# PE SEMESTERPROJEKT

Bericht

Stutz Aline, Isliker Pascal, Scherrer Marcel
ZHAW

# Inhaltsverzeichnis

# 1 Abstract

Dieser Bericht beinhaltet die gewonnen Erkenntnisse und Resultate der Experimente gemäss der Aufgabenstellung „PE_Semesterprojekt_FS22_v08.pdf“. Die Experimente wurden dabei mithilfe von Unity simuliert.

# 2 Einleitung

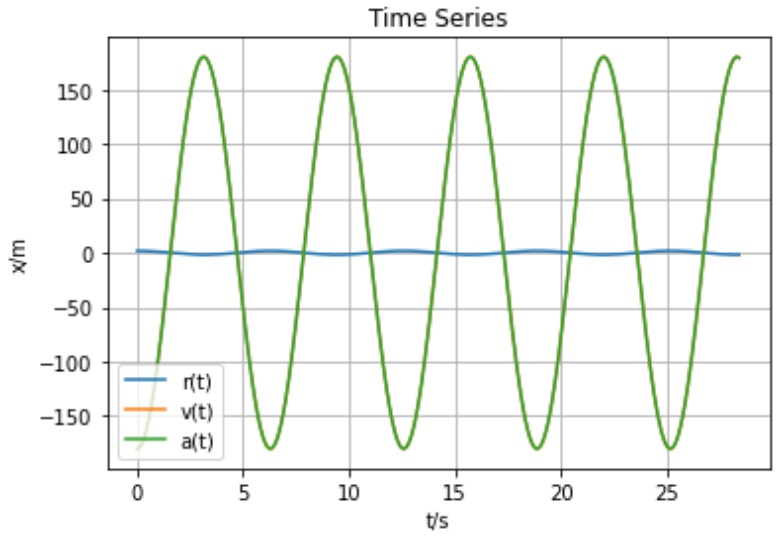
In diesem Bericht werden die gewonnenen Erkenntnisse aus den Experimenten gemäss der Anleitung Semesterprojekt PE FS 2022 dokumentiert. Dabei werden folgende Physikthemen behandelt:

- Harmonischer Oszillator
- Federkraft
- Kinetische Energie
- Federenergie
- Energieerhaltungsgesetz
- Impulsgesetz
- Elastischer Stoss
- Trockene Reibung
- Winkelgeschwindigkeit
- Arbeit
- Freier Fall

Für die Visualisierung der Experimente wird Unity verwendet. Zur Darstellung der Grafiken der erhaltenen Daten des Experimentes wird ein Python Skript verwendet.

# 3 Teil 1: Harmonischer Oszillators

Es soll ein Würfel erstellt werden, welcher eine Oszillation erfährt. Dieser soll dann eine Harmonische Schwingung widerspiegeln. In der unterstehenden Graphik, kann erkannt werden, dass die Reibung 0 beträgt und der Würfel eine gleiche Beschleunigung und Geschwindigkeit aufweist.



# 4 Teil 2: Würfel wird gestossen

## 4.1 Aufgabenstellung

Ein Würfel $W_1$ ($m = 1\ kg$ ) wird durch eine gespannte Feder von einer unendlich schweren Mauer $M_1$ ($m = \infty$) weggestossen. Dabei soll der Würfel, nachdem er abgestossen wurde, eine Geschwindigkeit ($\vec{v} = 1\ m/s$) erreichen. Um diese Bedingungen zu erfüllen, müssen die Federkonstant $k$ und die Kompression $x$ entsprechend gewählt werden.

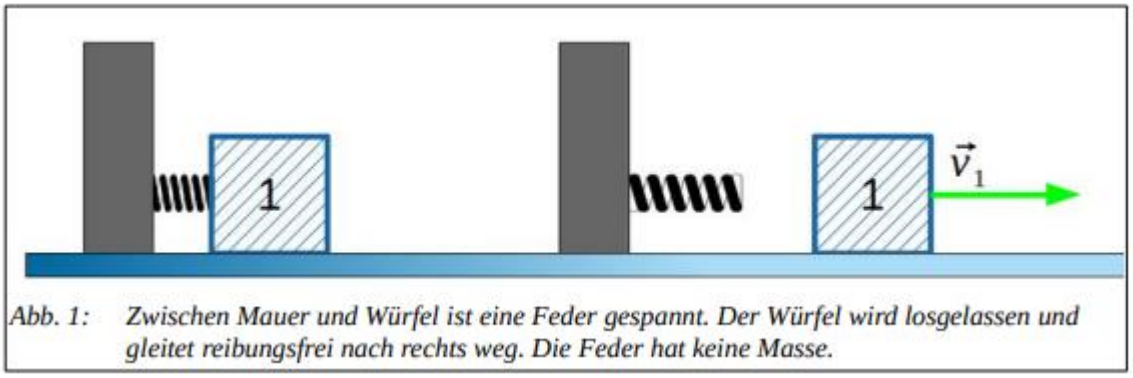



*Abb. 1: Zwischen Mauer und Würfel ist eine Feder gespannt. Der Würfel wird losgelassen und gleitet reibungsfrei nach rechts weg. Die Feder hat keine Masse.*

