Semesterprojekt Physik Engines

Kim Lan Vu, Michel Steiner, Asha Schwegler

22. April 2023

Inhaltsverzeichnis

1 Zusammenfassung

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 Reibungswiederstand ignoriert.
Ablauf des Experimentes:

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

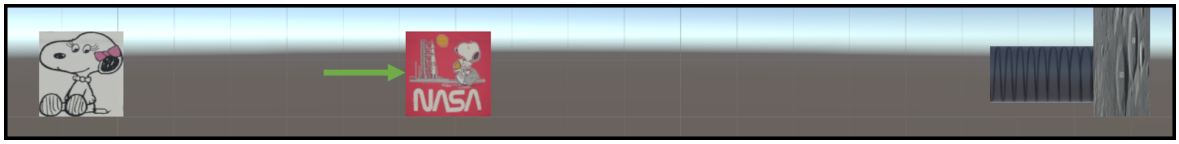

Abbildung 1: Beschleunigung des Würfels

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.

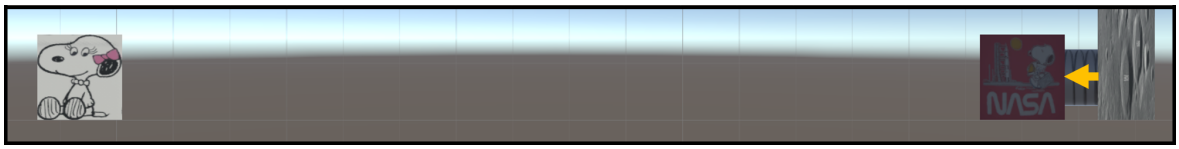

Abbildung 2: Elastischer Zusammenstoss mit der Feder

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 übertragener Energie (blaue Pfeile in Abbildung 3) weiter nacht links.

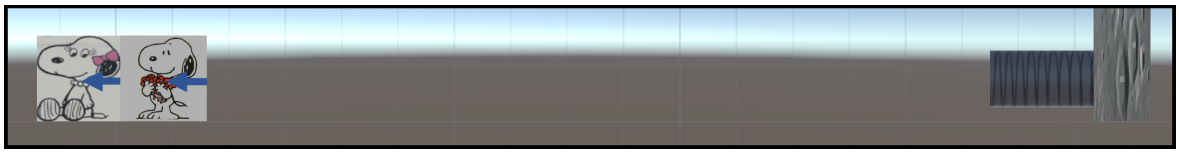

Abbildung 3: Inelastischer zusammenstoss mit dem anderen Würfel

3 Physikalische Beschreibung der einzelnen Vorgänge

In diesem Kapitel werden die physikalische Vorgänge des Versuches beschrieben. 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 eine 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 Anfangwertproblems zu lösen leiten wir die Geschwindigkeit ab [1]:

$$\dot{v} = a$$
$$2m * s^{-1} \rightarrow -2m * s^{-2} \rightarrow a = [\tfrac{2m}{s^2}]$$

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

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

Somit können wir nun die Kraft ausrechnen:

$$F = 2kg * \tfrac{2m}{s^2} => \tfrac{4kg*m}{s^2} = 4N$$

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 zusammenngedrückt wird. Sobald sie Romeo zurück stösst, 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 pontentielle Energie der Feder lautet[1]:

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

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

$$E_{kin_{Romeo}} = E_{pot_{Feder}}$$

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

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

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

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

$$\frac{2kg*(2m/s)^2}{1.3m^2} = 4.73N/m$$

Jetzt wo wir die Federkonstante und Länge haben, können wir einen langsamen 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 Impulserhaltungsatz 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 Implentierung inklusive Screenshots aus Unity

