

7. Abbildungsverzeichnis

8. Tabellenverzeichnis

9. Literaturverzeichnis

[1] Emiliana Luna Park, "Dancing Fly",
Available: https://www.emilianalunapark.com/ride/dancing-fly/.

[2] GeoGebra, "Curve in 3D", Available: https://www.geogebra.org/m/qVheAs2r.

[3] Suspension Secrets, Available:
https://suspensionsecrets.co.uk/calculating-ideal-spring-and-roll-bar-rates/.

[4] D. Anders, LV "Berechnungsmethoden für dynamische Systeme"
WS2023/24, FH Technikum Wien.

10. Anhang

10.1. Datenblätter

EMILIANA LUNA PARK S.r.l.
VIA PO, 12 PHONE:059-784017
41057 SPILAMBERTO (Modena) ITALY
www.emilianalunapark.com

DANCING FLY
park model
no scenery

THE RIDE HAS FOUR MOVEMENTS:
Rotation on its axis
Revolution on an eccentric axis
Lifting (tilting) up to 44 degrees
Swinging movement of the cars

MEASURES DURING WORKING:
Depth 18,50 meters approx.
Width 17,00 meters approx.
Height (cars) 5,50 meters approx.

In rotating operation: 12,00 meters approx.
Diameter of the rotary structure: 8,00 meters approx.
Maximum eccentric: 3,52 meters approx.
Maximum operative dimension: 12,00 meters approx.
Over the entire diameter described by the rotary movement plus revolution, the height of cars at the highest point is 5,50 meters approx.

SPEED:
Rides per minute: clockwise (rotation) : 21 rides
counterclockwise (revolution): 7 rides
THE RESULT IS: 14 R.P.M.
Circuit speed: 388 m/min
Lifting speed: 4 m/min (cylinder stroke: 1056 mm)

THE STRUCTURE:
it is made up by structural steel shapes, joined by electric welding and painted. the upper section of the rotary part is covered by coloured plastic panels. the lower section of the rotary part is covered with white plastic panels.

THE BASE:
the central piece of the equipment must be fixed on a reinforced-concrete base. The cement structure must be carried-out as per drawings supplied by us, but manufactured at purchaser's own care and expenses.

THE CARS are 20 and are made of fiberglass with incorporated colours, with stainless steel safety bar and safety belts.

The electrical equipment, lighting system and motive power are all connected to an electric panel installed in a suitable operator booth.

LIGHTING:

560 lighting points for the upper rotary part
40 fluorescent lamps 1,2 mt for the lower rotary part
16 iodine spotlights 500 w
40 spotlights, aircraft type, of 100 watt each
80 cars'lamps of 10 watt each

MOTIVE POWER:
2 x 7,5 hp motors operating the rotary movement.
1 x 7,5 hp motor operating the revolution movement.
1 x 10,0 hp motor operating the pump.

Lighting power installed:		29,00 kw
Lighting power absorbed:		19,00 kw
Motive power installed:	35,0 hp	25,70 kw
Motive power absorbed:	25,0 hp	18,00 kw

All motors and the electronic central unit are protected by automatic magneto-thermal systems.
Motors, wires, instruments, and the entire electrical equipment are according to the international regulations.

