Impact Shock Reduction by Elastic Rubber Bands

Jaime Almanza, Yiming Yu, Yuqing Zhai, and Jiaxuan Zhang

University of Illinois at Urbana-Champaign

December 2, 2022

Abstract

Many situations in everyday life involve the protection of falling objects. In our project, we used an Arduino data collection device to measure the acceleration of a falling object protected by elastic rubber bands. With qualitative analysis and simulations written in Python and Mathematica, our results showed that the maximum instantaneous acceleration is positively correlated with both the height and number of rubber bands. We also found that this protection causes the object to do more complicated non-single-dimensional motion and generate new problems for impact shock reductions.

1 Introduction

Many scenarios in our everyday life involve impact shock reduction: from preventing a phone from breaking when it accidentally drops, to protecting passengers when an airplane lands, or even keeping a rover safely landing on an extraterrestrial surface. Many previous works have studied the acceleration level and protection methods of falling objects [1, 2, 3]. Based on previous works, we believe shock absorption by spring is one feasible shock reduction device. Therefore, our project focuses on studying the correlation between shock absorption effect with different numbers of springs and different heights of dropping. In our experiment, we used rubber bands instead of springs. In following section, we will introduce the experimental setup, the software and hardware used to collect data, and the conclusion we derived from the experimental result.

2 Math and Simulation

2.1 Simple Calculation

The rubber bands could be approximated by a spring. The "linear rubber band" which only exerts force $F = -k \cdot x$ when stretched could be approximated by introducing "unidirectional spring constant" that equals to k when stretched, and 0 otherwise.

First, we ignored any loss of energy, and performed a simple mathematical calculation based on conservation of energy to estimate the maximum stretch length x and acceleration for an object of weight m falls from the height of H, when we attach n springs to the device.

$$x = \frac{mg}{nk} + \frac{1}{nk}\sqrt{m^2g^2 + 2mg \cdot nk \cdot H} \tag{1}$$

This estimate provides maximum possible stretch distances of the rubber bands. So, we could setup our experiment accordingly to prevent the device from hitting the ground. We measured k for the linear rubber bands we used. Plugging in $k = 2.5 \times 9.8 = 24.5\,\mathrm{N\,m^{-1}}$, Equation (1) becomes (shown in Figure 1)

$$x = \frac{2m}{5n}\left(1 + \sqrt{1 + 5H\frac{n}{m}}\right) \tag{2}$$

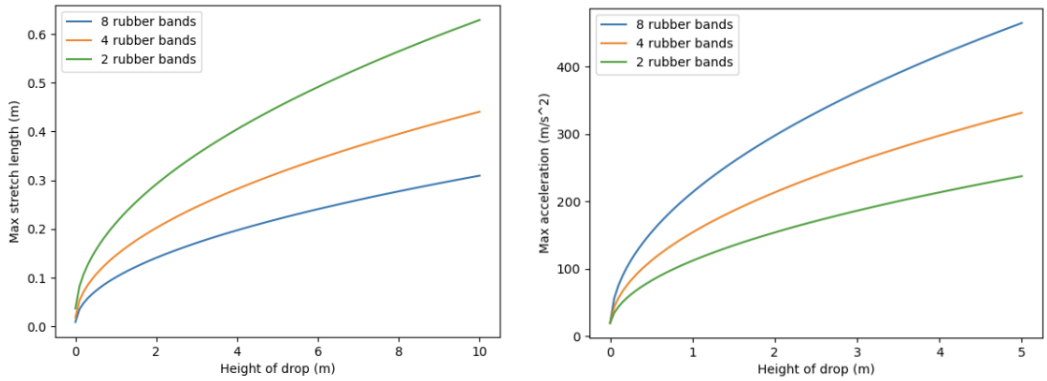

Figure 1: Left : Graph of the maximum stretch length x dependence on drop height H and number of rubber bands n for linear rubber bands with $k = 24.5\,\mathrm{N\,m^{-1}}$. Right: Graph of the maximum acceleration x dependence on drop height H and number of rubber bands n for linear rubber bands with $k = 24.5\,\mathrm{N\,m^{-1}}$

The above maximum acceleration is calculated in the scenarios when there is no energy loss. This maximum acceleration could be used as a sanity check for the simulation result. (All simulated acceleration should not exceed this value)

2.2 4th order Runge-Kutta Method

Then, we performed a more complicated calculation considering the oscillations and damping of the spring using the formula

$$m\ddot{x} + c\dot{x} + kx = -mg \tag{3}$$

Notice we used rubber bands instead of springs, so the kx term vanished when the device was above the equilibrium position.

We ran a numerical simulation in Python. Starting with the most intuitive approach where we calculate each step based on previous step:

$$F_n = k \cdot \Delta x_n \tag{4a}$$
$$a_n = \frac{F_n}{m} \tag{4b}$$
$$v_n = v_{n-1} + a_n\,dt \tag{4c}$$
$$x_n = x_{n-1} + v_n\,dt \tag{4d}$$

This simple simulation would accumulate errors over time, resulting in a graph as in Figure 2: the peaks of acceleration, velocity, and position get larger and larger over time, which violates the energy conservation and intuitions.

To reduce the error, we used a better simulation method, the 4th-order Runge-Kutta Method (RK4)[8]. If we want to simulate position x, let $f = dx/dt = v$, and

$$x_{n+1} = x_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\,dt \tag{5}$$

$k_i (i \in [4])$ are four slopes in the x-t diagram in Figure 3, which can be understood as the four "intermediate velocities" used to calculate the average v_n. k_1 is the slope at the starting point. k_2 and k_3 are both slopes at the middle point, but the formulas used to calculate them are different. k_4 is the slope of the endpoint. The formulas for k_i's are:

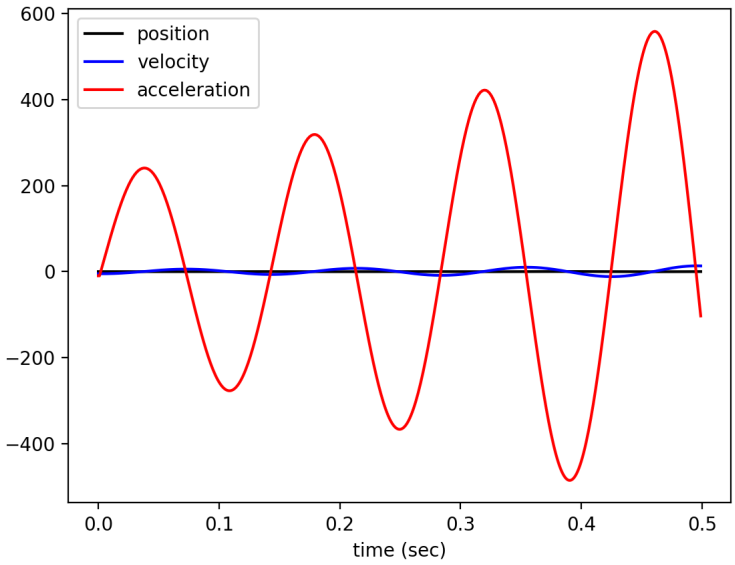

Figure 2: Our first simulation for linear rubber bands with accumulated errors proportional to dt. We let the damping factor $c = 0$ N· s/m here. As a result, the expected plot should be a sine curve. The increasing amplitudes in this graph are caused by accumulated errors.

$$k_1 = f(x_n, t_n) \tag{6a}$$

$$k_2 = f\left(x_n + k_1 \frac{dt}{2}, t_n + \frac{dt}{2}\right) \tag{6b}$$

$$k_3 = f\left(x_n + k_2 \frac{dt}{2}, t_n + \frac{dt}{2}\right) \tag{6c}$$

$$k_4 = f(x_n + k_3\, dt, t_n + dt) \tag{6d}$$

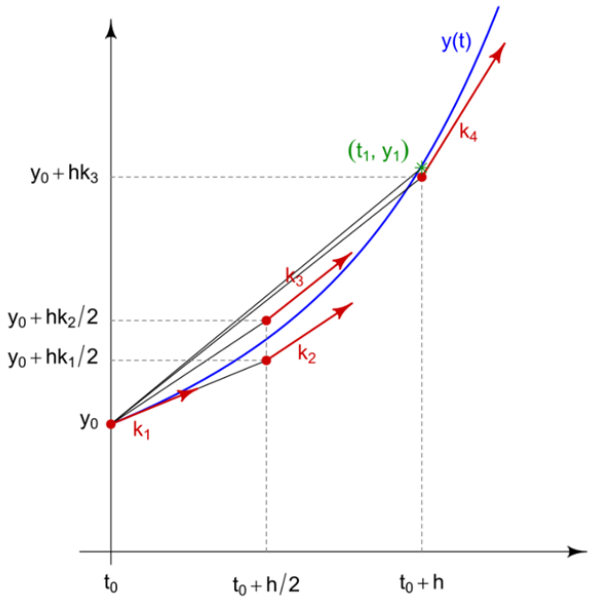

Figure 3: Diagram of RK4. In this figure, the position is labeled y, and dt is labeled h. The remaining parameters have the same notations as ours.

The total error of previous simple method is proportional to dt, while RK4 guarantees that the total error is proportional to dt^4, which is much smaller dt when dt is tiny.

Since our simulation involved the first and second derivatives of x (the v and a), RK4 method will be applied twice for each of them. The Python code for the simulation is provided in section A of the Appendix.

Figure 4 shows an example of the graphs of x, v, and a that we draw using RK4. The curves obey the laws of physics and meet our expectations. We also did simulations in Mathematica using its built-in function for solving differential equations. Detailed comparisons of the simulations with the experimental data will be shown in section 6.

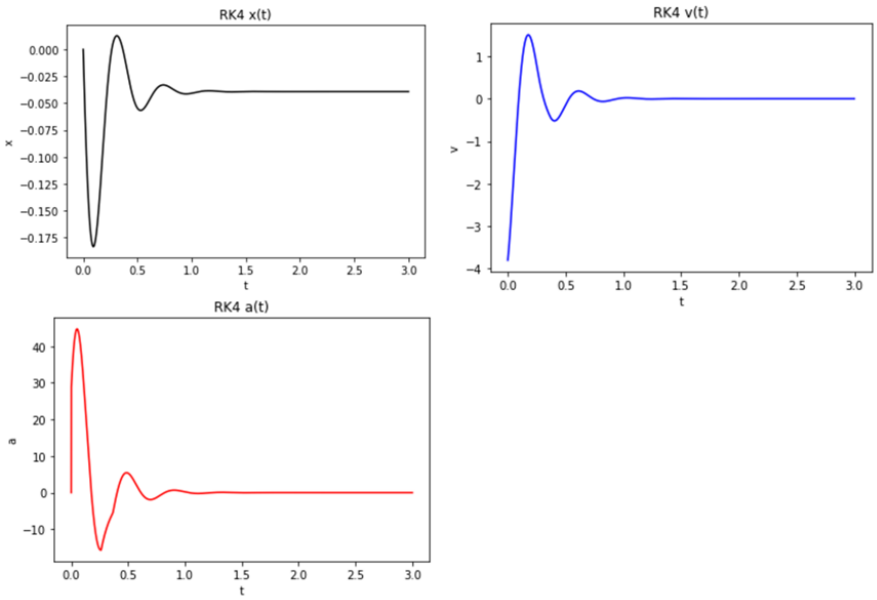

Figure 4: RK4 simulation for linear rubber bands in Python. We used $H = 0.737$ m, $c = 1$ N$\cdot$ s/m, $m = 0.1$ kg, $n = 1$, $k = 24.5$ N/m.

2.3 Force of Nonlinear Rubber Bands

We also used nonlinear rubber bands which did not behave with a linear restoring force. Shown in the Figure 6, when the system reaching equilibrium, $F_{\text{spring}} = mg$ and we could calculate the elastic force based on the mass we hang on the rubber band.

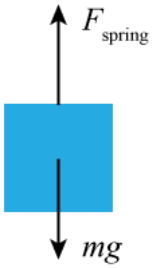

Figure 5: A model showing the force balance we used to determine the force from linear and nonlinear rubber bands.

Stretch Length (m)	Force of Gravity (N)
0.0762	0.686
0.165	1.18
0.279	1.67
0.457	2.16
0.648	2.65
0.902	3.14

Table 1: A table showing the experimental values for the stretch length of non-linear rubber bands beyond their initial length, and the force of gravity on the balancing mass.

As shown in Table 1, the values we determined from these rubber bands do not follow a linear pattern. To model the rubber band force by a function $F(x)$ that depends on the stretched distance, we plot the data in the Excel and preformed a fourth-order polynomial regression on the data, as show in the Figure 6.

