PROJECT 1 - PENDULUM OSCILLATORY MOTION

AUTHORS

CRISTIAN ALBERTO AGUDELO MARQUEZ

JUAN JOSE GOMEZ MEJIA

TEACHER

SEBASTIAN GUZMAN OBANDO

UNIVERSIDAD
DE ANTIOQUIA

1 8 0 3

UNIVERSITY OF ANTIOQUIA

SYSTEMS ENGINEERING

DIGITAL IMAGE PROCESSING

March 22, 2023

1 . INTRODUCTION

Digital image processing is a branch of computer science that deals with the acquisition, processing, and analysis of images using algorithms and mathematical techniques. This discipline is applicable in various areas, from medicine and biology to robotics and computer vision. In the field of physics, capturing and analyzing motion is a topic of great interest as it enables the study of physical phenomena and validation of theoretical models.

This report presents an analysis of the motion capture of a pendulum using digital image processing tools, specifically the Python OpenCV library. The pendulum is a physical system that has been studied for centuries and is still the subject of research in modern physics. The ability to measure and analyze its motion is essential to understanding its behavior and validating the physical theories that describe it.

The objective of this report is to demonstrate how the motion of a pendulum can be captured using video and image processing to obtain relevant physical variables such as position, velocity, and acceleration. In addition, a simulation of the pendulum's motion will be performed using the Euler method based on the pendulum's actual characteristics such as length and mass, and the results obtained through motion capture will be compared with those obtained through simulation.

The report is structured as follows: first, the procedure used to capture the motion of the pendulum and process the images will be described. Then, the edge detection method used to calculate the physical variables of the pendulum will be explained. Subsequently, the results obtained through motion capture and simulation will be presented, and the results will be compared to evaluate the consistency of the methods. Finally, the study's conclusions will be discussed, and possible improvements and future applications of the method will be proposed.

The source code repository for this project is hosted remotely on GitHub, and its link is: [juangomez177/Pendulum_Oscillatory_Motion (github.com)]

2. DESCRIPTION OF THE PROCEDURE

Initially, a video was recorded using a Christmas ball to represent the pendulum along with a nylon string.

The approximate details of the video are:

Video:

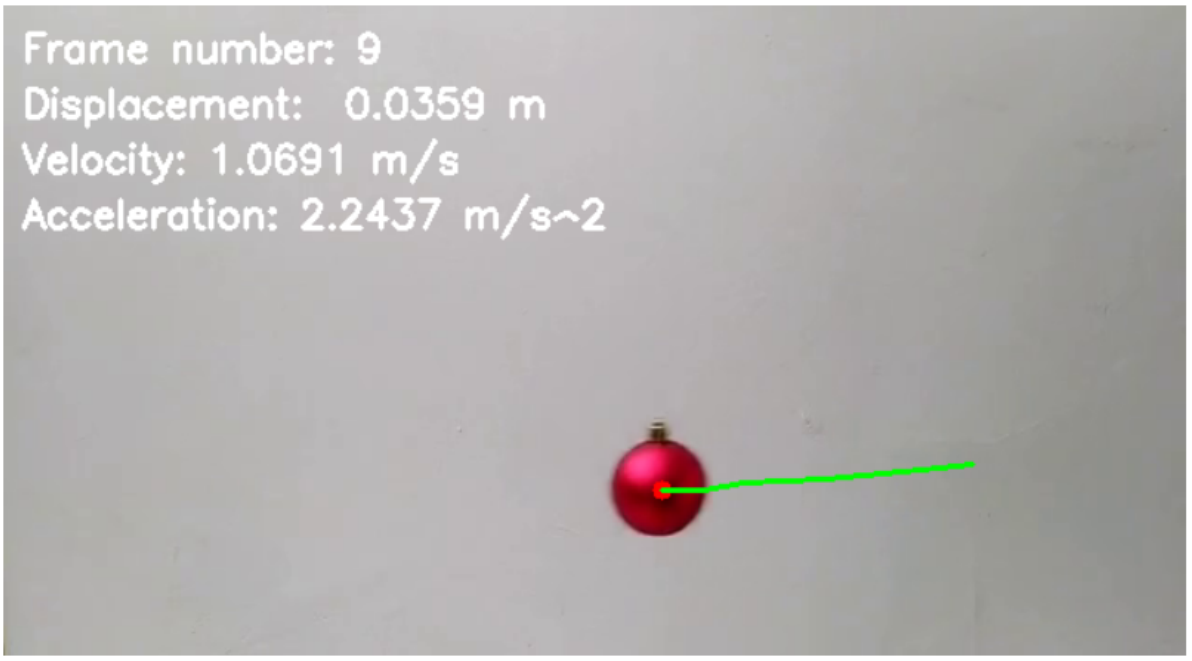
Format: mp4
Resolution: 640x352
Frames per second: 30
Duration: 12 seconds
Scale: 1 meter for horizontal frame, thus 640 pixels=1 meter