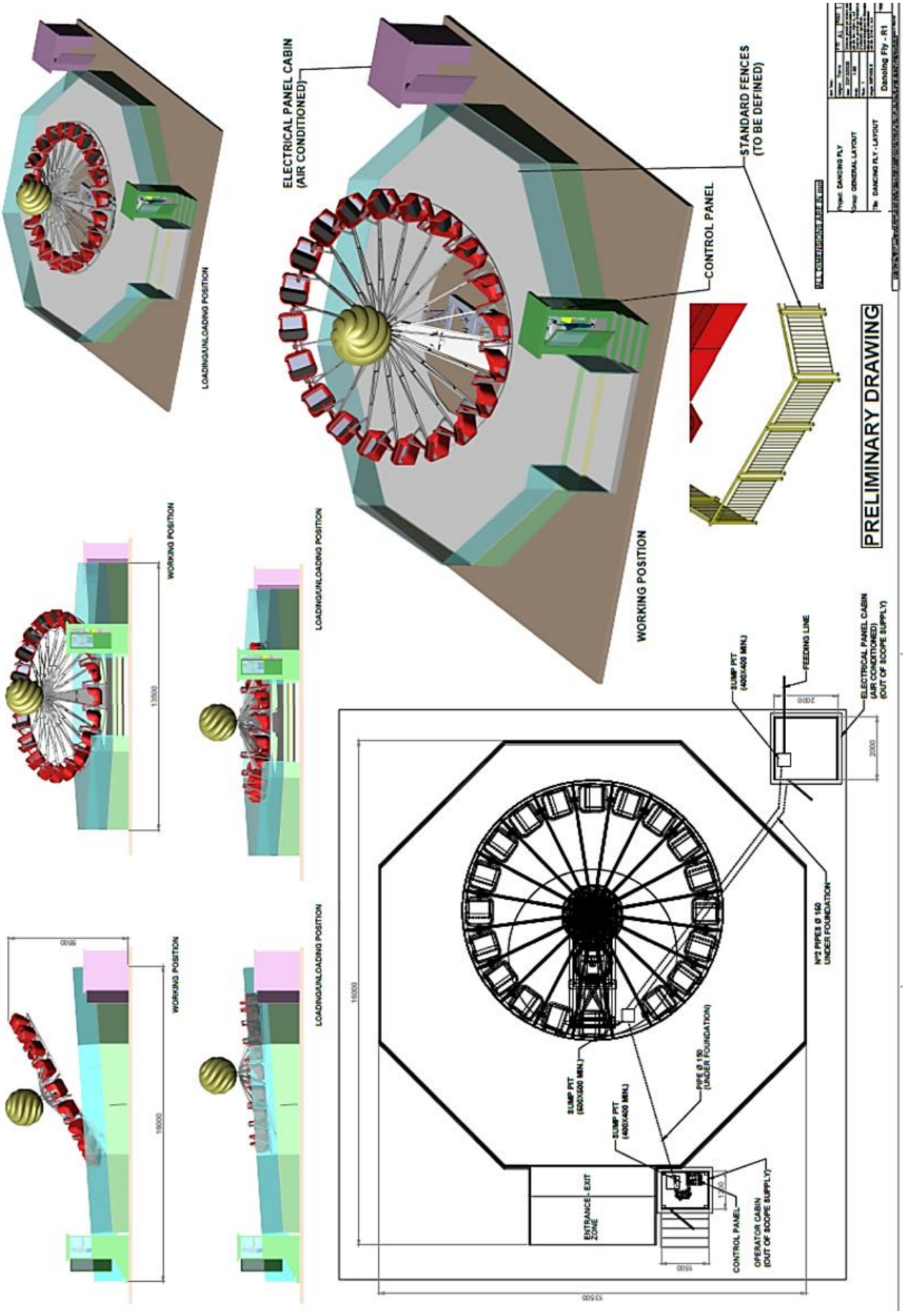
SEATS: 40 for adults or children.

INFORMATION FOR SHIPMENT

no. 2 containers open top 40 feet

TOTAL WEIGHT: Kg 20.000 apx.

AGE GROUP	SEATS	HOUR CAPACITY	RPM	MOTIVE POWER	WEIGHT	OPERATIONAL AREA
family	40	1000	14	35 KW	20.000 KG	Ø m 17 x x m 5,50 h


ELECTRICAL PANEL CABIN
(AIR CONDITIONED)
STANDARD FENCES
(TO BE DEFINED)
CONTROL PANEL
WORKING POSITION
LOADING/UNLOADING POSITION
PRELIMINARY DRAWING
Danoing Fly - R1
SUMP PIT
(400X400 MIN.)
FEEDING LINE
ELECTRICAL PANEL CABIN
(AIR CONDITIONED)
(OUT OF SCOPE SUPPLY)
N°2 PIPES Ø 160
UNDER FOUNDATION
SUMP PIT
(600X600 MIN.)
PIPE Ø 160
(UNDER FOUNDATION)
EXIT
ENTRANCE
ZONE
CONTROL PANEL
OPERATOR CABIN
(OUT OF SCOPE SUPPLY)

