УДК 531.3

Важенина Алина Витальевна, Коробанова Злата Олеговна, студенты
(КузГТУ, г. Кемерово)
Alina V. Vazhenina, Zlata O. Korobanova, students
(KuzSTU, Kemerovo)
Научный руководитель:
Пузынин Андрей Владимирович, к.ф-м.н., доцент кафедры физики
(КузГТУ, Кемерово)
Puzynin Andrey Vladimirovich, Candidate of Physical and Mathematical Sciences, Associate Professor, Department of Physics
(KuzSTU, Kemerovo)

АВТОМАТИЗИРОВАННЫЙ ВИДЕОАНАЛИЗ ДВИЖЕНИЯ МАЯТНИКА НЬЮТОНА
AUTOMATED VIDEO ANALYSIS OF NEWTON'S CRADLE MOTION

Аннотация. Исследование посвящено автоматическому отслеживанию шаров маятника Ньютона с использованием Python и OpenCV. На основе видеозаписи определяются координаты шаров, рассчитываются период и затухание колебаний, а также оценивается передача энергии при соударениях.

Abstract. This paper presents automated tracking of Newton's cradle balls using Python and OpenCV. Video data are used to determine ball coordinates and estimate the oscillation period, damping, and energy transfer during collisions.

Маятник Ньютона – система из нескольких одинаковых металлических шаров, наглядно демонстрирует законы сохранения импульса и энергии. Для точного анализа движения его шаров необходимо отслеживать их координаты все время. Обычному человеку это сделать достаточно проблематично и полученный результат может быть недостоверным. Для автоматизации данного процесса можно использовать компьютерное зрение, которое гарантирует точные данные для достоверного физического анализа. В работе были исследованы затухание колебаний и передача энергии. Для работы был выбран язык программирования Python и библиотека OpenCV.

Следует упомянуть о теории движения шаров Ньютона. Начнем с периода малых колебаний математического маятника, который определяется формулой: $T = 2\pi\sqrt{\frac{L}{g}}$, где L — длина нити, g ≈ 9,81 м/с² — ускорение свободного падения. По измеренному периоду можно вычислить длину нити по формуле: $L = \frac{gT^2}{4\pi^2}$.

В реальном мире колебания затухают из-за трения, деформации и потерь энергии. Амплитуда убывает по экспоненциальному закону:

$A(t) = A_0 e^{(-\beta t)}$, где β — коэффициент затухания. Логарифмический декремент δ характеризует уменьшение амплитуды за период, а добротность для слабого затухания оценивается как $Q \approx \frac{\pi}{\delta}$ [1].

Объектом исследования был маятник Ньютона, который состоит из 5 стальных шаров, подвешенных на нитях. Габариты используемого маятника: диаметр шара — 15 мм, масса — 15 г, длина нитей — 7 см. Чтобы данные были достовернее использовался зеленый хромакей, для того чтобы достигнуть наибольшей контрастности. Запись осуществлялась со смартфона с частотой 30 кадров в секунду, зафиксированного на штативе на расстоянии около метра. Левый шар отводили примерно на 30°.

На каждом кадре изображение переводили в цветовое пространство HSV, после чего выделяли зелёный фон и отсекали тёмные элементы рамы. После очищали маски, определяли контуры и по ним определяли центры шаров, координаты крайних сохраняли в файл.

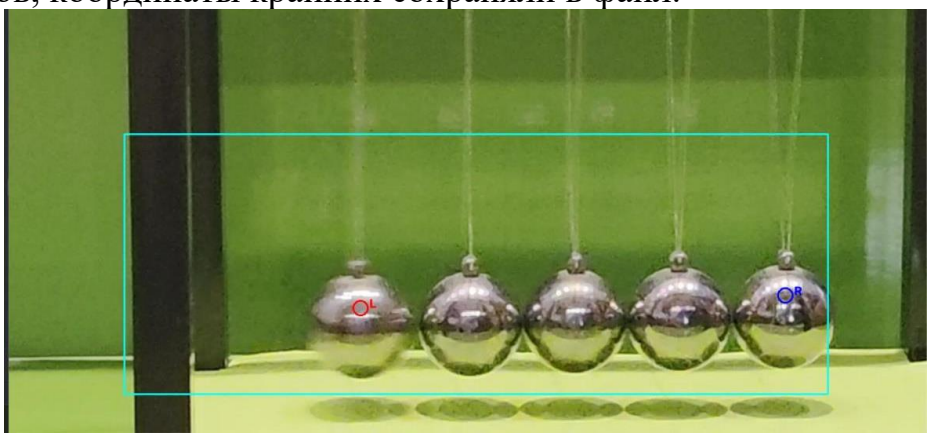

Рисунок 1 — Исходный кадр с выделенной областью интереса и обозначенными центрами шаров