And the approximate details of the pendulum are:

Pendulum:

Length: 1.7 meters
Mass: 0.05 kilograms or 50 grams
Coefficient of air friction: 0.0025
Initial angle with respect to the vertical: 15 degrees

Subsequently, the video was processed using digital image processing techniques, specifically the **OpenCV** library in **Python** language:



For each **frame**, we will obtain details of **Displacement**, **Velocity**, and **Acceleration**, along with a plot of the pendulum's trajectory that indicates that everything is correct.

The following is the source code where the video was processed:
Importing **libraries** and initializing necessary **variables**:

```
# Import libraries
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt
import pandas as pd
from scipy.ndimage import gaussian_filter
#from google.colab.patches import cv2_imshow

# Initialize video capture
cap = cv2.VideoCapture("pendulo.mp4")

# Check if video is opened correctly
if (cap.isOpened() == False):
  print("Error opening video stream or file")

# Calculate video frame rate
fps = cap.get(cv2.CAP_PROP_FPS)
minutes = 0
seconds = 0
frame_id = int(fps*(minutes*60 + seconds))
timefps = (1.0/fps)
cap.set(cv2.CAP_PROP_POS_FRAMES, frame_id)

# Initialize pixel displacements
positions = []
position = 0

# Initialize time in seconds
times = []
times.append(0)

# Initialize displacement in meters
displacements = []
displacements.append(0)
displacement = 0

# Initialize velocities in m/s
velocities = []
velocities.append(0)
velocity = 0

# Initialize tangential accelerations in m/s^2
accelerations = []
accelerations.append(0)
acceleration = 0

# Video scale is approximately 1 meter in the horizontal frame
scale = 1
```

As mentioned before, the **video** will be processed in each frame where necessary calculations will be made, such as converting each frame to grayscale, applying a mask, performing morphological operations, and calculating the object's center to subsequently calculate the physical variables.

```
while cap.isOpened():

    # Read a frame from the video stream
    ret, frame = cap.read()

    if ret==True:

      # Convert frame to grayscale and apply a mask
      gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
      threshold=85
      mask =np.uint8((gray<threshold)*255)

      # Perform morphological operations
      element_e = cv2.getStructuringElement(cv2.MORPH_RECT,(2,2))
      dilate= cv2.dilate(mask,element_e, iterations =9)

      # Find the contour of the pendulum
      contours, hierarchy = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
      largest_contour = max(contours, key=cv2.contourArea)
      M = cv2.moments(largest_contour)

      # Calculate the center of the pendulum using moments
      x = int(M["m10"] / M["m00"])
      y = int(M["m01"] / M["m00"])
      positions.append((x, y))

      # Draw a circle at the center of the pendulum contour
      cv2.circle(frame, (x, y), 5, (0, 0, 255), -1)

      # Draw the pendulum trajectory
      for i in range(1, len(positions)):
          cv2.line(frame, positions[i-1], positions[i], (0, 255, 0), 2)

     # Calculate the pixel displacement of the center of the contour between two consecutive frames
      if n!=1:
        cx = positions[-1][0] - positions[-2][0]
        cy = positions[-1][1] - positions[-2][1]
        distance = math.sqrt(cx**2 + cy**2)#Calculo de la distancia en pixeles entre el pixel actual y el anterior

        # Convert the distance traveled from pixels to meters, knowing that the video dimensions are 640x352,
        # and that the horizontal frame of 640 is approximately 1 meter
        pixel_per_meter = 640 / scale
        displacement = distance / pixel_per_meter
        displacements.append(displacement)

        # Calculate the time elapsed between two consecutive frames (Knowing that the video is at 30 fps)
        times.append(times[-1]+timefps)

        # Calculate the velocity
        velocity = displacement / timefps
        velocities.append(velocity)

        # Calculate the tangencial acceleration
        acceleration = (velocities[-1]-velocities[-2]) / timefps
        accelerations.append(acceleration)

      # Shows the actual frame, with the data captured
      print("Frame ", n)

      cv2.putText(frame, "Frame number: {}".format(n), (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
      cv2.putText(frame, "Displacement:  {:.4f} m".format(displacement), (10, 60), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
      cv2.putText(frame, "Velocity: {:.4f} m/s".format(velocity), (10, 90), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)
      cv2.putText(frame, "Acceleration: {:.4f} m/s^2".format(acceleration), (10, 120), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 255, 255), 2)

      cv2.imshow('Point Coordinates', frame)
      #cv2_imshow(frame)
      n=n+1

      # Exit if pess key 'q'
      if cv2.waitKey(25) & 0xFF == ord("q"):
          break
    else:
      break

# Release resources
cap.release()
cv2.destroyAllWindows()
```

