

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	5
3.4		5
4	Beschreibung der Implentierung inklusive Screenshots aus Unity	5
5	Rückblick und Lehren aus dem Versuch	5
6	Resultate mit grafischer Darstellung	5
6.1	Elastisch	5
6.2	Inelastisch	6
6.3	Inelastisch	7
7	Code	7
A	Anhang	11

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.

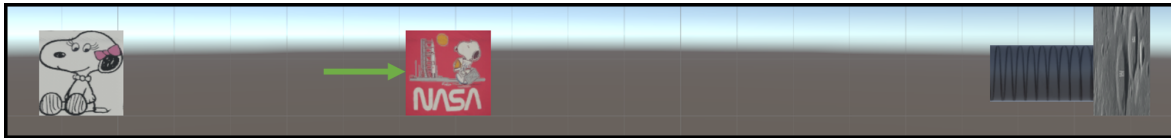


Abbildung 1: Beschleunigung des Würfels

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.

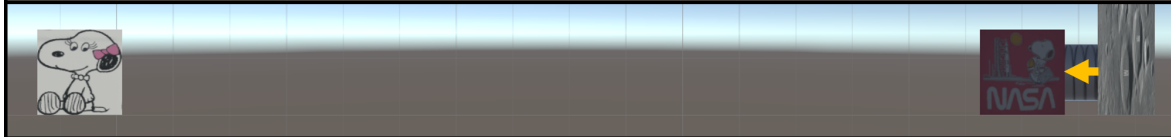


Abbildung 2: Elastischer Zusammenstoß mit der Feder

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.

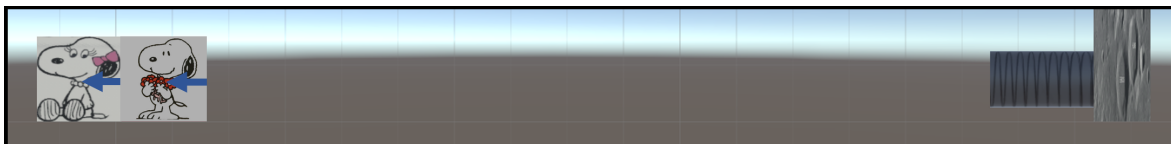


Abbildung 3: Inelastischer Zusammenstoß mit dem anderen Würfel

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

$$\dot{v} = a$$

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

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

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:

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

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

6.1 Elastisch

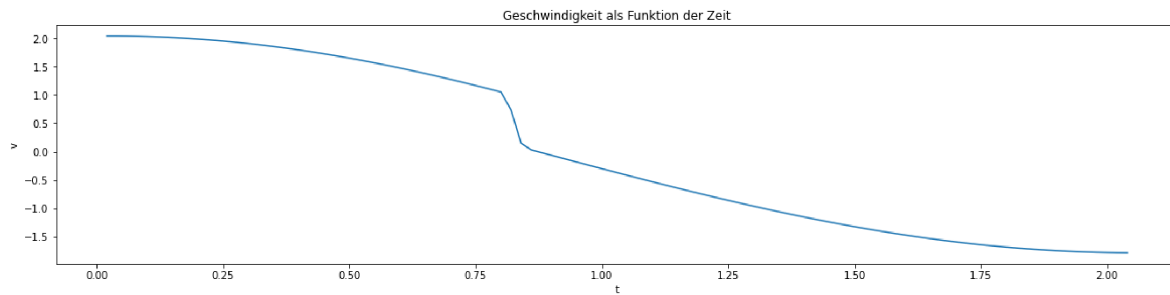


Abbildung 4: Geschwindigkeit als Funktion der Zeit

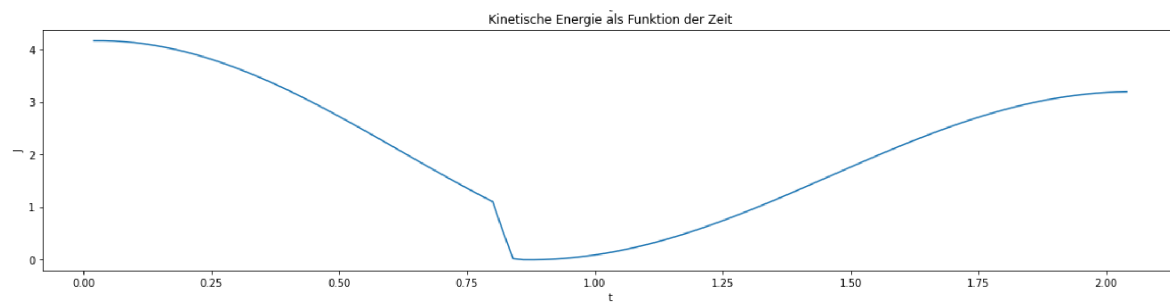


Abbildung 5: Kinetische Energie als Funktion der Zeit

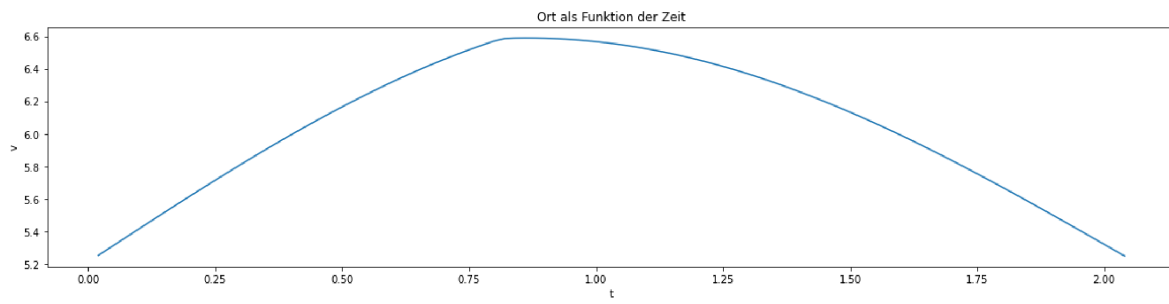


Abbildung 6: Ort als Funktion der Zeit

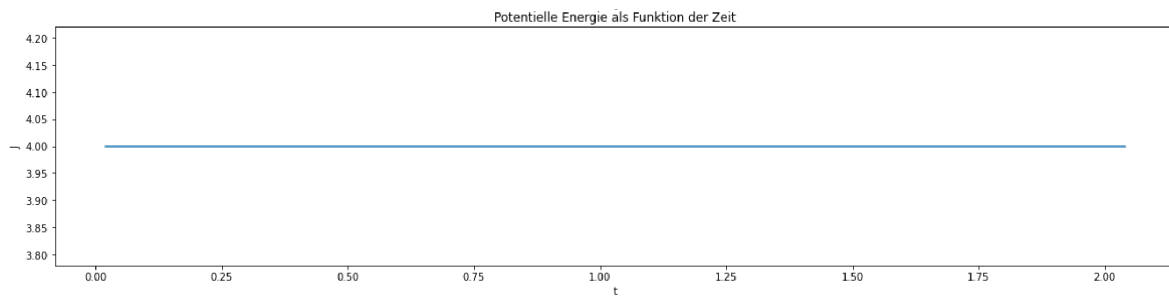


Abbildung 7: Potentielle Energie als Funktion der Zeit

6.2 Inelastisch

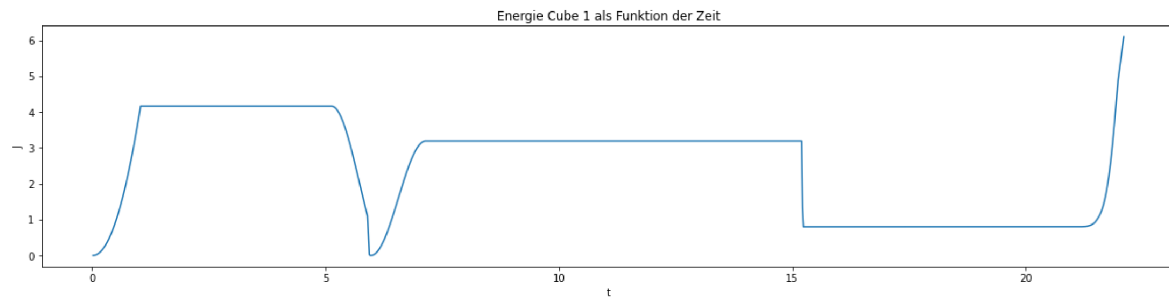


Abbildung 8: Energie Cube 1 als Funktion der Zeit

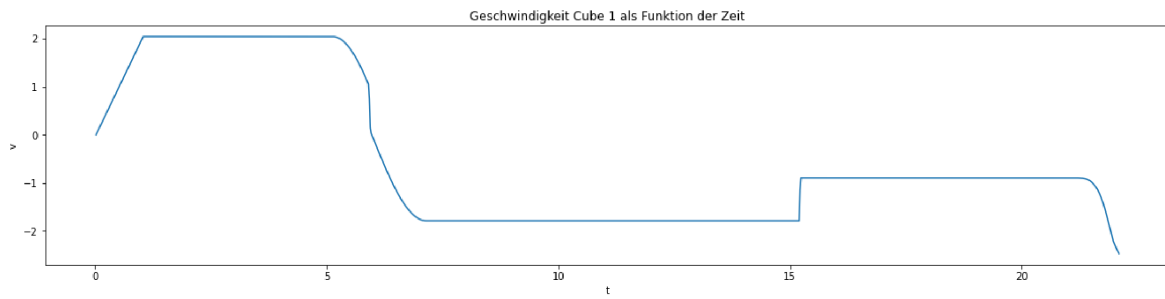


Abbildung 9: Geschwindigkeit Cube 1 als Funktion der Zeit

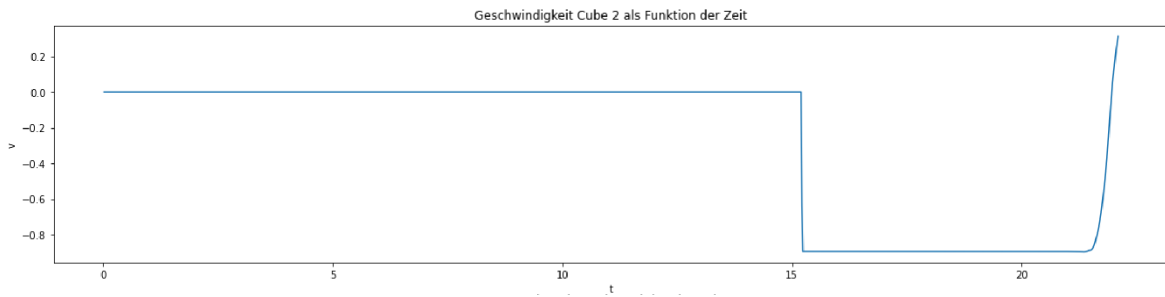


Abbildung 10: Geschwindigkeit Cube 2 als Funktion der Zeit

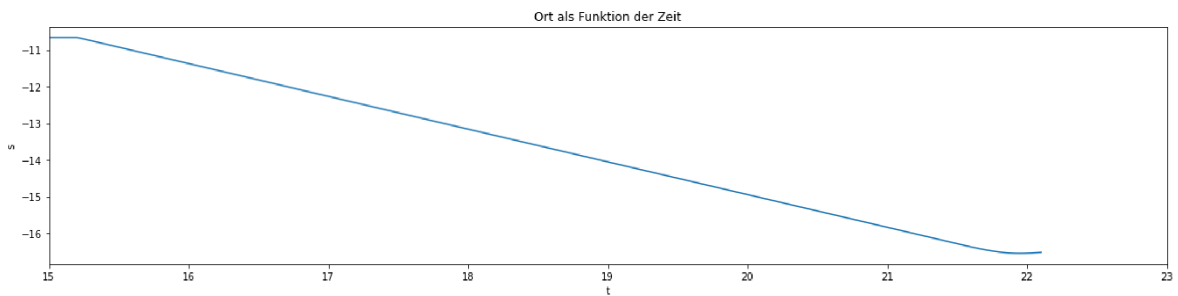


Abbildung 11: Ort als Funktion der Zeit

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

```

```

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 forceOnJulia = 0f;
26    float velocityEnd = 0f;
27    float cubeKineticEnd = 0f;
28    float constantForce = 4f;
29    double starttime = 0;
30    private double accelerationTime = 1.0;
31    float springConstant = 0f;
32    float springMaxDeviation = 0f;
33    float springContraction = 1.3f;
34
35    // Start is called before the first frame update
36    void Start()
37    {
38        starttime = Time.fixedTimeAsDouble;
39
40        timeSeriesElasticCollision = new List<List<float>>();
41        timeSeriesInelasticCollision = new List<List<float>>();
42
43        //Maximale Auslenkung gerechnet anhand der linken seite des Feders
44        springMaxDeviation = spring.transform.position.x -
45            spring.transform.localScale.y;
46        // Energieerhaltungsgesetz kinEnergie = PotEnergie :  $\frac{1}{2}mv^2 = \frac{1}{2}kx^2$ 
47        springConstant = (float)((cubeRomeo.mass * Math.Pow(2.0, 2)) /
48            (Math.Pow(springContraction, 2.0)));
49    }
50
51    // Update is called once per frame
52    void Update()
53    {
54    }
55
56    // FixedUpdate can be called multiple times per frame
57    void FixedUpdate()
58    {
59        double currentTime = Time.fixedTimeAsDouble - starttime;
60
61        if (accelerationTime >= currentTime)
62        {
63            //acceleration = velocity / time;
64            constantForce = 4f;
65        }
66    }

```