10.2. Handrechnung der Bewegungsgleichungen

$$T = \frac{1}{2} m_{Gondel} \cdot \underline{\dot{r}}^T \cdot \underline{\dot{r}} + 20 \cdot \frac{1}{2} I_{Gondel} \cdot \dot{\alpha}^2$$

$$\underline{r} = \begin{bmatrix} ls \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \\ \sin\alpha(t) \cdot x(t) \\ ls \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \end{bmatrix}$$

$$\dot{\underline{r}} = \begin{bmatrix} -\cos\varphi_0 \cdot \sin\alpha(t) \cdot x(t) \\ \cos\alpha(t) \cdot x(t) \\ \sin\varphi_0 \cdot \sin\alpha(t) \cdot x(t) \end{bmatrix} \cdot \dot{\alpha}(t) + \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) \\ \sin\alpha(t) \\ -\sin\varphi_0 \cdot \cos\alpha(t) \end{bmatrix} \cdot \dot{x}(t)$$

$$\begin{aligned} \dot{\underline{r}}^T \cdot \dot{\underline{r}} = {} & \Big[\cos^2\varphi_0 \cdot \sin^2\alpha(t) \cdot x^2(t) \cdot \dot{\alpha}^2(t) + \cos^2\varphi_0 \cdot \cos^2\alpha(t) \cdot \dot{x}^2(t) + \\ & - 2 \cdot \cos^2\varphi_0 \cdot \cos\alpha(t) \cdot \sin\alpha(t) \cdot x(t) \cdot \dot{x}(t) \cdot \dot{\alpha}(t) \Big] + \\ & + \Big[\cos^2\alpha(t) \cdot x^2(t) \cdot \dot{\alpha}^2(t) + \sin^2\alpha(t) \cdot \dot{x}^2(t) \\ & + 2 \cdot \cos\alpha(t) \cdot \sin\alpha(t) \cdot x(t) \cdot \dot{x}(t) \cdot \dot{\alpha}(t) \Big] + \Big[\sin^2\varphi_0 \cdot \sin^2\alpha(t) \cdot x^2(t) \cdot \dot{\alpha}^2(t) + \\ & + \sin^2\varphi_0 \cdot \cos^2\alpha(t) \cdot \dot{x}^2(t) - 2 \sin^2\varphi \cdot \cos\alpha(t) \cdot \sin\alpha(t) \cdot x(t) \cdot \dot{x}(t) \cdot \dot{\alpha}(t) \Big] \\ = {} & x^2(t) \cdot \dot{\alpha}^2(t) \Big[\overbrace{\cos^2\varphi_0 \cdot \sin^2\alpha(t) + \sin^2\varphi_0 \cdot \sin^2\alpha(t)}^{= \sin^2\alpha(t)} + \cos^2\alpha(t) \Big] + \\ & + \dot{x}^2(t) \Big[\overbrace{\cos^2\varphi_0 \cdot \cos^2\alpha(t) + \sin^2\varphi_0 \cdot \cos^2\alpha(t)}^{\cos^2\alpha(t)} + \overbrace{\sin^2\alpha(t)}^{=1} \Big] + \\ & + x(t) \cdot \dot{x}(t) \cdot \dot{\alpha}(t) \Big[\overbrace{-2\cos^2\varphi_0 \cdot \cos\alpha(t) \cdot \sin\alpha(t) - 2\sin^2\varphi_0 \cdot \cos\alpha(t) \cdot \sin\alpha(t)}^{-2\cos\alpha(t)\cdot\sin\alpha(t)} + \\ & + 2\cos\alpha(t) \cdot \sin\alpha(t) \Big] \\ = {} & x^2(t) \cdot \dot{\alpha}^2(t) + \dot{x}^2(t) \end{aligned}$$

$$V = \frac{1}{2} c \cdot \left(\sqrt{\underline{r}_F^T \cdot \underline{r}_F} - \sqrt{\underline{r}_0^T \cdot \underline{r}_0} \right)^2 + m_{Gondel} \cdot g \cdot r_z$$