*Abbildung 1 Ausschnitt der Aufgabe 2 aus der Aufgabenstellung*

## 4.2 Unser Experiment

**Beschreibung**

Nach dem Start des Programmes wird der Würfel auf eine Geschwindigkeit $v = 1\ m/s$ in Richtung der Wand beschleunigt. An der Position $x = 0$ trifft der Würfel auf die Feder. Dabei wird die Feder komprimiert und die Geschwindigkeit des Würfels verlangsamt sich, bis die maximale Kompression bei $x = -1$, erreicht wird. Ab diesem Zeitpunkt beschleunigt der Würfel in die entgegengesetzte Richtung (weg von der Wand). Die Beschleunigung erhöht sich bis der Würfel die Feder nicht mehr berührt. Die Geschwindigkeit beträgt ab diesem Zeitpunkt wieder konstant $v = 1\ m/s$.

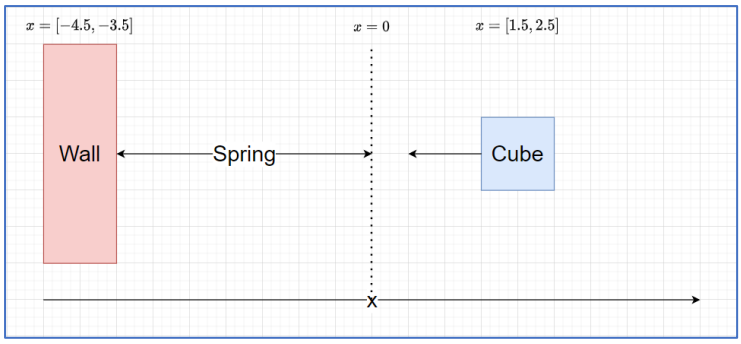



*Abbildung 2: Top-Down Ansicht des Experimentes*

**Umsetzung in Unity**

Um dem Würfel eine initiale Geschwindigkeit von $v = 1\ m/s$ zu geben, wendeten wir einmalig eine Kraft von 50 Newton auf den Würfel an. Dieser Kraftaufwand ist nötig, um den Würfel auf eine Geschwindigkeit $1\ m/s$ zu beschleunigen, da in Unity ein einziger Tick $(0.02\ s)$ beträgt.

Nach der initialen Beschleunigung, bewegt sich der Würfel auf die Feder $(x = 0)$ mit konstanter Geschwindigkeit von $1\ m/s$ zu. Sobald der Würfel auf die Feder trifft, können wir die Formel der Federkraft $(\vec{F} = -kx)$ anwenden. Die Feder wird maximal komprimiert und stösst den Würfel ab. Die Feder wächst anschliessend wieder auf ihre ursprüngliche Grösse an.

**Graphische Darstellung des Experiments**

Das nachfolgende Diagramm wurde basierend auf den Daten des Unity-Experiments generiert. Das Resultat entspricht unserer Erwartung. Die Geschwindigkeit nimmt zuerst ab und anschliessend nimmt sie wieder zu, bis eine Geschwindigkeit von $1\ m/s$ erreicht wird. Ab diesem Zeitpunkt bleibt die Geschwindigkeit konstant.

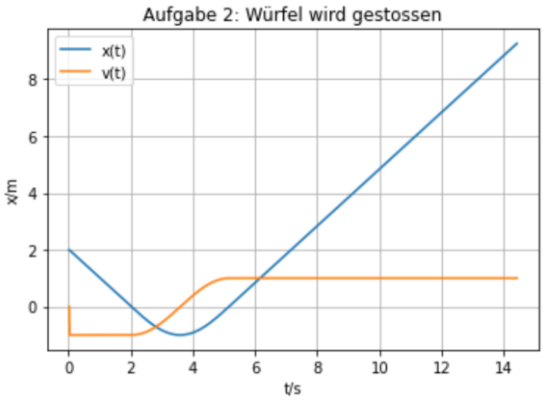



*Abbildung 3: Orts- und Geschwindigkeit Diagramm*

## 4.3 Theorie

Für die Umsetzung des Experimentes sind Kenntnisse einiger physikalische Formeln nötig, auf die wir in diesem Abschnitt genauer eingehen wollen.

**Federkraft**

Die Federkraft beträgt $\vec{F} = -kx = ma$. Sie zeigt entgegen der Auslenkung $(-kx)$. Die Beschleunigung $a$ ist proportional zum Ort $x$.

