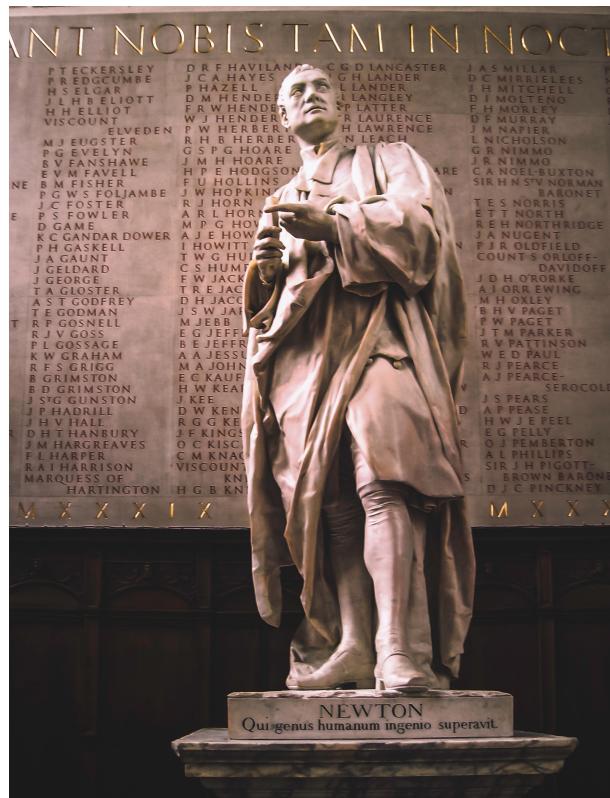


Semesterprojekt Physik Engines

Kim Lan Vu, Michel Steiner, Asha Schwegler

23. April 2023



Inhaltsverzeichnis

1 Zusammenfassung	3
2 Aufbau des Experiments	4
2.1 Aufbau des Lab 2, „Würfel 1 bewegt sich und stösst“	4
3 Physikalische Beschreibung der einzelnen Vorgänge	5
3.1 Lab 2: Würfel bewegt sich und stösst	5
3.1.1 Konstante Kraft	5
3.1.2 Elastischer Stoss	6
3.1.3 Inelastischer Stoss	7
4 Beschreibung der Implementierung inklusive Screenshots aus Unity	8
4.1 Lab 2: Würfel bewegt sich und stösst	8
5 Rückblick und Lehren aus dem Versuch	12
6 Resultate mit grafischer Darstellung	13
6.1 Grafiken zu Lab 2	13
6.1.1 Elastisch	13
6.1.2 Inelastisch	14
A Anhang	17
A.1 Code für das Lab 2	17
A.2 Code für die Datenaufbereitung des Elastischen Stosses des Lab 2	20
A.3 Code für die Datenaufbereitung des Inelastischen Stosses des Lab 2	21

1 Zusammenfassung

An der ZHAW wird im Physik Engine, kurz PE, physikalische Zusammenhänge von Bewegungsrichtungen und Beschleunigung, sowie potenzieller und kinetischer Energie im Raum und Zeit durch Simulationen auf Körper untersucht. Dabei werden über das Semester hinweg in Kleingruppen zwei Labs bearbeitet.

Im ersten Lab werden die inelastischen und elastischen Stöße untersucht.

Diese Untersuchungen werden mit den physikalischen Gesetzen in der Game-Engine Unity simuliert. Dabei werden laufend Daten zu den aktuellen Parameter einzelner Objekte gesammelt und mit diversen Grafiken veranschaulicht. Die aus den zwei Experimenten gewonnenen Erkenntnisse werden in diesem Bericht niedergeschrieben.

2 Aufbau des Experiments

2.1 Aufbau des Lab 2, „Würfel 1 bewegt sich und stösst“

Für den Aufbau des Experimentes sind zwei Würfel mit den Dimensionen von 1.5m Seitenlänge und dem Gewicht von 2 Kilogramm gegeben. Wie in der Abbildung 1 zu entnehmen ist, wird der linke Würfel Julia und der rechte Romeo benannt. Daneben existiert eine Feder die horizontal an einer Wand befestigt ist. Bei dem gesamten Experiment wird der Reibungswiderstand ignoriert.

Ablauf des Experimentes:

1. Romeo wird mit einer konstanten Kraft (grüner Pfeil in Abbildung 1) auf 2m/s nach rechts beschleunigt.



Abbildung 1: Beschleunigung des Würfels

2. Romeo trifft nun auf die Feder. Dabei soll die Federkonstante (gelber Pfeil in Abbildung 2) so gewählt werden, dass Romeo elastisch zurückprallt ohne die Wand zu berühren.



Abbildung 2: Elastischer Zusammenstoß mit der Feder

3. Nach dem abgefederten Stoss gleitet Romeo zurück in die Richtung aus der er gekommen ist und stösst inelastisch mit Julia zusammen. Über einen FixedJoint haften die Beiden nun zusammen und gleiten mit der übertragener Energie (blaue Pfeile in Abbildung 3) weiter nach links.



Abbildung 3: Inelastischer Zusammenstoß mit dem anderen Würfel

3 Physikalische Beschreibung der einzelnen Vorgänge

In diesem Kapitel werden die physikalischen Vorgänge des Versuches beschrieben. Die gegebenen Massen sind:

- Gewicht[m] = 2kg
- Velocity[v] = 2m/s
- Würfelseite = 1.5m

3.1 Lab 2: Würfel bewegt sich und stösst

Es werden drei Vorgänge beschrieben, die Beschleunigung durch die konstante Kraft, einen elastischen Stoß und einen inelastischen Stoß. Ein Würfel, namens Romeo, wird durch die konstante Kraft beschleunigt, bis maximal eine Geschwindigkeit von 2m/s erreicht wird. Romeo trifft auf eine Feder zu, die an einer Wand befestigt ist. Dabei geschieht ein elastischer Stoß und der Würfel gleitet wieder zurück und stößt dabei einen zweiten Würfel, Julia, diesmal passiert der Stoß inelastisch. Sämtliche Vorgänge erfolgen ohne Reibungskräfte.