$$r_z = l_s \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot x(t)$$

$$\underline{r}_1(t) = \begin{bmatrix} l_s \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot r_K \\ \sin\alpha(t) \cdot r_K \\ l_s \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot r_K \end{bmatrix}$$

$$\underline{r}_2(t) = \begin{bmatrix} l_s \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot (x(t) + r_K) \\ \sin\alpha(t) \cdot (x(t) + r_K) \\ l_s \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot (x(t) + r_K) \end{bmatrix}$$

$$\underline{r}_F(t) = r_2(t) - r_1(t) = \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \\ \sin\alpha(t) \cdot x(t) \\ -\sin\varphi_0 \cdot \cos\alpha(t) \cdot x(t) \end{bmatrix} ; \sqrt{\underline{r}_F^T \cdot \underline{r}_F} = \sqrt{x^2(t)} = x(t)$$

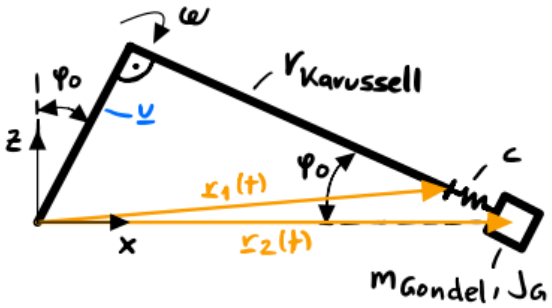
$$\underline{r}_1(0) = \begin{bmatrix} l_s \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot r_K \\ \sin\alpha(t) \cdot r_K \\ l_s \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot r_K \end{bmatrix}$$

$$\underline{r}_2(0) = \begin{bmatrix} l_s \cdot \sin\varphi_0 + \cos\varphi_0 \cdot \cos\alpha(t) \cdot 2 \cdot r_K \\ \sin\alpha(t) \cdot 2 \cdot r_K \\ l_s \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot 2 \cdot r_K \end{bmatrix}$$

$x(0) + r_K$, wobei $x(0) = r_K$ → $2 \cdot r_K$

$$\underline{r}_0(t) = r_2(0) - r_1(0) = \begin{bmatrix} \cos\varphi_0 \cdot \cos\alpha(t) \cdot r_K \\ \sin\alpha(t) \cdot r_K \\ -\sin\varphi_0 \cdot \cos\alpha(t) \cdot r_K \end{bmatrix} ; \sqrt{\underline{r}_0^T \cdot \underline{r}_0} = \sqrt{r_K^2} = r_K$$

$$\left(\sqrt{\underline{r}_F^T \cdot \underline{r}_F} - \sqrt{\underline{r}_0^T \cdot \underline{r}_0} \right)^2 = x^2(t) + r_K^2 - 2\,x(t) \cdot r_K$$



$$T = \frac{1}{2} m_{Gondel} \cdot (x^2(t) \cdot \dot{\alpha}^2(t) + \dot{x}^2(t)) + \frac{1}{2} (20 \cdot I_{Gondel} \cdot \dot{\alpha}^2)$$

$$V = \frac{1}{2} c \cdot (x^2(t) + r_K^2 - 2x(t) \cdot r_K) + m_{Gondel} \cdot g \cdot (l_s \cdot \cos\varphi_0 - \sin\varphi_0 \cdot \cos\alpha(t) \cdot x(t))$$

$$L = T - V = \frac{1}{2} m_{Gondel} \cdot x^2(t) \cdot \dot{\alpha}^2(t) + \frac{1}{2} m_{Gondel} \cdot \dot{x}^2(t) +$$
$$+ 10 \cdot I_{Gondel} \cdot \dot{\alpha}^2(t) - \frac{1}{2} c \cdot x^2(t) - \frac{1}{2} c \cdot r_K^2 + c \cdot x(t) \cdot r_K$$
$$- m_{Gondel} \cdot g \cdot l_s \cdot \cos\varphi_0 + m_{Gondel} \cdot g \cdot \sin\varphi_0 \cdot \cos\alpha(t) \cdot x(t)$$