Рисунок 2 — Бинарная маска, полученная после фильтрации фона

Перевод пиксельных координат в метры выполняли на основе известного диаметра шара. Начальную секунду записи исключили из анализа для минимизации влияния переходного процесса. Для устранения шумов применялся фильтр Савицкого — Голея, а скорость определялась численным дифференцированием координат. Амплитуды последовательных колебаний находили по локальным максимумам, что позволило рассчитать

декремент затухания и добротность. Кинетическую энергию вычисляли по формуле $E_{\text{кин}} = \frac{mV^2}{2}$.

В ходе обработки было проанализировано 479 кадров. Графики зависимости координат от времени демонстрируют затухающие колебания. Правый шар, изначально находившийся в покое, после удара, начинает движение, что соответствует законам сохранения импульса и энергии.

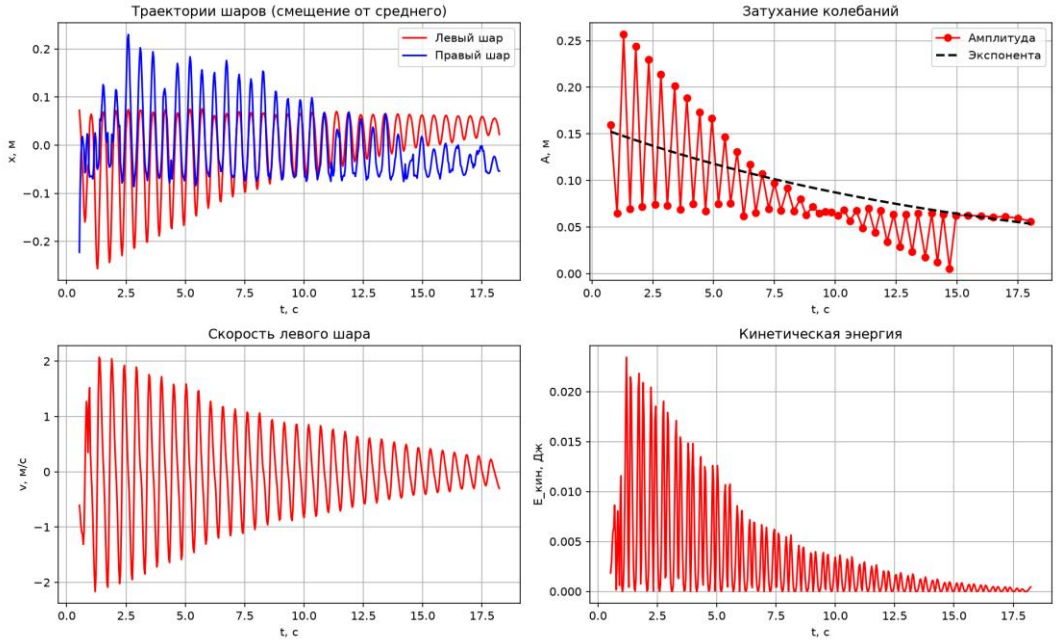

Рисунок 3 - результаты анализа: динамика координат, изменение амплитуды, скорость и кинетическая энергия.

По результатам обработки получены следующие значения: период колебаний T = 0,5028 с; логарифмический декремент δ = 0,0452; добротность Q ≈ 69,57. Оценка длины нити по измеренному периоду составила 6,28 см, тогда как реальная длина равна 7 см. Расхождение составляет около 10 %, что может быть связано с погрешностью выделения центров, калибровки.

Максимальная скорость левого шара составила 0,0607 м/с, максимальная кинетическая энергия — 0,0276 мДж. Полученные данные показывают уменьшение энергии во времени.

По графику амплитуда первого движения оценена примерно в 0,0025 м, а ответного — в 0,0015 м. При приближённом соотношении энергии и квадрата амплитуды доля потерь оценивается как $1 - (0{,}0015/0{,}0025)^2 \approx 0{,}64$, то есть около 64 %. Эта оценка является примерной, так как для более достоверного результата нужна более точная калибровка.

В ходе исследования был разработан метод автоматического отслеживания маятника Ньютона по видеозаписи с использованием Python и OpenCV. По координатам шаров были определены период колебаний, логарифмический декремент и добротность, выявлен экспоненциальный характер затухания. Длина нити рассчитанная по периоду, полученному с

помощью компьютерного анализа, отличалась примерно на 10 % от фактической, что показывает высокую точность такого метода исследований, а сравнение амплитуд позволило получить ориентировочную оценку потерь энергии.