```

62     cubeRomeo.AddForce(new Vector3(constantForce, 0f, 0f));
63 }
64
65     cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
66     Math.Pow(cubeRomeo.velocity.x, 2.0))); // 1/2*m*v^2
67     //currentTimeStep += Time.deltaTime;
68     //timeSeriesElasticCollision.Add(new List<float>() { currentTimeStep,
69     cubeRomeo.position.x, cubeRomeo.velocity.x, cubeRomeoKinetic,
70     springPotentialEnergy, cubeRomeoKinetic});
71
72     float collisionPosition = cubeRomeo.transform.position.x +
73     cubeRomeo.transform.localScale.x / 2;
74
75     if (collisionPosition >= springMaxDeviation)
76     {
77         float springForceX = (collisionPosition - springMaxDeviation) *
78         -springConstant;
79         springPotentialEnergy =(float)(0.5 * springConstant *
80         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,
85         cubeRomeo.position.x, cubeRomeo.velocity.x, springPotentialEnergy,
86         cubeRomeoKinetic, springForceX });
87     }
88
89     // 1/2*m*v^2
90     cubeRomeoKinetic = Math.Abs((float)(0.5 * cubeRomeo.mass *
91     Math.Pow(cubeRomeo.velocity.x, 2.0)));
92     cubeRomeoImpulse = Math.Abs(cubeRomeo.mass * cubeRomeo.velocity.x);
93     cubeJuliaImpulse = Math.Abs(cubeJulia.mass * cubeJulia.velocity.x);
94     velocityEnd = (cubeRomeoImpulse + cubeJuliaImpulse) / (cubeRomeo.mass +
95     cubeJulia.mass);
96     cubeKineticEnd = Math.Abs((float)(0.5 * (cubeRomeo.mass + cubeJulia.mass) *
97     Math.Pow(velocityEnd, 2.0)));
98     forceOnJulia = Math.Abs(cubeJulia.mass * velocityEnd - cubeJulia.velocity.x);
99
100     cubeJuliaTimeStep += Time.deltaTime;
101     timeSeriesInelasticCollision.Add(new List<float>() { cubeJuliaTimeStep,
102     cubeRomeo.position.x, cubeRomeo.velocity.x, cubeRomeo.mass,
103     cubeRomeoImpulse, cubeRomeoKinetic, cubeJulia.position.x,
104     cubeJulia.velocity.x, cubeJulia.mass, cubeJuliaImpulse, velocityEnd,
105     cubeKineticEnd, forceOnJulia });
106 }
107
108 void OnApplicationQuit()
109 {
110     WriteElasticTimeSeriesToCsv();
111     WriteInelasticTimeSeriesToCsv();
112 }
113
114 void WriteElasticTimeSeriesToCsv()
115 {
116     using (var streamWriter = new StreamWriter("time_seriesElastic.csv"))
117     {
118         streamWriter.WriteLine("currentTimeStep, cubeRomeo.position.x,
119         cubeRomeo.velocity.x, springPotentialEnergy, cubeRomeoKinetic,

```

```

        springForceX");
102
103     foreach (List<float> timeStep in timeSeriesElasticCollision)
104     {
105         streamWriter.WriteLine(string.Join(",", timeStep));
106         streamWriter.Flush();
107     }
108 }
109
110 }
111
112 void WriteInelasticTimeSeriesToCsv()
113 {
114     using (var streamWriter = new StreamWriter("time_seriesInelastic.csv"))
115     {
116         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 ");
117
118         foreach (List<float> timeStep in timeSeriesInelasticCollision)
119         {
120             streamWriter.WriteLine(string.Join(",", timeStep));
121             streamWriter.Flush();
122         }
123     }
124 }
125
126 void ChangeCubeTexture()
127 {
128     // the path of the image
129     filePath = "Assets/Images/snoopy-flower-cynthia-t-thomas.jpg";
130     // 1.read the bytes array
131     fileData = File.ReadAllBytes(filePath);
132     // 2.create a texture named tex
133     Texture2D tex = new Texture2D(2, 2);
134     // 3.load inside tx the bytes and use the correct image size
135     tex.LoadImage(fileData);
136     // 4.apply tex to material.mainTexture
137     GetComponent<Renderer>().material.mainTexture = tex;
138 }
139
140 void OnCollisionEnter(Collision collision)
141 {
142     if (collision.rigidbody != cubeJulia)
143     {
144         return;
145     }
146     if (collision.rigidbody == cubeJulia)
147     {
148         FixedJoint joint = gameObject.AddComponent<FixedJoint>();
149         ContactPoint[] contacts = new ContactPoint[collision.contactCount];
150         collision.GetContacts(contacts);
151         ContactPoint contact = contacts[0];
152         joint.anchor = transform.InverseTransformPoint(contact.point);

```

```

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

```

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	6
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	7
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).