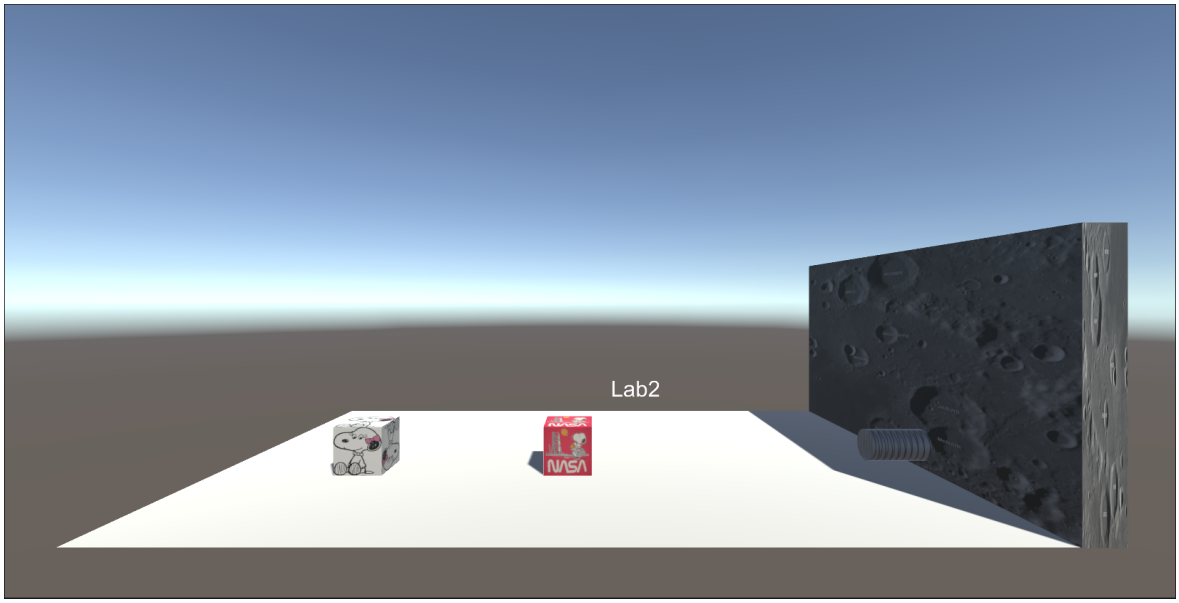

Abbildung 4: Experiment Übersicht

Im Code wird der Würfel, welcher am Anfang beschleunigt wird, Romeo benannt und der andere Würfel entsprechend Julia. Die Würfel werden in den nächsten Abschnitten ebenfalls so bezeichnet, damit man denn Zusammenhang mit den Screenshots des Codes leichter versteht. Zur Kontrolle der Werte und Grafik Erstellung wurde zwei verschiedene CSV Datei erstellt (timeseries genannt), eine für den elastischen Stoss relevanten Werte und eine für den inelastischen Stoss.
Für die Beschleunigung von Romeo wird gemäss Berechnung, eine konstante Kraft von 4 (Newton) im FixedUpdate hinzugefügt. Die Beschleunigunszeit von einer Sekunde wird von den Autoren bestimmt. Der Würfel braucht also eine Sekunde um die maximale Geschwindigkeit zu erreichen. Die konstante Kraft wirkt solange bis die Geschwindigkeit von 2m/s erreicht. Dabei wird Romeo wird dann auch die kinetische Energie berechnet und in beiden Timeseries notiert.
Beim Teil mit dem elastischen Stoss wurde ein Feder GameObject hinzugefügt. Die Federkonstante wird am Start berechnet bevor sich Romeo bewegt. Dies wird wie in der Berechnung erwähnt das Energieerhaltungsgesetz angewendet und für die Federstauchung wurde 1.3 (Meter) gewählt, sodass der berechnete Federkonstante Wert 4.733 (N/m) beträgt. Beim Start wird ebenfalls die Position der Feder in der Ruhelage, dort welche Romeo die Feder zuerst berühren würde, in der Variable springMaxDeviation gespeichert.
Diese wird in FixedUpdate gebraucht um zu überprüfen, ob Romeo bereits auf die Feder eintrifft. Sobald dies der Fall ist, verändert sich die Texture von Romeo und dann wird jeweils die Federkraft darin berechnet und an Romeo hinzugefügt. Dafür wird der Längenunterschied der Feder mit der Differenz aus der Position von Romeo und springMaxDeviation berechnet und danach mit der negativen Federkonstante multipliziert. Dadurch bewegt sich Romeo zurückt. Als Überprüfung der Energieerhaltung wird schlussendlich auch die potentiele Energie der Feder berechnet und in der elastischen Timeseries zusammen mit der Federkraft aufgeschrieben.
Für den inelastischen Stoss mit Julia wird im OnCollisionEnter jeweils geprüft, ob es sich bei der Kollision um Julia handelt. Nur wenn dies der Fall ist, wird dann ein FixedJoint an den Berührungspunkte zwischen Romeo und Julia hinzugefügt um die damit sie zusammenkleben. Zum Schluss muss noch .enableCollision auf false gesetzt werden, damit sich die beiden Würfeln nicht mehr kollidieren.
Im FixedUpdate wird dann für die inelastische Timeseries die Impulse berechnet und die gemeinsame Endgeschwindigkeit und kinetische Energie. Mit dieser Geschwindigkeit kann dann auch die Kraft

