

Semesterprojekt Physik Engines

Kim Lan Vu, Michel Steiner, Asha Schwegler

22. April 2023

Inhaltsverzeichnis

1	Zusammenfassung	3
2	Aufbau des Experiments	3
3	Physikalische Beschreibung der einzelnen Vorgänge	4
3.1	Raketenantrieb	4
3.2	Elastischer Stoss	4
3.3	Inelastischer Stoss	4
3.4		4
4	Beschreibung der Implentierung inklusive Screenshots aus Unity	4
5	Rückblick und Lehren aus dem Versuch	4
6	Resultate mit grafischer Darstellung	4
6.1	Elastisch	5
6.2	Inelastisch	6
7	Code	8
7.1	Code für das Experiment	8
7.2	Code für die Datenaufbereitung	11
A	Anhang	12

1 Zusammenfassung

2 Aufbau des Experiments

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 linke Würfel Julia und der Rechte Romeo benannt. Daneben existiert eine Feder die horizontal an einer Wand befestigt ist. Bei dem gesamten Experimentes wird der Reibungswiederstand ignoriert. Ablauf des Experimentes:

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

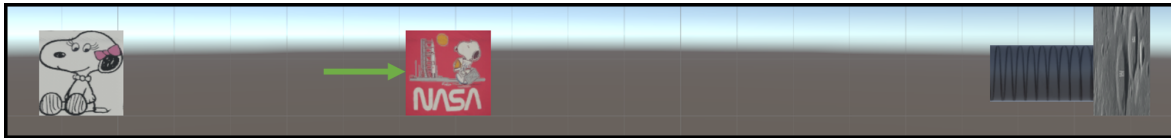


Abbildung 1: Beschleunigung des Würfels

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.

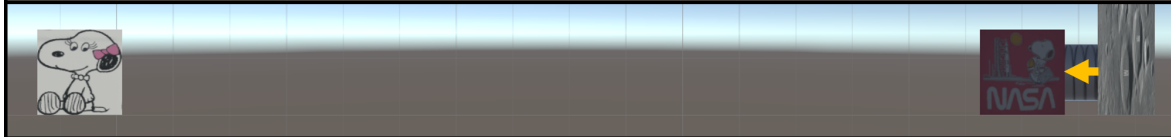


Abbildung 2: Elastischer Zusammenstoß mit der Feder

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 aufgeteilten Energie (blaue Pfeile in Abbildung 3) weiter nach links.

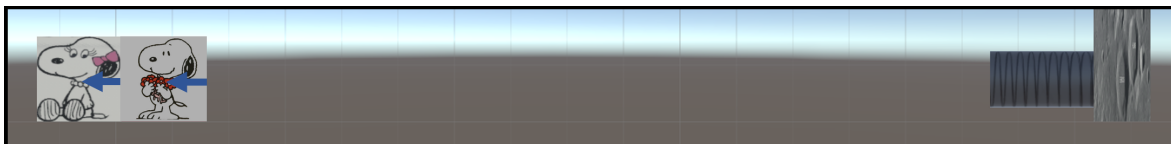


Abbildung 3: Inelastischer zusammenstoß mit dem anderen Würfel

3 Physikalische Beschreibung der einzelnen Vorgänge

In diesem Kapitel werden die physikalischen Vorgänge des Versuches beschrieben. Es geschehen drei Vorgänge, der Raketentrieb, einen elastischen Stoss und einen inelastischen Stoss. Die gegebenen Massen sind:

- Gewicht(m) = 2kg
- Velocity(v) = 2m/s
- Würfelseite = 1.5m

3.1 Raketenantrieb

Um die Kraft des Raketenantriebs zu berechnen nehmen wir die gewünschte Geschwindigkeit und berechnen damit die Beschleunigung, a. Da Kraft:

$$F = m * a.$$

Um dieses Anfangwertproblems zu lösen nehmen wir die Formel

$$\dot{v} = a$$

$$2m * s^{-1} \rightarrow -2m * s^{-2} \rightarrow a = \left[\frac{2m}{s^2}\right]$$

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

4N werden deshalb als konstante Kraft angewendet, damit auch die gewünschte Geschwindigkeit erreicht wird.