3.1.1 Konstante Kraft

Um die konstante Kraft zu berechnen nehmen wir die gewünschte Geschwindigkeit und berechnen damit die Beschleunigung, weil die Kraft sowohl von der Masse wie auch der Beschleunigung abhängt und gegeben ist durch die Formel[1]

$$F = m * a$$

Um dieses Anfangswertproblems zu lösen leiten wir die Geschwindigkeit ab [1]:

$$\begin{aligned} \dot{v} &= a \\ 2m * s^{-1} &\rightarrow -2m * s^{-2} \rightarrow a = [\frac{2m}{s^2}] \end{aligned}$$

Die Zeit, die gebraucht wird um den Würfel zu beschleunigen, wird durch folgende Formel beschrieben [1]:

$$v = a * t \rightarrow t = \frac{v}{a} \rightarrow \frac{2m/s}{2m/s^2} = 1s$$

Somit können wir nun die Kraft ausrechnen:

$$F = 2kg * \frac{2m}{s^2} \Rightarrow \frac{4kg*m}{s^2} = 4N$$

4N werden deshalb als konstante Kraft angewendet, damit auch die gewünschte Geschwindigkeit erreicht wird, danach wird keine Kraft mehr hinzugefügt und Romeo gleitet auf die Feder zu.

3.1.2 Elastischer Stoss

Beim elastischen Stoss ist die kinetische Energie vom Stosspartner vor und nach der Kollision gleich [1]. Gemäss Auftrag wird die Federlänge und Federkonstante so dimensioniert, dass der Würfel nicht auf die Wand trifft. Die kinetische Energie des Würfels wird mit folgender Formel berechnet [1]:

$$E_{kin_{Romeo}} = \frac{1}{2} * m * v^2$$

Setzt man die Massen von diesem Projekt ein erhält man:

$$\frac{1}{2} * 2kg * (\frac{2m}{s})^2 = 4J$$

Während des Stosses wird die kinetische Energie auf die Feder übetragen. Die Feder speichert diese Energie in Form von potentieller Energie, da sie zusammengedrückt wird. Sobald sie Romeo zurückstößt, wird diese Energie in eine kinetische zurückgewandelt.

Um die Federkonstante zu berechnen, nehmen wir die Tatsache der Energieerhaltung zu Nutze und setzen die ausgerechnete kinetische Energie gleich mit der potentiellen Energie der Feder.

Die Formel für die potentielle Energie der Feder lautet[1]:

$$E_{potFeder} = \frac{1}{2} * k * x^2$$

Die Gleichsetzung der Energien, sieht folgendermassen aus[1]:

$$E_{kin_{Romeo}} = E_{potFeder}$$

$$\frac{1}{2} * m * v^2 = \frac{1}{2} * k * x^2$$

Diese Gleichung stellen wir um und lösen nach der Federkonstante k auf:

$$k = \frac{m*v^2}{x^2}$$

Mit den eingesetzten Massen und die gewählte maximale Auslenkung erhalten wir:

$$\frac{2kg*(2m/s)^2}{(1.7m)^2} = 2.77N/m$$

Jetzt wo wir die Federkonstante und Länge haben, können wir einen langsam Stoss gewährleisten.

3.1.3 Inelastischer Stoss

Beim vollständigen inelastischen Stoss, werden beide Stosspartner nach der Kollision verbunden sein und dieselbe Geschwindigkeit haben[1]. Die Formeln, die wir für diesen Vorgang brauchen sind, die der Impulse der beiden Körper:

$$Impuls_{Romeo} = m_{Romeo} * v_{Romeo}$$

$$Impuls_{Julia} = m_{Julia} * v_{Julia}$$

Bei diesem Vorgang wird ein Teil des Impulses von Romeo auf Julia übertragen. Der Gesamtimpuls bleibt erhalten vor und nach dem Stoss und wird durch den Impulserhaltungssatz beschrieben[1]:

$$m_{Romeo} * v_{Romeo} + m_{Julia} * v_{Julia} = (m_{Romeo} + m_{Julia}) * v_{Ende}$$

Die Endgeschwindigkeit ist die Geschwindigkeit, die beide Körper gemeinsam haben nach dem Stoss. Die Relation zwischen der kinetischen Energie und des Impulses, können wir folgendermassen herleiten[1]:

$$E_{kin} = \frac{1}{2} * m * v^2 = \frac{(mv)^2}{2m} = \frac{p^2}{2m}$$

Wenden wir dies nach dem Stoss an, sehen wir, dass die kinetische Energie geringer wird:

$$E_{kin_{Ende}} = \frac{p^2}{2(m_{Romeo} + m_{Julia})}$$

4 Beschreibung der Implementierung inklusive Screenshots aus Unity

Der Programcode für Unity wird in C# geschrieben und im folgenden Kapitel wird auf die Implementation in Unity eingegangen.

4.1 Lab 2: Würfel bewegt sich und stösst

Alle Kräfte und Berechnungen befinden sich im Code CubeController.cs, welches im Anhang ersichtlich ist. Zur Kontrolle der Werte und Grafik Erstellung werden zwei verschiedene CSV Dateien erstellt, eine für den elastischen Stoss relevanten Werte und eine für den inelastischen Stoss.