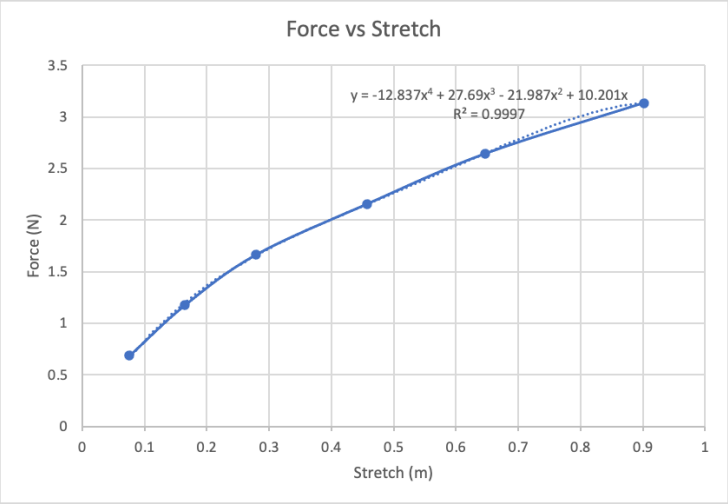

Figure 6: An Excel plot showing the force exerted by the nonlinear rubber bands we used as a function of its stretch.

The polynomial equation that models this is given by:

$$F(x) = n(-12.837x^4 + 27.69|x|^3 - 21.987x^2 + 10.201|x|) \tag{7}$$

For values past the last stretch length force we measured, we used a linear equation of the form:

$$F(x) = 1.9291|x| + 1.3965 \tag{8}$$

For multiple rubber bands tied in parallel, the resulting effective force is $F_{\text{eff}} = n \cdot F(x)$, where n is the number of rubber bands used. The corresponding differential equation governing the motion becomes

$$m\ddot{x} + c\dot{x} = -mg - F(x) \tag{9}$$

3 Hardware Design

To measure the acceleration of a falling object. We used three components:

- Accelerometer to record the acceleration. For the experiments, we used both LSM9DS1 (LSM) and H3LIS331 (LIS). LSM has a narrower range of 16g but less noise, while LIS has a wider range of 400g but more noise. Combine two accelerometers, we could measure a wide range of acceleration with decent accuracy. The LSM and LIS have 952 and 1000 Hz maximum data collection rate by design, which could be collecting around 100 data points

for a typical 0.1 seconds impact. The amount of data should be sufficient for later analysis [4, 6]. See Figure 7.

- Microcontroller to get data from accelerometer and store into SD breakout card. We used Arduino Mega 2560 initially with a clock rate of 16MHz. For our final design, we used Adafruit Feather M0 instead for the higher clock rate of 48MHz. Higher clock rate means quicker microcontroller program execution and potentially higher rate of data collection [10, 12]. See Figure 7 and 8.
- SD breakout board to store the data into. The Adafruit Feather M0 has a SD breakout on its main board already.

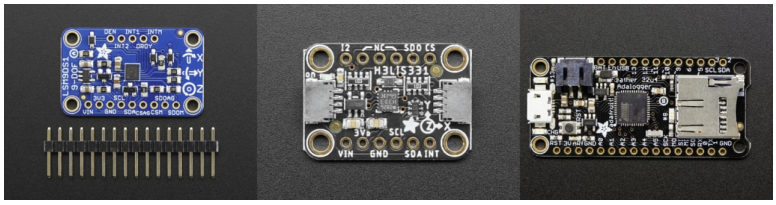

Figure 7: LSM9DS1 (Left); H3LIS331 (Middle); Feather M0 (Right)

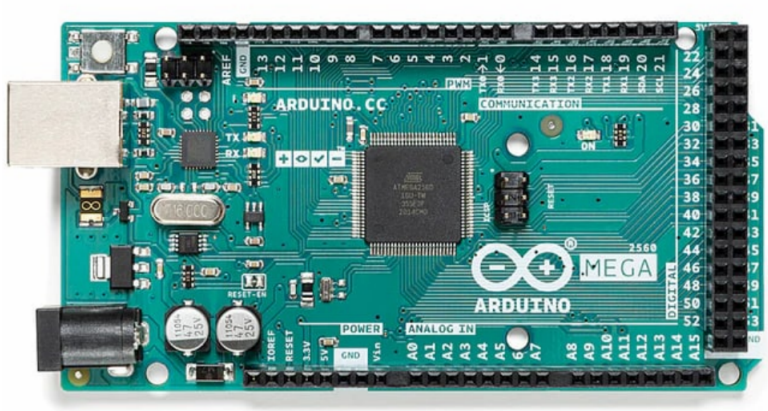

Figure 8: Mega 2560

For the communication between the microcontroller and accelerometer, there are two commonly used protocols: I2C and SPI. SPI uses 4 wires instead of 2 wires required by I2C. However, it generally has better communication speed and results in a higher data collection rate. So, we used SPI communication in our experiment. [7, 9, 5]

Our final circuit used Adafruit Feather M0, LSM9DS1 and H3LIS331. The LSM and LIS were connected via SPI. The Lithium battery was connected to provide power, and other necessary capacitors were added. Figure 10 shows the schematic of our final circuit board:

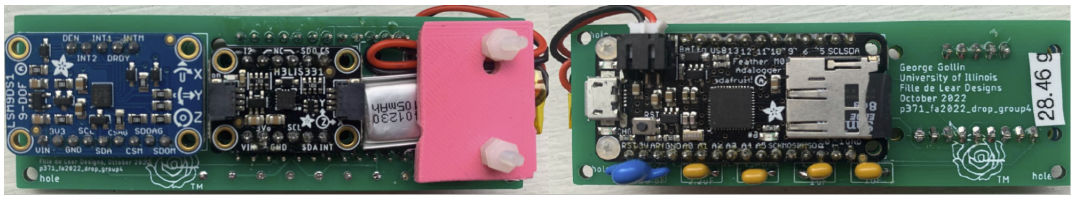

Figure 9: The two sides of the PCB board

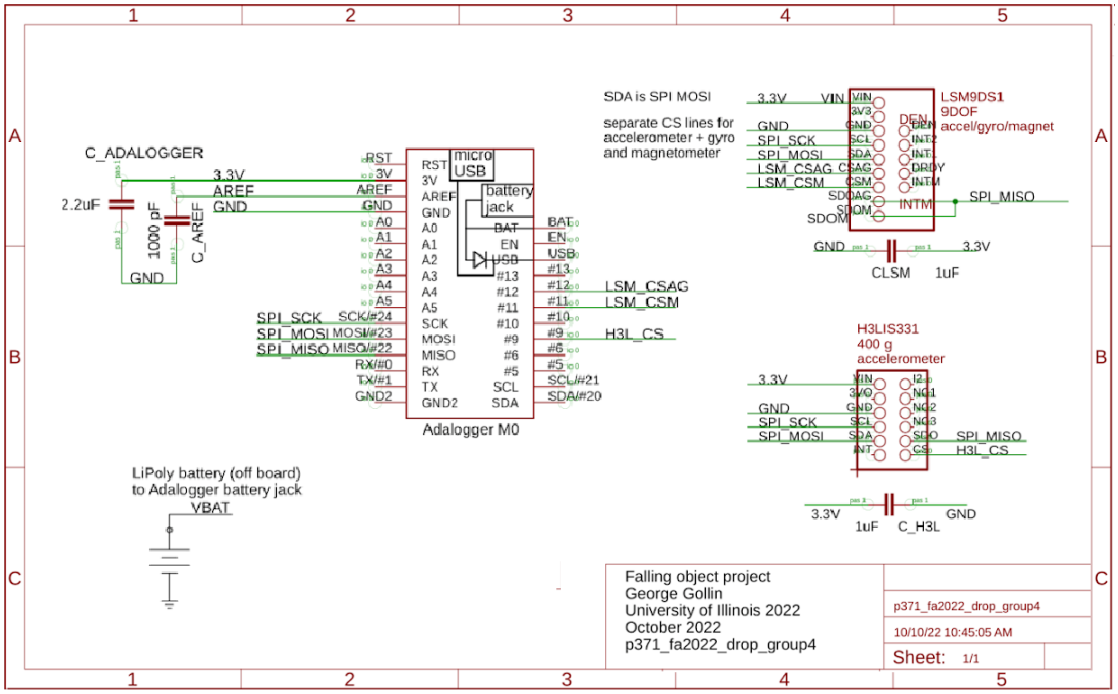

Figure 10: The schematic of the PCB board

4 Software Design

4.1 Data Acquisition Program

We wrote an Arudino data acquisition program to collect the data and store them on the SD card. The Arduino program was written in C++-like language [11]. Following the tutorial from the Adafruit website, the program was responsible for the following tasks:

- Measurement of the acceleration. The acceleration data in x, y, z direction was collected via the functions provided by the documentation of LSM9DS1 and H3LIS331 [4, 6].
- Calibration. When the box is at rest, the calibration function offset_test () in the file could collect hundreds of measurements from the accelerometer to determine its offset, this data could later be used for calibration.
- Storing the data into the SD card. Several optimization techniques were used.

Since writing to an SD card will block the program from reading the data from the accelerometer, it's best to write to the SD card as few times as possible. Therefore, a global buffer was created to store the data. The buffer could hold 512 measurements and then write them in chunks rather than individually. The logic of this program is described in Figure 11.

Then, the binary format is used. For each measurement, 7 pieces of information were collected: the time and the acceleration in x, y, and z direction for both LSM and LIS. Each of the information takes 4 bytes of storage and results in 28 bytes per measurement. The Figure 12 gives a more intuitive explanation. In comparison, if the data is stored in literal texts, not only the microcontroller needs to first change the binary data read from the accelerometer to string, but one piece of information stored in a string like "1.78301810198" could take 10 - 15 bytes of data. The binary format will not have the overhead of conversion from binary data, and also are smaller and thus quicker to write into the SD card.

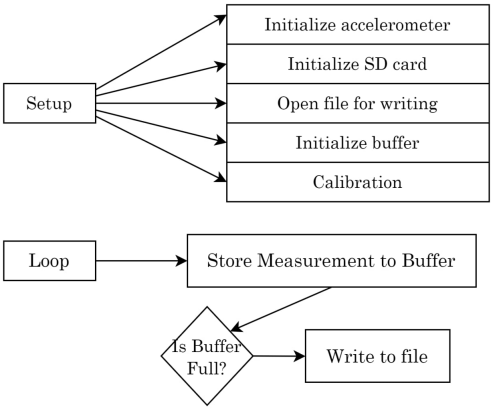

Figure 11: Flowchart of the data acquisition program.

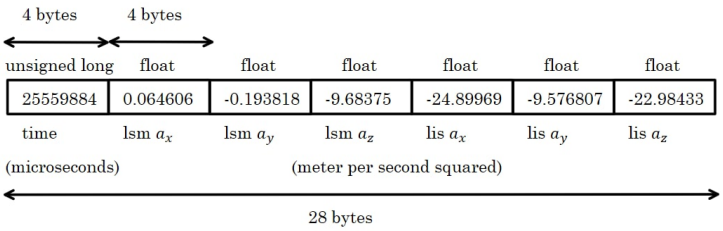

Figure 12: Binary layout of our data.

The Arduino source code is included in Appendix C.

4.2 Data Analysis and Graphing Program

We wrote a JavaScript program to analyze and display the data. The program consists of two parts. One part is running on the server side to decode the binary data into readable text, and the other part is on the browser to graph the plot according to the formatted data from the server. The user interface is shown in Figure 13. The program logic is illustrated in Figure 14.

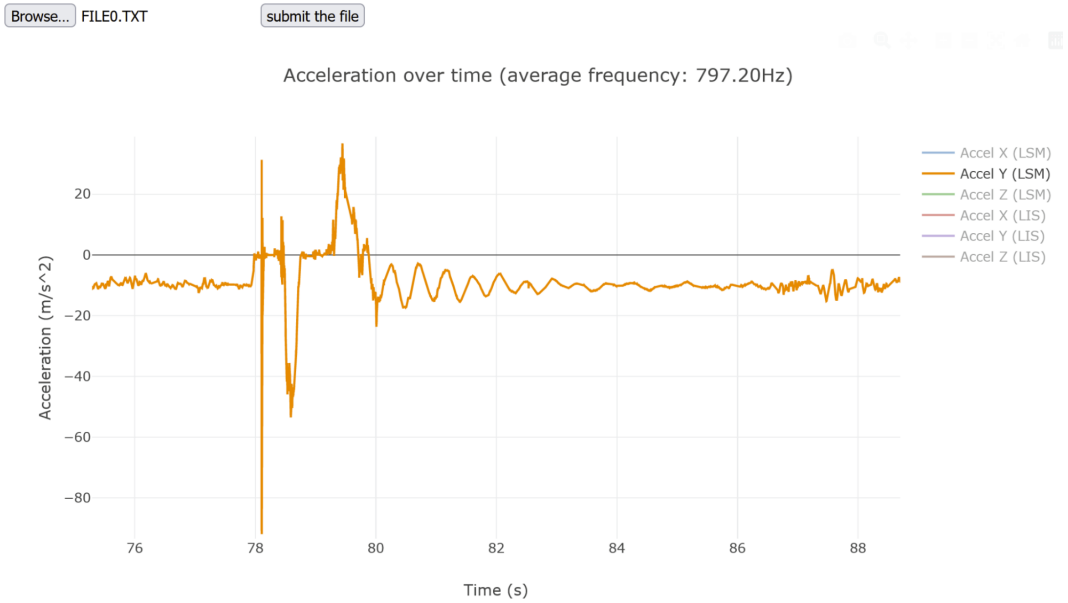

Figure 13: An example of a graph shown in the browser. A graph will be generated from a data file we submitted.

