# Motion of a falling slinky

Bharat Haria and Abdullah Khawer
Wallington County Grammar School

January 2024

# 1 Introduction and Practical Set Up

For our practical arrangement there were a few conditions that needed to be satisfied. For us to be able to observe the full motion of the falling of the slinky we needed quite the height to drop it from. To achieve a greater height the person dropping the slinky was placed upon a chair. For use as reference before we started dropping the slinky a picture of the wall we were dropping against was taken with the ruler's blue tacked to that wall. The reason we could not have the rulers against the wall was as we were planning to use a computer as the means to collect data from the video. By having as few objects in the frame as possible less post processing was needed to be done on the video. Another thing done in order to support this is to simplify the colour of the background. We chose a white wall to film this next to and the black skirting boards were covered up by paper. In some attempts we attempted to remove shadows of the slinky by holding a flashlight by our hand to remove the shadow. Additionally, in some attempts, we taped the top and bottom of the slinky with black electrical tape to make it easier to see at which point the slinky was released and when it made contact with the ground. In order to keep consistency in the footage a rudimentary tripod was mad by camping a phone to a chair on order to make sure all pictures and videos in a single session where taken from the same position

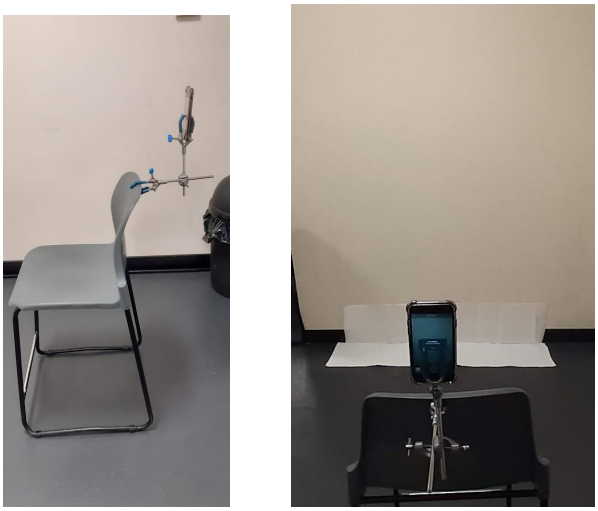

Figure 1: Practical set up

# 2 Video Processing

In order to have data as accurate as possible we write a program in order to read data out of the video automatically. Advantages of this method is that data samples will be much more accurate and we will be able to take many more samples of the data from the video. Below is the program that was written in order to process the video.

```
1  import cv2 as cv
2  import numpy as np
3  from turtle import textinput
4
5  vid = textinput("Video", "What is the name of the video")
6
7  cap = cv.VideoCapture(vid)
8
9  size = (int(cap.get(3)), int(cap.get(4)))
10 result = cv.VideoWriter("tracers.avi", cv.VideoWriter_fourcc
       (*"MJPG"), 60, size)
11
12 print(size)
13 currframe = 0
14 output = ""
15 while cap.isOpened():
16     ret, frame = cap.read()
17     if ret:
18         new = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)
19         bw = cv.threshold(new, 128, 255, cv.THRESH_BINARY)
       [1]
20         arr = np.array(bw)
21         loc = np.where(arr == 0)
22         top = loc[0][0], loc[1][0]
23         bottom = loc[0][len(loc[0]) - 1], loc[1][len(loc[1])
        - 1]
24         frame = cv.line(
25             frame,
26             (0, loc[0][0]),
27             (size[0] - 1, loc[0][0]),
28             color=[0, 0, 255],
29             thickness=2,
30         )
31         frame = cv.line(
32             frame,
33             (0, loc[0][len(loc[0]) - 1]),
34             (size[0] - 1, loc[0][len(loc[0]) - 1]),
35             color=[0, 0, 255],
36             thickness=2,
37         )
38         cv.putText(
```

```
39              frame, f"top {top},bottom = {bottom}", (50, 50),
      cv.FONT_HERSHEY_PLAIN, 2, 0
40          )
41          frame = cv.resize(frame, size)
42          result.write(frame)
43          cv.imshow("Black And White", bw)
44          cv.imshow("Tracers", frame)
45          cv.waitKey(100)
46          print(f"{currframe}: Top = {size[1] - top[0]} Bottom
      = {size[1]  - bottom[0]}")
47          output += f"{currframe} {size[1] - top[0]} {size[1]
      - bottom[0]} \n"
48          currframe += 1
49
50      else:
51          break
52
53 with open("data.txt", "w") as f:
54      f.write(output)
55
56 cap.release()
57 result.release()
58
59 cv.destroyAllWindows()
```

