Semesterprojekt Physik Engines

Kim Lan Vu, Michel Steiner, Asha Schwegler

22. April 2023

Inhaltsverzeichnis

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 gesammten 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.

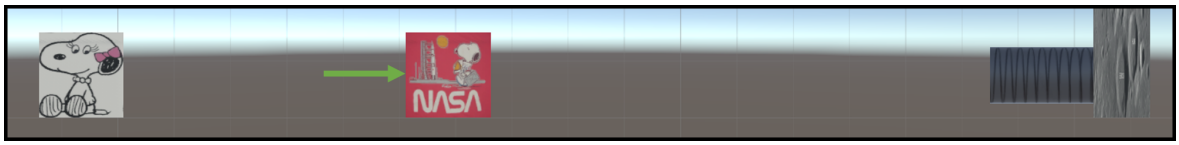

Abbildung 1: Beschleunigung des Würfels

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.

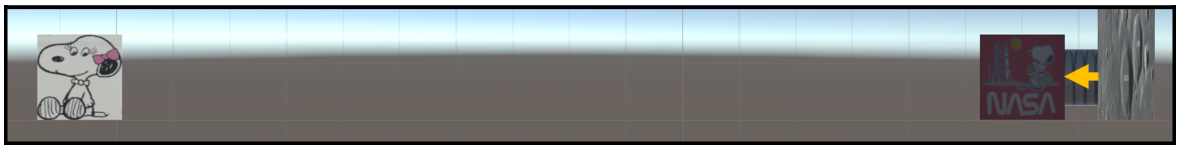

Abbildung 2: Elastischer Zusammenstoss mit der Feder

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 nacht links.

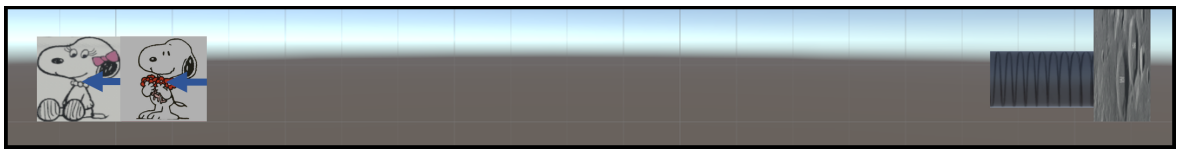

Abbildung 3: Inelastischer zusammenstoss mit dem anderen Würfel

3 Physikalische Beschreibung der einzelnen Vorgänge

In diesem Kapitel werden die physikalische 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
$\dot{v} = a$
$2m * s^{-1} \rightarrow -2m * s^{-2} \rightarrow a = [\frac{2m}{s^2}]$

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

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

3.3 Inelastischer Stoss

3.4

4 Beschreibung der Implentierung 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 Eleastischen 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.