3.2 Elastischer Stoss

Beim elastischen Stoss ist die kinetische Energie vom Stosspartner vor und nach der Kollision gleich. Kinetische Energie wird mit folgender Formel berechnet:

$$\frac{1}{2} * m * v^2$$

Setzt man die Massen von diesem Projekt ein bekommt man:

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

3.3 Inelastischer Stoss

3.4

4 Beschreibung der Implementierung inklusive Screenshots aus Unity

5 Rückblick und Lehren aus dem Versuch

6 Resultate mit grafischer Darstellung

Während dem Durchlauf des Experimentes werden diverse Daten zu allen im Experiment vorkommenden Objekten gesammelt. Diese Daten umfassen Geschwindigkeit, Kinetische, sowie Potentielle Energie. Dabei wird zwischen dem Elastischen sowie Inelastischen Zusammentoss unterschieden.

6.1 Elastisch

Nachfolgend werden alle Daten als Funktion der Zeit zu dem Elastischen Zusammenstoss in den Abbildungen Abbildung 4 bis Abbildung 7 aufgegliedert.

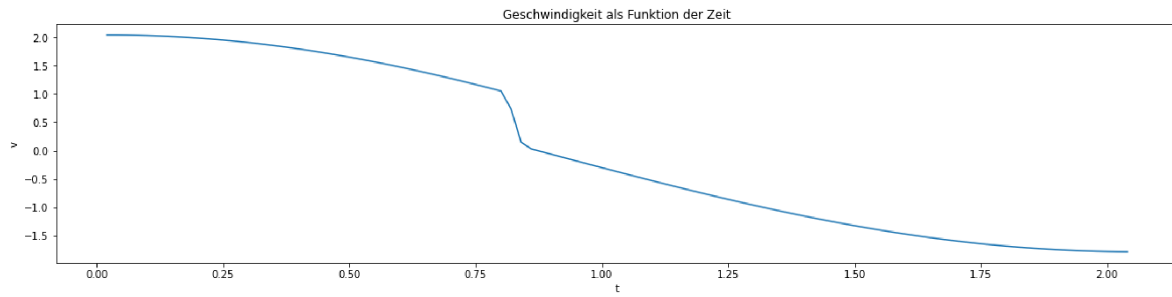


Abbildung 4: Geschwindigkeit als Funktion der Zeit

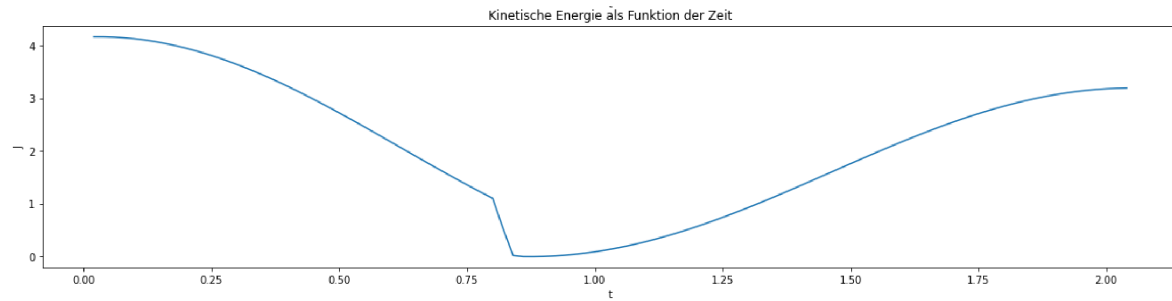


Abbildung 5: Kinetische Energie als Funktion der Zeit

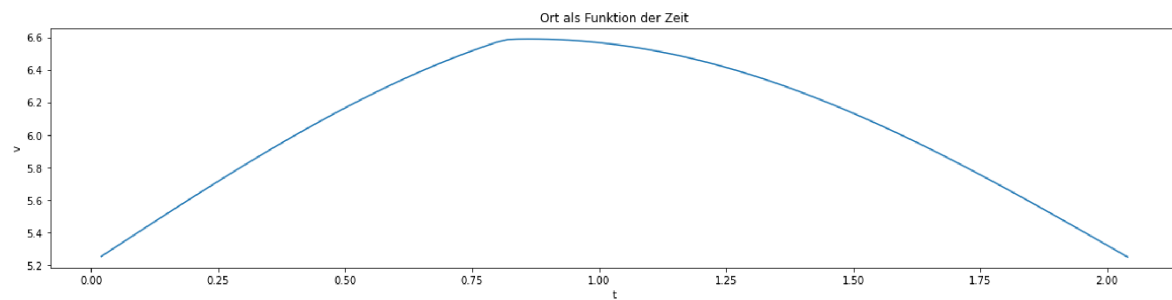


Abbildung 6: Ort als Funktion der Zeit

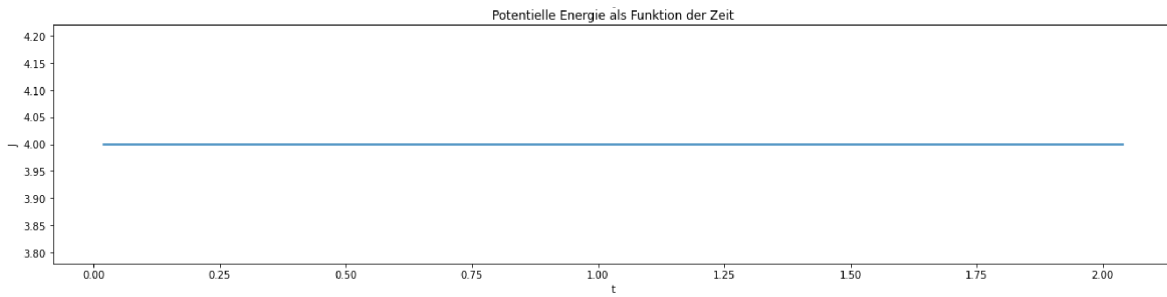


Abbildung 7: Potentielle Energie als Funktion der Zeit

6.2 Inelastisch

Nachfolgend werden alle Daten als Funktion der Zeit zu dem Inelastischen Zusammenstoss in den Abbildungen Abbildung 8 bis Abbildung 11 aufgegliedert.