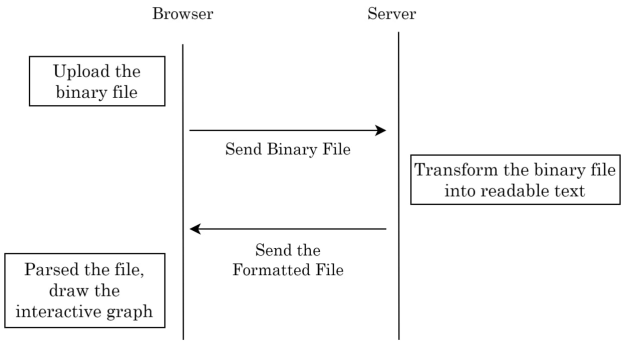

Figure 14: Flowchart of the data analysis and graphing program.

Since the source code is too lengthy for the limited space in this report, it's not included in the Appendix. For more details about this program, see the source folder.

5 Experiment

We used linear and nonlinear rubber bands as cushions to reduce the instantaneous acceleration of the falling object. The spring constant and properties of the rubber bands are already discussed in Section 2. An additional small box is used to protect the device. The mass of device is 0.028 kg and the box is 0.068 kg. The total mass is 0.096 kg The hole on the top of the box could be used to tie rubber bands. The file of the model for plastic box could be find in the source folder.

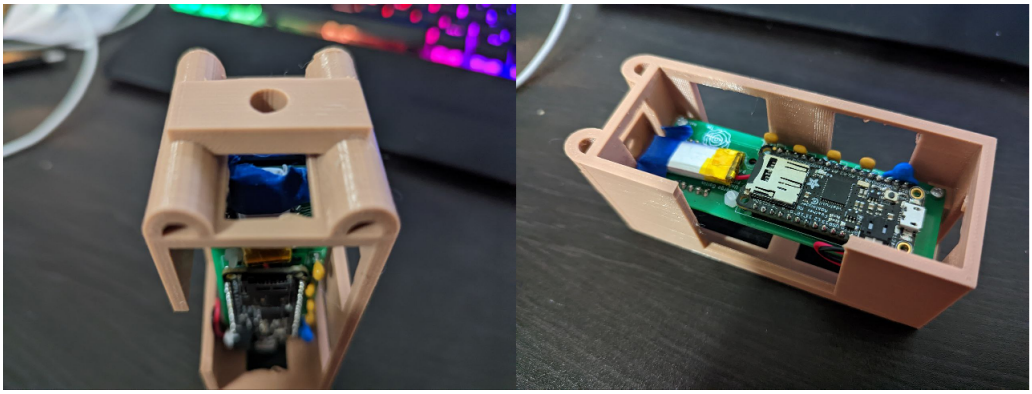

Figure 15: The top and side views of the box

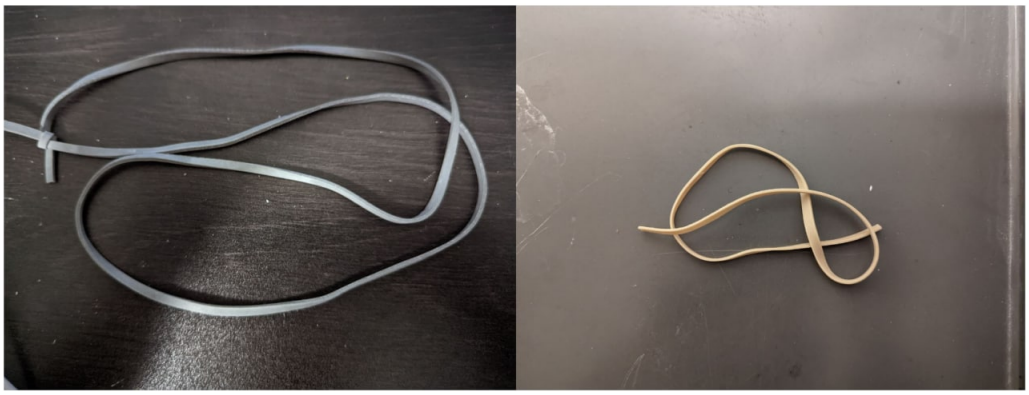

Figure 16: The linear rubber band (left) and the non-linear rubber band (right).

At first, we planned to drop both the falling object and its protection, together, as shown in Figure 17. The figure shows our original experiment setup. When dropping the tube to the ground, the device frequently bumps into the wall, resulting in unacceptable huge noise in all three dimensions of the measured acceleration.

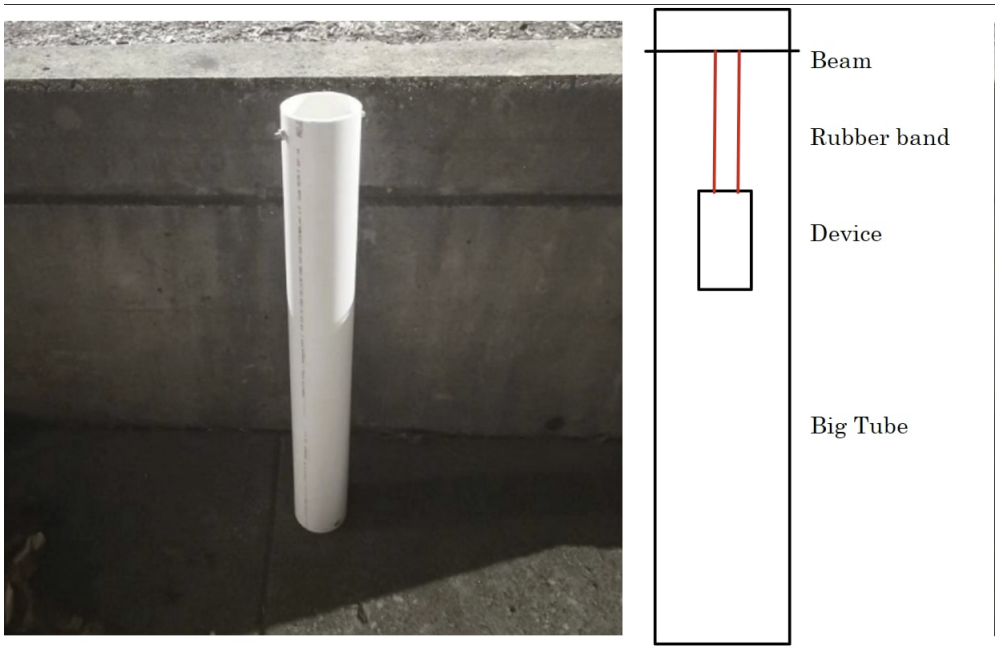

Figure 17: Our original setup. Left is a photo of the tube. Right is its internal design.

A more practical setup fixed the rubber bands on a beam and left a lot of room for the device to move freely without hitting the wall. The data was measured by the device, which is protected by the plastic box, as in Figure 18.

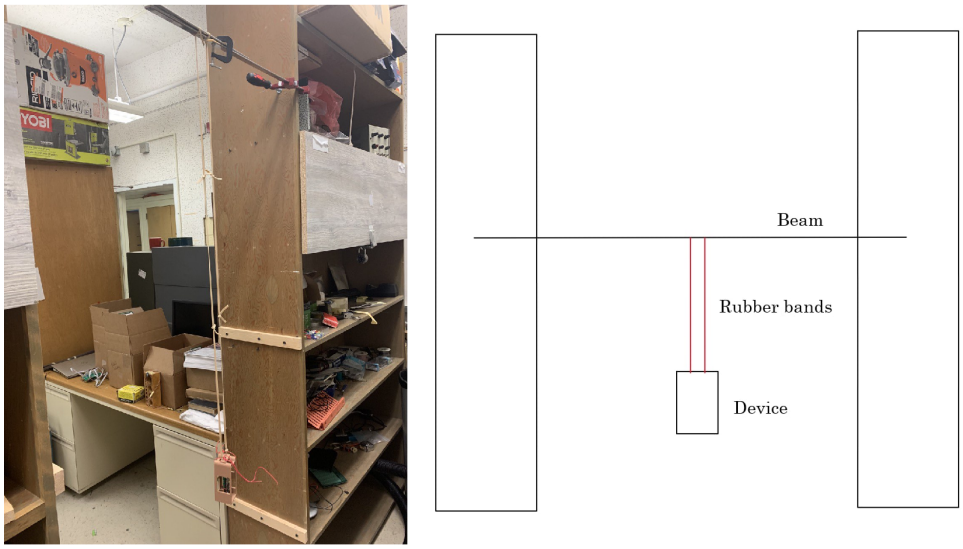

Figure 18: The new setup of the experiment.

Using our final design, we held the device to a certain height without stretching the rubber bands and then released the device. This motion is equivalent to dropping the rubber bands and device altogether in the tube (Figure 20).

Figure 19: Left: holding the device. Right: the device is falling.

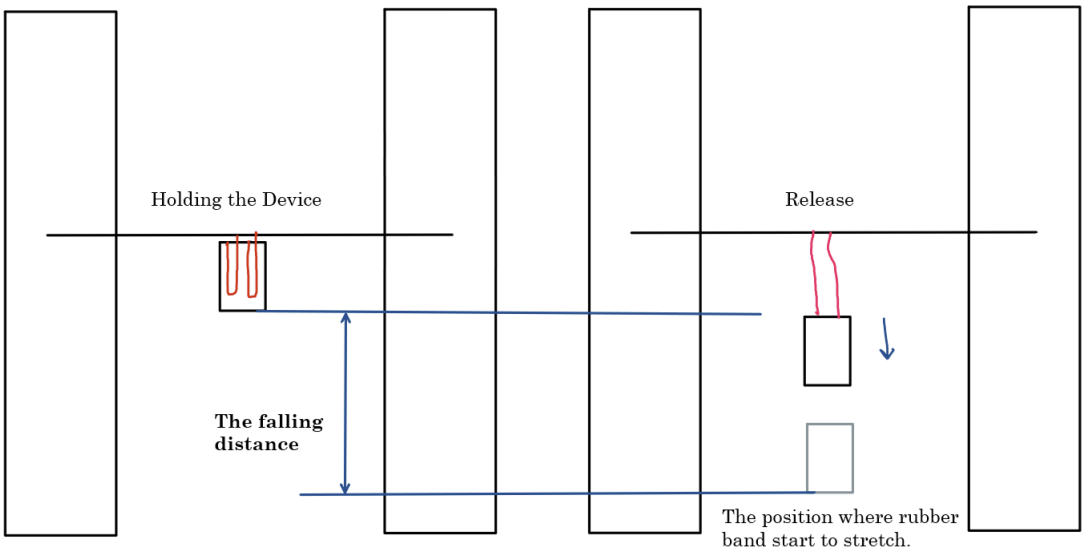

Figure 20: The illustration of the experiment

This experiment is repeated with

- Both linear and nonlinear rubber bands.
- Different number of rubber bands. For linear rubber bands, 1, 2, and 4 rubber bands are tested. For nonlinear rubber bands, 2 and 4 rubber bands are tested.
- Different heights. For linear rubber bands, the experiment is repeated with the height of 0.737 m and 1.14 m. For nonlinear rubber bands, the experiment is repeated with the height of 1.14 m and 2.16 m.

The video recording of some of the experiments can be found in the resources folder.

6 Result and Analysis

6.1 Linear Rubber Bands Dropping Data Example

We first show an example of the data we collected from the falling device protected by linear rubber bands (Figure 21).

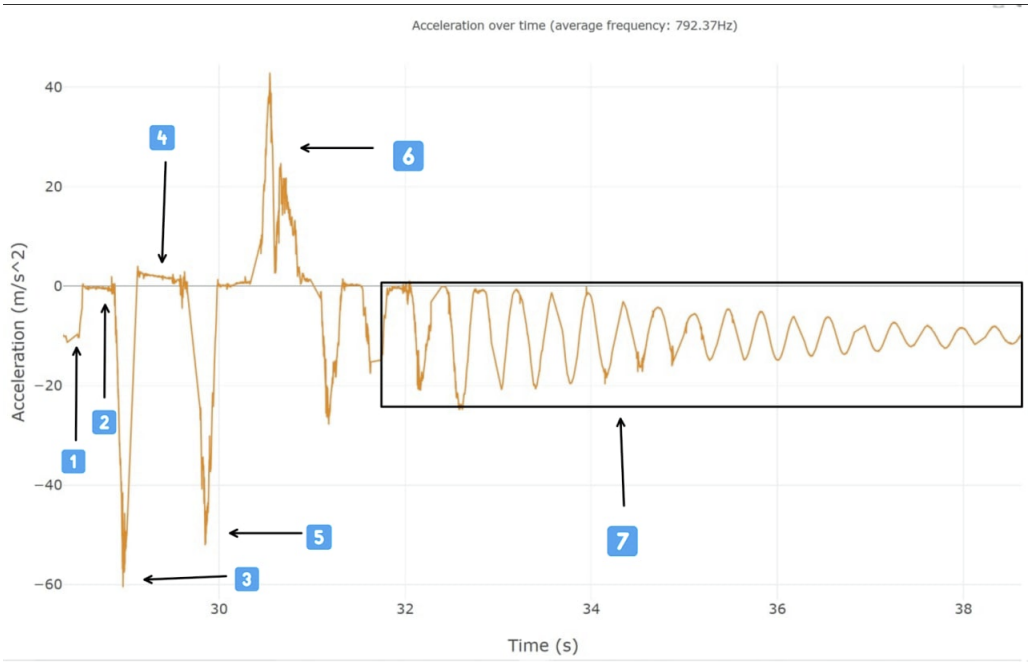

Figure 21: Vertical acceleration of the falling device protected by one linear rubber bands with $k = 24.5\,\mathrm{N\,m^{-1}}$, dropped from $H = 0.737\,\mathrm{m}$. Our data used the downward direction as the positive direction. The numbers in the graph indicate different phases of falling.