This code was written in python and uses the OpenCV image processing library as well as numpy in order to do the video processing. The program starts by opening a videostream which in this case would be the video file. Each frame is read and processed one by one. First the image is cast to grayscale then it is turned to black and white by means of a binary threshold. Pixels with a value above are turned white and below are black. The code then filters the image to get the indexes of all black pixels. The first and last black pixels should be the top and bottom of the object. This program has two outputs. It outputs a file with the frame (representing time) and the pixel y values of the top and bottom of the object giving the distance. Below is an example of a data table from the program

| Frame | Top | Bottom |
|---|---|---|
| 51 | 996 | 866 |
| 52 | 989 | 866 |
| 53 | 981 | 867 |
| 54 | 974 | 867 |
| 55 | 967 | 867 |

In addition to this a video is produced with red bars going across the original video file tracking the top and bottom of the object. This provides an easy visual of whether or not the tracking has been successful.

It is due to the fact that the program requires that during the casting to black and white that only the slinky is above the threshold that we need to make

sure that the Slinky stands out as much as possible. On initial runs this was done in past processing but attempts were made to instead darken core parts of the slinky and introduce extra lighting in order to remove shadows. Many separate runs were made with each of these ideas in play in order to see what would work and whether adding in additional markers would make it easier to process the video over simply editing the video to remove interference from other elements.

During the practical we noticed an interesting behaviour in the motion of the falling slinky. The entire slinky did not all fall at once, instead the top of the slinky would start to fall with the bottom of the slinky remaining suspended in the air. The bottom of the slinky would only start to fall when the slinky was fully compressed at which point the whole slinky would start to fall as a single unit. The slinky also did not always drop in a straight line. In our initial takes with a totally unmodified slinky it fell in mostly a straight line. When adding black tape we needed to use string to drop in in order to make sure the tape was being covered up. The way the string needed to be held in order to allow the slinky to be held meant that it tended to tip over and not fall straight when being dropped.

The post processing of the videos with the black tape was done by increasing the exposure of the videos to brighten the scene. The problem with this was that when increasing the exposure the tape itself became brighter meaning it stood out less.

Figure 2: Frames from a rejected video

This means that the processing done on these videos would need to be the same as what would be done on the non black tape videos. Combining this with the fact that they did not fall entirely straight during the experiment the

decision was made to go ahead with the videos that had the unmodified slinky.

Here is what the postprocessing was like on the video we used for the generation of the final data as well as tracers overlaid on the video by the analysis program.