↳ nicht zeitabhängig

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = 0$$

$$\frac{\partial L}{\partial \dot{x}(t)} = m_{Gondel} \cdot \dot{x}(t) \quad ; \quad \frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}(t)}\right) = m_{Gondel} \cdot \ddot{x}(t)$$
$$\frac{\partial L}{\partial x(t)} = m_{Gondel} \cdot x(t) \cdot \dot{\alpha}^2(t) - c \cdot x(t) + c \cdot r_K$$
$$+ m_{Gondel} \cdot g \cdot \sin\varphi_0 \cdot \cos\alpha(t)$$

$$\Rightarrow m_{Gondel} \cdot \ddot{x}(t) - m_{Gondel} \cdot x(t) \cdot \dot{\alpha}^2(t) + c \cdot x(t) - c \cdot r_K -$$
$$- m_{Gondel} \cdot g \cdot \sin\varphi_0 \cdot \cos\alpha(t) = 0$$

$$\ddot{x}(t) = x(t) \cdot \dot{\alpha}^2(t) + g \cdot \sin\varphi_0 \cdot \cos\alpha(t) + \frac{c}{m_{Gondel}} \cdot (r_K - x(t)) \quad \left[\frac{m}{s^2}\right] \checkmark$$

$$\frac{\partial L}{\partial \dot{\alpha}(t)} = m_{Gondel} \cdot \dot{\alpha}(t) \cdot x^2(t) + 20 \cdot I_{Gondel} \cdot \dot{\alpha}(t)$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\alpha}(t)}\right) = m_{Gondel} \cdot \ddot{\alpha}(t) \cdot x^2(t) + 2 \cdot m_{Gondel} \cdot \dot{\alpha}(t) \cdot x(t) \cdot \dot{x}(t) + 20 \cdot I_{Gondel} \cdot \ddot{\alpha}(t)$$

$$\frac{\partial L}{\partial \alpha(t)} = - m_{Gondel} \cdot g \cdot \sin\varphi_0 \cdot \sin\alpha(t) \cdot x(t)$$

$$\Rightarrow \ddot{\alpha}(t) \left(m_{Gondel} \cdot x^2(t) + 20 \cdot I_{Gondel}\right) + 2 \cdot m_{Gondel} \cdot \dot{\alpha}(t) \cdot \dot{x}(t) +$$
$$+ m_{Gondel} \cdot g \cdot \sin\varphi_0 \cdot \sin\alpha(t) \cdot x(t) = 0$$

$I_{Gondel} = m_{Gondel} \cdot \left[\frac{1}{12}\left(b_{Gondel}^2 + h_{Gondel}^2\right) + r_{Karussell}^2\right]$

$$\ddot{\alpha}(t) = - \frac{2 \cdot \dot{\alpha}(t) \cdot x(t) \cdot \dot{x}(t) + g \cdot \sin\varphi_0 \cdot \sin\alpha(t) \cdot x(t)}{x^2(t) + \frac{5}{3}\left(b_{Gondel}^2 + h_{Gondel}^2\right) + 20 \cdot r_{Karussell}^2} \quad \left[\frac{1}{s^2}\right] \checkmark$$