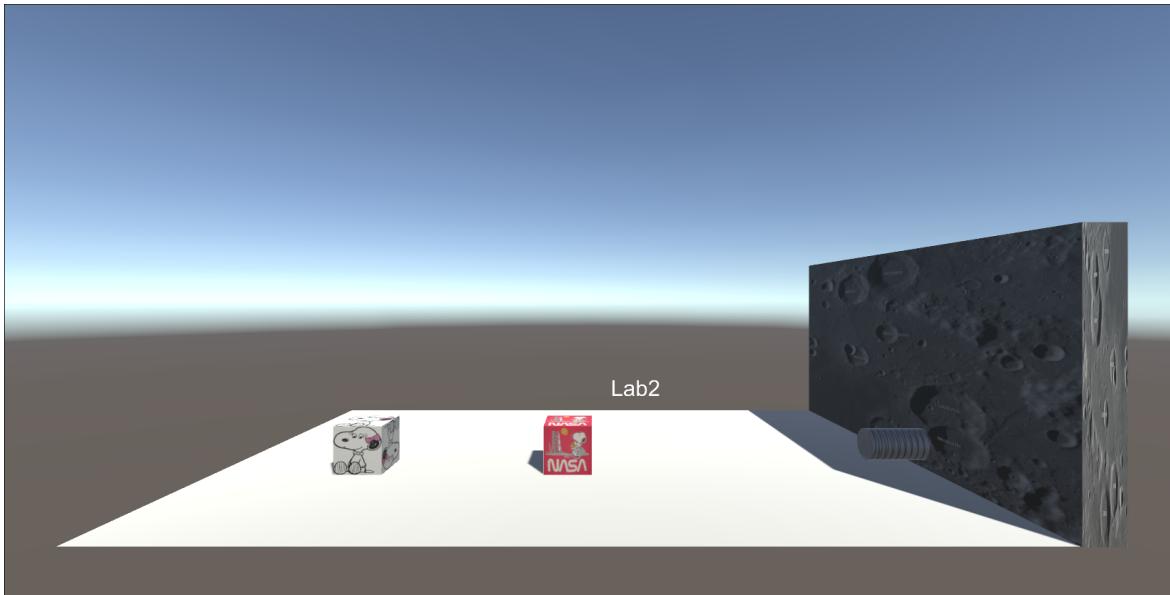


Abbildung 4: Experiment Übersicht

Folgend sind einige wichtige Eigenschaften der Lab relevanten Objekte in Unity aufgelistet:

- Julia:

Für die Seitenlänge ist die Scale auf 1.5 angepasst, da in Unity beim Würfel eine Seitenlänge von 1 gilt. Wichtig ist nicht zu vergessen, das Material auf reibungslos zu ändern, sonst gleiten die Würfel nicht korrekt.

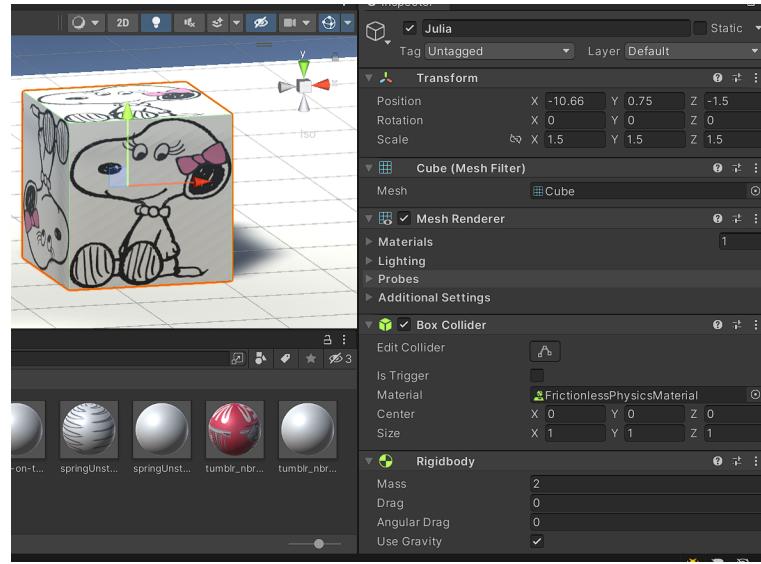


Abbildung 5: Einstellung Julia

- Romeo:

Die Eigenschaften sind ausser der Position und Farbe gleich wie bei Julia. Romeo besitzt zudem Variabel, über welcher gewisse Parameter an den ihm angehängten Code übergeben werden kann.

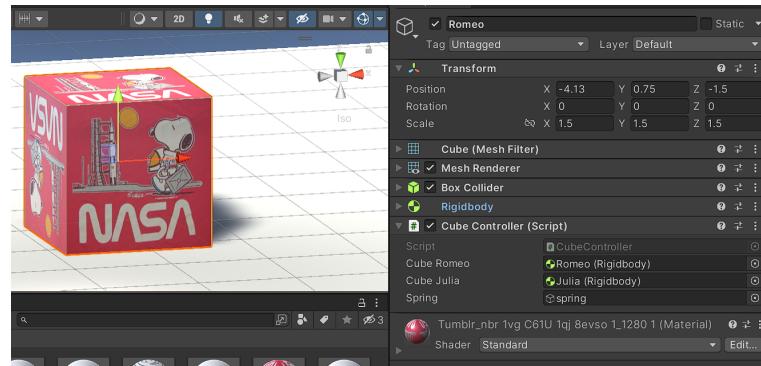


Abbildung 6: Einstellung Romeo

- Spring:

Wie in Abbildung 6 zu sehen ist die Feder nur als GameObject und nicht als Rigidbody im Code angegeben. Die Ausrichtung wurde entlang der Y-Achse belassen und um 90 Grad rotiert damit der Zylinder liegend erscheint.

- Mesh: Cylinder
- Collider direction: Y-Axis

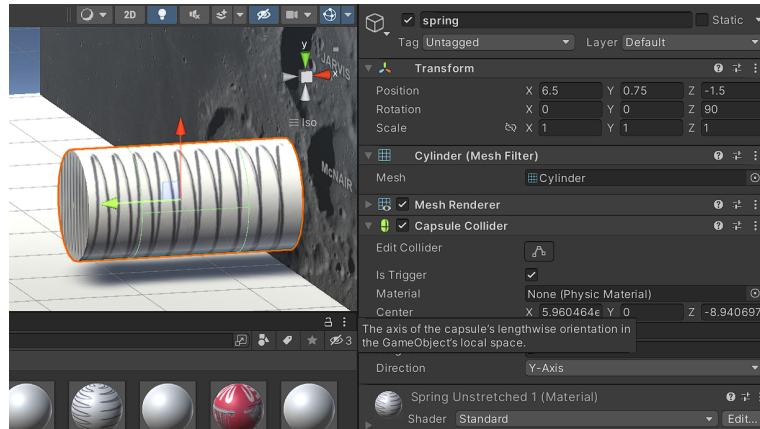


Abbildung 7: Einstellung Feder

- Plane
 - Collider Material: FrictionlessPhysicsMaterial

Da es sich beim CubeController um ein Unity Code handelt, wird von der Klasse Monobeviour geerbt, damit Methoden wie FixedUpdate oder OnCollisionEnter benutzt werden kann. Die im vorhinein berechnete konstante Kraft 4, sowie Beschleunigungszeit 1 wird im Code als Konstanten am Anfang deklariert. Für die Federauslenkung wird 1.7 gewählt, da die Feder eine Länge von 2 hat und vorher gestoppt werden muss bevor Romeo auf die Wand auftrefft. In der Start Methode wird aus den gegebenen Werten die Federkonstante berechnet.

```

1 //Maximale Auslenkung gerechnet anhand der linken seite des Feders
2 springMaxDeviation = spring.transform.position.x - springLength / 2;
```