welche auf Julia wirkt errechnet werden.

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 Vorgängen 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 5 bis Abbildung 6 aufgegliedert.

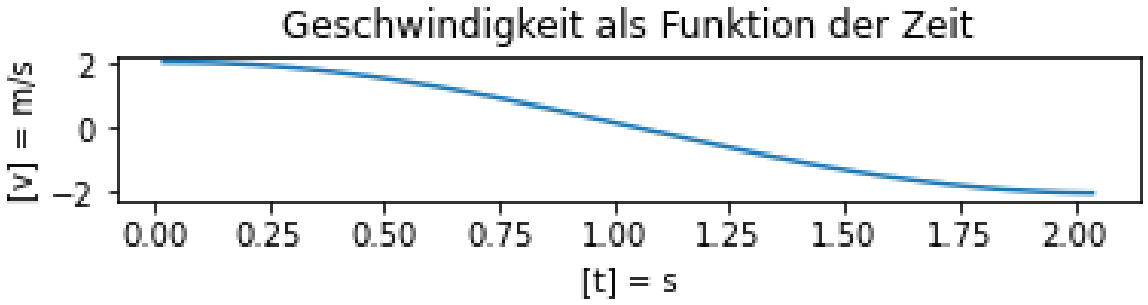

Abbildung 5: Geschwindigkeit als Funktion der Zeit

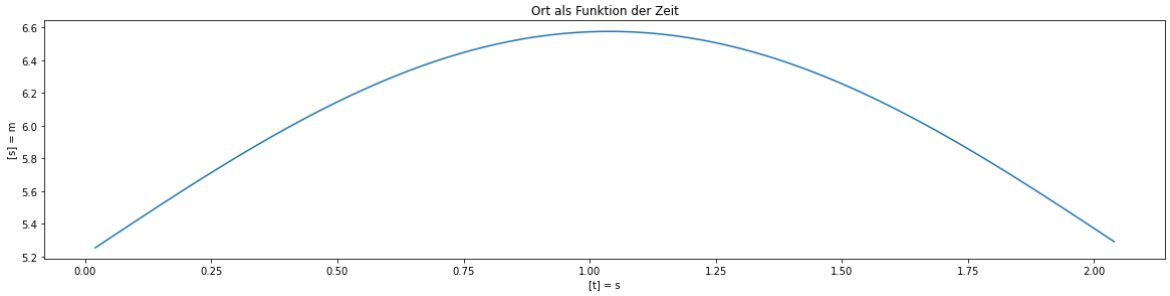

Abbildung 6: Ort als Funktion der Zeit

6.1.2 Inelastisch

Nachfolgend werden alle Daten als Funktion der Zeit zu dem inelastischen Zusammenstoss in der Abbildung 7 bis Abbildung 13 aufgegliedert.

