**Цель работы**

1. Сформировать XML-модель для *Tendon connected 2R planar mechanism* (рис. 1) используя данные из таблицы (табл. 1);
2. Написать скрипт на Python с использованием методов *model*, *data* и *viewer*;
3. Запустить симуляцию работы механизма, используя физический движок *MuJoCo (Multi-Joint dynamics with Contact)*.

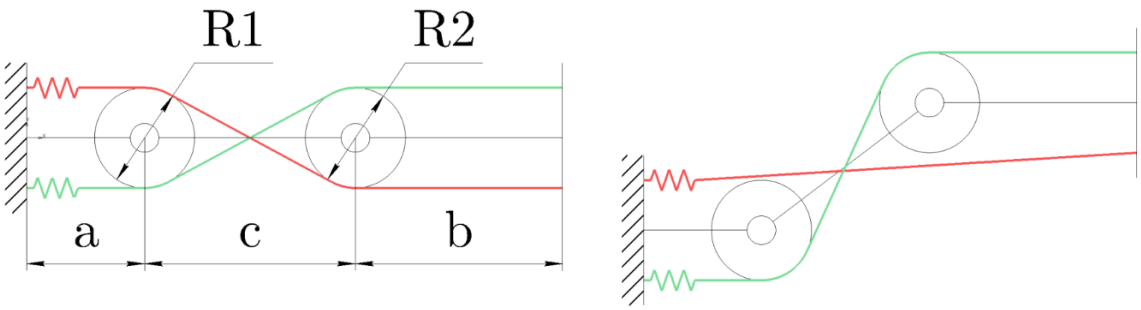


Рисунок 1. плоский механизм с сухожильным соединением, где $R_1$ – радиус первого шкифа, $R_2$ – радиус второго шкифа, a – длина первого звена, b – длина второго звена, c – смещение точки крепления сухожилия второго звена

Таблица 1 – Данные геометрических параметров

| $R_1$, м | $R_2$, м | $a$, м | $b$, м | $c$, м |
|---|---|---|---|---|
| 0.015 | 0.032 | 0.032 | 0.1 | 0.015 |

## Анализ XML-модели

XML-модель механизма 2R с сухожильным соединением была создана с использованием фреймворка MuJoCo. Модель манипулятора создается динамически с помощью отдельной функции, что позволяет легко изменять параметры, не редактируя напрямую XML-структуру. На рисунке 2 изображен фрагмент XML-файла, с помощью которого и происходит “создание” механизма.

```
<mujoco>
    <option timestep="1e-4"/>
    <option gravity="0 0 -9.8"/>

    <asset>
        <texture type="skybox" builtin="gradient" rgb1="1 1 1" rgb2="0.5 0.5 0.5" width="265" height="256"/>
        <texture name="grid" type="2d" builtin="checker" rgb1="0.1 0.1 0.1" rgb2="0.6 0.6 0.6" width="300" height="300"/>
        <material name="grid" texture="grid" texrepeat="10 10" reflectance="0.2"/>
    </asset>

    <worldbody>
        <light pos="0 0 10"/>
        <geom type="plane" size="1 1 0.1" material="grid"/>

        <camera name="side view" pos="0.1 -0.3 1" euler="60 0 25" fovy="50"/>
        <camera name="front view" pos="0 -2 1" euler="0 0 0"/>


        <body name="base" pos="0 0 0.75">
            <geom type="box" size="0.01 0.01 0.05" rgba="0.8 0.3 0.3 1"/>
            <site name="top_attachment_0" pos="0 0 0.035" size="0.001" rgba="1 0 0 1"/>
            <site name="middle_attachment_0" pos="0 0 0" size="0.001" rgba="0 1 0 1"/>
            <site name="bottom_attachment_0" pos="0 0 -0.035" size="0.001" rgba="0 0 1 1"/>
        </body>


        <body name="R1" pos="0 0.032 0.75">
            <geom name="R1_geom" type="sphere" size="0.015" rgba="0.8 0.3 0.3 1"/>
            <joint name="R1_joint" type="slide" axis="0 0 1" range="-0.05 0.05"/>
            <site name="top_attachment_1" pos="0 0 0.015" size="0.001" rgba="1 0 0 1"/>
            <site name="middle_attachment_1" pos="0 0 0" size="0.001" rgba="0 1 0 1"/>
            <site name="bottom_attachment_1" pos="0 0 -0.015" size="0.001" rgba="0 0 1 1"/>
        </body>
```