Es wird auch die Position ermittelt, welche Romeo zum ersten Mal besitzt, wenn er auf die Feder auftrefft (springMaxDeviation) wie in Abbildung 8 rot markiert ist.

```

1 // Energieerhaltungsgesetz kinEnergie = PotEnergie : 1/2*m*v^2 = 1/2k * x^2
2 springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
    (Math.Pow(springContraction, 2.0)));
```

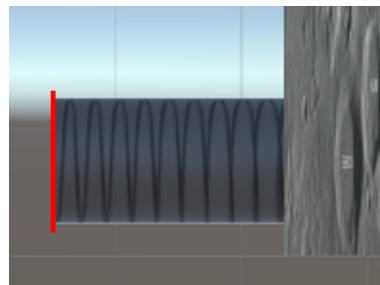


Abbildung 8: Einstellung Feder

Danach werden die Timeseries Listen deklariert, für die CSV Dateien. Das Hinzufügen der Werte passiert kontinuierlich in der Methode FixedUpdate.

In FixedUpdate gibt es zwei If-Bedingungen. Die erste ist zum Hinzufügen der konstanten Kraft mit der Methode .AddForce bis die Beschleunigungszeit vorüber ist. Die zweite If-Bedingung ist zur

Überprüfung, ob Romeo die Feder berührt. Dafür muss die Position der rechten Kante von Romeo berechnet werden und mit der springMaxDeviation verglichen werden. Für den inelastischen Stoss wird die Komponente FixedJoint in der Methode OnCollision implementiert. So bleiben Romeo und Julia nach ihrer Kollision zusammen und gleiten gemeinsam in den Sonnenuntergang.

5 Rückblick und Lehren aus dem Versuch

6 Resultate mit grafischer Darstellung

6.1 Grafiken zu Lab 2

Während dem Durchlauf des Experimentes des Lab 2, werden diverse Daten physikalische Vorgänge gesammelt. Diese Daten umfassen Ort und Geschwindigkeit, sowie kinetische und potentielle Energie. Dabei wird zwischen dem elastischen sowie inelastischen Zusammenstoss unterschieden.

6.1.1 Elastisch

Nachfolgend werden alle Daten als Funktion der Zeit zu dem elastischen Zusammenstoss in der Abbildung 9 bis Abbildung 10 aufgegliedert.

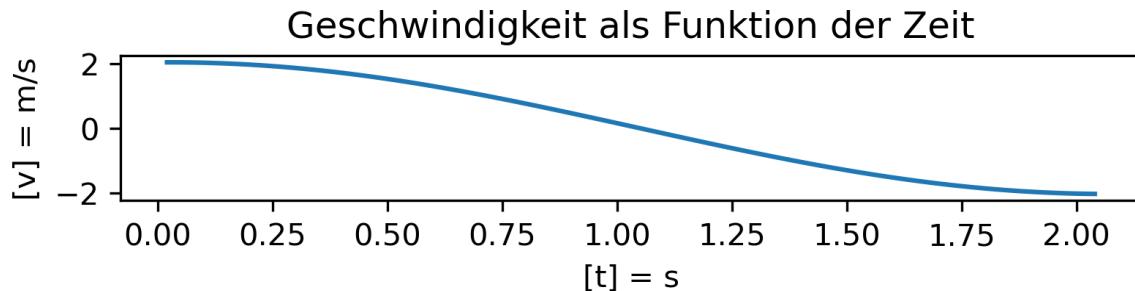


Abbildung 9: Geschwindigkeit als Funktion der Zeit

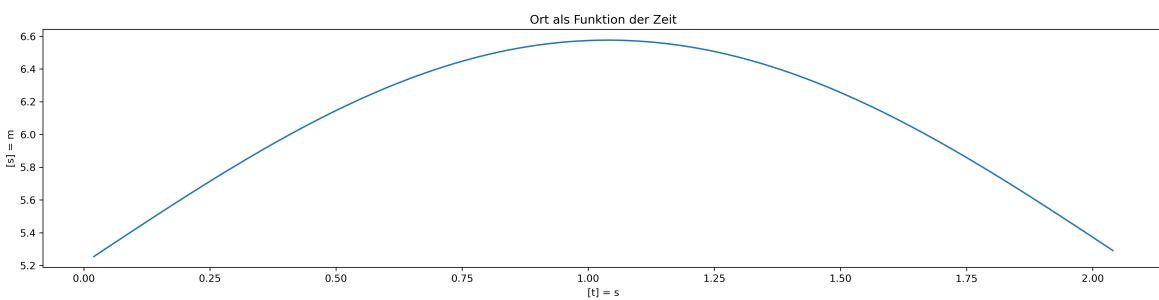


Abbildung 10: Ort als Funktion der Zeit

6.1.2 Inelastisch

Nachfolgend werden alle Daten als Funktion der Zeit zu dem inelastischen Zusammenstoß in der Abbildung 11 bis Abbildung 17 aufgegliedert.

