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.

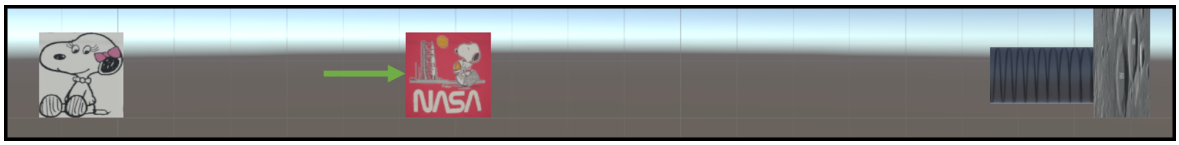

Abbildung 1: Beschleunigung des Würfels

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.

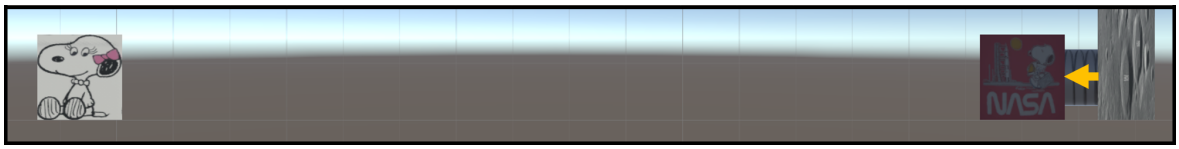

Abbildung 2: Elastischer Zusammenstoss mit der Feder

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.