**Kinetische-Energie und Feder-Energie**

Bei unserem Experiment entsteht sowohl Kinetische- als auch Feder-Energie. Diese Energien sind durch folgende Formeln definiert.

$$E_{spring} = \frac{1}{2} \cdot kx^2, \qquad E_{kin} = \frac{1}{2} \cdot mv^2$$

**Energie-Erhaltungssatz**

Da es sich bei unserem Versuch um ein abgeschlossenes System handelt, kommt der Energie-Erhaltungssatz zum Zuge. Dieser besagt, dass sich die Energie in einem abgeschlossenen System mit der Zeit nicht verändert. In unserem Fall gibt es kinetische- $E_{kin}$ und Feder-Energie $E_{spring}$, diese müssen zusammen konstant bleiben. Daraus lässt sich folgende Formel ableiten.

$$E_{spring} + E_{kin} = E_{tot}, \qquad E_{tot} = konstant$$

## 4.4 Berechnung der Energien

In diesem Bereich werden wir die Energien, mithilfe der vorhergehenden Formel berechnen.

**Vor Kompression** $(x = 0.5)$

$$E_{kin} = \frac{1}{2} \cdot m \cdot v^2 = \frac{1}{2} \cdot 1 \cdot 1^2 = 0.5\,J, \qquad E_{spring} = \frac{1}{2} \cdot k \cdot x^2 = \frac{1}{2} \cdot 1 \cdot 0^2 = 0\,J$$
$$E_{kin} + E_{spring} = E_{tot} = 0.5 + 0 = 0.5\,J$$

**Während Kompression** $(x = -0.4)$

Berechnung der Feder-Energie:

- $E_{spring} = \frac{1}{2} \cdot k \cdot x^2 = \frac{1}{2} \cdot 1 \cdot (-0.4)^2 = 0.08\,J$

Berechnung der Geschwindigkeit:

$$E_{tot} - E_{spring} = \frac{1}{2} \cdot m \cdot v^2 \rightarrow v = \sqrt{2 \cdot \frac{E_{tot} - E_{spring}}{m}} = \sqrt{2 \cdot \frac{0.42}{1}} = 0.917\,m/s$$

**Maximale Kompression** $(x = -1)$

$$E_{kin} = \frac{1}{2} \cdot m \cdot v^2 = \frac{1}{2} \cdot 1 \cdot 0^2 = 0\,J, \qquad E_{spring} = \frac{1}{2} \cdot k \cdot x^2 = \frac{1}{2} \cdot 1 \cdot (-1)^2 = 0.5\,J$$
$$E_{kin} + E_{spring} = E_{tot} = 0 + 0.5 = 0.5\,J$$

## 4.5 Vergleich mit Experiment

Vergleichen wir die berechnete Geschwindigkeit von $v = 0.917\ m/s$ , welche während der Kompression besteht, mit der Geschwindigkeit in unseren Experiment, so stellen wir fest, dass diese an der Position $x = -0.407$ ebenfalls eine Geschwindigkeit $v = 0.917\ m/s$ aufweist.

# 5 Teil 3a: Würfel stossen verzögert elastisch

## 5.1 Beschreibung

Würfel 1 stösst mit Würfel 2 zusammen.

- Würfel 1 startet mit einer Geschwindigkeit $v_1 = 1\ m/s$
- Würfel 2 startet mit einer Geschwindigkeit von $v_0 = 0\ m/s$

Zwischen den Würfeln wird eine Feder komprimiert.

Wenn die Feder maximal komprimiert ist, werden die beiden Würfel fest miteinander verbunden und gleiten gemeinsam weiter.

Nach einer Strecke von 5 Meter oder nach einigen Sekunden wird die feste Verbindung gelöst, die Würfel stossen sich voneinander ab. Würfel 1 gleitet nach links, Würfel 2 nach rechts.

Eigenschaften:

- Würfel 2 ($m_2 = 2\ kg$) ist doppelt so schwer wie Würfel 1 ($m_1 = 1\ kg$).
- Würfel 2 ($V_2 = 1\ m^3$) ist gleich gross wie Würfel 1 ($V_1 = 1\ m^3$).