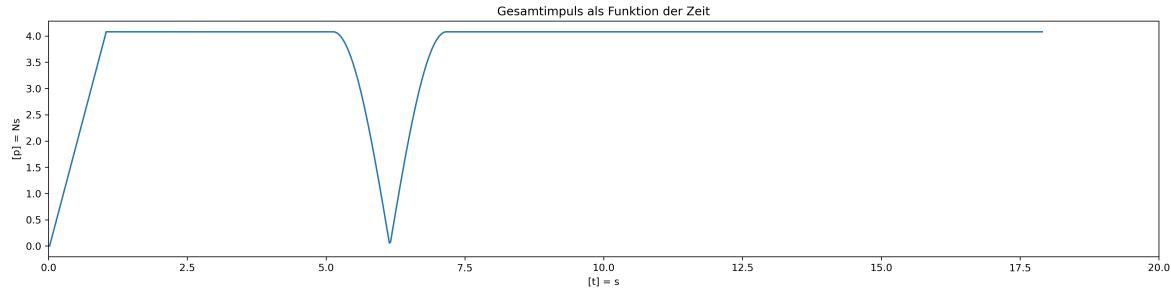


Abbildung 11: GesamtImpuls als Funktion der Zeit

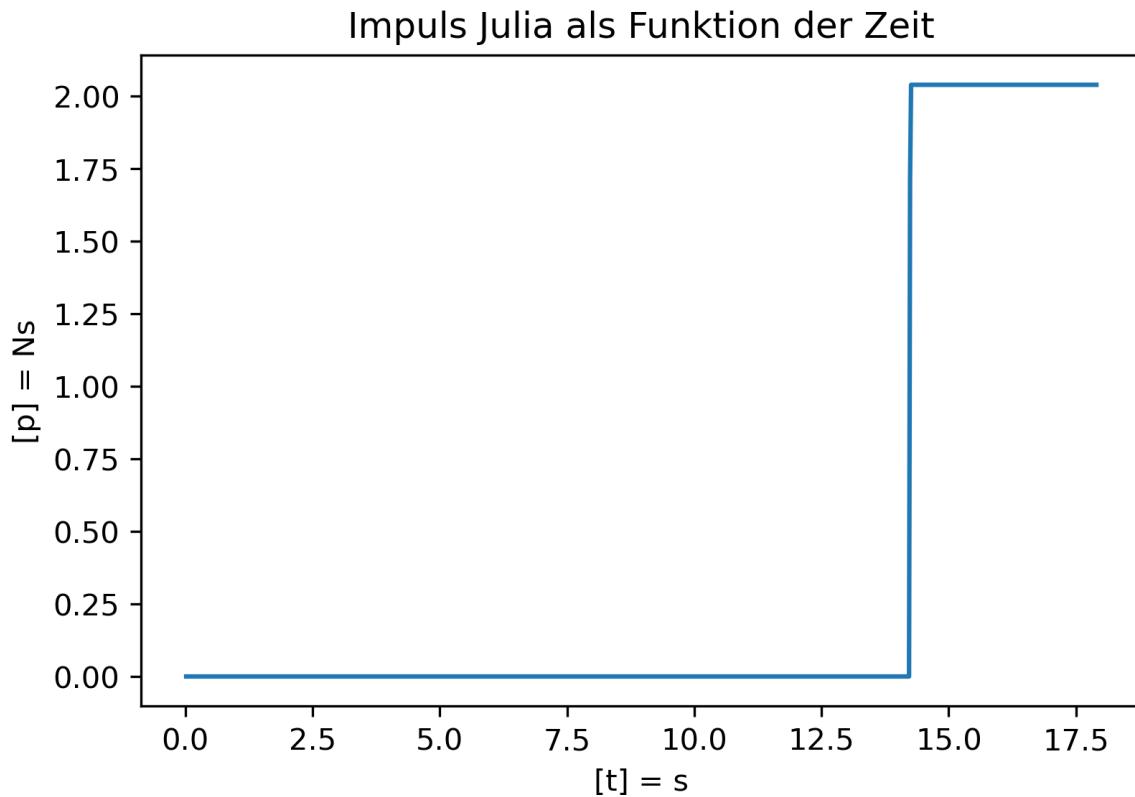


Abbildung 12: Impuls Julia als Funktion der Zeit

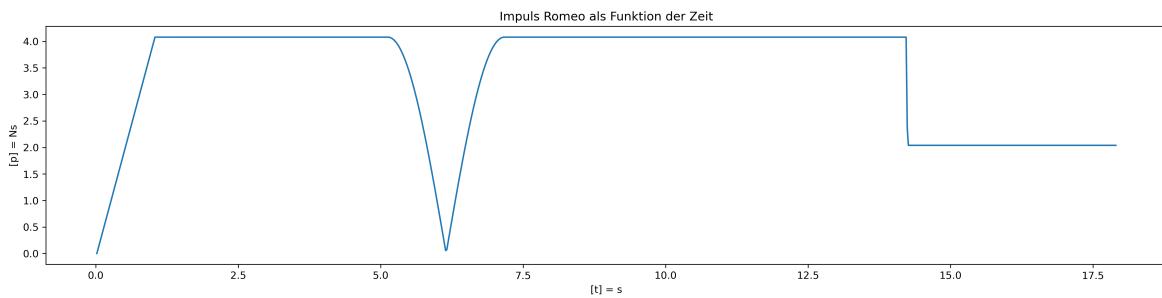


Abbildung 13: Impuls Romeo als Funktion der Zeit

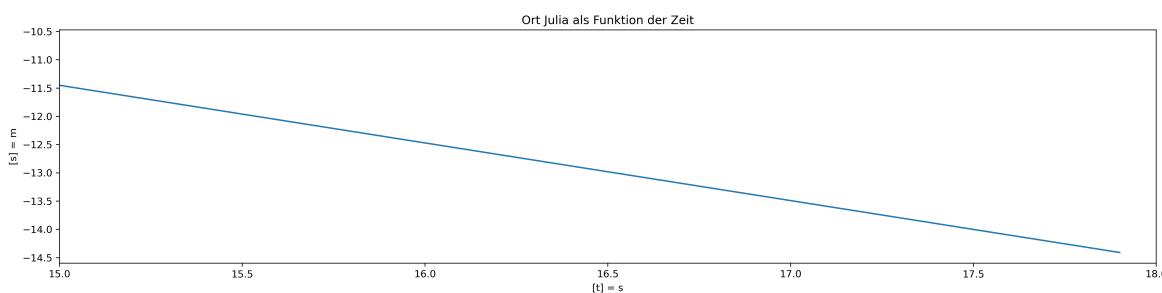


Abbildung 14: Ort Julia als Funktion der Zeit

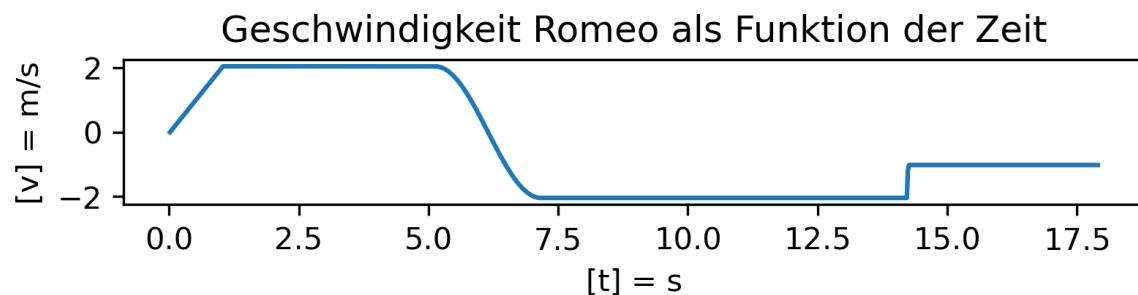


Abbildung 15: Geschwindigkeit Romeo als Funktion der Zeit

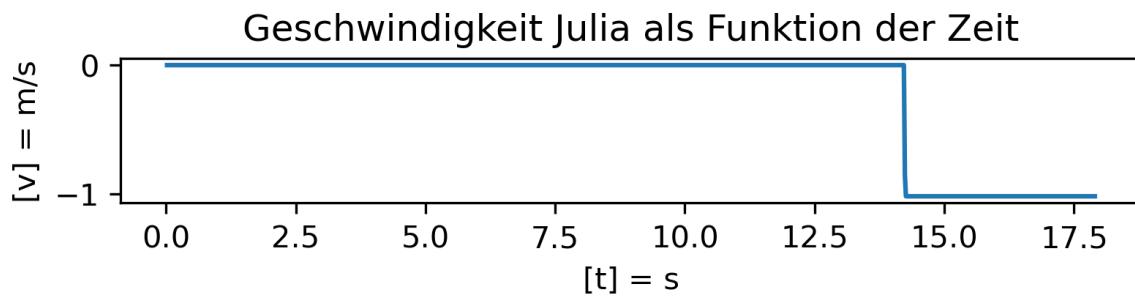


Abbildung 16: Geschwindigkeit Julia als Funktion der Zeit

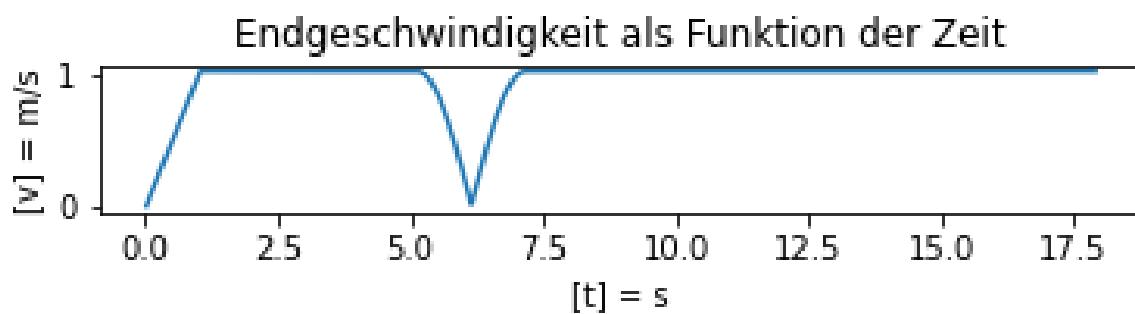


Abbildung 17: Endgeschwindigkeit als Funktion der Zeit

A Anhang

A.1 Code für das Lab 2

Nachfolgend ist der für das Experiment benötigte Code abgebildet. Dieser umfasst alle physikalischen Berechnungen, sowie die Bewegungen.

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using UnityEngine;
5
6
7 public class CubeController : MonoBehaviour
8 {
9     public Rigidbody cubeRomeo;
10    public Rigidbody cubeJulia;
11    public GameObject spring;
12
13    private float currentTimestep; // s
14    private float cubeJuliaTimestep;
15
16    private List<List<float>> timeSeriesElasticCollision;
17    private List<List<float>> timeSeriesInelasticCollision;
18
19    private string filePath;
20    private byte[] fileData;