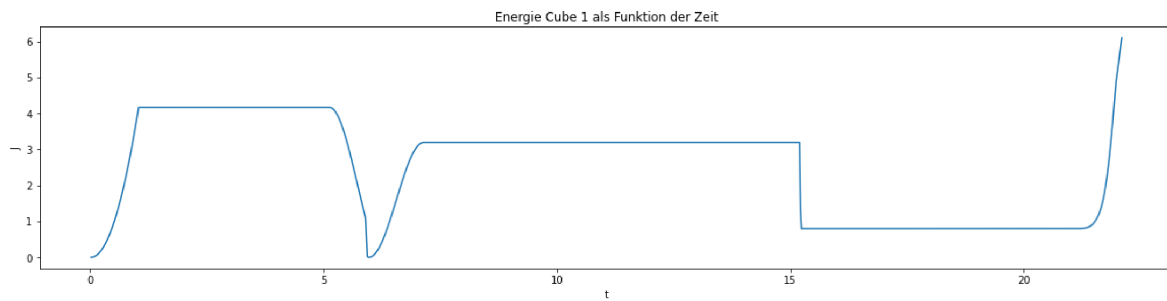


Abbildung 8: Energie Cube 1 als Funktion der Zeit

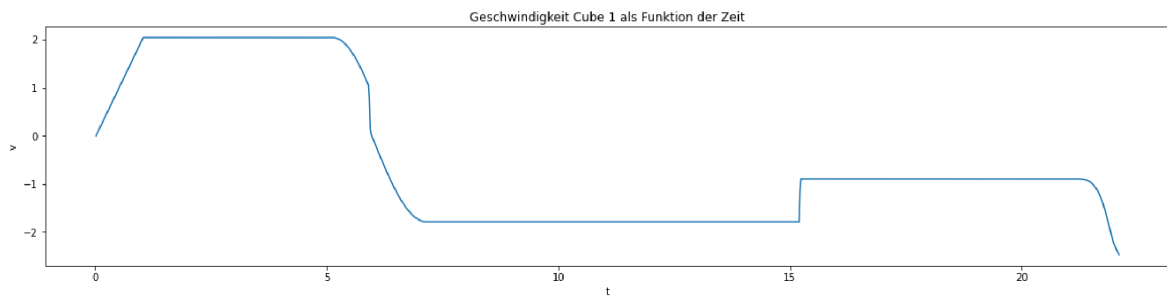


Abbildung 9: Geschwindigkeit Cube 1 als Funktion der Zeit

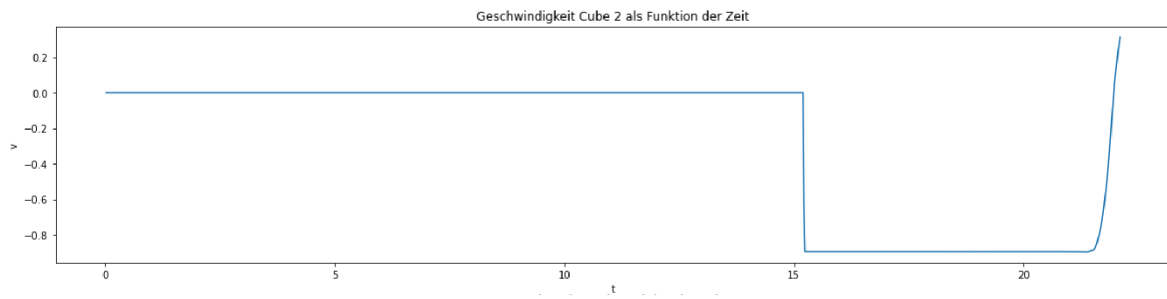


Abbildung 10: Geschwindigkeit Cube 2 als Funktion der Zeit

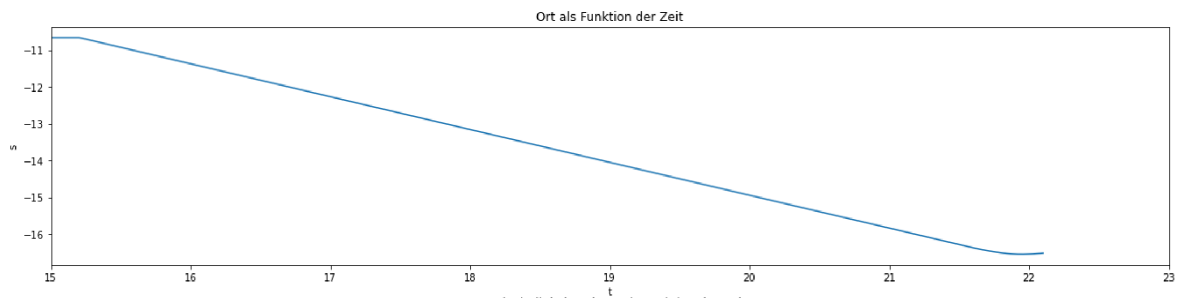


Abbildung 11: Ort als Funktion der Zeit

7 Code

7.1 Code für das Experiment

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 GesamtImpuls = 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 = 1.3f;
36
37    // Start is called before the first frame update
38    void Start()
39    {
40        starttime = Time.fixedTimeAsDouble;
41
42        timeSeriesElasticCollision = new List<List<float>>();
43        timeSeriesInelasticCollision = new List<List<float>>();
44
45
46        springMaxDeviation = spring.transform.position.x -
47            spring.transform.localScale.y/2; //Maximale Auslenkung gerechnet anhand
48            der linken Seite des Feders
49
50        springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
```



```

(Math.Pow(springContraction, 2.0))); // Energieerhaltungsgesetz
kinEnergie = PotEnergie :  $1/2 * m * v^2 = 1/2k * x^2$ 
49
50 //Maximale Auslenkung gerechnet anhand der linken seite des Feders
51 springMaxDeviation = spring.transform.position.x -
    spring.transform.localScale.y;
52 // Energieerhaltungsgesetz kinEnergie = PotEnergie :  $1/2 * m * v^2 = 1/2k * x^2$ 
53 springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
    (Math.Pow(springContraction, 2.0)));
54
55 }
56
57 // Update is called once per frame
58 void Update()
59 {
60 }