Finally, we calculate the **period** and **frequency** based on the length of the pendulum, as it can be difficult to calculate them from the data captured in the video processing.

```
# Calculation of the period and frequency (Only if the system is not subjected to air resistance, the period and frequency will be constant)
L = 1.7 # Length of pendulum in meters
g = 9.81 # Gravity
# Calculate the angular velocity
omega = np.sqrt(g/L)
# Calculate the period and frequency
T = 2*np.pi/omega
f=1/T
print("The approximate period of the pendulum is:", T, "seconds")
print("The frequency is:", f, "Hz")
```

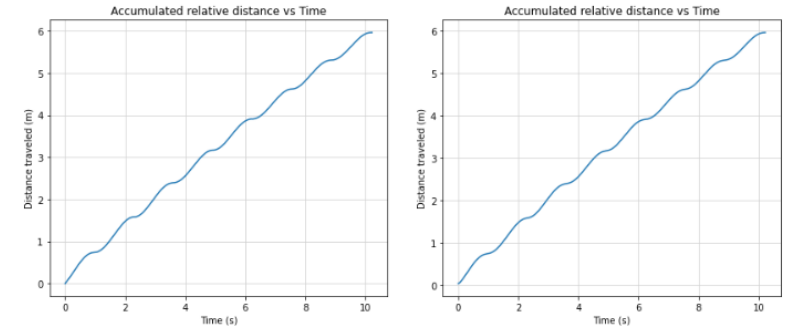
And now, another code snippet was used to **simulate** the motion of the pendulum from its physical variables such as length and mass.

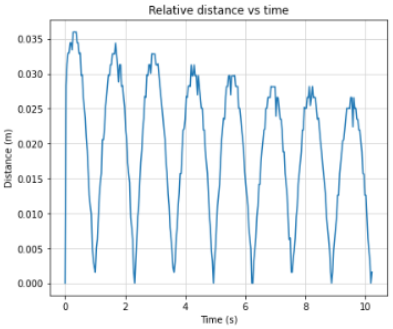
```
# Initial conditions
g = 9.8  # Acceleration due to gravity (m/s^2)
L = 1.7  # Length of the pendulum (m)
m = 0.05  # Mass of the pendulum (kg), aprox 50 gr
b = 0.0025 # Damping coefficient (kg/s)
theta0 = 15  # Initial angle grad
theta0 = math.radians(theta0)
omega0 = 0  # Initial angular velocity(rad/s)

# Time array
dt = timefps  # Time step
n_steps = n-1 # iterations number
cumulative_time = np.cumsum(np.ones(n_steps))
t = dt * cumulative_time  # time vector

theta = np.zeros_like(t)
omega = np.zeros_like(t)
displacements2 = np.zeros_like(t)
velocities2 = np.zeros_like(t)
accelerations2 = np.zeros_like(t)

theta[0] = theta0
omega[0] = omega0
dt = t[1] - t[0]

# Solve for theta(t) and omega(t) using Euler's method
for i in range(1, len(t)):
    accelerations2[i-1] = -(g/L)*np.sin(theta[i-1]) - (b/m)*omega[i-1]
    omega[i] = omega[i-1] + accelerations2[i-1]*dt
    theta[i] = theta[i-1] + omega[i]*dt
    displacements2[i] = L*(1 - np.cos(theta[i]))
    velocities2[i] = L*omega[i]
```