- In initial state (1), the device is at rest. The acceleration is approximately $-10\,\mathrm{m\,s^{-2}}$, i.e. gravity.
- In (2), the acceleration is zero. That means the object is undergoing free fall. The small fluctuation around zero is due to the noise of the accelerometer.
- In (3), the rubber bands are stretched to the maximum, as indicated by the maximum acceleration the device received.
- In (4), the rubber bands are no longer being stretched, and the device jumps up and again gets in the free fall state.
- In (5), the rubber bands are stretched again. But the max acceleration is smaller since the potential energy has been dissipated.
- In (6), the direction of the acceleration suddenly changes. This is because the box has flipped in the direction. See Figure 22.
- In (7), the gradual damping of the acceleration is reflected in the video recording as the pendulum motion of the rubber-band device system. See Figure 23.

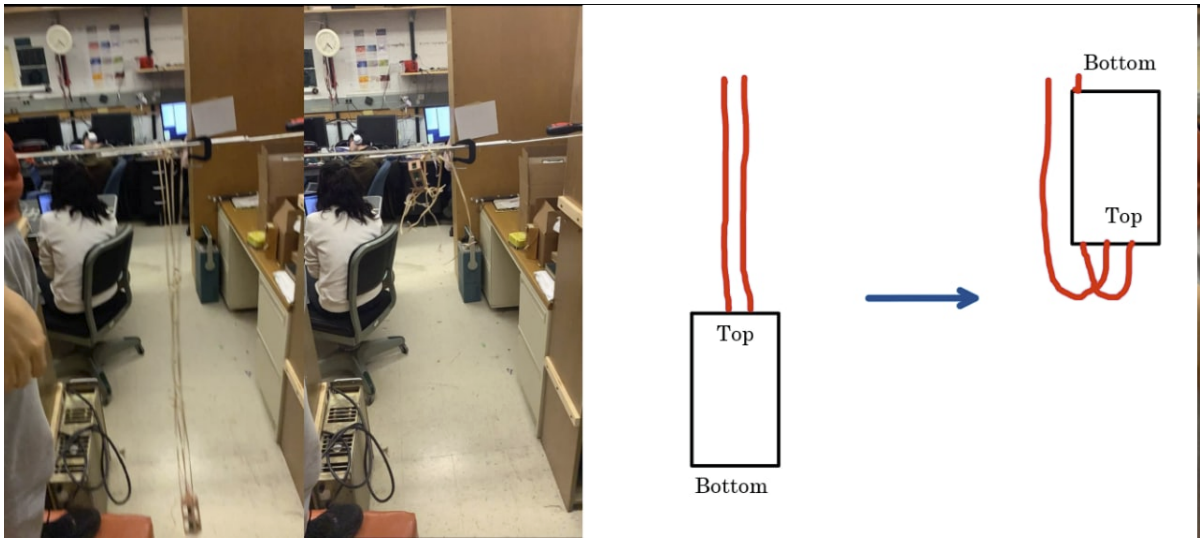

Figure 22: Left: the device being pulled up and flipped. Right: the graph indicates the device motion.

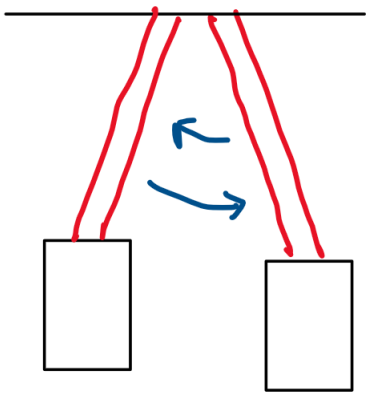

Figure 23: The device moves like a pendulum.

- In (8), the "perfect" straight line is due to the lack of data within a few milliseconds. The graph just jumps to the next available point, resulting in a straight line. The device is writing the data into the SD cards, so the data collection is paused at that time. See Figure 24 in the next page.

From these graphs, we see the maximum acceleration in each impulse is gradually decreasing, this is because the potential energy is transferred into other forms of energy in the following ways:

- Internal energy of the rubber bands due to the stretching.
- Rotational energy of the box. This rotation motion could be seen in the video recording. Figure 25 illustrates it in a sketch.
- Translational energy of the box in other directions like the pendulum motion mentioned above.

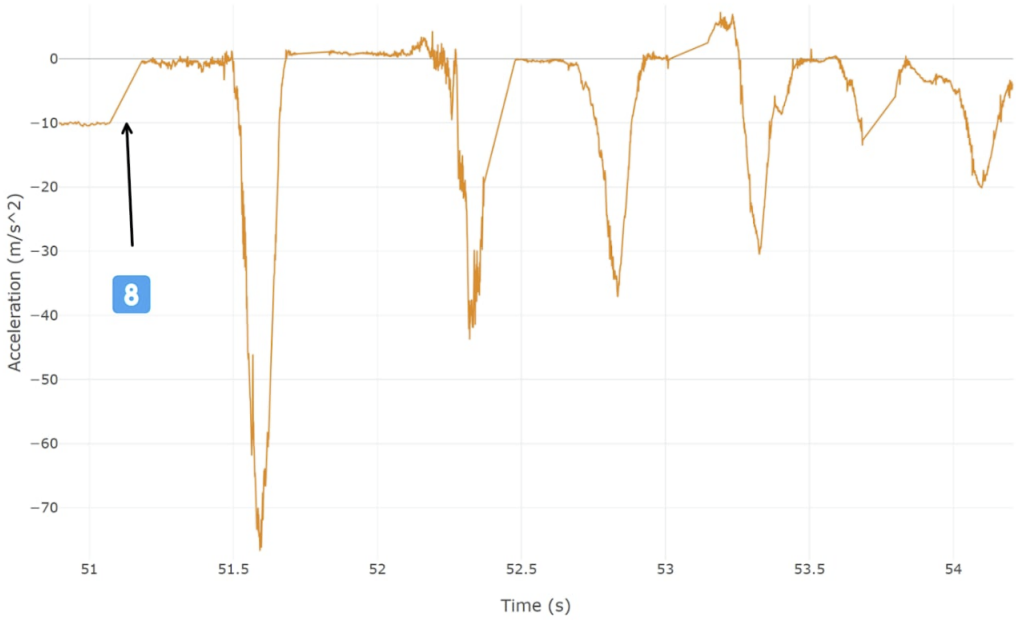

Figure 24: Vertical acceleration of the falling device protected by 4 linear rubber bands with $k = 24.5\,\mathrm{N\,m^{-1}}$, dropped from $H = 0.737\,\mathrm{m}$. Our data used the downward direction as the positive direction. The numbers in the graph indicate different phases of dropping.

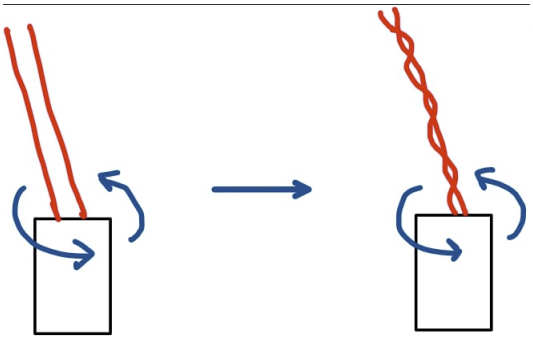

Figure 25: The device experiences a rotational motion.

6.2 Linear Rubber Bands Dropping Data

The following table contains all the first maximum accelerations obtained in the experiments for linear rubber band dropping.

Linear/Non-Linear	Height (m)	n	$a_{\max}$ ($\mathrm{m\,s^{-2}}$)
Linear	0.737	1	46.8
Linear	0.737	1	43.6
Linear	0.737	1	43.6
Linear	0.737	2	60.5
Linear	0.737	2	58.2
Linear	0.737	4	60.4
Linear	0.737	4	66.5
Linear	0.737	4	76.7
Linear	1.14	1	54.0
Linear	1.14	1	52.9
Linear	1.14	1	54.8
Linear	1.14	1	55.1
Linear	1.14	2	73.3
Linear	1.14	2	80.2
Linear	1.14	2	75.9
Linear	1.14	4	71.3
Linear	1.14	4	88.1

Table 2: First maximum acceleration of each experiment protected using linear bands. Here, n represents the number of rubber bands, and $a_{\max}$ is the maximum acceleration when the device reaches the lowest point at the first time.

6.3 Analysis of Linear Rubber Bands

Below is a table of the average first peak acceleration for each configuration. The rows are different heights we drop the device at. And the columns are different numbers of rubber bands we bind to the device in parallel.

H(m) \ n	1	2	4
0.737	44.7	59.4	67.9
1.14	54.2	76.5	79.7

Table 3: Average of first maximum accelerations, the unit is $\mathrm{m\,s^{-2}}$. Here H represents the initial height and n represents the number of rubber bands.

As shown in the Table 3, we can see two trends:

- First, maximum acceleration increases as the initial height increases. As the height increases, the initial gravitational potential energy $PE = mgh$ increases. When the object falls into the lowest point, it completely stops moving, and all of its initial energy changes into elastic potential energy $1/2kx^2$. Since k does not change, higher initial height results in a bigger stretching distance x. Therefore, the bigger elastic force $F = -kx = ma$ will result in bigger instantaneous acceleration.
- Second, maximum acceleration increases as the number of rubber bands increases. The effective elastic coefficient is the number of rubber bands in parallel times the spring constant of each individual rubber band: $k_{\text{eff}} = nk$. As the number of rubber bands increases, the effective coefficient increases. By energy conservation, the elastic potential energy $P = 1/2kx^2$ should be equal to the initial potential energy. As k is scaled by n, x should be scaled by $1/\sqrt{n}$. Now that force is given by Hooke's Law: $F = -kx$, so the acceleration is scaled by $\sqrt{n}$.

6.4 Simulation for Linear Rubber Bands

We used Equation 3 for RK4 simulation. We also run the simulation in the Mathematica. The Python code is in Appendix A. The Mathematica code is in Appendix B. The maximum acceleration predicted by both simulations is listed in the table 4

$\bar{a}_{\max}$	$n=1$				$n=2$				$n=4$			
H	Data	Max	RK4	M.	Data	T.M.	RK4	Max	Data	Max	RK4	M.
0.737	44.7	72.3	44.2	44.7	59.4	97.6	65.4	66.1	67.9	134	92.3	93.3
1.14	54.2	87.1	56.0	56.6	76.5	119	82.0	82.9	79.7	164	116	117

Table 4: Comparison of experimental average and simulated maximum acceleration ($\mathrm{m\,s^{-2}}$) of the first peak. Here $\bar{a}_{\max}$ represents the average of maximum acceleration in each configuration, and H (m) is the initial height. Max is the theoretical maximum of acceleration given by 1. M. is the abbreviation for Mathematica.

From Table 4, We could see that our RK4 and Mathematica simulation fits closely to the experimental data when there is only rubber band. The simulation data diverges from the experiment data as the number of rubber bands increases. One possible reason for this is that our simulation is one-dimensional. We only consider the motion of the device in the vertical direction. When there is only one rubber band, the real situation is close to one-dimensional motion, and the simulated acceleration is close to the experiment data. But when more rubber bands are attached, the rubber bands can bump, twist, and interfere with each other, causing more chaos. The real situation is far more complicated than a one-dimensional motion, and energy is dissipated during this process and transferred into other forms of motion mentioned in Section 6.1. Thus, the real acceleration is smaller than the simulated acceleration as the rubber band increases.

In our simulation, we used bigger damping factor c for bigger n (the number of rubber band). In this motion, the damping comes from (1) the complex movements of the bands (2) the internal resistance of the rubber bands (3) the air resistance, this terms is extremely small in comparison. As we increase the number of rubber bands, the movement of the rubber bands becomes more complicated and the internal resistance increases, so it's reasonable to use bigger c.

We could also see that both our experiment and simulation acceleration is smaller than the theoretical maximum, which justify the validity of our simulation.

The graph also shows that the acceleration is oscillate periodically at the end. This is mostly due to the pendulum motion of the rubber bands mentioned in the Section 6.1 rather than the up-and-down stretching of the rubber band. So, this periodic oscillation is not in the consideration of simulation.

One sample of the simulation graph could be seen below. More is given in the Appendix D.