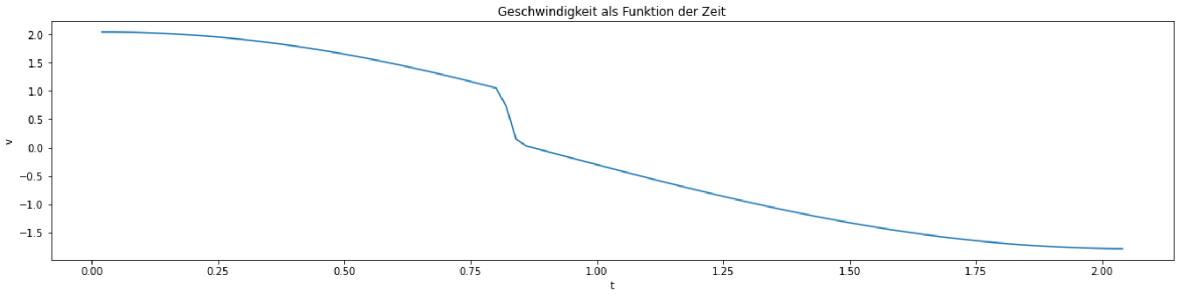

Abbildung 4: Geschwindigkeit als Funktion der Zeit

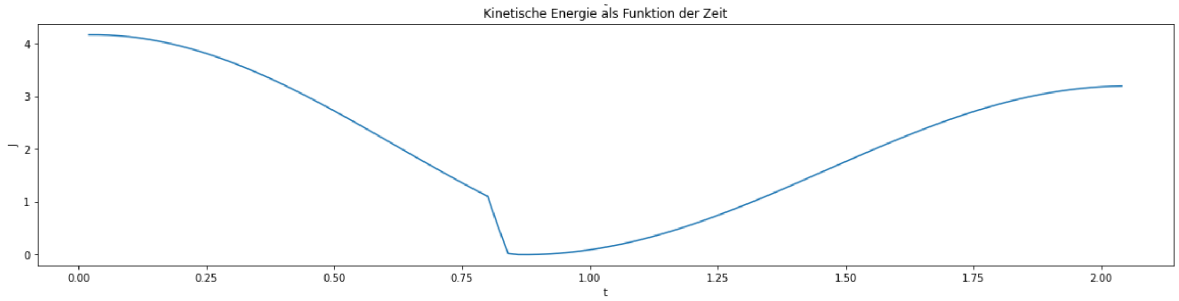

Abbildung 5: Kinetische Energie als Funktion der Zeit

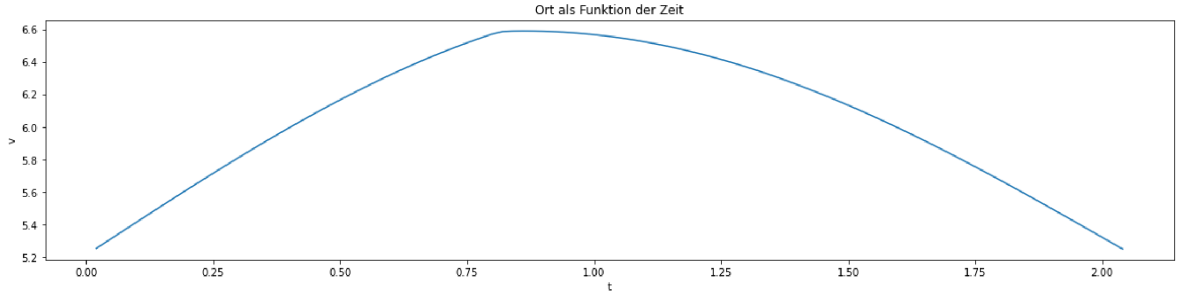

Abbildung 6: Ort als Funktion der Zeit

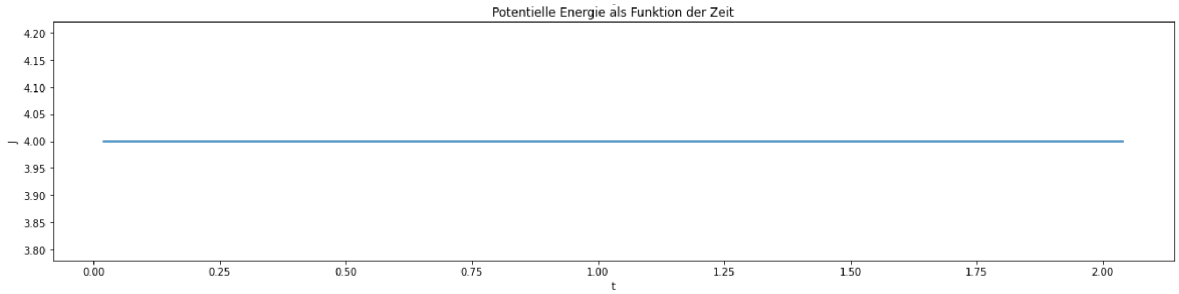

Abbildung 7: Potentielle Energie als Funktion der Zeit

6.2 Inelastisch

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

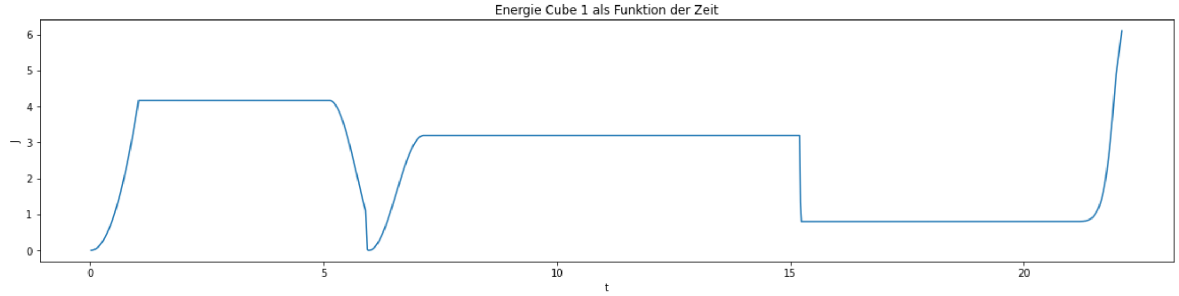

Abbildung 8: Energie Cube 1 als Funktion der Zeit

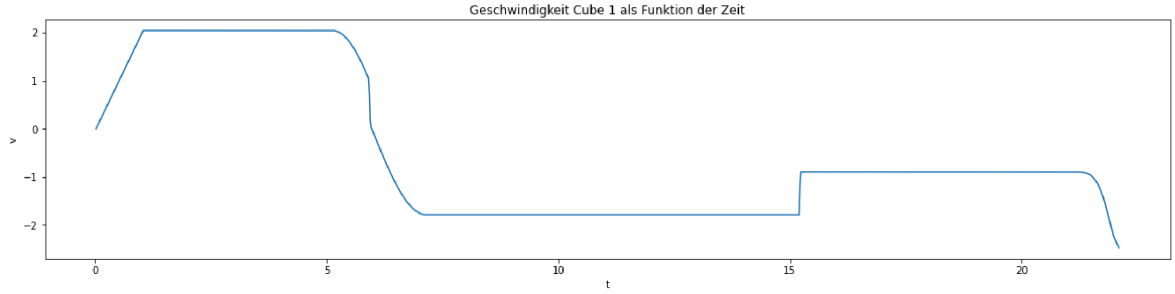

Abbildung 9: Geschwindigkeit Cube 1 als Funktion der Zeit

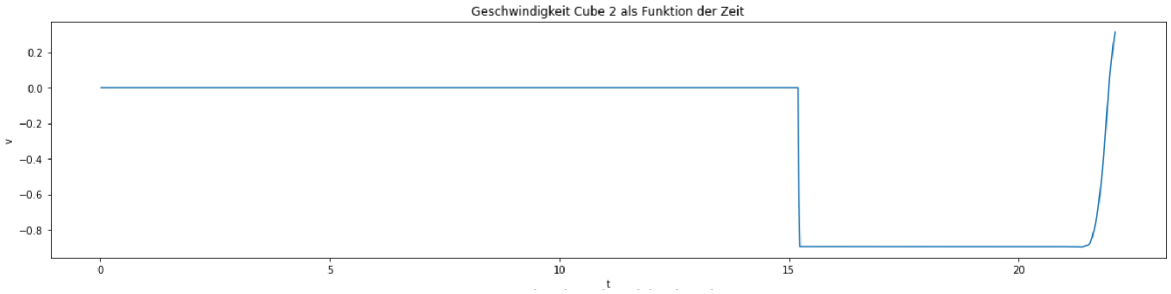

Abbildung 10: Geschwindigkeit Cube 2 als Funktion der Zeit

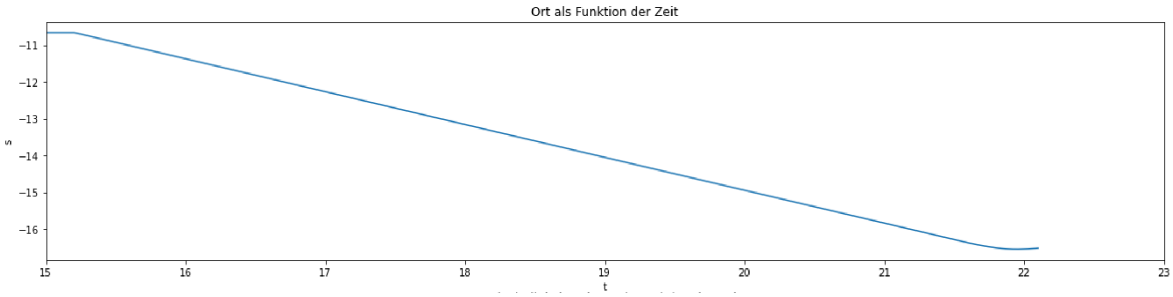

Abbildung 11: Ort als Funktion der Zeit

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.

```
using System;
using System.Collections.Generic;
using System.IO;
using UnityEngine;


public class CubeController : MonoBehaviour
{
    public Rigidbody cubeRomeo;
    public Rigidbody cubeJulia;
    public GameObject spring;

    private float currentTimeStep; // s
    private float cubeJuliaTimeStep;

    private List<List<float>> timeSeriesElasticCollision;
    private List<List<float>> timeSeriessInelasticCollision;

    private string filePath;
    private byte[] fileData;
    float springPotentialEnergy = 0f;
    float cubeRomeoKinetic = 0f;
    float cubeRomeoImpulse = 0f;
    float cubeJuliaImpulse = 0f;
    float GesamtImpluls = 0f;
    float ImpulsCheck = 0f;
    float forceOnJulia = 0f;
    float velocityEnd = 0f;
    float cubeKineticEnd = 0f;
    float constantForce = 4f;
    double startime = 0;
    double accelarationTime = 1.0;
    float springConstant = 0f;
    float springMaxDeviation = 0f;
    float springContraction = 1.3f;

    // Start is called before the first frame update
    void Start()
    {
        startime = Time.fixedTimeAsDouble;

        timeSeriesElasticCollision = new List<List<float>>();
        timeSeriessInelasticCollision = new List<List<float>>();

        //Maximale Auslenkung gerechnet anhand der linken seite des Feders
        springMaxDeviation = spring.transform.position.x -
            spring.transform.localScale.y/2;
        // Energieerhaltungsgesetz kinEnergie = PotEnergie : 1/2*m*v^2 = 1/2k * x^2
        springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