Рисунок 2. Фрагмент XML-модели

**Анализ Python-скрипта**

Программный код осуществляет запуск и контроль работы системы, фрагмент которого представлен на рис. 3. Ключевая особенность реализации заключается в нестандартном подходе к управлению приводами. Вместо того, чтобы подавать постоянный сигнал для движения в одну сторону, алгоритм реализует регулирование на основе синусоидальных колебаний. Приводы получают изменяющиеся во времени сигналы, причём управляющие воздействия для сгибающих и разгибающих механизмов находятся в противофазе.

```python
import mujoco
import mujoco_viewer
import matplotlib.pyplot as plt
import numpy as np
from lxml import etree
import time


R1 = 0.015
R2 = 0.032
a = 0.032
b = 0.1
c = 0.078
base_height = 0.75


R1_pos = a
R2_pos = a + c
S_pos = R2_pos + b


original_xml = "2R_tendon_mechanism.xml"
modified_xml = "2R_tendon_mechanism_modified.xml"

def create_2R_model():
```

```python
def main():

    create_2R_model()

    parameterize_model()

    model = mujoco.MjModel.from_xml_path(modified_xml)
    data = mujoco.MjData(model)

    SIMEND = 20
    TIMESTEP = 0.001
    STEP_NUM = int(SIMEND / TIMESTEP)


    ee_pos_x = []
    ee_pos_z = []

    viewer = mujoco_viewer.MujocoViewer(model, data, title="2R Tendon Mechanism")
```

Рисунок 3. Фрагменты кода

Благодаря такому принципу действия манипулятор выполняет плавные колебательные движения вдоль рабочей траектории. Система непрерывно отслеживает положение звеньев и накапливает данные о их перемещениях.

По завершении процесса симуляции, собранные данные обрабатываются для построения графика траектории движения *end-effector* (рис. 4). Такой подход позволяет оценить эффективность работы системы и точность выполнения заданных движений.

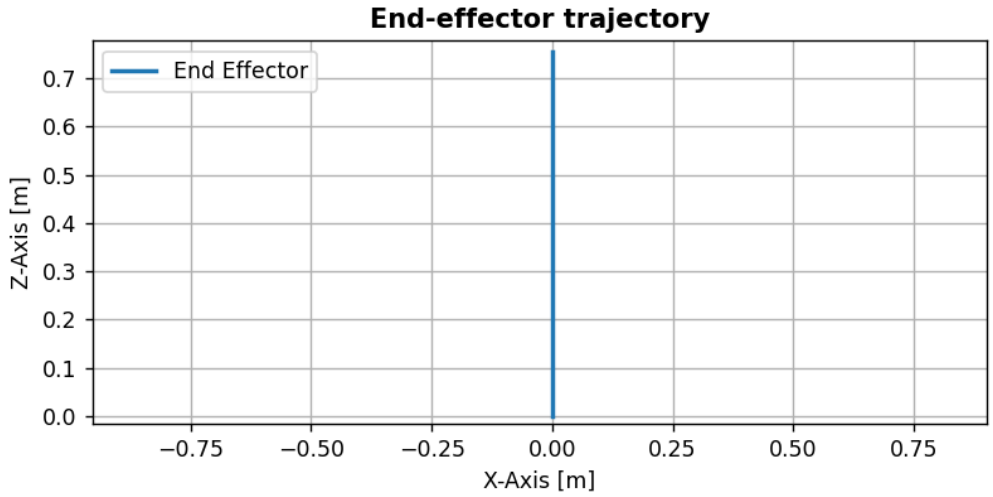


Рисунок 4. Траектория движения энд-эффектора

## Результаты симуляции

Механизм демонстрирует плавное движение благодаря сухожильным соединениям и синусоидальному управлению (рис. 5, 6). Траектория энд-эффектора имеет линейную форму. Анализ ошибки позиционирования показал значение: *0.049032*.