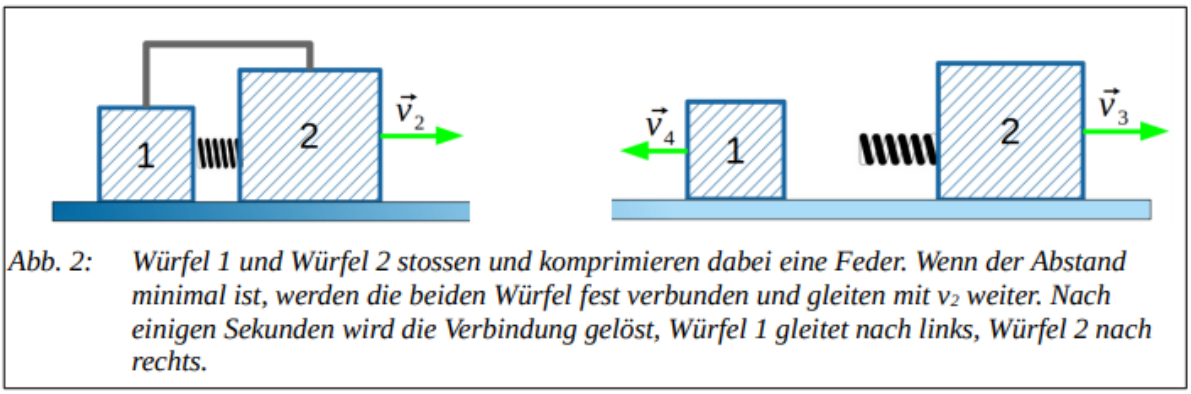



*Abb. 2: Würfel 1 und Würfel 2 stossen und komprimieren dabei eine Feder. Wenn der Abstand minimal ist, werden die beiden Würfel fest verbunden und gleiten mit $v_2$ weiter. Nach einigen Sekunden wird die Verbindung gelöst, Würfel 1 gleitet nach links, Würfel 2 nach rechts.*

## 5.2 Theorie

**Massen und Geschwindigkeitsänderungen**

$$\frac{m_1}{m_2} = -\frac{\Delta v_2}{\Delta v_1}$$

**Impuls**

- Impuls eines Körpers $\vec{p} = m \cdot \vec{v}$
- Impuls vor Stoss $\overrightarrow{p_{vor}} = m_1 \cdot \overrightarrow{v_1} + m_2 \cdot \overrightarrow{v_2}$
- Impuls nach Stoss $\overrightarrow{p_{nach}} = m_1 \cdot \overrightarrow{v_1'} + m_2 \cdot \overrightarrow{v_2'}$

**Elastischer Stoss**

$$\overrightarrow{v_1'} = \frac{m_1 - m_2}{m_1 + m_2} \cdot \overrightarrow{v_1} + \frac{2m_2}{m_1 + m_2} \overrightarrow{v_2}$$

$$\overrightarrow{v_2'} = \frac{m_2 - m_1}{m_1 + m_2} \cdot \overrightarrow{v_2} + \frac{2m_1}{m_1 + m_2} \overrightarrow{v_1}$$

## 5.3 Umsetzung in Unity

Für die Umsetzung in Unity, haben wir zwei gleich grosse Würfelobjekte erstellt. Diesen Würfeln haben wir unterschiedliche Materialien zugewiesen, sodass der eine Würfel doppelt so schwer ist wie der andere. Wenn wir das Experiment starten, wird der leichte Würfel, mit m = 1kg, in Richtung des schwereren Würfels, mit m = 2 kg, beschleunigt. Die Feder zwischen den Würfel wird komprimiert und die beiden Würfel gleiten gemeinsam für 4 Sekunden weiter. Nach Ablauf dieser 4 Sekunden, wird die Federkraft wieder aktiviert und die beiden Würfel stossen sich voneinander ab.

## 5.4 Berechnungen

**Impuls**

- Impuls von Körper $m_1$ $\quad \overrightarrow{p_1} = m_1 \cdot \overrightarrow{v_1} = 1 \cdot (1 \quad 0 \quad 0)^T = (1 \quad 0 \quad 0)^T$
- Impuls von Körper $m_2$ $\quad \overrightarrow{p_2} = m_2 \cdot \vec{v}_2 = 2 \cdot (0 \quad 0 \quad 0)^T = (0 \quad 0 \quad 0)^T$

**Elastischer Stoss**

$$\overrightarrow{v_1'} = \frac{m_1 - m_2}{m_1 + m_2} \cdot \overrightarrow{v_1} + \frac{2m_2}{m_1 + m_2} \overrightarrow{v_2} = \frac{1-2}{1+2} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} + \frac{2 \cdot 2}{1+2} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} = -\frac{1}{3} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

$$\overrightarrow{v_2'} = \frac{m_2 - m_1}{m_1 + m_2} \cdot \overrightarrow{v_2} + \frac{2m_1}{m_1 + m_2} \overrightarrow{v_1} = \frac{2-1}{1+2} \cdot \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \frac{2 \cdot 1}{1+2} \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} = \frac{2}{3} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$$

## 5.5 Vergleich mit Experiment

Nun vergleichen wir die berechnete mit der tatsächlichen Geschwindigkeit der beiden Würfel.