10.3. Python-Code: explizites 1-Schritt-Euler-Verfahren

```
import numpy as np
import matplotlib.pyplot as plt
import math


def setup_simulation(simulation_time, step_time):
    number_of_steps = int(simulation_time / step_time)
    time_array = np.zeros(number_of_steps)
    x = np.zeros(number_of_steps)          # Displacement in meter
    x_dot = np.zeros(number_of_steps)      # Velocity in meter/second
    alpha = np.zeros(number_of_steps)      # Angular displacement in radians
    alpha_dot = np.zeros(number_of_steps)  # Angular velocity in radians/second
    return number_of_steps, time_array, x, x_dot, alpha, alpha_dot


def setup_inital_conditions(initial_x, initial_x_dot, initial_alpha,
initial_alpha_dot):
    x[0] = initial_x
    x_dot[0] = initial_x_dot
    alpha[0] = initial_alpha
    alpha_dot[0] = initial_alpha_dot


def run_simulation():
    for i in range(0, number_of_steps-1):
        x[i + 1] = x_dot[i] * STEP_TIME + x[i]
        x_dot[i + 1] = (x[i] * (alpha_dot[i]**2) + G * np.sin(PHI_0_RAD) *
                        np.cos(alpha[i]) + ((C / M_GONDEL) *
                            (R_KARUSELL - x[i]))) * STEP_TIME + x_dot[i]
        alpha[i + 1] = alpha_dot[i] * STEP_TIME + alpha[i]
        alpha_dot[i + 1] = (-(2 * alpha_dot[i] * x_dot[i] * x[i] + G *
                              np.sin(PHI_0_RAD) * np.sin(alpha[i]) * x[i]) /
                                (x[i]**2 + (5/3) * (B_GONDEL**2 + H_GONDEL**2) +
                                    20 * R_KARUSELL**2)) * STEP_TIME + alpha_dot[i]
        time_array[i + 1] = (i + 1) * STEP_TIME


def plot_results() -> None:
    fig, axs = plt.subplots(4, 1, figsize=(10, 12))

    axs[0].plot(time_array, x, label=r'$x(t)$')
    axs[0].set_ylabel(r'$x$ [m]')
    axs[0].legend(loc='upper right')
    axs[0].grid()
    axs[0].set_xlim(min(time_array), max(time_array))

    axs[1].plot(time_array, x_dot, label=r'$\dot{x}(t)$')
    axs[1].set_ylabel(r'$\dot{x}$ [m/s]')
    axs[1].legend(loc='upper right')
    axs[1].grid()
    axs[1].set_xlim(min(time_array), max(time_array))

    degrees_alpha = np.degrees(alpha)
```

```
    axs[2].plot(time_array, degrees_alpha, label=r'$\alpha(t)$')
    axs[2].set_ylabel(r'$\alpha$ [deg]')
    axs[2].legend(loc='upper right')
    axs[2].grid()
    axs[2].set_xlim(min(time_array), max(time_array))

    degrees_alpha_dot = np.degrees(alpha_dot)
    axs[3].plot(time_array, degrees_alpha_dot, label=r'$\dot{\alpha}(t)$')
    axs[3].set_ylabel(r'$\dot{\alpha}$ [deg/s]')
    axs[3].legend(loc='upper right')
    axs[3].grid()
    axs[3].set_xlim(min(time_array), max(time_array))

    for i in range(1, int(max(degrees_alpha) / 360) + 1):
        idx = np.where(np.isclose(degrees_alpha, 360 * i, atol=1))[0]
        if len(idx) > 0:
            axs[2].plot(time_array[idx[0]], degrees_alpha[idx[0]], 'ro')
            for ax in axs:
                ax.axvline(x=time_array[idx[0]],
                           color='r', linestyle='--', linewidth=1)

    y_ticks = np.arange(0, max(degrees_alpha) + 360, 360)
    axs[2].set_yticks(y_ticks)
    axs[2].set_yticklabels([f'{int(y)}' for y in y_ticks])

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.text(0.5, 0.01, 'Time (t)', ha='center', fontsize=12)
    fig.suptitle(f"{C=} N/m", fontsize=16)
    plt.show()

if __name__ == "__main__":
    G = 9.81                            # Gravity acceleration [m/s^2]
    PHI_0_DEG = 30                      # Angle of tilted carousel [°]
    PHI_0_RAD = math.radians(PHI_0_DEG) # Angle of tilted carousel [radians]
    R_KARUSELL = 6                      # Radius of carousel [m]
    C = 3000000                         # Spring stiffness [N/m]
    M_GONDEL = 300                      # Mass of gondola [kg]
    N_GONDEL = 0.2333                   # Rotational speed [radians/second]
    B_GONDEL = 1.5                      # Width of gondola [m]
    H_GONDEL = 1.5                      # Height of gondola [m]

    SIMULATION_TIME = 10  # [s]
    STEP_TIME = 0.0001     # [s]
    number_of_steps, time_array, x, x_dot, alpha, alpha_dot = setup_simulation(
        SIMULATION_TIME, STEP_TIME)
    setup_inital_conditions(initial_x=R_KARUSELL, # Initial displacement (6 meter)
                            initial_x_dot=0, # Initial velocity ()
                            initial_alpha=0, # Initial angular displacement ()
                            initial_alpha_dot=2*np.pi*N_GONDEL) # Initial angular
velocity ()
    run_simulation()
    plot_results()
```