21    float springPotentialEnergy = 0f;
22    float cubeRomeoKinetic = 0f;
23    float cubeRomeoImpulse = 0f;
24    float cubeJuliaImpulse = 0f;
25    float GesamtImpluls = 0f;
26    float ImpulsCheck = 0f;
27    float forceOnJulia = 0f;
28    float velocityEnd = 0f;
29    float cubeKineticEnd = 0f;
30    float constantForce = 4f;
31    double starttime = 0;
32    double accelerationTime = 1.0;
33    float springConstant = 0f;
34    float springMaxDeviation = 0f;
35    float springContraction = 1.7f;
36    float springLength = 0f;
37
38    // Start is called before the first frame update
39    void Start()
40    {
41        starttime = Time.fixedTimeAsDouble;
42
43        timeSeriesElasticCollision = new List<List<float>>();
44        timeSeriesInelasticCollision = new List<List<float>>();
45
46
47        springLength = spring.GetComponent<MeshFilter>().mesh.bounds.size.y *
48            spring.transform.localScale.y;
49
50        // Maximale Auslenkung gerechnet anhand der linken Seite des Feders
51        springMaxDeviation = spring.transform.position.x - springLength / 2;
52        // Energieerhaltungsgesetz kinEnergie = PotEnergie : 1/2*m*v^2 = 1/2k * x^2
53        springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
54            (Math.Pow(springContraction, 2.0)));
55
56
57
58
59
59
```