            (Math.Pow(springContraction, 2.0)));
```

```
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-startime;
65
66          if (accelarationTime >= currentTime)
67          {
68              constantForce = 4f;
69              cubeRomeo.AddForce(new Vector3(constantForce, 0f, 0f));
70          }
71
72          // 1/2*m*v^2
73          cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
                Math.Pow(cubeRomeo.velocity.x, 2.0)));
74
75
76          float collisionPosition = cubeRomeo.transform.position.x +
                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 - springMaxDeviation, 2.0));
82              cubeRomeo.AddForce(new Vector3(springForceX, 0f, 0f));
83              ChangeCubeTexture();
84              currentTimeStep += Time.deltaTime;
85              timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
                    cubeRomeo.position.x, cubeRomeo.velocity.x, springPotentialEnergy,
                    cubeRomeoKinetic, springForceX });
86          }
87
88          // 1/2*m*v^2
89          cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
                Math.Pow(cubeRomeo.velocity.x, 2.0)));
90          cubeRomeoImpulse = Math.Abs(cubeRomeo.mass * cubeRomeo.velocity.x);
91          cubeJuliaImpulse = Math.Abs(cubeJulia.mass * cubeJulia.velocity.x);
92          GesamtImpluls = cubeJuliaImpulse + cubeRomeoImpulse;
93          velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (cubeRomeo.mass +
                cubeJulia.mass);
94          ImpulsCheck = (cubeRomeo.mass + cubeJulia.mass )* velocityEnd;
```

```
95          cubeKineticEnd = Math.Abs((float)(0.5 * (cubeRomeo.mass + cubeJulia.mass) *
                Math.Pow(velocityEnd, 2.0)));
96          forceOnJulia = Math.Abs(cubeJulia.mass * velocityEnd - cubeJulia.velocity.x);
97
98          cubeJuliaTimeStep += Time.deltaTime;
99          timeSeriessInelasticCollision.Add(new List<float>() { cubeJuliaTimeStep,
                cubeRomeo.position.x, cubeRomeo.velocity.x,cubeRomeo.mass,
                cubeRomeoImpulse, cubeRomeoKinetic, cubeJulia.position.x,
                cubeJulia.velocity.x,cubeJulia.mass, cubeJuliaImpulse, velocityEnd,
                cubeKineticEnd, forceOnJulia, GesamtImpluls, ImpulsCheck });
100     }
101     void OnApplicationQuit()
102     {
103         WriteElasticTimeSeriesToCsv();
104         WriteInelasticTimeSeriesToCsv();