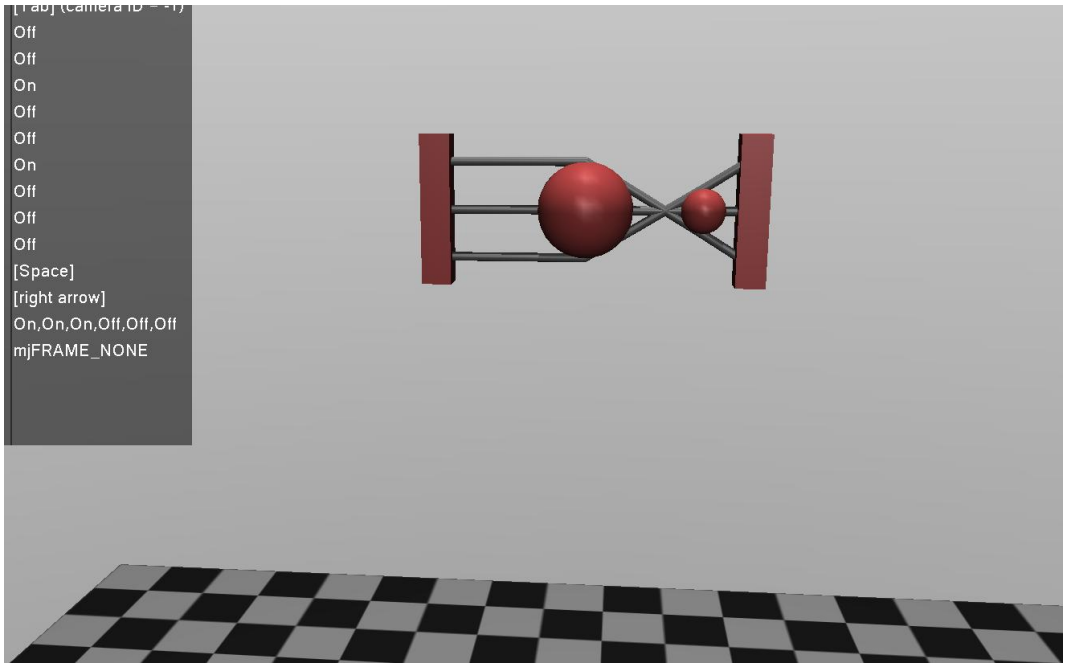


Рисунок 5. Механизм в начальном положении

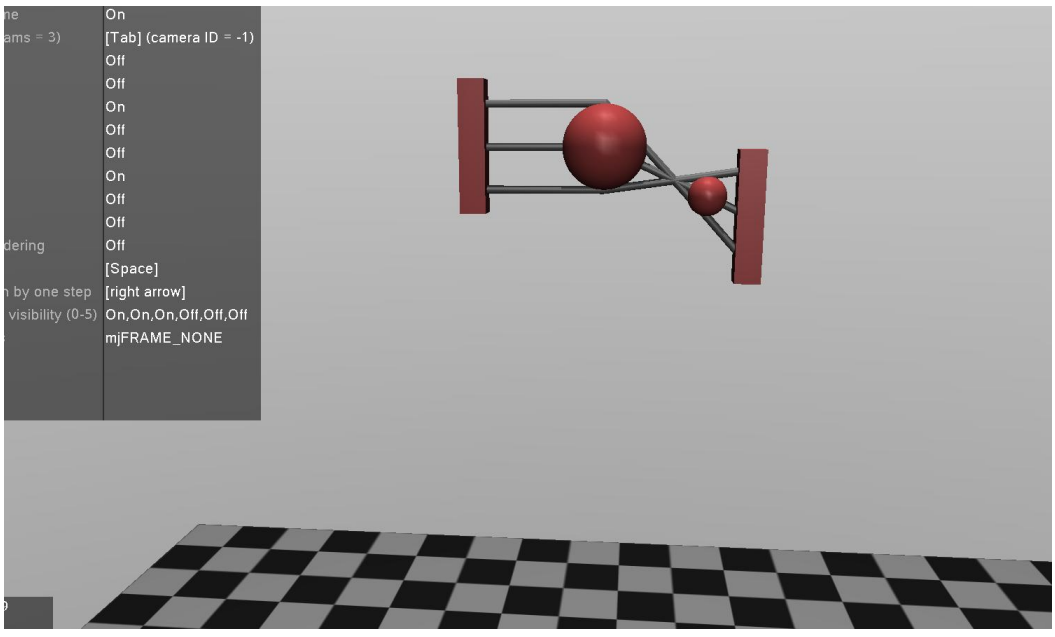


Рисунок 6. Момент работы механизма

## Вывод

В ходе выполнения лабораторной работы была создана XML-модель механизма с сухожильными соединениями, реализован скрипт на Python для симуляции с использованием MuJoCo, визуализирована траектория движения конечного эффектора, а также рассчитана ошибка позиционирования системы.