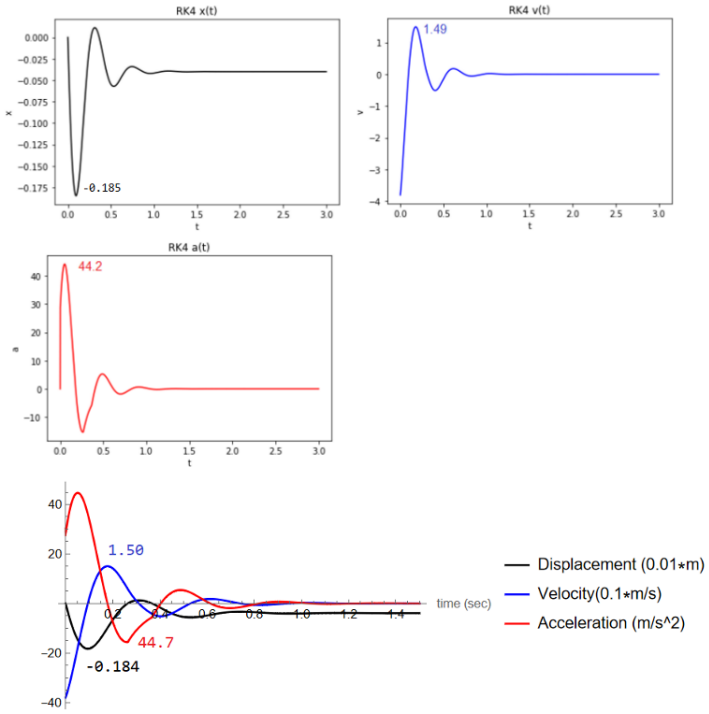

Figure 26: RK4 (above) and Mathematica (below) simulation results for the device dropped from 0.737 m protected by one linear rubber band. We used $H = 0.737\,\mathrm{m}$, $n = 1$, $c = 1\,\mathrm{N\,s\,m^{-1}}$, $m = 0.1\,\mathrm{kg}$, $k = 24.5\,\mathrm{N\,m^{-1}}$ in our functions. The values of first peaks are shown in the graphs. Notice for simulations, we used upwards direction as the positive direction, which is opposite to the data.

6.5 Non-Liner Rubber Bands Dropping Data

As in the previous section, we first give an example of a non-linear rubber band dropping experiment with an acceleration-time graph. The force of the non-linear rubber band was given in Table 1.

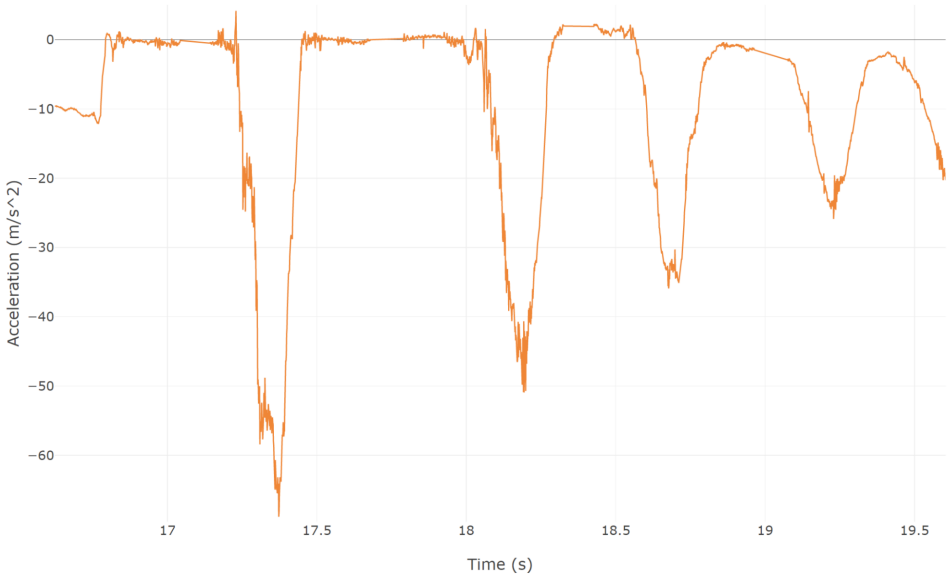

Figure 27: Vertical acceleration of the falling device protected by four non-linear rubber bands, dropped from $H = 1.14\,\mathrm{m}$. Our data used the downward direction as the positive direction.

The patterns of the graph highly resemble the ones we have in Figure 21. Again, the maximum

magnitude of acceleration is the most important data we want. The following Table 5 includes the data collected from the experiments.

Linear/Non-Linear	Height (m)	n	$a_{\max}$ ($\mathrm{m\,s^{-2}}$)
Non-Linear	1.14	2	56.7
Non-Linear	1.14	2	56.3
Non-Linear	1.14	2	59.5
Non-Linear	1.14	4	64.7
Non-Linear	1.14	4	75.5
Non-Linear	1.14	4	68.8
Non-Linear	1.14	4	83.0
Non-Linear	1.14	4	76.1
Non-Linear	2.16	4	89.6
Non-Linear	2.16	4	93.2
Non-Linear	2.16	4	77.4
Non-Linear	2.16	2	64.4

Table 5: Maximum acceleration of each experiment of non-linear bands. Here, n represents the number of rubber bands, and $a_{\max}$ is the maximum acceleration when the device reaches the lowest point at the first time.

Calculating the averages of maximum accelerations, we got Table 6.

H(m) \ n	2	4
1.14	57.50	73.62
2.16	64.4	86.73

Table 6: Average of maximum acceleration by group, the unit is $\mathrm{m\,s^{-2}}$. Here H represents the initial height and n represents the number of rubber bands.

We can observe the same relation we had for linear rubber bands: higher initial heights and more rubber bands result in greater maximum accelerations.

6.6 Simulation for Non-Linear Rubber Bands

We simulated the motion in Python and Mathematica using the same methods as the linear case. For nonlinear rubber bands, we used Equation 7 for the force exerted. The comparison of our simulation and experimental results are in Table 7.

$\bar{a}_{\max}$ ($\mathrm{m\,s^{-2}}$)	$n=2$			$n=4$		
Height (m)	Data	RK4	Mathematica	Data	RK4	Mathematica
1.14	57.5	56.9	56.9	73.6	66.9	66.9
2.16	64.4	81.6	81.6	86.7	92.6	92.6

Table 7: Comparison of experimental average and simulated maximum acceleration of the first peak. In our simulations, we use $\mathrm{c} = 1.4\,\mathrm{N\,m^{-1}}$.

Notice that although the simulation results are close to our experimental data, the errors are bigger than in the linear case. The main reason for this is that we did the experiment for nonlinear rubber bands first, and we had some operations that increased the systematic error. For example, we did not let the device fall as vertically as possible. This operation was improved when we did the experiment for linear bands later.

The code we used is in Appendix A and B. Figure 28 shows a sample of our simulation result.

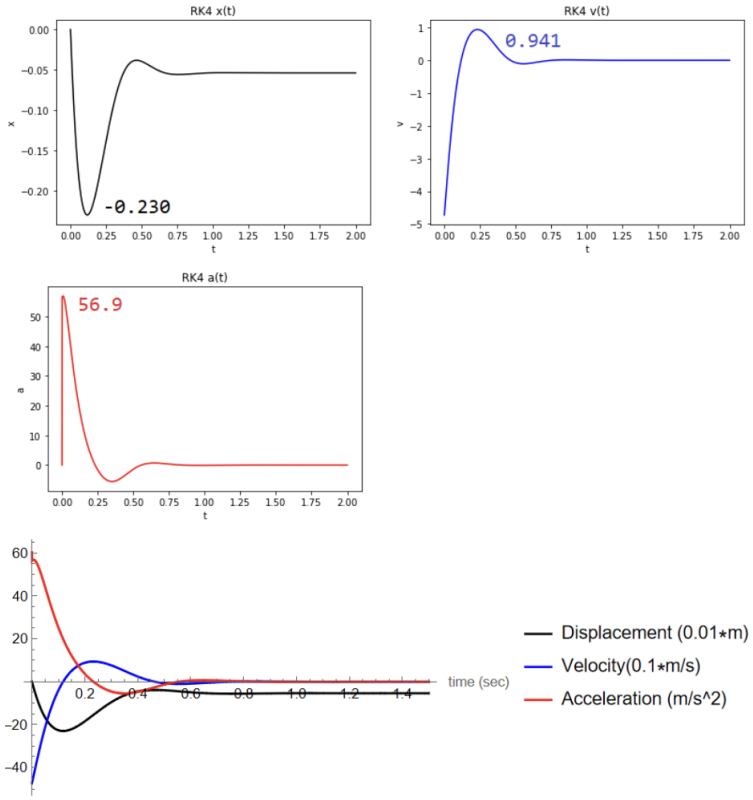

Figure 28: RK4 (above) and Mathematica (below) simulation for $H = 1.14\,\mathrm{m}$, $n = 2$.

7 Discussion

We measured the acceleration of a falling object protected by a rubber band, which can be regarded as a one-side spring. From our data and analysis, we can conclude that the first peak acceleration increase with the height H and number of rubber bands n, which matches our intuition. We also derive a method of simulation the motion of an object with such protection using numerical simulations, which can be useful for similar future studies.

One problem in our experiment is that the motion of the falling object is not one-dimensional. In the original experiment setup, the object frequently bumps into the tube and produce completely noisy and useless data. We only come up with the better version of the setup at the end of the semester, so there is no time to improve the design and confine the object into one dimensional motion. The increasing complexity makes the experiment data diverge from simulation data which assumes the object only move in one direction. Also, since we only come with the better version at the end of the semester, we only test two kinds of rubber with few different configurations. The lack of data make it harder to model the motions. Together, due to the complexity in the motion and lack of data, we were unable to make strong, quantitative conclusions.

In our simulations, we assume the damping only depends on the velocity of the motion v. We have mentioned the possible source of damping in the Section 6.4. The damping could also depends on x, a and other factors. The simulation based on this simple damping would result in different maximum acceleration than the experiment result.

The data acquisition program also using blocking SD write to collect data. This could result in a loss of data for a small period of time mentioned in the Section 6.1. We have attempted to change the program to perform non-blocking write but failed. The loss of data could potentially cause some important features in the graphs missing.

We hope the future study could improve the experiment setup, simulation model, and data acquisition program, and they could repeat experiment in more different configurations to get a better quantitative understanding of the motion and reach a stronger, quantitative conclusion in the end.

8 Conclusion

We studied the impact shock reduction provided by linear and nonlinear rubber bands in our project. We found that the maximum acceleration increases with the height H and number of rubber bands n. Even though using spring as cushion reduce the maximum acceleration, it could complicate the motion of the object. This indicates that in real life, we need to worry about the damage caused by the object hitting other objects around it when we want to apply this protection. We also derived methods of simulating the motion of an object. Although our simulations are one-dimensional, it fits quite well with the experimental data. Because of limitations of time and conditions, we could only do mostly qualitative analysis on our results. We hope our experiment and discoveries could be inspiring and beneficial for more thorough studies on the protection for falling objects using spring and elastic rubber bands.

A RK4 Simulation in Python

We used RK4 for simulation in Python. The code for linear rubber band is as follows:

```
H = 1.14
n = 4
cv = 1.
# 1 for H = 0.7377, 1,14, n = 1,2
# 1.85 for H = 0.7377, 1.14, n = 4
c2 = 0
m = 0.1
g = 9.8
k = n*2.5*g

x0 = 0
v0 = -(2*g*H)**.5
a0 = 0

from numpy import zeros, linspace
import matplotlib.pyplot as plt
def f1(x, v, t):
    f1 = v
    return f1

def f2(x, v, t):
    if x > 0:
        f2 = - cv/m*v - c2/m*v**2 - g     # acceleration
        return f2
    else:
        f2 = - cv/m*v - c2/m*v**2 - g - (k/m)*x
        return f2

T = 3

dt = 0.0001
N = int(round(T/dt))

t = linspace(0, T, N+1)

x = zeros(N+1)
v = zeros(N+1)
a = zeros(N+1)

x[0] = x0
v[0] = v0
a[0] = a0

for n in range(N):
 k11 = f1(x[n],v[n],t[n])
 k12 = f2(x[n],v[n],t[n])
 k21 = f1(x[n] + k11 * dt/2, v[n] + k12*dt/2,t[n]+dt/2)
 k22 = f2(x[n]+k11*dt/2,v[n]+k12*dt/2,t[n]+dt/2)
 k31 = f1(x[n]+k21*dt/2,v[n]+k22*dt/2,t[n]+dt/2)
 k32 = f2(x[n]+k21*dt/2,v[n]+k22*dt/2,t[n]+dt/2)

 k41 = f1(x[n]+k31*dt,v[n]+k32*dt,t[n]+dt)
 k42 = f2(x[n]+k31*dt,v[n]+k32*dt,t[n]+dt)
 x[n+1] = x[n] + (dt/6)*(k11 + 2*k21 + 2*k31 + k41)
 v[n+1] = v[n] + (dt/6)*(k12 + 2*k22 + 2*k32 + k42)
```

```
 a[n+1] = k12

plt.plot(t, x, 'k', label='RK4')
plt.title("RK4 x(t)")
plt.xlabel("t")
plt.ylabel("x")
plt.show()

plt.plot(t, v, 'b', label='RK4')
plt.title("RK4 v(t)")
plt.xlabel("t")
plt.ylabel("v")
plt.show()

plt.plot(t, a, 'r', label='RK4')
plt.title("RK4 a(t)")
plt.xlabel("t")
plt.ylabel("a")
plt.show()
```