Список литературы

1. Савельев И. В. Курс общей физики. Т. 1. Механика. — М.: Наука, 2005.
2. Bradski G. The OpenCV Library // Dr. Dobb's Journal of Software Tools. — 2000.
3. OpenCV. Документация библиотеки OpenCV. — URL: https://docs.opencv.org/ (дата обращения: 24.10.2026).
4. Сивухин Д. В. Общий курс физики. Т. 1. Механика. — М.: ФИЗМАТЛИТ, 2005.

ПРИЛОЖЕНИЕ. ЛИСТИНГ ПРОГРАММЫ

Программа выполняет выделение шаров на видеокадрах, определяет координаты центров крайних шаров и сохраняет результаты в файл data.json.

Листинг 1 — Программа track.py.

```
import cv2
import numpy as np
import json
VIDEO = 'video60.mp4'
OUTPUT = 'data.json'
MIN_AREA = 150
TRACK_SIDE = 'both'
ROI_X1_FRAC = 0.10
ROI_X2_FRAC = 0.71
ROI_Y1_FRAC = 0.40
ROI_Y2_FRAC = 0.80
GREEN_LOW = np.array([25, 40, 40])
GREEN_HIGH = np.array([95, 255, 255])
BLACK_LOW = np.array([0, 0, 0])
BLACK_HIGH = np.array([180, 255, 60])
cap = cv2.VideoCapture(VIDEO)
if not cap.isOpened():
    raise SystemExit(f'Не могу открыть {VIDEO}')
fps = cap.get(cv2.CAP_PROP_FPS)
W = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
H = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
x1 = int(W * ROI_X1_FRAC)
x2 = int(W * ROI_X2_FRAC)
y1 = int(H * ROI_Y1_FRAC)
y2 = int(H * ROI_Y2_FRAC)
print(f'ROI: x=[{x1},{x2}], y=[{y1},{y2}]  кадр {W}x{H}')
data = {'left': [], 'right': [], 'fps': fps}
frame_id = 0
while True:
    ret, frame = cap.read()
    if not ret:
        break
```

```
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    mask_green = cv2.inRange(hsv, GREEN_LOW, GREEN_HIGH)
    mask_black = cv2.inRange(hsv, BLACK_LOW, BLACK_HIGH)
    mask = cv2.bitwise_not(mask_green)
    mask = cv2.bitwise_and(mask, cv2.bitwise_not(mask_black))
    mask[:y1, :] = 0
    mask[y2:, :] = 0
    mask[:, :x1] = 0
    mask[:, x2:] = 0
    kernel = np.ones((5, 5), np.uint8)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel)
    mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel)
    mask = cv2.erode(mask, kernel, iterations=1)
    mask = cv2.dilate(mask, kernel, iterations=1)
    contours, _ = cv2.findContours(
        mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE )
    t = frame_id / fps
    big = [c for c in contours if cv2.contourArea(c) > MIN_AREA]
    circles = []
    for c in big:
        bx, by, bw, bh = cv2.boundingRect(c)
        if bh > 0 and 0.6 < bw / bh < 1.8:
            M = cv2.moments(c)
            if M['m00'] > 0:
                circles.append((M['m10'] / M['m00'], M['m01'] / M['m00']))
    if circles:
        left_c = min(circles, key=lambda p: p[0])
        right_c = max(circles, key=lambda p: p[0])
        if TRACK_SIDE in ('left', 'both'):
            data['left'].append((t, left_c[0], left_c[1]))
            cv2.circle(frame, (int(left_c[0]), int(left_c[1])),
                       12, (0, 0, 255), 2)
            cv2.putText(frame, 'L',
                        (int(left_c[0]) + 15, int(left_c[1])),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        if TRACK_SIDE in ('right', 'both'):
            data['right'].append((t, right_c[0], right_c[1]))
            cv2.circle(frame, (int(right_c[0]), int(right_c[1])),
                       12, (255, 0, 0), 2)
            cv2.putText(frame, 'R',
                        (int(right_c[0]) + 15, int(right_c[1])),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (255, 0, 0), 2)
    cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 255, 0), 2)
    cv2.imshow('mask', mask)
    cv2.imshow('frame', frame)
    if cv2.waitKey(1) & 0xFF == 27:
        break
    frame_id += 1
cap.release()
cv2.destroyAllWindows()
with open(OUTPUT, 'w') as f:
    json.dump(data, f)
print(f"Сохранено: left={len(data['left'])}, right={len(data['right'])}")
```