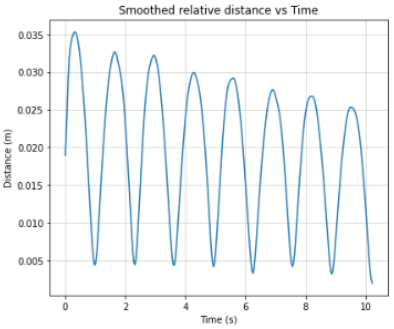
Here, a calculation is performed using the **Euler method**, based on the calculation of the angle theta and the angular velocity omega at each time instant.

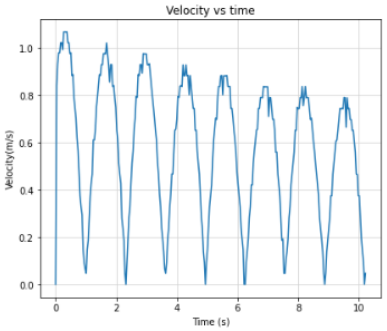
3. ANALYSIS AND RESULTS

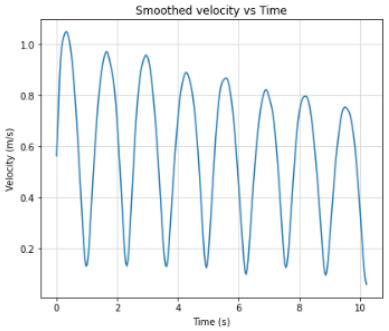
By processing the motion of the **pendulum**, we obtain the following results:

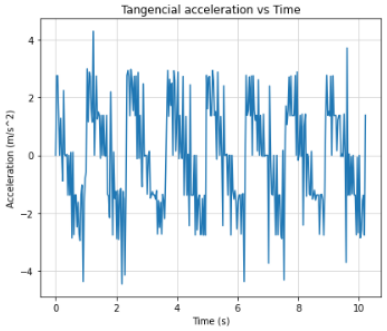


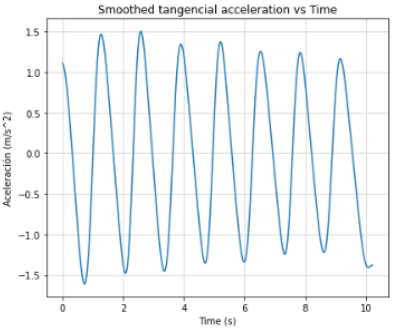


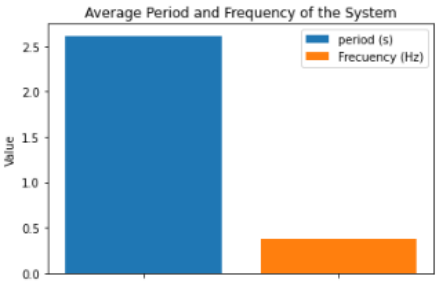






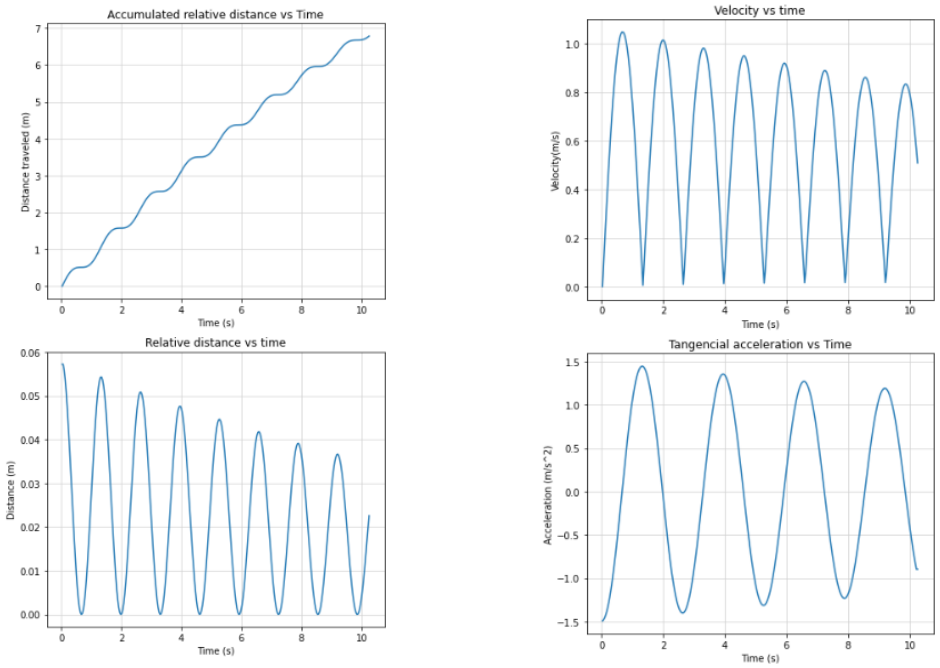




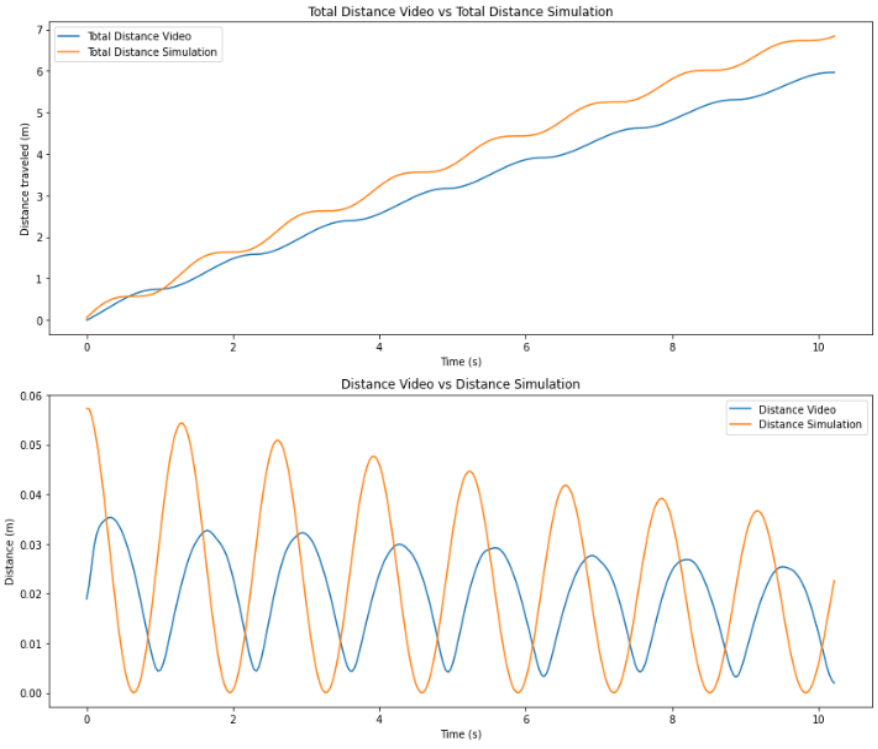


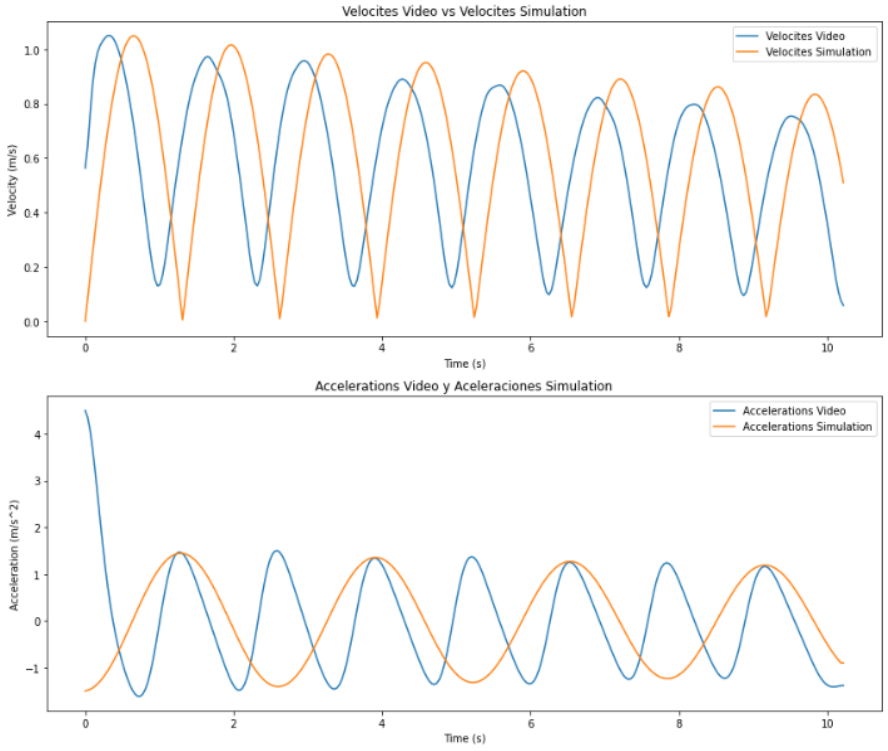
It can be observed that the **first** column corresponds to the data captured exactly, therefore the **second** column shows the same data, but with a small smoothing applied using a **Gaussian** filter in the areas where some noise was generated, especially in the acceleration. This gives us oscillatory movements for each variable, describing the oscillatory motion of the pendulum.

Meanwhile, the **simulation** shows the following results:



Finally, if we compare **both methods**, we obtain:





The results seem to be correct, however there are small differences between the different procedures:

On one hand, the displacement traveled by the pendulum seems to be greater in the **simulation**, approximately about 7 meters, while for the **video** it's 6.

It can be better observed that the relative displacements, in fact a greater distance is being traveled between each oscillation of the pendulum. Possibly the cause is that the actual **video** scaling is not being taken into account when calculating the pixels and the conversion is not being done correctly, as well as the **simulation** may fail due to a wrong parameter such as the coefficient of air friction or even the mass of the pendulum.

However, there is agreement with the velocities and accelerations, the small gaps show a phase shift between each oscillation, possibly due to the smoothing of the data, or because the data initialization in the video is zero.

While in the acceleration it can be observed that there are fewer undulations in the simulation, but this happens because the accelerations for both cases are being considered negative, so in the **simulation** the accelerations are vectorially described and in the **video** the accelerations are described when there is a decrease in the accelerations, but generally the magnitudes agree.

4. CONCLUSIONS