```

54 }
55
56 // Update is called once per frame
57 void Update()
58 {
59 }
60 // FixedUpdate can be called multiple times per frame
61 void FixedUpdate()
62 {
63     double currentTime = Time.fixedTimeAsDouble-startime;
64
65     if (accelarationTime >= currentTime)
66     {
67         constantForce = 4f;
68         cubeRomeo.AddForce(new Vector3(constantForce, 0f, 0f));
69     }
70
71 // 1/2*m*v^2
72 cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
73                               Math.Pow(cubeRomeo.velocity.x, 2.0)));
74
75 float collisionPosition = cubeRomeo.transform.position.x +
76     cubeRomeo.transform.localScale.x / 2;
77
78 if (collisionPosition >= springMaxDeviation)
79 {
80     float springForceX = (collisionPosition - springMaxDeviation) * -springConstant;
81     springPotentialEnergy =(float)(0.5 * springConstant * Math.Pow(collisionPosition -
82                                     springMaxDeviation, 2.0));
83     cubeRomeo.AddForce(new Vector3(springForceX, 0f, 0f));
84     ChangeCubeTexture();
85     currentTimeStep += Time.deltaTime;
86     timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
87                                                 cubeRomeo.position.x, cubeRomeo.velocity.x, springPotentialEnergy,
88                                                 cubeRomeoKinetic, springForceX });
89 }
90
91 // 1/2*m*v^2
92 cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
93                               Math.Pow(cubeRomeo.velocity.x, 2.0)));
94 cubeRomeoImpulse = Math.Abs(cubeRomeo.mass * cubeRomeo.velocity.x);
95 cubeJuliaImpulse = Math.Abs(cubeJulia.mass * cubeJulia.velocity.x);
96 GesamtImpluls = cubeJuliaImpulse + cubeRomeoImpulse;
97 velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (cubeRomeo.mass +
98             cubeJulia.mass);
99 ImpulsCheck = (cubeRomeo.mass + cubeJulia.mass)* velocityEnd;
100 cubeKineticEnd = Math.Abs((float)(0.5 * (cubeRomeo.mass + cubeJulia.mass) *
101                             Math.Pow(velocityEnd, 2.0)));
102 forceOnJulia = Math.Abs(cubeJulia.mass * velocityEnd - cubeJulia.velocity.x);
103
104 cubeJuliaTimeStep += Time.deltaTime;
105 timeSeriesInelasticCollision.Add(new List<float>() { cubeJuliaTimeStep,
106                                         cubeRomeo.position.x, cubeRomeo.velocity.x,cubeRomeo.mass, cubeRomeoImpulse,
107                                         cubeRomeoKinetic, cubeJulia.position.x, cubeJulia.velocity.x,cubeJulia.mass,
108                                         cubeJuliaImpulse, velocityEnd, cubeKineticEnd, forceOnJulia, GesamtImpluls,
109                                         ImpulsCheck });
110 }
111 void OnApplicationQuit()
112 {
113     WriteElasticTimeSeriesToCsv();
114     WriteInelasticTimeSeriesToCsv();

```

```

104 }
105 void WriteElasticTimeSeriesToCsv()
106 {
107     using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
108     {
109         streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
110             cubeRomeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic, springForceX");
111
112         foreach (List<float> timeStep in timeSeriesElasticCollision)
113         {
114             streamWriter.WriteLine(string.Join(", ", timeStep));
115             streamWriter.Flush();
116         }
117     }
118 }
119
120 void WriteInelasticTimeSeriesToCsv()
121 {
122     using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))
123     {
124         streamWriter.WriteLine("cubeJuliaTimeStep, cubeRomeo.position.x,
125             cubeRomeo.velocity.x,cubeRomeo.mass, cubeRomeoImpulse, cubeRomeoKinetic,
126             cubeJulia.position.x, cubeJulia.velocity.x,cubeJulia.mass, cubeJuliaImpulse,
127             velocityEnd, cubeKineticEnd, forceOnJulia, GesamtImpluls, ImpulsCheck ");
128
129         foreach (List<float> timeStep in timeSeriesInelasticCollision)
130         {
131             streamWriter.WriteLine(string.Join(", ", timeStep));
132             streamWriter.Flush();
133         }
134     }
135
136 void ChangeCubeTexture()
137 {
138     // the path of the image
139     filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";
140     // 1.read the bytes array
141     fileData = File.ReadAllBytes(filePath);
142     // 2.create a texture named tex
143     Texture2D tex = new Texture2D(2, 2);
144     // 3.load inside tx the bytes and use the correct image size
145     tex.LoadImage(fileData);
146     // 4.apply tex to material.mainTexture
147     GetComponent<Renderer>().material.mainTexture = tex;
148 }
149
150 void OnCollisionEnter(Collision collision)
151 {
152     if (collision.rigidbody != cubeJulia)
153     {
154         return;
155     }
156     if (collision.rigidbody == cubeJulia)
157     {
158         FixedJoint joint = gameObject.AddComponent<FixedJoint>();
159         ContactPoint[] contacts = new ContactPoint[collision.contactCount];
160         collision.GetContacts(contacts);
161         ContactPoint contact = contacts[0];
162         joint.anchor = transform.InverseTransformPoint(contact.point);
163         joint.connectedBody =

```

```

        collision.contacts[0].otherCollider.transform.GetComponent<Rigidbody>();

162
163
164     // Stops objects from continuing to collide and creating more joints
165     joint.enableCollision = false;
166 }
167 }
168 }
```

A.2 Code für die Datenaufbereitung des Elastischen Stosses des Lab 2

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Elastischen Stosses der Grafiken benötigt wurde.