The next is for the non-linear rubber bands:

```
H = 1.14
#H = 2.16
nr = 4  # number of rubber bands
cv = 1.4
c2 = 0
m = 0.1
g = 9.8

x0 = 0
v0 = -(2*g*H)**.5
a0 = 0

from numpy import zeros, linspace
import matplotlib.pyplot as plt
def f1(x, v, t):
    f1 = v
    return f1

def f2(x, v, t):
    F = nr*(-12.837*x**4 + 27.69*abs(x**3) - 21.987*x**2 + 10.201*abs(x))
    if x > 0:
        f2 = - cv/m*v - c2/m*v**2 - g     # acceleration
        return f2
    else:
        f2 = - cv/m*v - c2/m*v**2 - g + F/m
        return f2

T = 2

dt = 0.0001
N = int(round(T/dt))

t = linspace(0, T, N+1)

x = zeros(N+1)
v = zeros(N+1)
a = zeros(N+1)
```

```
x[0] = x0
v[0] = v0
a[0] = a0

for n in range(N):
 k11 = f1(x[n],v[n],t[n])
 k12 = f2(x[n],v[n],t[n])
 k21 = f1(x[n] + k11 * dt/2, v[n] + k12*dt/2,t[n]+dt/2)
 k22 = f2(x[n]+k11*dt/2,v[n]+k12*dt/2,t[n]+dt/2)
 k31 = f1(x[n]+k21*dt/2,v[n]+k22*dt/2,t[n]+dt/2)
 k32 = f2(x[n]+k21*dt/2,v[n]+k22*dt/2,t[n]+dt/2)

 k41 = f1(x[n]+k31*dt,v[n]+k32*dt,t[n]+dt)
 k42 = f2(x[n]+k31*dt,v[n]+k32*dt,t[n]+dt)
 x[n+1] = x[n] + (dt/6)*(k11 + 2*k21 + 2*k31 + k41)
 v[n+1] = v[n] + (dt/6)*(k12 + 2*k22 + 2*k32 + k42)
 a[n+1] = k12

plt.plot(t, x, 'k', label='RK4')
plt.title("RK4 x(t)")
plt.xlabel("t")
plt.ylabel("x")
plt.show()

plt.plot(t, v, 'b', label='RK4')
plt.title("RK4 v(t)")
plt.xlabel("t")
plt.ylabel("v")
plt.show()

plt.plot(t, a, 'r', label='RK4')
plt.title("RK4 a(t)")
plt.xlabel("t")
plt.ylabel("a")
plt.show()

print(max(a))
```

B Simulation in Mathematica

The code we used in Mathematica for simulating linear rubber bands is:

```
m = 0.1; H = 1.14; cv = 1.85; g = 9.8; n = 4; k =
  n*25; v = -(2*g*H)^.5;
Clear[u]
Clear[y]
f[x_] := If[x < 0, k*Abs[x], 0]
(*Plot[{f[-z]},{z,-1,1},PlotRange->Full]*)
p = NDSolve[{f[y[t]] - m g - cv (y'[t]) (**Sign[y'[t]]*)== m y''[t],
     y'[0] == v, y[0] == 0}, y, {t, 0, 100}];
pos[t_] := Evaluate[y[t] /. p];
vol[t_] := D[pos[u], u] /. u -> t;
acc[t_] := D[D[pos[u], u], u] /. u -> t;
Plot[{100*pos[t], 10*vol[t], acc[t]}, {t, 0, 1.5}, PlotRange -> Full,
  AxesLabel -> {"time (sec)"},
  PlotLegends -> {"Displacement (0.01*m)", "Velocity(0.1*m/s)",
    "Acceleration (m/s^2)"}, PlotStyle -> {Black, Blue, Red}]
FindMinimum[pos[t], {t, 0, 1.5}]
```

```
FindMaximum[vol[t], {t, 0, 1.5}]
FindMaximum[acc[t], {t, 0, 1.5}]
```

The other one for the non-linear rubber bands is:

```
m = 0.1; H = 1.14; cv = 1.4; g = 9.8; n = 2; k = n*25; v = -(2*g*H)^.5;
Clear[u]
Clear[y]
f[x_] :=
  If[x < 0,
   If[Abs[x] < 0.6477,
    n *(-12.837*Abs[x]^4 + 27.69*Abs[x]^3 - 21.987 Abs[x]^2 +
       10.201 Abs[x])], 0];
p = NDSolve[{f[y[t]] - m g - cv (y'[t]) (**Sign[y'[t]]*)-
      cn*Sign[y'[t]] == m y''[t], y'[0] == -(2*9.8*H)^.5, y[0] == 0},
   y, {t, 0, 100}];
pos[t_] := Evaluate[y[t] /. p]
vol[t_] := D[pos[u], u] /. u -> t;
acc[t_] := D[D[pos[u], u], u] /. u -> t
Plot[{100*pos[t], 10*vol[t], acc[t]}, {t, 0, 1.5}, PlotRange -> Full,
 AxesLabel -> {"time (sec)"},
 PlotLegends -> {"Displacement (0.01*m)", "Velocity(0.1*m/s)",
   "Acceleration (m/s^2)"}, PlotStyle -> {Black, Blue, Red}]
FindMaximum[acc[t], {t, 0.001, 1.5}]
FindMinimum[pos[t], {t, 0, 1.5}]
FindMaximum[vol[t], {t, 0, 1.5}]
```

C Arduino Code

```
// Falling Object Group 4, Based on George Gollin (Professor) code.

// Headers
#include <SD.h>
#include <SPI.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM9DS1.h>
#include <Adafruit_H3LIS331.h>


////////////////////////// LSM9DS1 parameters //////////////////////
// PINS
#define LSM9DS1_XGCS 12
#define LSM9DS1_MCS 11

const double G_EARTH = 9.8067;

// CONSTS
const int LSM9DS1_SAMPLE_INTERVAL = 1051; // in micros (frequency = 952 Hz)
const double LSM9DS1_SATURATION = 15.9 * G_EARTH;

// SPI
Adafruit_LSM9DS1 lsm = Adafruit_LSM9DS1(LSM9DS1_XGCS, LSM9DS1_MCS);

// Time counter
uint32_t lsm_last_sampled_time = 0;

////////////////////////// H3LIS331 parameters //////////////////////
// PINS
```

```
#define H3LIS331_CS 9
#define H3LIS331_SCK 24
#define H3LIS331_MISO 22
#define H3LIS331_MOSI 23

// CONSTS
const int H3LIS331_SAMPLE_INTERVAL = 1000; // in micros (frequency = 1000 Hz)
const double H3LIS331_SATURATION = 15 * G_EARTH;
// SPI
Adafruit_H3LIS331 lis = Adafruit_H3LIS331();

// Time counter
uint32_t lis_last_sampled_time = 0;

////////////////////// GLOBALS ////////////////////////////////////

// for storing
struct offset_t {
  double ax = 0.0, ay = 0.0, az = 0.0;
  double ax_rms = 0.0, ay_rms = 0.0, az_rms = 0.0;
};

struct data_t {
  unsigned long time = 0UL;
  float ax_lsm = 0.0f, ay_lsm = 0.0f, az_lsm = 0.0f;
  float ax_lis = 0.0f, ay_lis = 0.0f, az_lis = 0.0f;
};

// CONSTANTS
const int LED_GREEN_PIN = 8;
const int LED_RED_PIN = 13;
const int LOOPS = 300000; // 5 minutes

////////////////////// SD card parameters ////////////////////////
#define SD_PIN 4


Sd2Card card;
File file;
File count_file;
String FILE_NAME;
const char COUNT_FILE_NAME[15] = "COUNT.TXT";

// buffers for non-blocking write
char buffer[sizeof(data_t) * 512];


////////////////////// DECLARATIONS ////////////////////////////////
void open_file();
void offset_test(int);
offset_t offset_lsm_test(int);
offset_t offset_lis_test(int);
void delay_lsm();
void delay_lis();
void setup_lis();
void setup_lsm();
void setup_sd();
void blink(uint32_t, int = 50);
```

```
void setup() {
  Serial.begin(115200);

  uint32_t start = millis();
  // wait for 10 seconds for Serial otherwise just keep going
  while (!Serial && millis() - start < 10000) delay(1);

  Serial.println("------------ Setting Up ------------");

  setup_sd();
  setup_lsm();
  setup_lis();

  // open file
  open_file();

  // offset test
  offset_test(1);
  blink(1145);
}

void loop() {
  static char* buffer_ptr = buffer;
  static int index = 0;
  static data_t data;

  unsigned long micros_now = micros();

  if (index > LOOPS) {
    // just close the file and stop
    file.close();
    while (1) {
      blink(10000, 100);
    }
  } else {
    index ++;
  }

  // if (using_lsm) {
  sensors_event_t accel, mag, gyro, temp;
  delay_lsm();

  lsm.getEvent(&accel, &mag, &gyro, &temp);
  lsm.readAccel();

  data.time = micros_now;
  data.ax_lsm = float(accel.acceleration.x);
  data.ay_lsm = float(accel.acceleration.y);
  data.az_lsm = float(accel.acceleration.z);

  // } else {
  sensors_event_t accel_two;
  delay_lis();

  lis.getEvent(&accel_two);

  data.ax_lis = float(accel_two.acceleration.x);
```

```
  data.ay_lis = float(accel_two.acceleration.y);
  data.az_lis = float(accel_two.acceleration.z);

  memcpy(buffer_ptr, &data, sizeof(data_t));
  buffer_ptr += sizeof(data_t);

  // write to file

  if (buffer_ptr - buffer >= sizeof(buffer)) {
    Serial.println("Writing to file... ");
    file.write(buffer, sizeof(buffer));
    file.close();
    file = SD.open(FILE_NAME, FILE_WRITE);
    buffer_ptr = buffer;
  }
}

void open_file() {
  Serial.println("---------- Opening Count File ----------");

  if (!SD.exists(COUNT_FILE_NAME)) {
    count_file = SD.open(COUNT_FILE_NAME, FILE_WRITE);
    count_file.print("0");
    count_file.close();
  }

  count_file = SD.open(COUNT_FILE_NAME);

  String count_string = "";

  if (count_file) {
    while (count_file.available()) {
      Serial.println("Am I here?");
      count_string += (char)count_file.read();
    }
  }

  int count = count_string.toInt();

  Serial.println(count_string);
  Serial.println(count);

  count_file.close();

  SD.remove(COUNT_FILE_NAME);
  count_file = SD.open(COUNT_FILE_NAME, FILE_WRITE);
  count_file.print(String(count + 1));
  count_file.close();

  Serial.println("---------- Opening File ----------");

  FILE_NAME = String("FILE") + count + ".TXT";

  if (SD.exists(FILE_NAME)) {
    Serial.println("File exists, deleting...");
    SD.remove(FILE_NAME);
  }
```

```
  file = SD.open(FILE_NAME, FILE_WRITE);
  if (!file) {
    Serial.println("Failed to open file");
    blink(1000, 150);
    while (1);
  }
}

void offset_test(int times) {
  Serial.println("---------- Calculating Offset ----------");

  auto offset_lsm = offset_lsm_test(times);
  auto offset_lis = offset_lis_test(times);

  file.write((char*) &offset_lsm, sizeof(offset_lsm));
  file.write((char*) &offset_lis, sizeof(offset_lis));
}

offset_t offset_lsm_test(int times) {
  Serial.println("Calculating LSM Offset...");

  sensors_event_t accel, mag, gyro, temp;

  offset_t lsm_offset {};

  for (int i = 0; i < times; i++) {
    delay_lsm();
    lsm.getEvent(&accel, &mag, &gyro, &temp);
    lsm.readAccel();

    lsm_offset.ax += accel.acceleration.x;
    lsm_offset.ay += accel.acceleration.y;
    lsm_offset.az += accel.acceleration.z;
    lsm_offset.ax_rms += accel.acceleration.x * accel.acceleration.x;
    lsm_offset.ay_rms += accel.acceleration.y * accel.acceleration.y;
    lsm_offset.az_rms += accel.acceleration.z * accel.acceleration.z;
  }

  lsm_offset.ax /= times;
  lsm_offset.ay /= times;
  lsm_offset.az /= times;
  lsm_offset.ax_rms /= times;
  lsm_offset.ay_rms /= times;
  lsm_offset.az_rms /= times;

  lsm_offset.ax_rms = sqrt(lsm_offset.ax_rms - lsm_offset.ax * lsm_offset.ax);
  lsm_offset.ay_rms = sqrt(lsm_offset.ay_rms - lsm_offset.ay * lsm_offset.ay);
  lsm_offset.az_rms = sqrt(lsm_offset.az_rms - lsm_offset.az * lsm_offset.az);

  // Az should minus the gravity
  lsm_offset.az -= G_EARTH;

  return lsm_offset;
}

offset_t offset_lis_test(int times) {
  Serial.println("Calculating LIS Offset...");
```

```
  sensors_event_t accel;

  offset_t lis_offset {};

  for (int i = 0; i < times; i++) {
    delay_lis();
    lis.getEvent(&accel);

    lis_offset.ax += accel.acceleration.x;
    lis_offset.ay += accel.acceleration.y;
    lis_offset.az += accel.acceleration.z;
    lis_offset.ax_rms += accel.acceleration.x * accel.acceleration.x;
    lis_offset.ay_rms += accel.acceleration.y * accel.acceleration.y;
    lis_offset.az_rms += accel.acceleration.z * accel.acceleration.z;
  }

  lis_offset.ax /= times;
  lis_offset.ay /= times;
  lis_offset.az /= times;
  lis_offset.ax_rms /= times;
  lis_offset.ay_rms /= times;
  lis_offset.az_rms /= times;

  lis_offset.ax_rms = sqrt(lis_offset.ax_rms - lis_offset.ax * lis_offset.ax);
  lis_offset.ay_rms = sqrt(lis_offset.ay_rms - lis_offset.ay * lis_offset.ay);
  lis_offset.az_rms = sqrt(lis_offset.az_rms - lis_offset.az * lis_offset.az);

  // Az should minus the gravity
  lis_offset.az -= G_EARTH;

  return lis_offset;
}

void delay_lsm() {
  while (micros() - lsm_last_sampled_time < LSM9DS1_SAMPLE_INTERVAL);
  lsm_last_sampled_time = micros();
}

void delay_lis() {
  while (micros() - lis_last_sampled_time < H3LIS331_SAMPLE_INTERVAL);
  lis_last_sampled_time = micros();
}

void setup_lis() {
  if (!lis.begin_SPI(H3LIS331_CS)) {
    Serial.println("Unable to Initialize H3LIS331, Exiting...");
    while (1) yield();
  }

  lis.setRange(H3LIS331_RANGE_400_G);
  lis.setDataRate(LIS331_DATARATE_1000_HZ);
}

void setup_lsm() {
  if (!lsm.begin()) {
    Serial.println("Unable to Initialize LSM9DS1, Exiting...");
    while (1);
  }
```

```
  lsm.setupAccel(lsm.LSM9DS1_ACCELRANGE_16G);
  lsm.setupMag(lsm.LSM9DS1_MAGGAIN_4GAUSS);
  lsm.setupGyro(lsm.LSM9DS1_GYROSCALE_245DPS);
}

void setup_sd() {
  if (!card.init(SPI_FULL_SPEED, SD_PIN)) {
    Serial.println("Unable to Initialize SD, Exiting...");
    while (1);
  }

  SD.begin(SD_PIN);
}

void blink(uint32_t duration, int interval) {
  Serial.print("---------- Blinking green and red lights for ");
  Serial.print(duration / 1000);
  Serial.print(" seconds ----------\n");

  uint32_t start = millis();

  pinMode(LED_GREEN_PIN, OUTPUT);
  pinMode(LED_RED_PIN, OUTPUT);

  while (millis() - start < duration) {
    digitalWrite(LED_GREEN_PIN, HIGH);
    digitalWrite(LED_RED_PIN, LOW);

    delay(interval);

    digitalWrite(LED_GREEN_PIN, LOW);
    digitalWrite(LED_RED_PIN, HIGH);

    delay(interval);
  }

  // restore the led
  digitalWrite(LED_GREEN_PIN, LOW);
  digitalWrite(LED_RED_PIN, LOW);
}
```