10.4. Python-Code: Runge-Kutta-Verfahren

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import solve_ivp
import math
from termcolor import colored
import colorama
colorama.init()


def system_dynamics(t, y):
    x, x_dot, alpha, alpha_dot = y
    dxdt = x_dot
    dx_dotdt = (x * (alpha_dot**2) + G * np.sin(PHI_0_RAD) *
                np.cos(alpha) + ((C / M_GONDEL) * (R_KARUSELL - x)))
    dalphadt = alpha_dot
    dalpha_dotdt = (-(2 * alpha_dot * x_dot * x + G * np.sin(PHI_0_RAD) *
                      np.sin(alpha) * x) / (x**2 + (5/3) *
                        (B_GONDEL**2 + H_GONDEL**2) + 20 * R_KARUSELL**2))
    return [dxdt, dx_dotdt, dalphadt, dalpha_dotdt]


def run_simulation():
    t_span = [0, SIMULATION_TIME]
    solution = solve_ivp(system_dynamics, t_span,
                         initial_conditions, method='RK45', rtol=1e-3, atol=1e-6)

    """ Solver keeps local error estimates less than `atol + rtol * abs(y) `
        - rtol controls relative accuracy (number of correct digits)
        - atol controls absolute accuracy (number of correct decimal places)
        To achieve desired `rtol`, set `atol` smaller than the smallest value of
            `rtol * abs(y)`
        Conversely, to achieve desired `atol`, set `rtol` smaller than smallest
            value of `rtol * abs(y)`
        Default values used for rtol and atol.
    """
    time_array = solution.t
    x, x_dot, alpha, alpha_dot = solution.y
    return time_array, x, x_dot, alpha, alpha_dot


def plot_results(time_array, x, x_dot, alpha, alpha_dot) -> None:
    fig, axs = plt.subplots(4, 1, figsize=(10, 12))

    axs[0].plot(time_array, x, label=r'$x(t)$')
    axs[0].set_ylabel(r'$x$ [m]')
    axs[0].legend(loc='upper right')
    axs[0].grid()
    axs[0].set_xlim(min(time_array), max(time_array))

    axs[1].plot(time_array, x_dot, label=r'$\dot{x}(t)$')
    axs[1].set_ylabel(r'$\dot{x}$ [m/s]')
    axs[1].legend(loc='upper right')
    axs[1].grid()
```

```
    axs[1].set_xlim(min(time_array), max(time_array))

    degrees_alpha = np.degrees(alpha)
    axs[2].plot(time_array, degrees_alpha, label=r'$\alpha(t)$')
    axs[2].set_ylabel(r'$\alpha$ [deg]')
    axs[2].legend(loc='upper right')
    axs[2].grid()
    axs[2].set_xlim(min(time_array), max(time_array))

    degrees_alpha_dot = np.degrees(alpha_dot)
    axs[3].plot(time_array, degrees_alpha_dot, label=r'$\dot{\alpha}(t)$')
    axs[3].set_ylabel(r'$\dot{\alpha}$ [deg/s]')
    axs[3].legend(loc='upper right')
    axs[3].grid()
    axs[3].set_xlim(min(time_array), max(time_array))

    for i in range(1, int(max(degrees_alpha) / 360) + 1):
        idx = np.where(np.isclose(degrees_alpha, 360 * i, atol=1))[0]
        if len(idx) > 0:
            axs[2].plot(time_array[idx[0]], degrees_alpha[idx[0]], 'ro')
            for ax in axs:
                ax.axvline(x=time_array[idx[0]],
                           color='r', linestyle='--', linewidth=1)

    y_ticks = np.arange(0, max(degrees_alpha) + 360, 360)
    axs[2].set_yticks(y_ticks)
    axs[2].set_yticklabels([f'{int(y)}' for y in y_ticks])