- Würfel 1: Berechnet: $v_1 = (-0.33 \quad 0 \quad 0)^T$ Experiment: $v_1 = (-0.29 \quad 0 \quad 0)^T$
- Würfel 2: Berechnet: $v_1 = (0.7 \quad 0 \quad 0)^T$ Experiment: $v_1 = (0.66 \quad 0 \quad 0)^T$

Die berechneten Werte unterscheiden sich somit nur leicht von den tatsächlichen Werten.

# 6 Teil 3b: Würfel 1 wird gefangen

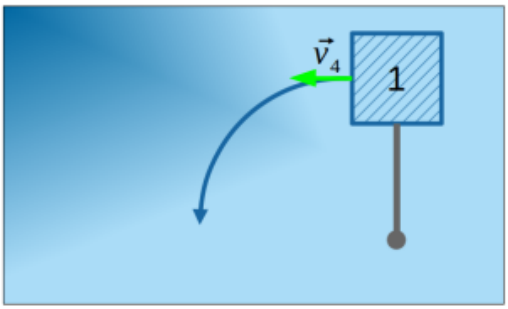
## 6.1 Beschreibung

Würfel 1 gleitet (mit einer Geschwindigkeit von $v_2'$) eine kurze Strecke nach links.

Dann wird ein virtuelles masseloses Seil von 5 m Länge am Schwerpunkt des Würfels befestigt. Würfel 1 dreht in der Ebene um den Befestigungspunkt des Seils. Der Befestigungspunkt befindet sich seitlich, 5 m senkrecht zur Geschwindigkeit.

Am Boden wirkt trockene Reibung, sodass der Würfel nach einem Viertelkreis zum Stehen kommt.

*Abb. 3: Ansicht von oben. Würfel 1 kreist um den Befestigungspunkt des Seils und kommt nach einem Viertelkreis zum Stehen.*



## 6.2 Theorie

**Trockene Reibung**

Trockene Reibung ist proportional zur Normalkraft

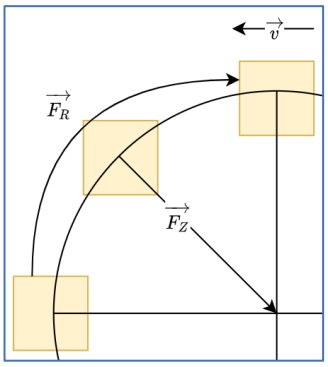
- $\mu =$ Proportionalitätsfaktor / Gleitreibungskoeffizient

$$\overrightarrow{F_R} = \mu \overrightarrow{F_N}$$

**Arbeit**

Die physikalische Definition für Kraft lautet:

$$Arbeit = Kraft \cdot Weg, \qquad E = F \cdot s$$




**Lösungsweg**

Da das ganze System betrachtet wird, müssen wir die Zentripetalkraft zur Berechnung verwenden. Diese ist folgendermassen definiert

$F_Z$: Zentripetalkraft

$$\overrightarrow{F_Z} = m \cdot \frac{v^2}{R}, \qquad z = (x_0 - x, y_0 - y)$$

$$\overrightarrow{F_z} = \left|\overrightarrow{F_z}\right| \cdot \overrightarrow{e_z}$$

$F_R$: Reibungskraft

Die Formel für die Reibungskraft lässt sich aus der Formel der Trockenen Reibung und der Arbeit ableiten. Dies muss danach auf die Aufgabenstellung angewendet werden. Dies resultiert in folgende Formel:

$$E = F_R \cdot s = mg\mu \cdot \frac{2}{4}\pi R = \frac{1}{2}mv^2, \qquad g\mu \cdot \pi R = v^2$$

$$\mu = \frac{v^2}{g\pi R}$$

$$\left|\overrightarrow{F_R}\right| = mg\mu, \qquad \overrightarrow{F_R} = mg\mu \cdot -\overrightarrow{e_z}$$

$F_{res}$: Resultierende Kraft

Die resultierende Kraft aus der Zentripetalkraft und Reibungskraft ist äquivalent und hat einen Wert von 0.

$$\overrightarrow{F_{res}} = \overrightarrow{F_z} + \overrightarrow{F_R}$$

## 6.3 Umsetzung in Unity

Bei der Umsetzung in Unity haben wir jeweils die Reibungskraft und die Zentripetalkraft berechnet und auf den Würfel angewendet. Damit der Würfel auch tatsächlich nach ungefähr einer viertel Rotation anhält haben wir zuerst den Reibungskoeffizienten $\mu$ berechnet (siehe Berechnung).