D Simulation Graphs for Linear Rubber Bands

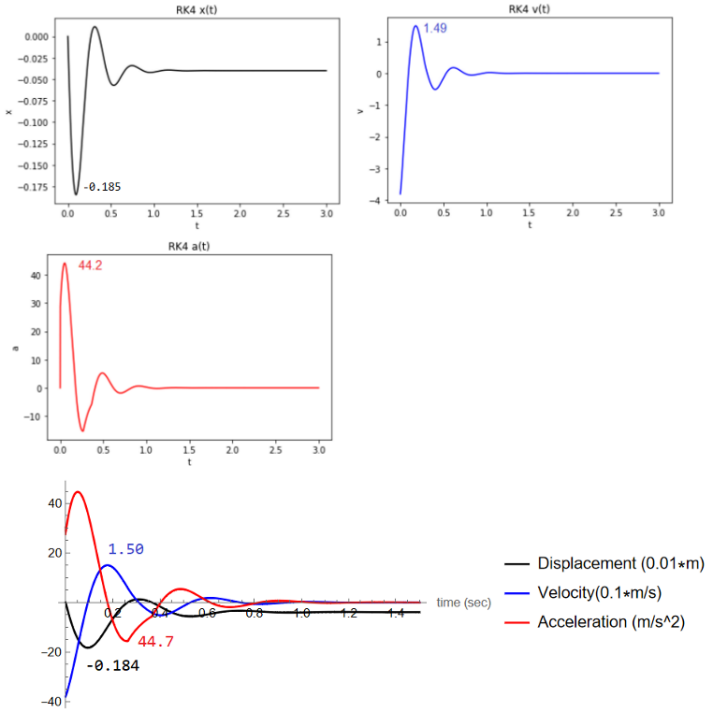

Figure 29: RK4 (above) and Mathematica (below) simulation results for the device dropped from 0.737 m protected by one linear rubber band. We used $H = 0.737\,\mathrm{m}$, $n = 1$, $c = 1\,\mathrm{N\,s\,m^{-1}}$, $m = 0.1\,\mathrm{kg}$, $k = 24.5\,\mathrm{N\,m^{-1}}$ in our functions. The values of first peaks are shown in the graphs. Notice for simulations, we used upwards direction as the positive direction, which is opposite to the data.

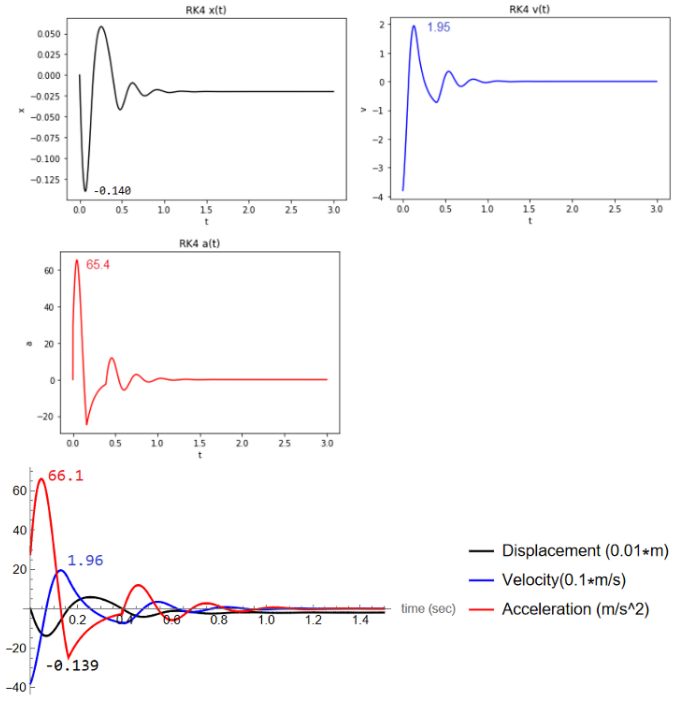

Figure 30: RK4 (above) and Mathematica (below) simulation results for the device dropped from 0.737 m protected by two linear rubber bands. $H = 0.737\,\mathrm{m}$, $n = 2$, $c = 1\,\mathrm{N\,s\,m^{-1}}$, $m = 0.1\,\mathrm{kg}$, $k = 24.5\,\mathrm{N\,m^{-1}}$

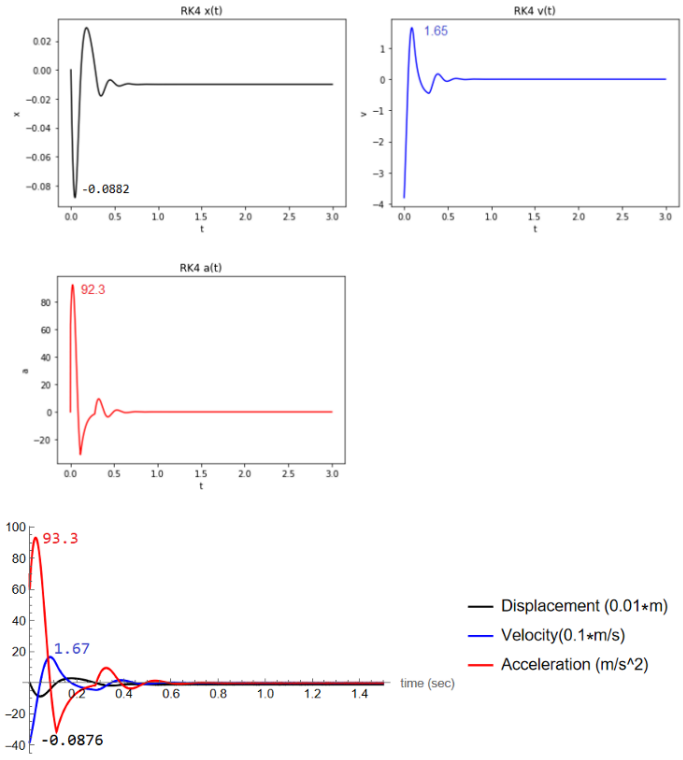

Figure 31: RK4 (above) and Mathematica (below) simulation results for the device dropped from 0.737 m protected by four linear rubber bands. $H = 0.737\,\mathrm{m}$, $n = 4$, $c = 1\,\mathrm{N\,s\,m^{-1}}$, $m = 0.1\,\mathrm{kg}$, $k = 24.5\,\mathrm{N\,m^{-1}}$

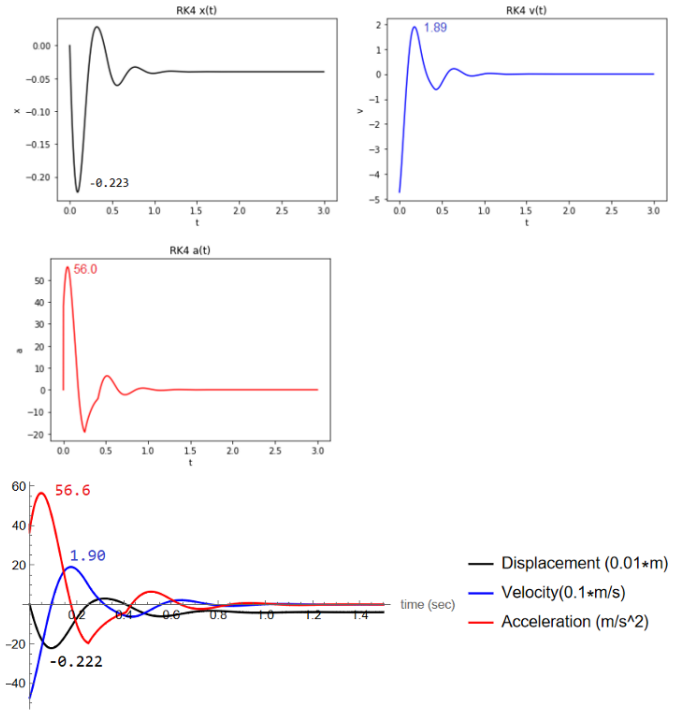

Figure 32: RK4 (above) and Mathematica (below) simulation results for the device dropped from 1.14 m protected by one linear rubber band. $H = 1.14\,\mathrm{m}$, $n = 1$, $c = 1\,\mathrm{N\,s\,m^{-1}}$, $m = 0.1\,\mathrm{kg}$, $k = 24.5\,\mathrm{N\,m^{-1}}$

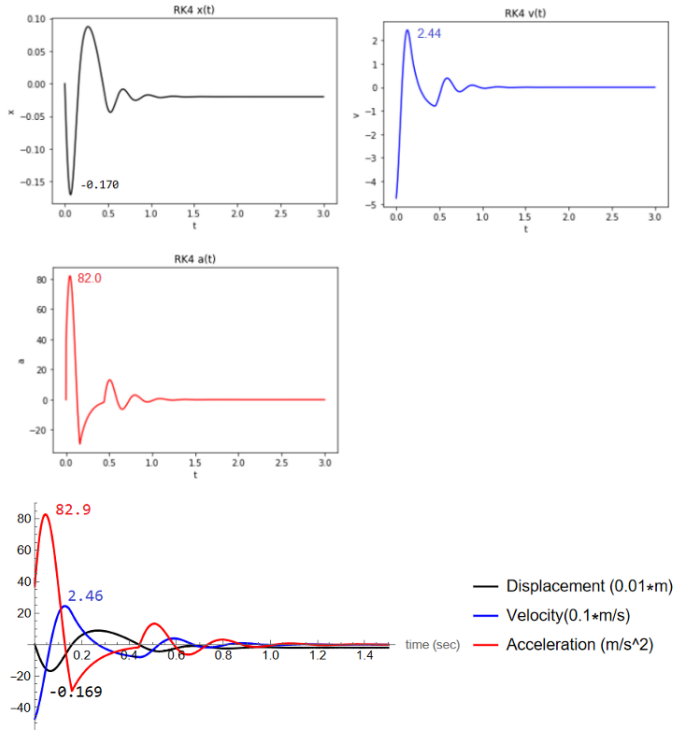

Figure 33: RK4 (above) and Mathematica (below) simulation results for the device dropped from 1.14 m protected by two linear rubber bands. $H = 1.14\,\mathrm{m}$, $n = 2$, $c = 1\,\mathrm{N\,s\,m^{-1}}$, $m = 0.1\,\mathrm{kg}$, $k = 24.5\,\mathrm{N\,m^{-1}}$