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    fig.text(0.5, 0.01, 'Time (t)', ha='center', fontsize=12)
    fig.suptitle(f"{C=} N/m", fontsize=16)
    plt.show()


def get_local_extremes(data_series):
    local_maxima = []
    local_minima = []
    for i in range(1, len(data_series) - 1):
        if data_series[i] > data_series[i-1] and data_series[i] > data_series[i+1]:
            local_maxima.append(data_series[i])
        if data_series[i] < data_series[i-1] and data_series[i] < data_series[i+1]:
            local_minima.append(data_series[i])
    return local_minima, local_maxima


def get_global_extremes(local_minima, local_maxima):
    try:
        global_minima = round(min(local_minima), 5)
        global_maxima = round(max(local_maxima), 5)
        difference = round(global_maxima - global_minima, 5)
        return global_minima, global_maxima, difference

    except ValueError:
        return None, None, None
```

```
def find_optimal_C(start: int, step_size: int, MAX_RADIAL_DISPLACEMENT: float,
amount_of_results: int, show_results: bool) -> None:
    """
    This function finds and prints the optimal values of C (stiffness constant)
    that result in a radial displacement less than or equal to a specified maximum
    value. The search starts from a given initial value and increments by a
    specified step size. It stops when a specified number of valid results are
    found.

    Parameters:
    - start (int): The initial value of C to start the search from.
    - step_size (int): The step size to increment C in each iteration.
    - MAX_RADIAL_DISPLACEMENT (float): The maximum allowable radial displacement.
    - amount_of_results (int): The number of valid results to find before stopping
    the search.
    - show_results (bool): If True, plots the results using the plot_results
function.

    Returns:
    - None
    """

    global C
    results_counter = 0

    for C in np.arange(start, 999_999_999_999, step_size):
        try:
            time_array, x, x_dot, alpha, alpha_dot = run_simulation()
            local_minima, local_maxima = get_local_extremes(x)
            global_minima, global_maxima, difference = get_global_extremes(
                local_minima, local_maxima)
            if difference is not None and difference <= MAX_RADIAL_DISPLACEMENT:
                print(colored(
                    f"[C: {C:<8} N/m] Global maxima: {global_maxima:<8} | Global
minima: {global_minima:<8} | Difference is {difference:<8}", 'green'))
                results_counter += 1
                if show_results:
                    plot_results(time_array, x, x_dot, alpha, alpha_dot)
                if results_counter >= amount_of_results:
                    break
            elif difference is None:
                print(
                    colored(f"[C: {C:<8} N/m] no Global minima or maxima found!",
'red'))
            else:
                print(colored(
                    f"[C: {C:<8} N/m] Global maxima: {global_maxima:<8} | Global
minima: {global_minima:<8} | Difference is {difference:<8}", 'red'))
                results_counter = 0

        except Exception as e:
            print(colored(f"[C: {C:<8} N/m] Error: {str(e)}", 'red'))
            results_counter = 0
```

```python
if __name__ == "__main__":
    G = 9.81                                # Gravity acceleration [m/s^2]
    PHI_0_DEG = 30                          # Angle of tilted carousel [°]
    PHI_0_RAD = math.radians(PHI_0_DEG)     # Angle of tilted carousel [radians]
    R_KARUSELL = 6                          # Radius of carousel [m]
    M_GONDEL = 300                          # Mass of gondola [kg]
    N_GONDEL = 0.2333                       # Rotational speed [radians/second]
    B_GONDEL = 1.5                          # Width of gondola [m]
    H_GONDEL = 1.5                          # Height of gondola [m]

    SIMULATION_TIME = 10                    # [s]
    MAX_RADIAL_DISPLACEMENT = 0.005         # Maximum radial displacement [m]
    initial_conditions = [R_KARUSELL, 0, 0, 2 * np.pi * N_GONDEL]

    find_optimal_C(
        start=0,
        step_size=100_000,
        MAX_RADIAL_DISPLACEMENT=MAX_RADIAL_DISPLACEMENT,
        amount_of_results=1,
        show_results=True
    )
```