61 // FixedUpdate can be called multiple times per frame
62 void FixedUpdate()
63 {
64     double currentTime = Time.fixedTimeAsDouble-starttime;
65
66     if (accelarationTime >= currentTime)
67     {
68         constantForce = 4f;
69         cubeRomeo.AddForce(new Vector3(constantForce, 0f, 0f));
70     }
71
72     cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
        Math.Pow(cubeRomeo.velocity.x, 2.0))); //  $1/2 * m * v^2$ 
73
74
75     float collisionPosition = cubeRomeo.transform.position.x +
        cubeRomeo.transform.localScale.x / 2;
76
77     if (collisionPosition >= springMaxDeviation)
78     {
79         float springForceX = (collisionPosition - springMaxDeviation) *
            -springConstant;
80         springPotentialEnergy =(float)(0.5 * springConstant *
            Math.Pow(collisionPosition - springMaxDeviation, 2.0));
81         cubeRomeo.AddForce(new Vector3(springForceX, 0f, 0f));
82         ChangeCubeTexture();
83         currentTimeStep += Time.deltaTime;
84         timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
            cubeRomeo.position.x, cubeRomeo.velocity.x, springPotentialEnergy,
            cubeRomeoKinetic, springForceX });
85     }
86
87 //  $1/2 * m * v^2$ 
88 cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
        Math.Pow(cubeRomeo.velocity.x, 2.0)));
89 cubeRomeoImpulse = Math.Abs(cubeRomeo.mass * cubeRomeo.velocity.x);
90 cubeJuliaImpulse = Math.Abs(cubeJulia.mass * cubeJulia.velocity.x);
91 GesamtImpluls = cubeJuliaImpulse + cubeRomeoImpulse;
92 velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (cubeRomeo.mass +
        cubeJulia.mass);

```