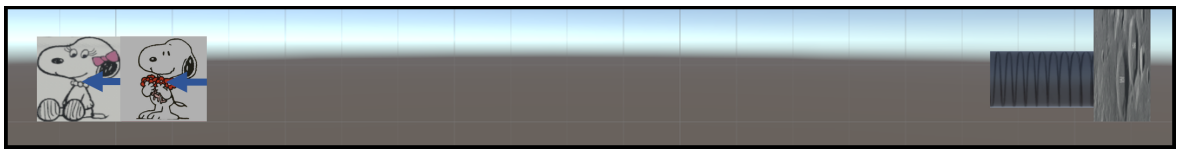

Abbildung 3: Inelastischer zusammenstoss mit dem anderen Würfel

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

6.1 Elastisch

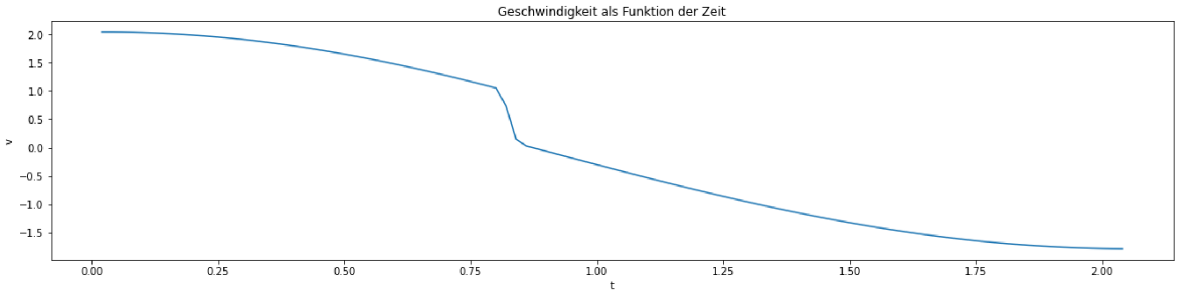

Abbildung 4: Geschwindigkeit als Funktion der Zeit

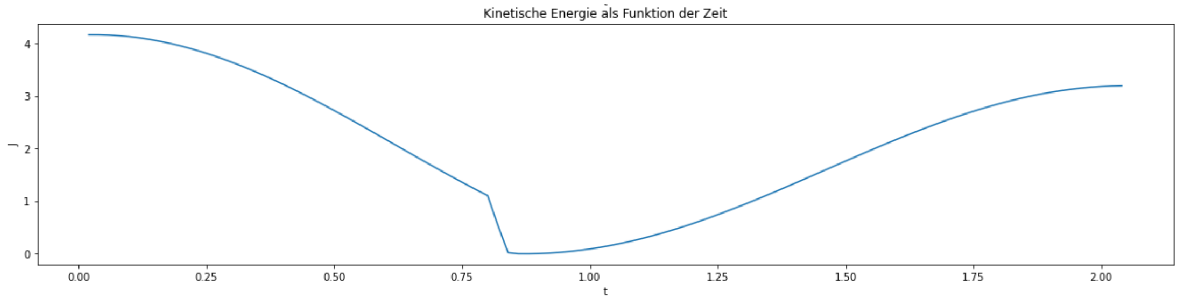

Abbildung 5: Kinetische Energie als Funktion der Zeit

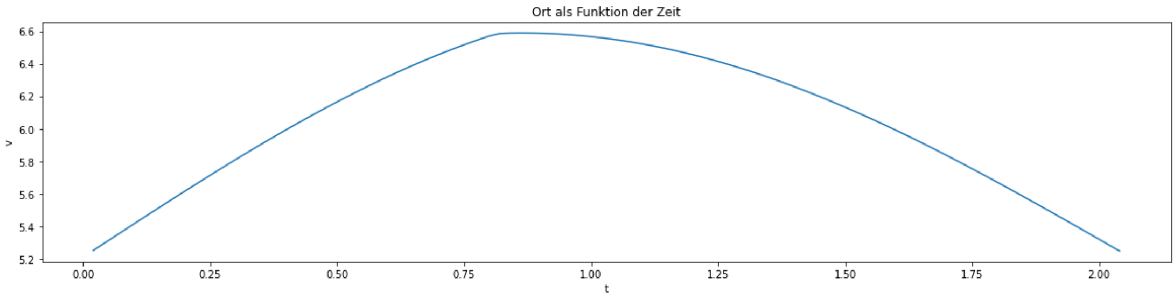

Abbildung 6: Ort als Funktion der Zeit

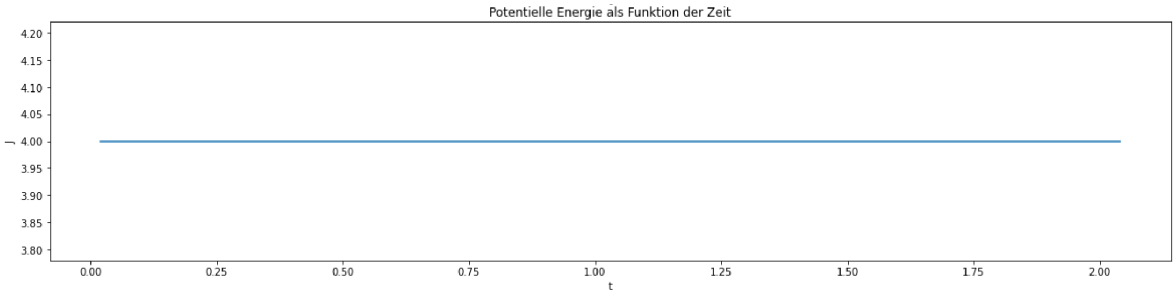

Abbildung 7: Potentielle Energie als Funktion der Zeit

6.2 Inelastisch

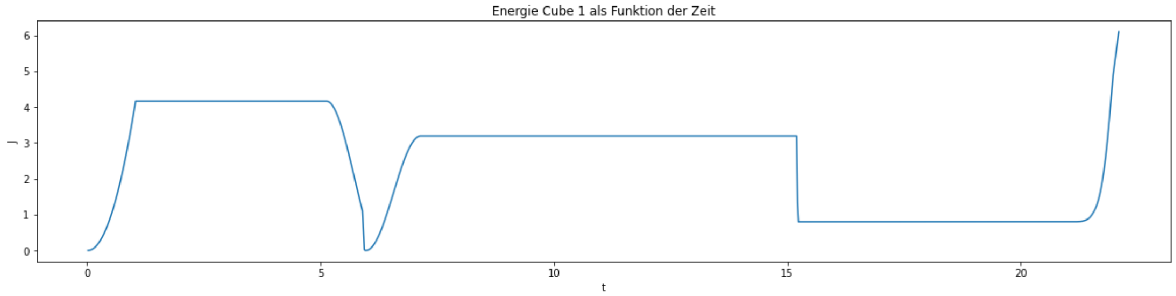

Abbildung 8: Energie Cube 1 als Funktion der Zeit

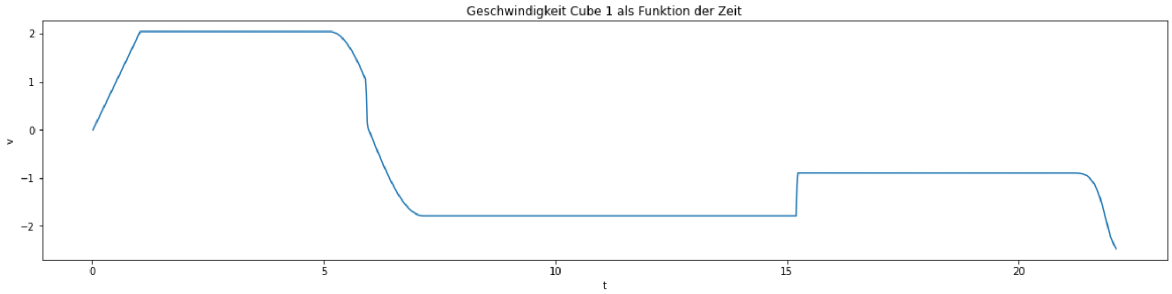

Abbildung 9: Geschwindigkeit Cube 1 als Funktion der Zeit

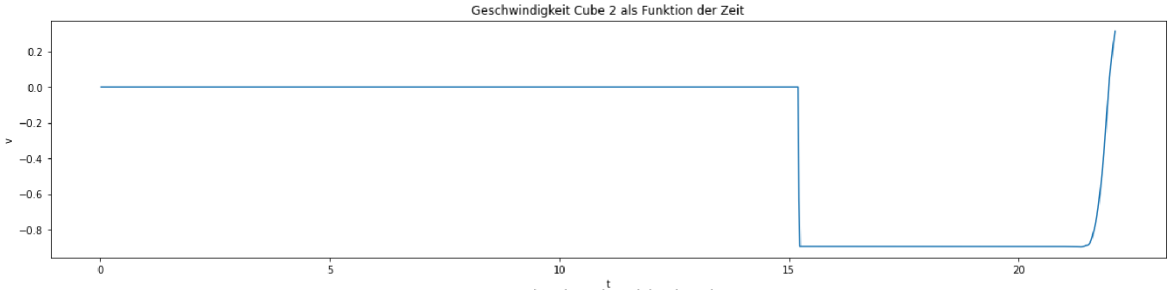

Abbildung 10: Geschwindigkeit Cube 2 als Funktion der Zeit

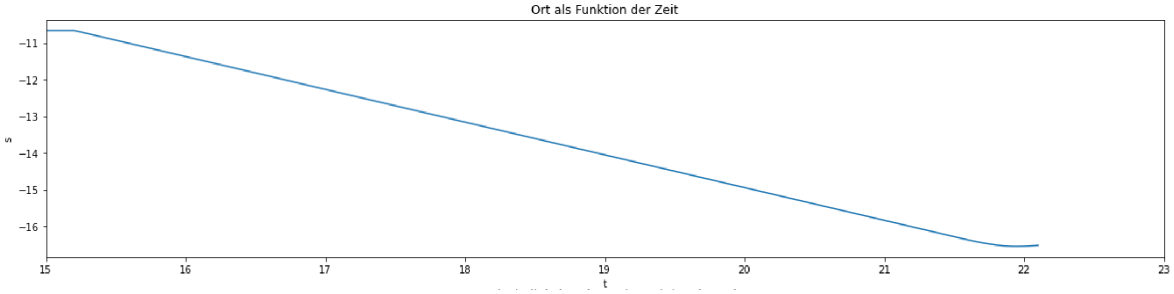

Abbildung 11: Ort als Funktion der Zeit

6.3 Inelastisch

7 Code

```
1 using System;
2 using System.Collections.Generic;
3 using System.IO;
4 using UnityEngine;
5
6
7 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 forceOnJulia = 0f;
    float velocityEnd = 0f;
    float cubeKineticEnd = 0f;
    float constantForce = 4f;
    double startime = 0;
    private double accelarationTime = 1.0;
    float springConstant = 0f;
    float springMaxDeviation = 0f;


    float springContraction = 1.5f;
    // 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)));
    }

    // Update is called once per frame
    void Update()