105     }
106     void WriteElasticTimeSeriesToCsv()
107     {
108         using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
109         {
110             streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
                    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
121     void WriteInelasticTimeSeriesToCsv()
122     {
123         using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))
124         {
125             streamWriter.WriteLine("cubeJuliaTimeStep, cubeRomeo.position.x,
                    cubeRomeo.velocity.x,cubeRomeo.mass, cubeRomeoImpulse,
                    cubeRomeoKinetic, cubeJulia.position.x,
                    cubeJulia.velocity.x,cubeJulia.mass, cubeJuliaImpulse, velocityEnd,
                    cubeKineticEnd, forceOnJulia, GesamtImpluls, ImpulsCheck }");
126
127             foreach (List<float> timeStep in timeSeriessInelasticCollision)
128             {
129                 streamWriter.WriteLine(string.Join(",", timeStep));
130                 streamWriter.Flush();
131             }
132         }
133     }
134
135     void ChangeCubeTexture()
136     {
137         // the path of the image
138         filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";
139         // 1.read the bytes array
```

```
        fileData = File.ReadAllBytes(filePath);
        // 2.create a texture named tex
        Texture2D tex = new Texture2D(2, 2);
        // 3.load inside tx the bytes and use the correct image size
        tex.LoadImage(fileData);
        // 4.apply tex to material.mainTexture
        GetComponent<Renderer>().material.mainTexture = tex;
    }

    void OnCollisionEnter(Collision collision)
    {
        if (collision.rigidbody != cubeJulia)
        {
            return;
        }
        if (collision.rigidbody == cubeJulia)
        {
            FixedJoint joint = gameObject.AddComponent<FixedJoint>();
            ContactPoint[] contacts = new ContactPoint[collision.contactCount];
            collision.GetContacts(contacts);
            ContactPoint contact = contacts[0];
            joint.anchor = transform.InverseTransformPoint(contact.point);
            joint.connectedBody =
                collision.contacts[0].otherCollider.transform.GetComponent<Rigidbody>();


            // Stops objects from continuing to collide and creating more joints
            joint.enableCollision = false;
        }
    }
}
```