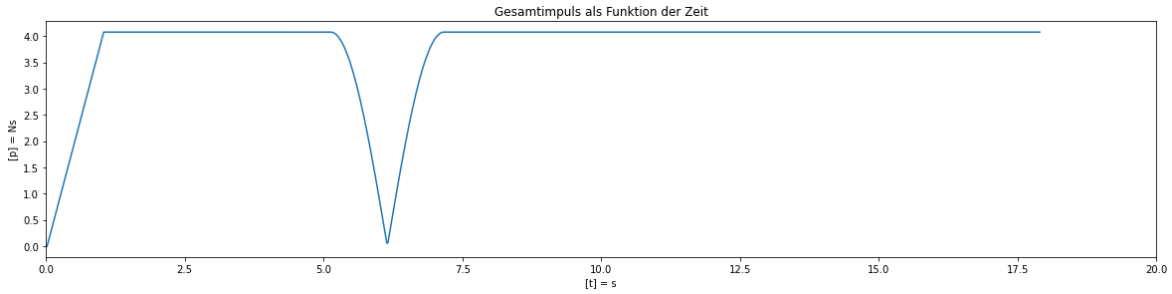

Abbildung 7: GesamtImpluls als Funktion der Zeit

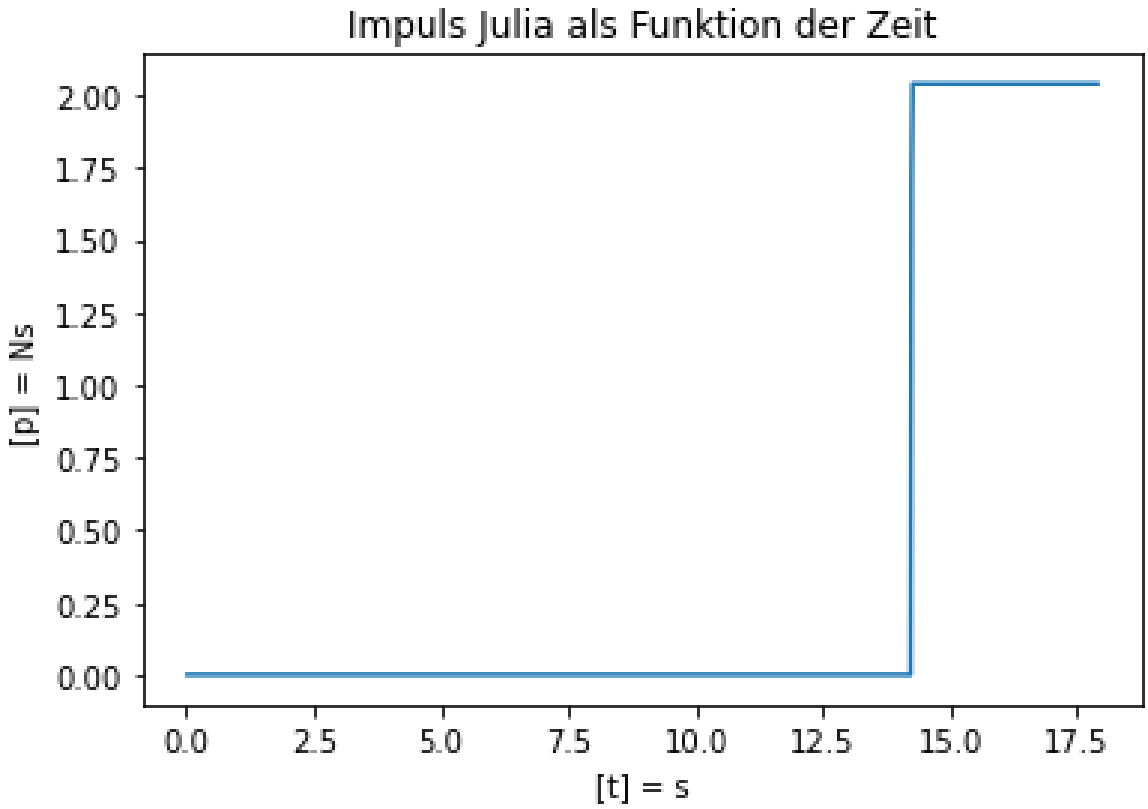

Abbildung 8: Impuls Julia als Funktion der Zeit

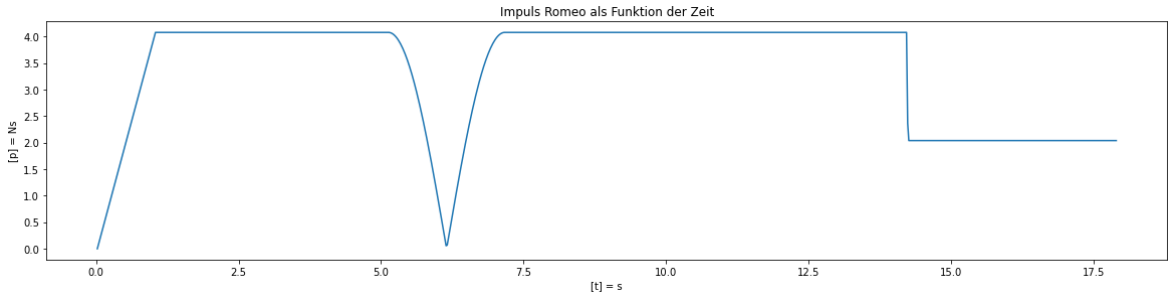

Abbildung 9: Impuls Romeo als Funktion der Zeit

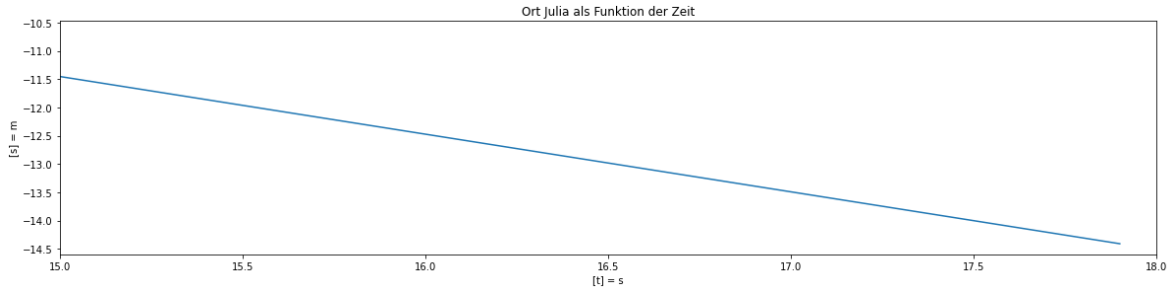

Abbildung 10: Ort Julia als Funktion der Zeit

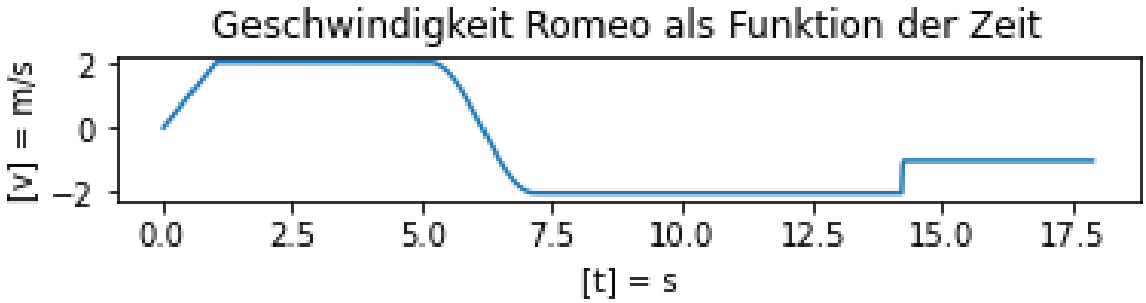

Abbildung 11: Geschwindigkeit Romeo als Funktion der Zeit

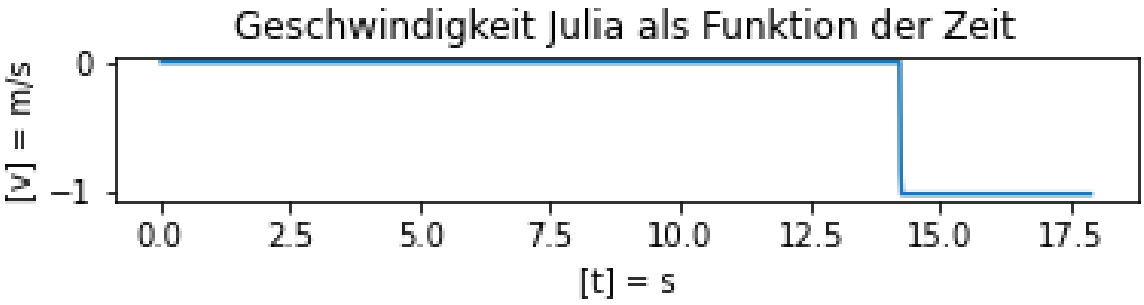

Abbildung 12: Geschwindigkeit Julia als Funktion der Zeit

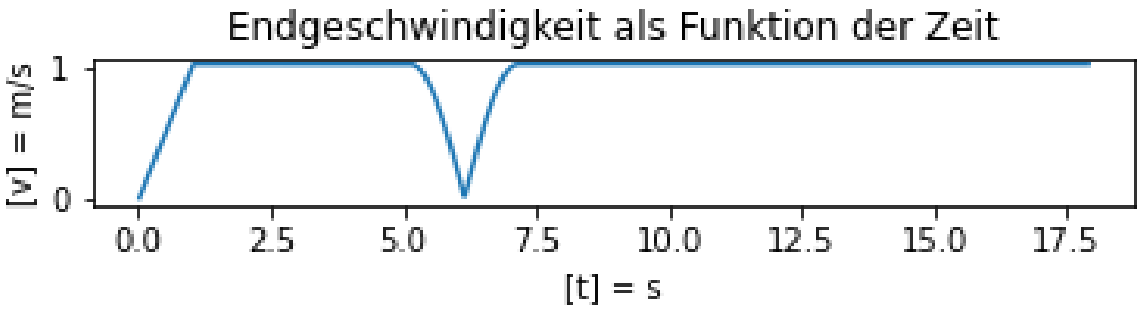

Abbildung 13: Endgeschwindigkeit als Funktion der Zeit

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)));

        //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)));
```

```
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();
```

```
        WriteInelasticTimeSeriesToCsv();
    }
    void WriteElasticTimeSeriesToCsv()
    {