This report has presented a method for capturing the motion of a pendulum using digital image processing tools and obtaining relevant physical variables such as position, velocity, and acceleration. In addition, a simulation of the pendulum's motion was performed using the Euler method, and the results obtained through motion capture were compared with those obtained through simulation.

The results obtained show that the motion capture method is capable of providing precise measurements of the physical variables of the pendulum. The observed variations in the results may be due to systematic errors in the motion capture procedure, such as camera calibration or image processing, and limitations in capturing each frame, and the simulation may not be entirely accurate due to the use of incorrect parameters. However, a good consistency has been observed between the results obtained through motion capture and those obtained through simulation.

So, the method presented in this report has potential applications in experimental physics and physics education. The use of digital image processing tools to measure the motion of physical objects can provide an alternative to traditional measurement methods, enabling measurements through multimedia content.

5. REFERENCES

- "Analyzing the Periodic Motion of a Pendulum" by el MIT OpenCourseWare: https://ocw.mit.edu/resources/res-8-003-physics-i-fall-2004/exams/pendulum.pdf
- "Simple Pendulum" por Real Python: https://realpython.com/pendulum-visualization-python/
- "How to Detect Motion in a Video Stream with Python" by PyImageSearch: https://www.pyimagesearch.com/2015/05/25/basic-motion-detection-and-tracking-with-python-and-opencv/
- "Detecting Motion from Video Streams using Python and OpenCV" by Towards Data Science: https://towardsdatascience.com/detecting-motion-from-video-streams-using-python-and-opencv-ed1158e7df41
- "Analyzing Motion in Videos using Python" by Towards Data Science: https://towardsdatascience.com/analyzing-motion-in-videos-using-python-5e70dd4c526c
- "Analyzing Simple Harmonic Motion of a Pendulum with Python" by Physics Forums: https://www.physicsforums.com/insights/analyzing-simple-harmonic-motion-pendulum-python/