## 6.4 Berechnung

- Würfel-Geschwindigkeit $v = -0.33\ m/s$ (gemäss Resultat aus Kapitel 5.4)
- Seillänge (Radius) $R = 5\ m$
- Reibungskoeffizient $\mu$

$$\mu = \frac{v^2}{g\pi R} = \frac{(-0.33)^2}{9.81 \cdot \pi \cdot 5} = 0.0007$$

## 6.5 Vergleich mit Experiment

Nun vergleichen wir die berechnete mit der tatsächlichen Werten des Experiments.

- Reibungskoeffizient Berechnet: $\mu = 0.0007$ Experiment: $\mu = 0.00054$

In Unity haben wir die Geschwindigkeit des Würfels auf 0 gesetzt, nachdem dieser die Geschwindigkeit von $0.001\ m/s$ unterschritten hatte. Zu diesem Zeitpunkt hatte der Würfel beinahe eine Viertel Rotation durchlaufen.

Vergleich der Position am Ende des Experiments

- Erwartete Position: $(0, 0.5, -4)^T$ Tatsächliche Position: $(0.002, 0.5, -3.91)^T$

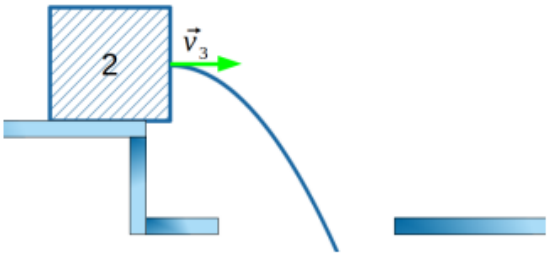
# 7 Teil 3c: Würfel 2 fällt

## 7.1 Beschreibung

Würfel 2 gleitet (mit einer Geschwindigkeit von $v_2'$) auf eine Kante zu und fällt ins Leere ($g = 9.81\ m/s$).

Weit unten, $5\ m$ tiefer, gibt es ein Loch im Boden, durch das der Würfel fallen soll. Wählen Sie die Position des Lochs so, dass der Würfel mittig trifft.

*Abb. 4: Ansicht von der Seite. Würfel 2 fällt über die Kante und durch ein Loch im 5 m tieferen nächsten Boden.*




## 7.2 Theorie

Um die Position des Loches richtig zu wählen, müssen wir die Falldistanz des Würfels berechnen. Dazu brauchen wir die Formel zu:

**Falldistanz $s_1$ und Fall-Geschwindigkeit $v_1$ nach einer Zeit $t$**

Da die Beschleunigung des Würfels durch die Gravitation bestimmt wird, müssen wir $a = g$ setzen.

$$s = s_0 + \frac{g}{2} \cdot t^2, \qquad v = at = gt$$

**Horizontale Distanz $s_2$ und Geschwindigkeit $v_2$ nach einer Zeit $t$**

Um die horizontale Distanz auszurechnen, müssen wir $v_3$ bestimmen, welche äquivalent zu $v_2$ ist.

$$s_2 = s_0 + v_3 \cdot t, \qquad v_2 = v_3$$

**Berechnung der Zeit**

Die Zeit die der Würfel braucht um von der Kante in das Loch zu fallen kann mithilfe folgender Formel berechnet werden:

$$t = \sqrt{\frac{2s}{g}}$$

# 8 Rückblick

In diesem Projekt konnten wir verschiedene physikalische Formel kennen lernen und mithilfe von Unity visualisieren. In einem ersten Teil lernten wir, wie sich ein harmonischer Oszillator verhält und konnten diesen dann im zweiten Teil verwenden und die Federkraft kennen zu lernen, indem wir ein Objekt oszillierten und dann von einer Feder abstossen. Im dritten Teil verwendeten wir das Wissen aus Teil 1 und 2, um den elastischen Stoss kennenzulernen. Ausserdem machten wir ein Experiment, das Aufzeigte, wie die Reibungskraft und Zentripetalkraft einen sich bewegenden Körper abbremsen. Zu guter letzte betrachteten wir uns noch den freien Fall, indem wir ein Objekt über die Kante fallen liessen.

Obwohl die Experimente zu Anfang kompliziert erschienen, konnten wir mit den Unterlagen und Formel, welche wir erlernten, die Theorie dafür nachvollziehen. Dies erlaubte es uns diese Experimente in Unity zu visualisieren, welches das Verständnis der physikalisch wirkenden Kräfte verbesserte. Die Arbeit an diesem Auftrag war sehr lehrreich und erlaubte uns einen guten Einblick in die Welt der Physik. Ausserdem konnte das Informatiker Herz sich ebenfalls erfreuen beim Code schreiben und kennenlernen des Unity Tools.

# 9 Abbildungsverzeichnis

# 10 Anhang

## 10.1 Aufgabe 2

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

using System.IO;

/*
    Accelerates the cube to which it is attached, modelling an harmonic
oscillator.
    Writes the position, velocity and acceleration of the cube to a CSV file.