Листинг 2 — Программа analyze.py.

```
import json
```

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import savgol_filter, find_peaks
with open('data.json') as f:
    data = json.load(f)
fps = data.get('fps', 60.0)
print('Данные:', {k: (len(v) if isinstance(v, list) else v)
                  for k, v in data.items()})
def to_arrays(key):
    arr = np.array(data.get(key, []))
    if arr.ndim != 2 or arr.shape[0] < 5:
        return None
    return arr[:, 0], arr[:, 1], arr[:, 2]
left  = to_arrays('left')
right = to_arrays('right')
DIAMETER_PX = 500
PX_PER_M = DIAMETER_PX / 0.015
MASS_KG = 0.015
T_CUT_START = 1.0
def smooth(y, window=11, poly=3):
    if len(y) < window:
        return y
    return savgol_filter(y, window, poly)
def deriv(t, x):
    return np.gradient(x, t)
def prepare(arr):
    if arr is None:
        return None
    t, x, y = arr
    mask = t > T_CUT_START
    t, x, y = t[mask], x[mask], y[mask]
    if len(t) < 5:
        return None
    x_m = smooth(x) / PX_PER_M
    return t, x_m, y
L = prepare(left)
R = prepare(right)
fig, axes = plt.subplots(2, 2, figsize=(13, 8))
ax = axes[0, 0]
if L is not None:
    t_l, x_l, _ = L
    ax.plot(t_l, x_l - np.mean(x_l), 'r-', label='Левый шар')
if R is not None:
    t_r, x_r, _ = R
    ax.plot(t_r, x_r - np.mean(x_r), 'b-', label='Правый шар')
ax.set_xlabel('t, c')
ax.set_ylabel('x, м')
ax.set_title('Траектории шаров (смещение от среднего)')
ax.legend(); ax.grid(True)
ax = axes[0, 1]
delta = None
T_period = None
if L is not None:
    t_l, x_l, _ = L
    x_centered = x_l - np.mean(x_l)
```

```
    abs_x = np.abs(x_centered)
    peaks, _ = find_peaks(abs_x, distance=5)
    if len(peaks) > 3:
        t_peaks = t_l[peaks]
        A_peaks = abs_x[peaks]
        ax.plot(t_peaks, A_peaks, 'ro-', label='Амплитуда')
        ratios = A_peaks[:-1] / A_peaks[1:]
        ratios = ratios[ratios > 0]
        if len(ratios) > 0:
            delta = np.mean(np.log(ratios))
            T_period = np.mean(np.diff(t_peaks)) * 2
            A0 = A_peaks[0]
            fit = A0 * np.exp(-abs(delta) * t_peaks / abs(T_period))
            ax.plot(t_peaks, fit, 'k--', linewidth=2, label='Экспонента')
            print(f'Период T ≈ {T_period:.4f} c')
            print(f'Лог. декремент δ ≈ {delta:.4f}')
            print(f'Добротность Q = π/δ ≈ {np.pi/abs(delta):.2f}')
ax.set_xlabel('t, c')
ax.set_ylabel('A, м')
ax.set_title('Затухание колебаний')
ax.legend(); ax.grid(True)
ax = axes[1, 0]
if L is not None:
    t_l, x_l, _ = L
    v_l = deriv(t_l, x_l)
    ax.plot(t_l, v_l, 'r-')
    print(f'Макс. скорость левого шара: {np.max(np.abs(v_l)):.4f} м/c')
ax.set_xlabel('t, c')
ax.set_ylabel('v, м/c')
ax.set_title('Скорость левого шара')
ax.grid(True)
ax = axes[1, 1]
if L is not None:
    t_l, x_l, _ = L
    v_l = deriv(t_l, x_l)
    E = 0.5 * MASS_KG * v_l ** 2
    ax.plot(t_l, E, 'r-')
    print(f'Макс. кин. энергия: {np.max(E)*1000:.4f} мДж')
    print(f'Макс. кин. энергия: {np.max(E):.6f} Дж')
ax.set_xlabel('t, c')
ax.set_ylabel('E_кин, Дж')
ax.set_title('Кинетическая энергия')
ax.grid(True)
plt.tight_layout()
plt.savefig('results20.png', dpi=150)
plt.show()
print()
print('===== ФИЗИЧЕСКИЕ ВЕЛИЧИНЫ =====')
if T_period is not None:
    print(f'Период колебаний T = {T_period:.4f} c')
    L_est = 9.81 * T_period ** 2 / (4 * np.pi ** 2)
    print(f'Оценка длины нити L ≈ {L_est*100:.2f} см (по периоду)')
if delta is not None:
    print(f'Лог. декремент δ = {delta:.4f}')
        print(f'Коэф. затухания β = δ/T = {abs(delta)/T_period:.4f} 1/c')
```