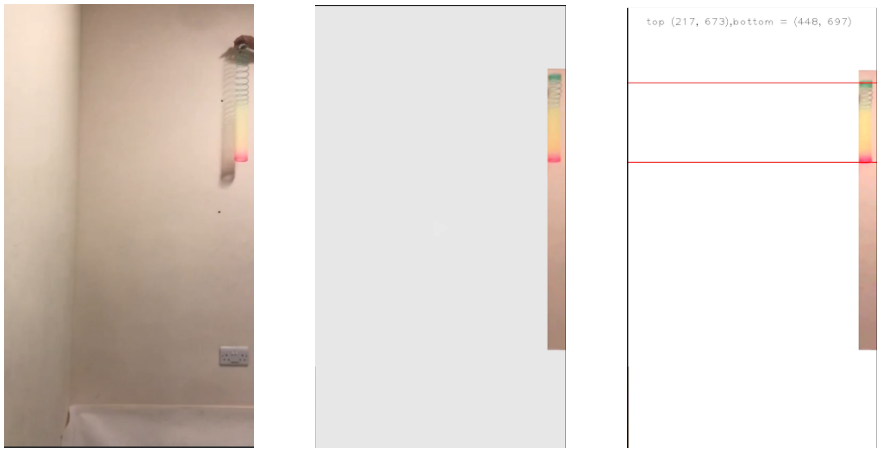


Figure 3: Frames from the used video

Here below is a sample of the data produced by the algorithm

| Frame | Top | Bottom |
|---|---|---|
| 0 | 836 | 832 |
| 20 | 941 | 840 |
| 40 | 845 | 797 |
| 60 | 737 | 645 |
| 80 | 576 | 483 |
| 100 | 372 | 230 |
| 120 | 128 | 53 |

# 3 Data Processing

The next step was to then generate graphs of this data and then extract coefficients of graphs for velocity displacement and acceleration. This was also done by a computer program made to directly read the data produced by the video analysis algorithms.

```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 from scipy.optimize import curve_fit
4 import sympy
5 from turtle import textinput
6
```

```
time = []
top = []
bottom = []

framerate = int(textinput("Framrate", "Please enter
    Framerate:"))
heightmax = int(textinput("2m", "What is the pixel at 2m"))
heightmin = int(textinput("0m", "What is the pixel at 0m"))


def FrameToS(x):
    return x / framerate


def PixeltoHeight(x):
    m = 2 / (heightmax - heightmin)
    return m * x - m * heightmin


with open("data.txt", "r") as f:
    for i in f.readlines():
        x, y, z = list(map(int, i.split()))
        time.append(x)
        top.append(y)
        bottom.append(z)

QuickCheck = plt.figure(0)
plt.scatter(time, top, c="blue")
plt.scatter(time, bottom, c="red")
QuickCheck.show()

time = list(map(FrameToS, time))
top = list(map(PixeltoHeight, top))
bottom = list(map(PixeltoHeight, bottom))

tot_time = sympy.ceiling(time[-1] * 100)
timeLine = list(map(lambda x: x / 100, range(tot_time)))


def lineFunc(x, a, b, c):
    return (a * (x**2)) + (b * x) + c


t = sympy.symbols("t")


start = int(textinput("Start Bottom", "What Frame does the
    bottom start moving"))
bottomTime = list(filter(lambda x: x >= FrameToS(start),