```

93     ImpulsCheck = (cubeRomeo.mass + cubeJulia.mass )* velocityEnd;
94     cubeKineticEnd = Math.Abs((float)(0.5 * (cubeRomeo.mass + cubeJulia.mass) *
95         Math.Pow(velocityEnd, 2.0)));
96     forceOnJulia = Math.Abs(cubeJulia.mass * velocityEnd - cubeJulia.velocity.x);
97
98     cubeJuliaTimeStep += Time.deltaTime;
99     timeSeriesInelasticCollision.Add(new List<float>() { cubeJuliaTimeStep,
100         cubeRomeo.position.x, cubeRomeo.velocity.x, cubeRomeo.mass,
101         cubeRomeoImpulse, cubeRomeoKinetic, cubeJulia.position.x,
102         cubeJulia.velocity.x, cubeJulia.mass, cubeJuliaImpulse, velocityEnd,
103         cubeKineticEnd, forceOnJulia, GesamtImpuls, ImpulsCheck });
104 }
105 void OnApplicationQuit()
106 {
107     WriteElasticTimeSeriesToCsv();
108     WriteInelasticTimeSeriesToCsv();
109 }
110 void WriteElasticTimeSeriesToCsv()
111 {
112     using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
113     {
114         streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
115             cubeRomeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic,
116             springForceX");
117
118         foreach (List<float> timeStep in timeSeriesElasticCollision)
119         {
120             streamWriter.WriteLine(string.Join(",", timeStep));
121             streamWriter.Flush();
122         }
123     }
124 }
125 void WriteInelasticTimeSeriesToCsv()
126 {
127     using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))
128     {
129         streamWriter.WriteLine("cubeJuliaTimeStep, cubeRomeo.position.x,
130             cubeRomeo.velocity.x, cubeRomeo.mass, cubeRomeoImpulse,
131             cubeRomeoKinetic, cubeJulia.position.x,
132             cubeJulia.velocity.x, cubeJulia.mass, cubeJuliaImpulse, velocityEnd,
133             cubeKineticEnd, forceOnJulia, GesamtImpuls, ImpulsCheck ");
134
135         foreach (List<float> timeStep in timeSeriesInelasticCollision)
136         {
137             streamWriter.WriteLine(string.Join(",", timeStep));
138             streamWriter.Flush();
139         }
140     }
141 }
142 void ChangeCubeTexture()
143 {
144     // the path of the image
145     filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";

```

```

138     // 1.read the bytes array
139     fileData = File.ReadAllBytes(filePath);
140     // 2.create a texture named tex
141     Texture2D tex = new Texture2D(2, 2);
142     // 3.load inside tx the bytes and use the correct image size
143     tex.LoadImage(fileData);
144     // 4.apply tex to material.mainTexture
145     GetComponent<Renderer>().material.mainTexture = tex;
146 }
147
148 void OnCollisionEnter(Collision collision)
149 {
150     if (collision.rigidbody != cubeJulia)
151     {
152         return;
153     }
154     if (collision.rigidbody == cubeJulia)
155     {
156         FixedJoint joint = gameObject.AddComponent<FixedJoint>();
157         ContactPoint[] contacts = new ContactPoint[collision.contactCount];
158         collision.GetContacts(contacts);
159         ContactPoint contact = contacts[0];
160         joint.anchor = transform.InverseTransformPoint(contact.point);
161         joint.connectedBody =
            collision.contacts[0].otherCollider.transform.GetComponent<Rigidbody>();
162
163         // Stops objects from continuing to collide and creating more joints
164         joint.enableCollision = false;
165     }
166 }
167 }
168 }

```

7.2 Code für die Datenaufbereitung

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