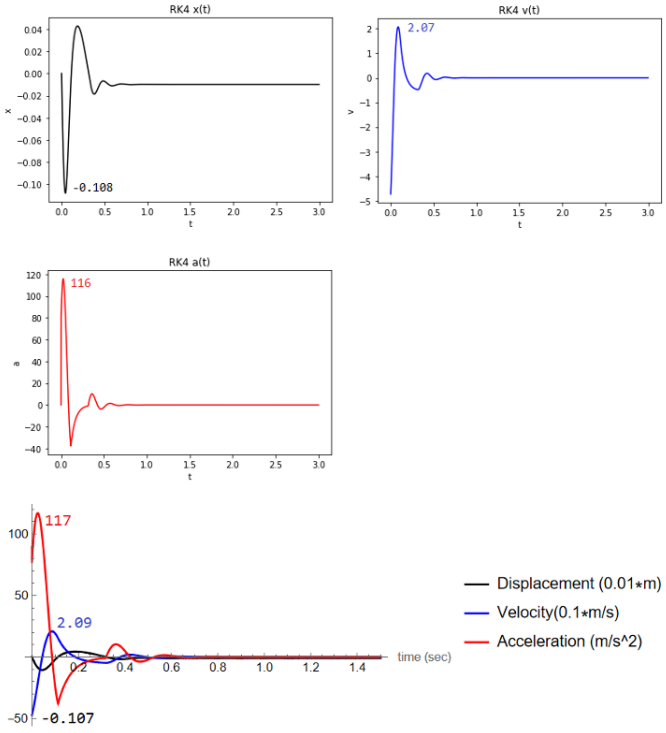

Figure 34: RK4 (above) and Mathematica (below) simulation results for the device dropped from 1.14 m protected by four linear rubber bands. $H = 1.14\,\mathrm{m}$, $n = 4$, $c = 1\,\mathrm{N\,s\,m^{-1}}$, $m = 0.1\,\mathrm{kg}$, $k = 24.5\,\mathrm{N\,m^{-1}}$

E Simulation Graphs for Non-Linear Rubber Bands

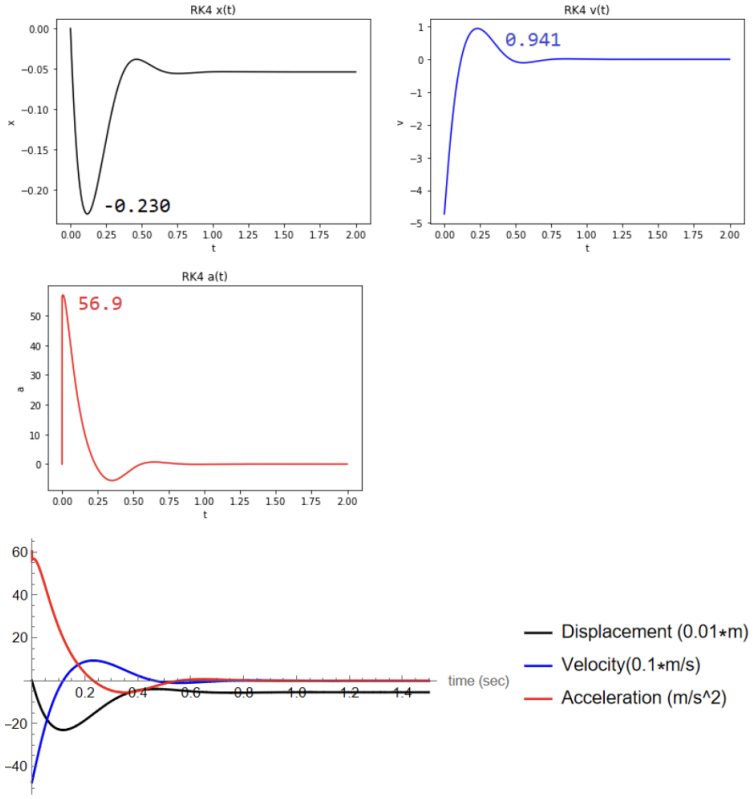

Figure 35: RK4 (above) and Mathematica (below) simulation for $H = 1.14\,\text{m}$, $n = 2$.

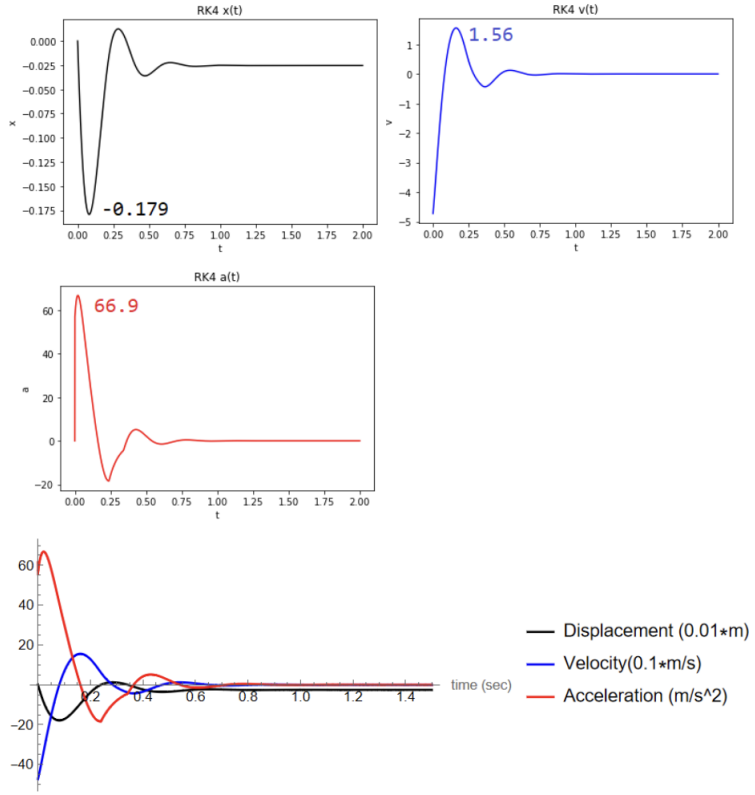

Figure 36: RK4 (above) and Mathematica (below) simulation for $H = 1.14\,\text{m}$, $n = 4$.

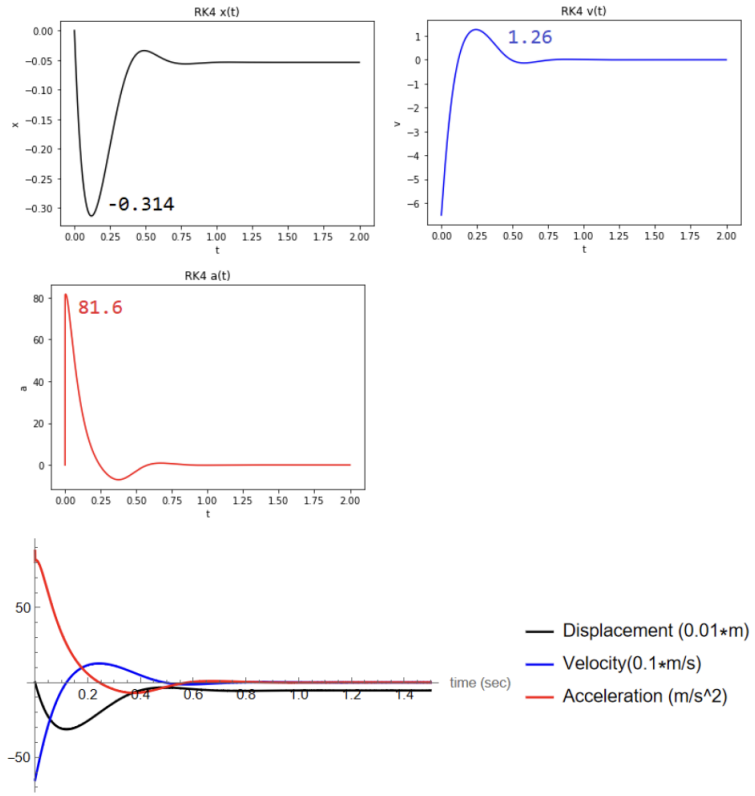

Figure 37: RK4 (above) and Mathematica (below) simulation for $H = 2.16\,\text{m}$, $n = 2$.

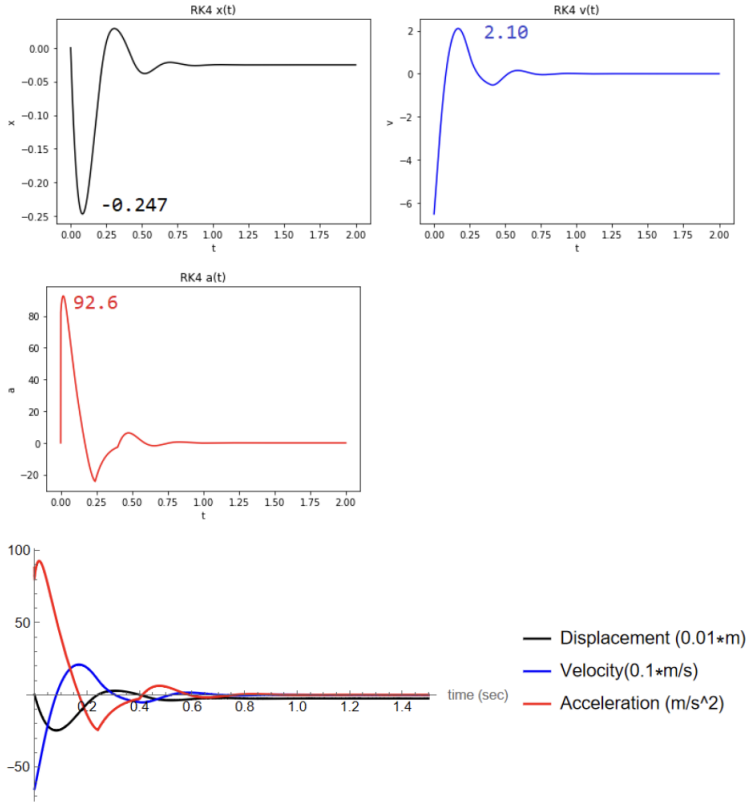

Figure 38: RK4 (above) and Mathematica (below) simulation for $H = 2.16\,\mathrm{m}$, $n = 4$.

References

[1] E. Suhir. "Dynamic response of a one-degree-of-freedom linear system to a shock load during drop tests: Effect of viscous damping". In: *IEEE Transactions on Components, Packaging, and Manufacturing Technology: Part A* 19.3 (1996), pp. 435–440. DOI: 10.1109/95.536846.

[2] PM Leonhardt. "Acceleration levels of dropped objects". In: *Endevco Corporation* (2001).

[3] R Lernbeiss and M Plöchl. "Simulation model of an aircraft landing gear considering elastic properties of the shock absorber". In: *Proceedings of the Institution of Mechanical Engineers, Part K: Journal of Multi-body Dynamics* 221.1 (2007), pp. 77–86. DOI: 10.1243/1464419JMBD63. eprint: https://doi.org/10.1243/1464419JMBD63. URL: https://doi.org/10.1243/1464419JMBD63.

[4] Adafruit. 2017. URL: https://learn.adafruit.com/adafruit-lsm9ds1-accelerometer-plus-gyro-plus-magnetometer-9-dof-breakout/overview.

[5] Amlendra. *Difference between I2C and SPI (I2C vs SPI), you should know.* 2018. URL: https://aticleworld.com/difference-between-i2c-and-spi/.

[6] Adafruit. 2020. URL: https://learn.adafruit.com/adafruit-h3lis331-and-lis331hh-high-g-3-axis-accelerometers.

[7] I²C. *I²C — Wikipedia, The Free Encyclopedia.* 2022. URL: https://en.wikipedia.org/wiki/I%C2%B2C.

[8] Runge–Kutta methods. *Runge–Kutta methods — Wikipedia, The Free Encyclopedia.* 2022. URL: https://en.wikipedia.org/wiki/Runge%E2%80%93Kutta_methods.

[9] Serial Peripheral Interface. *Serial Peripheral Interface — Wikipedia, The Free Encyclopedia.* 2022. URL: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface.

[10] Arduino. URL: https://docs.arduino.cc/hardware/mega-2560.

[11] Arduino. URL: https://www.arduino.cc/reference/en/.

[12] Arudino. URL: https://docs.arduino.cc/retired/boards/arduino-m0.