    timeLine))
```

```
54
55 Sta, Stb, Stc = curve_fit(lineFunc, time, top)[0]
56 Sba, Sbb, Sbc = curve_fit(lineFunc, time[start:], bottom[
       start:])[0]
57
58 st_eq = (Sta * t**2) + (Stb * t) + Stc
59 vt_eq = sympy.diff(st_eq)
60 at_eq = sympy.diff(vt_eq)
61
62 sb_eq = (Sba * t**2) + (Sbb * t) + Sbc
63 vb_eq = sympy.diff(sb_eq)
64 ab_eq = sympy.diff(vb_eq)
65
66 st_est = list(map(lambda x: (Sta * x**2) + (Stb * x) + Stc,
       timeLine))
67 vt_est = [vt_eq.subs(t, i) for i in timeLine]
68 at_est = [at_eq.subs(t, i) for i in timeLine]
69
70 sb_est = list(map(lambda x: (Sba * x**2) + (Sbb * x) + Sbc,
       bottomTime))
71 vb_est = [vb_eq.subs(t, i) for i in bottomTime]
72 ab_est = [ab_eq.subs(t, i) for i in bottomTime]
73
74 data = plt.figure(1)
75 plt.title("$Displacement (Scatter)$")
76 plt.xlabel("$Time (s)$")
77 plt.ylabel("$Distance (m)$")
78 plt.scatter(time, top, c="blue")
79 plt.scatter(time, bottom, c="red")
80 data.savefig("data.png")
81 data.show()
82
83 s = plt.figure(2)
84 plt.title("$Displacement$")
85 plt.xlabel("$Time (s)$")
86 plt.ylabel("$Distance (m)$")
87 plt.plot(timeLine, st_est, c="blue")
88 plt.plot(bottomTime, sb_est, c="red")
89 s.savefig("Displacement.png")
90 s.show()
91
92 v = plt.figure(3)
93 plt.title("$Velocity$")
94 plt.xlabel("$Time (s)$")
95 plt.ylabel("$Velocity (m/s)$")
96 plt.plot(timeLine, vt_est, c="blue")
97 plt.plot(bottomTime, vb_est, c="red")
98 v.savefig("Velocity.png")
99 v.show()
100
```

```
101 a = plt.figure(4)
102 plt.title("$Acceleration$")
103 plt.xlabel("$Time (s)$")
104 plt.ylabel("$Acceleration (m/s^{2})$")
105 plt.plot(timeLine, at_est, c="blue")
106 plt.plot(bottomTime, ab_est, c="red")
107 a.savefig("Acceleration.png")
108 a.show()
109
110 print(st_eq)
111 print(vt_eq)
112 print(at_eq)
113 print(sb_eq)
114 print(vb_eq)
115 print(ab_eq)
116 input()
```