    {
    }
    // FixedUpdate can be called multiple times per frame
    void FixedUpdate()
    {
        double currentTime = Time.fixedTimeAsDouble-startime;

        if (accelarationTime >= currentTime)
        {
```

```
            //accelaration = velocity / time;
            constantForce = 4f;
            cubeRomeo.AddForce(new Vector3(constantForce, 0f, 0f));
        }

        cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
            Math.Pow(cubeRomeo.velocity.x, 2.0))); // 1/2*m*v^2
        springPotentialEnergy = Math.Abs((float)(0.5 * springConstant *
            Math.Pow(springContraction,2.0)));
        //currentTimeStep += Time.deltaTime;
        //timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
            cubeRomeo.position.x, cubeRomeo.velocity.x, cubeRomeoKinetic,
            springPotentialEnergy, cubeRomeoKinetic});

        float collisionPosition = cubeRomeo.transform.position.x +
            cubeRomeo.transform.localScale.x / 2;

        if (collisionPosition >= springMaxDeviation)
        {
            float springForceX = (collisionPosition - springMaxDeviation) *
                -springConstant;
            cubeRomeo.AddForce(new Vector3(springForceX, 0f, 0f));
            ChangeCubeTexture();
            currentTimeStep += Time.deltaTime;
            timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
                cubeRomeo.position.x, cubeRomeo.velocity.x, cubeRomeoKinetic,
                springPotentialEnergy, cubeRomeoKinetic, springForceX });