    Remark: For use in "Physics Engines" module at ZHAW, part of physics lab
    Author: kemf
    Version: 1.0
*/

public class CubeController : MonoBehaviour
{
    private Rigidbody rigidBody;

    public int springConstant; // N/m

    private float currentTimeStep; // s

    private List<List<float>> timeSeries;

    // Start is called before the first frame update
    void Start()
    {
        rigidBody = GetComponent<Rigidbody>();
        timeSeries = new List<List<float>>();
    }

    // Update is called once per frame
    void Update()
    {

    }

    private float featherPosXFrom = 0; // m
    private double featherPosXTo = -3.5; // m
    private bool featherEnabled = false;
    private bool setInitialForce = false;

    // FixedUpdate can be called multiple times per frame
    void FixedUpdate() {
        float forceX = 0; // N

        rigidBody.mass = 1; // kg
        springConstant = 1;  // N/m

        // Assuming the Feather starts at position x = 0
        float compression = rigidBody.position.x;

        //Set Initial Force before Feather is touched
        if (!setInitialForce)
        {
            //Apply Force
```

```
            forceX = -50; // N
            rigidBody.AddForce(new Vector3(forceX, 0f, 0f));
            setInitialForce = true;
        }

        //Pre-Feather
        if (rigidBody.position.x < featherPosXFrom && rigidBody.position.x >
featherPosXTo)
        {
            featherEnabled = true;
        } else
        {
            featherEnabled = false;
        }

        Debug.Log("Position: " + rigidBody.position.x
            + ", Velocity: " + rigidBody.velocity.x
            + ", Acceleration: " + forceX / rigidBody.mass);

        //Feather
        if (featherEnabled)
        {
            //F = -x*k
            forceX = -compression * springConstant;


            //Apply Force
            rigidBody.AddForce(new Vector3(forceX, 0f, 0f));
        }

        //Post-Feather

        currentTimeStep += Time.deltaTime;
        timeSeries.Add(new List<float>() {currentTimeStep, rigidBody.position.x,
rigidBody.velocity.x, forceX });
    }

    void OnApplicationQuit() {
        WriteTimeSeriesToCSV();
    }


    void WriteTimeSeriesToCSV() {
        using (var streamWriter = new StreamWriter("time_series.csv")) {
            streamWriter.WriteLine("t,z(t),v(t),a(t) (added)");

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

## 10.2 Aufgabe 3

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

using System.IO;

/*
    Accelerates the cube to which it is attached, modelling an harmonic
oscillator.
    Writes the position, velocity and acceleration of the cube to a CSV file.

    Remark: For use in "Physics Engines" module at ZHAW, part of physics lab
    Author: kemf
    Version: 1.0
*/
public class CubeController : MonoBehaviour
{
    private const int DELAY = 4;

    public float springLength;   // m
    public float springConstant; // N/m
    public float startVelocity;  // m/s
    private float velocity1;     // m/s
    private float velocity2;     // m/s
    public Rigidbody lightCube;
    private Rigidbody heavyCube;
    private float previousDistance = 10.0f; // m
    private float currentTimeStep; // s
    private float pushTime;        // s
    private float RADIUS = 5.0f;   // m
    private float GRAVITY = 9.81f; // m/s^2
    private float mu;
    private List<List<float>> timeSeries;
    private Vector3 vectorInit = new Vector3(5f, 0.5f, -4f);
    private State state;

    private bool isRotating = false;


    // Start is called before the first frame update
    void Start()
    {
        heavyCube = GetComponent<Rigidbody>();
        timeSeries = new List<List<float>>();
        velocity1 = startVelocity; // m/s
        velocity2 = 0; // m/s
        lightCube.velocity = new Vector3(velocity1, 0, 0);
        heavyCube.velocity = new Vector3(velocity2, 0, 0);
        state = State.PRE_SPRING;

        //Root position of the string
        vectorInit = new Vector3(5f, 0.5f, -4f);
    }

    // Update is called once per frame
    void Update()
    {

    }
```

```
    // FixedUpdate can be called multiple times per frame
    void FixedUpdate()
    {
        currentTimeStep += Time.deltaTime;
        timeSeries.Add(new List<float>() { currentTimeStep, lightCube.velocity.x,
heavyCube.velocity.x, lightCube.position.x, heavyCube.position.x });

        float distanceBetweenCubes = heavyCube.position.x - 0.5f -
(lightCube.position.x + 0.5f);
        float forceX = springConstant * (springLength - distanceBetweenCubes);