```

1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4 df = pd.read_csv("../UnityProj/time_seriesElastic.csv")
5
6 plt.figure(figsize=(20,20))
7 plt.subplot(4,1,1)
8 plt.plot(df["currentTimeStep"], df[" cubeRomeo.position.x"])
9 plt.ylabel("[s] = m")
10 plt.xlabel("[t] = s")
11 plt.title("Ort als Funktion der Zeit")
12 plt.savefig('../Semesterprojekt Physik Engines/images/Elastisch/OrtAlsFunktionDerZeit.png',
13             dpi=300, bbox_inches='tight')
14 plt.show()
15
16 plt.subplot(4,1,2)
17 plt.plot(df["currentTimeStep"], df[" cubeRomeo.velocity.x"])
18 plt.ylabel("[v] = m/s")
19 plt.xlabel("[t] = s")
20 plt.title("Geschwindigkeit als Funktion der Zeit")
21 plt.savefig('../Semesterprojekt Physik
22             Engines/images/Elastisch/GeschwindigkeitAlsFunktionDerZeit.png', dpi=300,
23             bbox_inches='tight')
24 plt.show()
25 plt.subplot(4,1,3)
26 plt.plot(df["currentTimeStep"], df[" cubeRomeoKinetic"])
27 plt.ylabel("[J] = Nm")
28 plt.xlabel("[t] = s")
29 plt.title("Kinetische Energie als Funktion der Zeit")
30 plt.show()
31
32 plt.subplot(4,1,4)
33 plt.plot(df["currentTimeStep"], df[" springPotentialEnergy"])
34 plt.ylabel("[J] = Nm")
35 plt.xlabel("[t] = s")
36 plt.title("Potentielle Energie als Funktion der Zeit")
37
38 plt.show()
```

A.3 Code für die Datenaufbereitung des Inelastischen Stosses des Lab 2

Nachfolgend ist der Code abgebildet, welche für die Visuelle Datenaufbereitung des Inelastischen Stosses der Grafiken benötigt wurde.

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3
4
5 df = pd.read_csv("../UnityProj/time_seriesInelastic.csv")
6
7 plt.figure(figsize=(20,20))
8 plt.subplot(4,1,1)
9 plt.plot(df["cubeJuliaTimeStep"], df[" GesamtImpluls"])
10 plt.ylabel("[p] = Ns")
11 plt.xlabel("[t] = s")
12 plt.xlim(0,20)
13 plt.title("Gesamtimpuls als Funktion der Zeit")
14 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/GesamtImpluls.png',
15             dpi=300, bbox_inches='tight')
16 plt.show()
17
18 plt.figure(figsize=(20,20))
19 plt.subplot(4,1,1)
20 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.position.x"])
21 plt.ylabel("[s] = m")
22 plt.xlabel("[t] = s")
23 plt.xlim(15,18)
24 plt.title("Ort Julia als Funktion der Zeit")
25 plt.savefig('../Semesterprojekt Physik
Engines/images/Inelastisch/OrtJuliaAlsFunktionDerZeit.png', dpi=300, bbox_inches='tight')
26 plt.show()
27 plt.subplot(4,1,2)
28 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeo.velocity.x"])
29 plt.ylabel("[v] = m/s")
30 plt.xlabel("[t] = s")
31 plt.title("Geschwindigkeit Romeo als Funktion der Zeit")
32 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/GeschwindigkeitRomeo.png',
            dpi=300, bbox_inches='tight')
33 plt.show()
34
35 plt.subplot(4,1,3)
36 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.velocity.x"])
37 plt.ylabel("[v] = m/s")
38 plt.xlabel("[t] = s")
39 plt.title("Geschwindigkeit Julia als Funktion der Zeit")
40 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/GeschwindigkeitJulia.png',
            dpi=300, bbox_inches='tight')
41 plt.show()
42
43 plt.subplot(4,1,4)
44 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoKinetic"])
45 plt.ylabel("[J] = Nm")
46 plt.xlabel("[t] = s")
47 plt.title("Energie Romeo als Funktion der Zeit")
48
49 plt.show()
50
51 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJuliaImpulse"])
52 plt.ylabel("[p] = Ns")
53 plt.xlabel("[t] = s")
```

```

54 plt.title("Impuls Julia als Funktion der Zeit")
55 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/ImpulsJulia.png', dpi=300,
56         bbox_inches='tight')
57 plt.show()
58
59 plt.figure(figsize=(20,20))
60 plt.subplot(4,1,1)
61 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoImpulse"])
62 plt.ylabel("[p] = Ns")
63 plt.xlabel("[t] = s")
64 plt.title("Impuls Romeo als Funktion der Zeit")
65 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/ImpulsRomeo.png', dpi=300,
66         bbox_inches='tight')
67 plt.show()
68
69 plt.subplot(4,1,2)
70 plt.plot(df["cubeJuliaTimeStep"], df[" velocityEnd"])
71 plt.ylabel("[v] = m/s")
72 plt.xlabel("[t] = s")
73 plt.title("Endgeschwindigkeit als Funktion der Zeit")
74 plt.show()
75
76 plt.subplot(4,1,3)
77 plt.plot(df["cubeJuliaTimeStep"], df[" cubeKineticEnd"])
78 plt.ylabel("[J] = Nm")
79 plt.xlabel("[t] = s")
80 plt.title("Endkinetik als Funktion der Zeit")
81 plt.show()
82 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/cubeKineticEnd.png',
83             dpi=300, bbox_inches='tight')
84
85 plt.subplot(4,1,4)
86 plt.plot(df["cubeJuliaTimeStep"], df[" forceOnJulia"])
87 plt.ylabel("N")
88 plt.xlabel("[t] = s")
89 plt.title("Kraft auf Julia als Funktion der Zeit")
90 plt.savefig('../Semesterprojekt Physik Engines/images/Inelastisch/forceOnJulia.png',
91             dpi=300, bbox_inches='tight')

```

Abbildungsverzeichnis

1	Beschleunigung des Würfels	4
2	Elastischer Zusammenstoss mit der Feder	4
3	Inelastischer zusammenstoss mit dem anderen Würfel	4
4	Experiment Übersicht	8
5	Einstellung Julia	9
6	Einstellung Romeo	9
7	Einstellung Feder	10
8	Einstellung Feder	10
9	Geschwindigkeit als Funktion der Zeit	13
10	Ort als Funktion der Zeit	13
11	GesamtImpluls als Funktion der Zeit	14
12	Impuls Julia als Funktion der Zeit	14
13	Impuls Romeo als Funktion der Zeit	15
14	Ort Julia als Funktion der Zeit	15
15	Geschwindigkeit Romeo als Funktion der Zeit	15
16	Geschwindigkeit Julia als Funktion der Zeit	16
17	Endgeschwindigkeit als Funktion der Zeit	16

Literatur

- [1] Tipler, Paul A. u.a. *Physik Für Studierende Der Naturwissenschaften Und Technik*. Springer Spektrum. ISBN: 978-3-662-58280-0.