```

1  # -*- coding: utf-8 -*-
2  """
3  Created on Thu Apr 20 21:06:43 2023
4
5  @author: Asha
6  """
7  import pandas as pd
8  import matplotlib.pyplot as plt
9
10
11  df =
12      pd.read_csv("C:/Users/ashas/ZHAW_Code/PE_Lab/UnityProj/TimeSeries/time_seriesInelastic.csv")
13
14  plt.figure(figsize=(20,20))
15  plt.subplot(4,1,1)
16  plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.position.x"])
17  plt.ylabel("s")

```

```

17 plt.xlabel("t")
18 plt.xlim(15,23)
19 plt.title("Ort als Funktion der Zeit")
20
21
22 plt.subplot(4,1,2)
23 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeo.velocity.x"])
24 plt.ylabel("v")
25 plt.xlabel("t")
26 plt.title("Geschwindigkeit Cube 1 als Funktion der Zeit")
27
28
29 plt.subplot(4,1,3)
30 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.velocity.x"])
31 plt.ylabel("v")
32 plt.xlabel("t")
33 plt.title("Geschwindigkeit Cube 2 als Funktion der Zeit")
34
35
36 plt.subplot(4,1,4)
37 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoKinetic"])
38 plt.ylabel("J")
39 plt.xlabel("t")
40 plt.title("Energie Cube 1 als Funktion der Zeit")
41
42 plt.show()
43
44 plt.plot(df["cubeJuliaTimeStep"], df[" cubeJuliaImpulse"])
45 plt.ylabel("Ns")
46 plt.xlabel("t")
47 plt.title("Impuls Cube 2 als Funktion der Zeit")
48
49
50 plt.figure(figsize=(20,20))
51 plt.subplot(4,1,1)
52 plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoImpulse"])
53 plt.ylabel("Ns")
54 plt.xlabel("t")
55 plt.title("Impuls Cube 1 als Funktion der Zeit")
56
57 plt.subplot(4,1,2)
58 plt.plot(df["cubeJuliaTimeStep"], df[" velocityEnd"])
59 plt.ylabel("s")
60 plt.xlabel("t")
61 plt.title("Endgeschwindigkeit als Funktion der Zeit")
62
63
64 plt.subplot(4,1,3)
65 plt.plot(df["cubeJuliaTimeStep"], df[" cubeKineticEnd"])
66 plt.ylabel("J")
67 plt.xlabel("t")
68 plt.title("Endkinetik als Funktion der Zeit")
69
70
71 plt.subplot(4,1,4)
72 plt.plot(df["cubeJuliaTimeStep"], df[" forceOnJulia "])

```

```

73 plt.ylabel("N")
74 plt.xlabel("t")
75 plt.title("Kraft auf Cube 2 als Funktion der Zeit")
76 plt.show()

```

A Anhang

test

Abbildungsverzeichnis

1	Beschleunigung des Würfels	3
2	Elastischer Zusammenstoss mit der Feder	3
3	Inelastischer zusammenstoss mit dem anderen Würfel	3
4	Geschwindigkeit als Funktion der Zeit	5
5	Kinetische Energie als Funktion der Zeit	5
6	Ort als Funktion der Zeit	5
7	Potentielle Energie als Funktion der Zeit	6
8	Energie Cube 1 als Funktion der Zeit	6
9	Geschwindigkeit Cube 1 als Funktion der Zeit	6
10	Geschwindigkeit Cube 2 als Funktion der Zeit	7
11	Ort als Funktion der Zeit	7

Literatur

- [1] Catherine Brendow. *LibGuides: Zotero: Zotero and LaTeX*. URL: <https://libguides.graduateinstitute.ch/zotero/LaTeX> (besucht am 05.03.2023).