        }

        // 1/2*m*v^2
        cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
            Math.Pow(cubeRomeo.velocity.x, 2.0)));
        cubeRomeoImpulse = Math.Abs(cubeRomeo.mass * cubeRomeo.velocity.x);
        cubeJuliaImpulse = Math.Abs(cubeJulia.mass * cubeJulia.velocity.x);
        velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (cubeRomeo.mass +
            cubeJulia.mass);
        cubeKineticEnd = Math.Abs((float)(0.5 * (cubeRomeo.mass + cubeJulia.mass) *
            Math.Pow(velocityEnd, 2.0)));
        forceOnJulia = Math.Abs(cubeJulia.mass * velocityEnd - cubeJulia.velocity.x);

        cubeJuliaTimeStep += Time.deltaTime;
       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 });
    }
    void OnApplicationQuit()
    {
        WriteElasticTimeSeriesToCsv();
        WriteInelasticTimeSeriesToCsv();
    }
    void WriteElasticTimeSeriesToCsv()
    {
        using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
        {
```

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

```
153             ContactPoint contact = contacts[0];
154             joint.anchor = transform.InverseTransformPoint(contact.point);
155             joint.connectedBody =
                    collision.contacts[0].otherCollider.transform.GetComponent<Rigidbody>();
156
157
158             // Stops objects from continuing to collide and creating more joints
159             joint.enableCollision = false;
160         }
161     }
162 }
```

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).