7.2 Code für die Datenaufbereitung des Elastischen Stosses

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

```
import pandas as pd
import matplotlib.pyplot as plt

df = pd.read_csv("../UnityProj/TimeSeries/time_seriesElastic.csv")

plt.figure(figsize=(20,20))
plt.subplot(4,1,1)
plt.plot(df["currentTimeStep"], df[" cubeRomeo.position.x"])
plt.ylabel("v")
plt.xlabel("t")
plt.title("Ort als Funktion der Zeit")



plt.subplot(4,1,2)
plt.plot(df["currentTimeStep"], df[" cubeRomeo.velocity.x"])
plt.ylabel("v")
plt.xlabel("t")
```

```
plt.title("Geschwindigkeit als Funktion der Zeit")



plt.subplot(4,1,3)
plt.plot(df["currentTimeStep"], df[" cubeRomeoKinetic"])
plt.ylabel("J")
plt.xlabel("t")
plt.title("Kinetische Energie als Funktion der Zeit")


plt.subplot(4,1,4)
plt.plot(df["currentTimeStep"], df[" springPotentialEnergy"])
plt.ylabel("J")
plt.xlabel("t")
plt.title("Potentielle Energie als Funktion der Zeit")

plt.show()
```

7.3 Code für die Datenaufbereitung des Inelastischen Stosses

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

```
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_csv("../UnityProj/TimeSeries/time_seriesInelastic.csv")

plt.figure(figsize=(20,20))
plt.subplot(4,1,1)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.position.x"])
plt.ylabel("s")
plt.xlabel("t")
plt.xlim(15,23)
plt.title("Ort als Funktion der Zeit")


plt.subplot(4,1,2)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeo.velocity.x"])
plt.ylabel("v")
plt.xlabel("t")
plt.title("Geschwindigkeit Cube 1 als Funktion der Zeit")


plt.subplot(4,1,3)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeJulia.velocity.x"])
plt.ylabel("v")
plt.xlabel("t")
plt.title("Geschwindigkeit Cube 2 als Funktion der Zeit")


plt.subplot(4,1,4)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoKinetic"])
```

```
plt.ylabel("J")
plt.xlabel("t")
plt.title("Energie Cube 1 als Funktion der Zeit")

plt.show()

plt.plot(df["cubeJuliaTimeStep"], df[" cubeJuliaImpulse"])
plt.ylabel("Ns")
plt.xlabel("t")
plt.title("Impuls Cube 2 als Funktion der Zeit")


plt.figure(figsize=(20,20))
plt.subplot(4,1,1)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeRomeoImpulse"])
plt.ylabel("Ns")
plt.xlabel("t")
plt.title("Impuls Cube 1 als Funktion der Zeit")

plt.subplot(4,1,2)
plt.plot(df["cubeJuliaTimeStep"], df[" velocityEnd"])
plt.ylabel("s")
plt.xlabel("t")
plt.title("Endgeschwindigkeit als Funktion der Zeit")


plt.subplot(4,1,3)
plt.plot(df["cubeJuliaTimeStep"], df[" cubeKineticEnd"])
plt.ylabel("J")
plt.xlabel("t")
plt.title("Endkinetik als Funktion der Zeit")


plt.subplot(4,1,4)
plt.plot(df["cubeJuliaTimeStep"], df[" forceOnJulia "])
plt.ylabel("N")
plt.xlabel("t")
plt.title("Kraft auf Cube 2 als Funktion der Zeit")
plt.show()
```

A Anhang

test

Abbildungsverzeichnis

Literatur

[1] Catherine Brendow. *LibGuides: Zotero: Zotero and LaTeX*. URL: https://libguides.graduateinstitute.ch/zotero/LaTex (besucht am 05. 03. 2023).