        switch (state)
        {
            case State.PRE_SPRING:
                if (cubesAreTouchingSpring(distanceBetweenCubes, springLength))
                {
                    state = State.TOUCHING_BEFORE_LOCK;
                    goto case State.TOUCHING_BEFORE_LOCK;
                }
                break;
            case State.TOUCHING_BEFORE_LOCK:
                applyForce(-forceX, forceX);
                if (maxSpringCompression(distanceBetweenCubes, previousDistance))
                {
                    state = State.CONNECTED;
                    pushTime = currentTimeStep;
                }
                else
                {
                    previousDistance = distanceBetweenCubes;
                }
                break;
            case State.CONNECTED:
                if (currentTimeStep - pushTime >= DELAY)
                {
                    state = State.TOUCHING_AFTER_LOCK;
                    goto case State.TOUCHING_AFTER_LOCK;
                }
                applyForce(0, 0);
                lightCube.velocity = heavyCube.velocity;
                break;
            case State.TOUCHING_AFTER_LOCK:
                if (distanceBetweenCubes > springLength)
                {
                    state = State.POST_SPRING;
                    goto case State.POST_SPRING;
                }
                applyForce(-forceX, forceX);
                break;
            case State.POST_SPRING:
                rotate();
                break;
            default:
                break;
        }
    }

    void rotate()
    {
        debugCube(lightCube, state);

        if (lightCube.position.x <= 5 && !isRotating)
        {
```

```
            isRotating = true;
            mu = lightCube.velocity.sqrMagnitude / (RADIUS * (float) Math.PI *
GRAVITY); // v^2/2*s => m*v^2/r*pi
        }
        if (isRotating)
        {
            if (hasVirtuallyStopped(lightCube))
            {
                lightCube.velocity = Vector3.zero;
                isRotating = false;
            }

            // Einheitsvektor der Zentripetalkraft
            Vector3 ez = Vector3.Normalize(lightCube.position - vectorInit);
            Debug.Log("Mu: " + mu);
            // Zentripetalkraft: F_{Z} = m * v^2 / R
            float forceZ = lightCube.mass * lightCube.velocity.sqrMagnitude /
RADIUS;
            Vector3 forceCentriputal = -ez * forceZ;
            //Debug.Log("F_Z: " + vectorString(forceCentriputal));

            // Reibungskraft: F_{R} = g * m * mu
            Vector3 forceFriction = -Vector3.Normalize(lightCube.velocity) *
GRAVITY * lightCube.mass * mu;
            //Debug.Log("F_R: " + vectorString(forceFriction));

            // Resultierende Kraft: F_{Res} = F_{Z} + F_{R}
            Vector3 forceResulting = forceCentriputal + forceFriction;
            lightCube.AddForce(forceResulting);
        }
    }

    bool hasVirtuallyStopped(Rigidbody lightCube)
    {
        return lightCube.velocity.sqrMagnitude < 0.001f;
    }

    bool cubesAreTouchingSpring(float distanceBetweenCubes, float springLength)
    {
        return (distanceBetweenCubes <= springLength);
    }

    void applyForce(float lightCubeX, float heavyCubeX)
    {
        lightCube.AddForce(new Vector3(lightCubeX, 0, 0));
        heavyCube.AddForce(new Vector3(heavyCubeX, 0, 0));
    }

    bool maxSpringCompression(float distanceBetweenCubes, float previousDistance)
    {
        return (distanceBetweenCubes > previousDistance);
    }

    void OnApplicationQuit()
    {
        WriteTimeSeriesToCSV();
    }

    void debugCube(Rigidbody cube, State state)
    {
        Debug.Log("State = " + state
        + ", Light-Cube Velocity: " + cube.velocity.magnitude + " (X, Y, Z): " +
cube.velocity.x + ", " + cube.velocity.y + ", " + cube.velocity.z
```

```
        + ", Light-Cube Position (X, Y, Z): " + cube.position.x + ", " +
cube.position.y + ", " + cube.position.z);
    }

    String vectorString(Vector3 vector)
    {
        return " " + vector.magnitude + " (X, Y, Z): " + vector.x + ", " +
vector.y + ", " + vector.z;
    }

    void WriteTimeSeriesToCSV()
    {
        using (var streamWriter = new StreamWriter("time_series.csv"))
        {
            streamWriter.WriteLine("t, v_lightCube, v_heavyCube, x(t)_lightCube,
x(t)_heavyCube");

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

enum State
{
    ///<summary>This state is applied until the light (left) cube touches the
spring the first time.</summary>
    PRE_SPRING,
    ///<summary>This state is used while the spring is still being compressed.
</summary>
    TOUCHING_BEFORE_LOCK,
    ///<summary>This state is used while the spring is staying at max compression
for a certain time period. </summary>
    CONNECTED,
    ///<summary>This state is used once the spring is "unlocked" but the cubes
are still touching the spring. </summary>
    TOUCHING_AFTER_LOCK,
    ///<summary>This state is used once the spring spring doesn't touch the cubes
anymore. </summary>
    POST_SPRING
}
```