        using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
        {
            streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
                cubeRomeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic, springForceX");

            foreach (List<float> timeStep in timeSeriesElasticCollision)
            {
                streamWriter.WriteLine(string.Join(",", timeStep));
                streamWriter.Flush();
            }
        }

    }

    void WriteInelasticTimeSeriesToCsv()
    {
        using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))
        {
            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 }");

            foreach (List<float> timeStep in timeSeriessInelasticCollision)
            {
                streamWriter.WriteLine(string.Join(",", timeStep));
                streamWriter.Flush();
            }
        }
    }

    void ChangeCubeTexture()
    {
        // the path of the image
        filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";
        // 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/time_seriesElastic.csv")

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


plt.subplot(4,1,2)
plt.plot(df["currentTimeStep"], df[" cubeRomeo.velocity.x"])
plt.ylabel("[v] = m/s")
plt.xlabel("[t] = s")
plt.title("Geschwindigkeit als Funktion der Zeit")
plt.show()


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

plt.subplot(4,1,4)
plt.plot(df["currentTimeStep"], df[" springPotentialEnergy"])
plt.ylabel("[J] = Nm")
plt.xlabel("[t] = s")
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
```

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

```
plt.title("Endgeschwindigkeit als Funktion der Zeit")
plt.show()

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

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

Abbildungsverzeichnis

Literatur

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