This program works by first converting the units of the data ,which are pixels and frames, into real world units of metres and seconds. It also asks for a start time of when the bottom of the slinky starts falling. This means that the Graphs of the bottom of the slinky start later than the graphs of the top of the slinky but both should stop at about the same time.

From looking at a scatter plot of the data we can see that the Displacement over time looks somewhat like a downwards quadratic curve. This makes a lot of sense as the shape of the curve as acceleration is generally a constant. As displacement is the result of the double integration of acceleration we would expect it to be two orders higher which is a quadratic curve which we see. First we generate a curve of displacement by fitting a quadratic to the data using the SciPy curve fit function. I then use the SymPy differentiation function to give me an algebraic derivative of and second derivative of the displacement curve which are velocity and acceleration respectively. These are then plotted using MatPlotLib.

# 4 Data and Conclusions

## 4.1 Plots

Here are the final plots of Velocity Distance and accelerations as functions of time where the blue line represents the top of the slinky and the red the bottom.

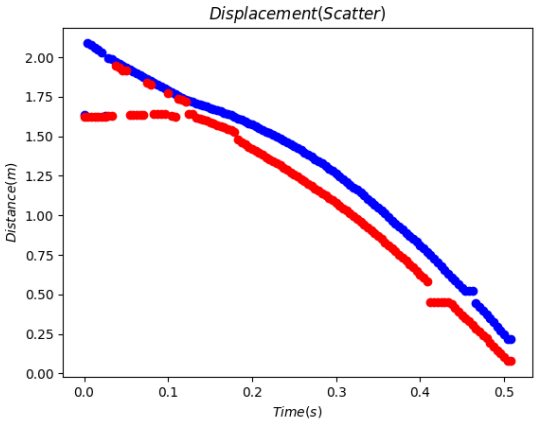


Figure 4: Raw Data

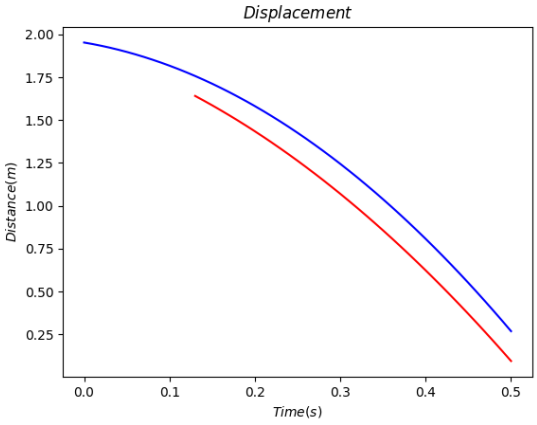


Figure 5: Displacement

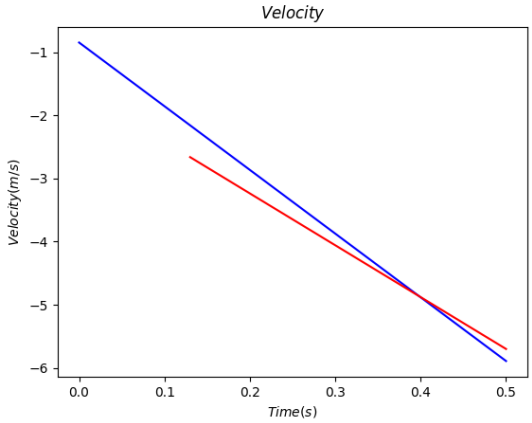


Figure 6: Velocity

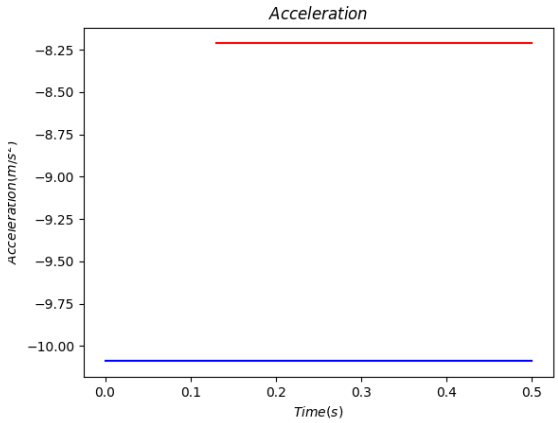


Figure 7: Acceleration

### 4.2 Equations

The equations for all are given as follows

$$s_t = -5.044782367476 77t^2 - 0.846329659090045t + 1.95262822033282$$

$$v_t = -10.0895647349535t - 0.846329659090045$$

$$a_t = -10.0895647349535$$

$$s_b = -4.105906440545 49t^2 - 1.59414344464529t + 1.91770040329421$$

$$v_b = -8.21181288109098t - 1.59414344464529$$

$$a_b = -8.21181288109098$$

One interesting thing that we can see is that the bottom does not start moving or accelerating at the same rate as the top of the slinky when it is in motion. Once the bottom starts to move the bottom of the slinky is moving at a lower velocity but also then experiences a higher acceleration meaning by the end it is moving faster than the top of the slinky. This suggests that there is likely another force acting likely from a collision occurring from the top and bottom of the slinky or even tensile forces from within the slinky.

## 5 Appendices

### 5.1 Appendix 1: Summary of error sources

For this experiment there are a few sources of error. These include the fact that the slinky doesn't fall perfectly straight meaning that some of its velocity towards the sides are not factored in as the code only tracks the top and the bottom of the object. The program also does sometimes briefly lose tracking though this is not too much of a problem as it takes a new measurement every single frame so errors get smoothed out. Errors in the accuracy of the measurements dont stem from inaccuracies in measurement as the program can "see" to an individual pixel though there can be inaccuracies if whereabouts 2m and the 0m are.

### 5.2 Appendix 2: Videos and Data

Falling Slinky Video : https://youtube.com/shorts/HG4UeAhDvP4
Falling Slinky Processed : https://youtube.com/shorts/byjEyY-lvn0
Falling Slinky Tracers: https://youtube.com/shorts/WUflNMSh2r8
Code + Data: https://github.com/xohanthetrader/bpho-git