Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по практической работе №4

Дисциплина: “Имитационное моделирование робототехнических систем”

Автор: Балакин А.Р.

Вариант: 6

Факультет: СУиР

Преподаватель: Ракшин Е.А.

Санкт-Петербург, 2025

Входные данные

AMP_1, град	$FREQ_1$, Гц	$BIAS_1$, град	AMP_2, град	$FREQ_2$, Гц	$BIAS_2$, град
27.19	3.79	-42.1	18.73	3.11	42.4

Таблица 1 - Значения входных данных

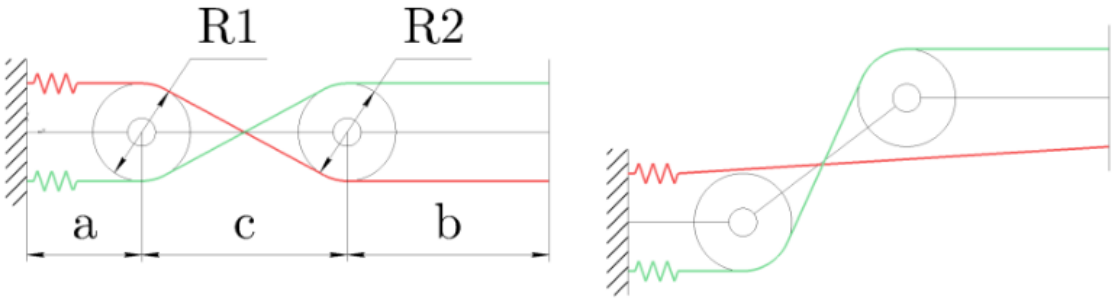

Рисунок 1 - Моделируемый механизм

К сожалению, заданные данные немного выходили за пределы рабочей области механизма, поэтому я незначительно поменял значения амплитуды и смещений, задающих желаемую траекторию.

Задание

a) Добавление актуаторов и сенсоров

Соответственно, в рамках третьего задания необходимо было создать XML-файл механизма, приведенного на рисунке 1. Теперь немного модифицируем его, добавив актуаторы и сенсоры:

```
<actuator>
  <motor name="motor_q1" joint="joint1" gear="1"/>
  <motor name="motor_q2" joint="joint2" gear="1"/>
</actuator>
```

Рисунок 2 - Блок актуаторов

```xml
<sensor>
  <jointpos name="j1_q" joint="joint1"/>
  <jointpos name="j2_q" joint="joint2"/>
  <jointvel name="j1_v" joint="joint1"/>
  <jointvel name="j2_v" joint="joint2"/>
</sensor>
```

Рисунок 3 - Блок сенсоров

Таким образом актуаторы будут приводить в движение наш механизм, а с помощью датчиков можно будет получить информацию о положении и скорости определенных частей механизма.

b) Скрипт симуляции

В данном задании, помимо запуска самой симуляции, также необходимо реализовать ПД-регулятор. В итоге получился следующий код:

```python
q1_desired = []
q2_desired = []

kp1 = 1.5
kd1 = 0.01
kp2 = 1.1
kd2 = 0.08

with viewer.launch_passive(model, data) as v:
  while v.is_running():
    for _ in range(steps_per_frame):

      q1_d = np.deg2rad(24.19)*np.sin(3.79*data.time/100)-np.deg2rad(32.1)
      q2_d = np.deg2rad(18.73)*np.sin(3.11*data.time/100)+np.deg2rad(32.4)
      q1_desired.append(q1_d)
      q2_desired.append(q2_d)

      qpos1 = data.sensor("j1_q").data[0]
      qpos2 = data.sensor("j2_q").data[0]
      qvel1 = data.sensor("j1_v").data[0]
      qvel2 = data.sensor("j2_v").data[0]

      diff1 = q1_d-qpos1
      diff2 = q2_d-qpos2

      data.ctrl[0] = kp1*diff1-kd1*qvel1
      data.ctrl[1] = kp2*diff2-kd2*qvel2

      mujoco.mj_step(model, data)
      time_data.append(data.time)
      qpos_data.append(data.qpos.copy())
      ctrl_data.append(data.ctrl.copy())
    v.sync()

time_data = np.array(time_data)
qpos_data = np.array(qpos_data)
ctrl_data = np.array(ctrl_data)
```

Рисунок 4 - Скрипт симуляции

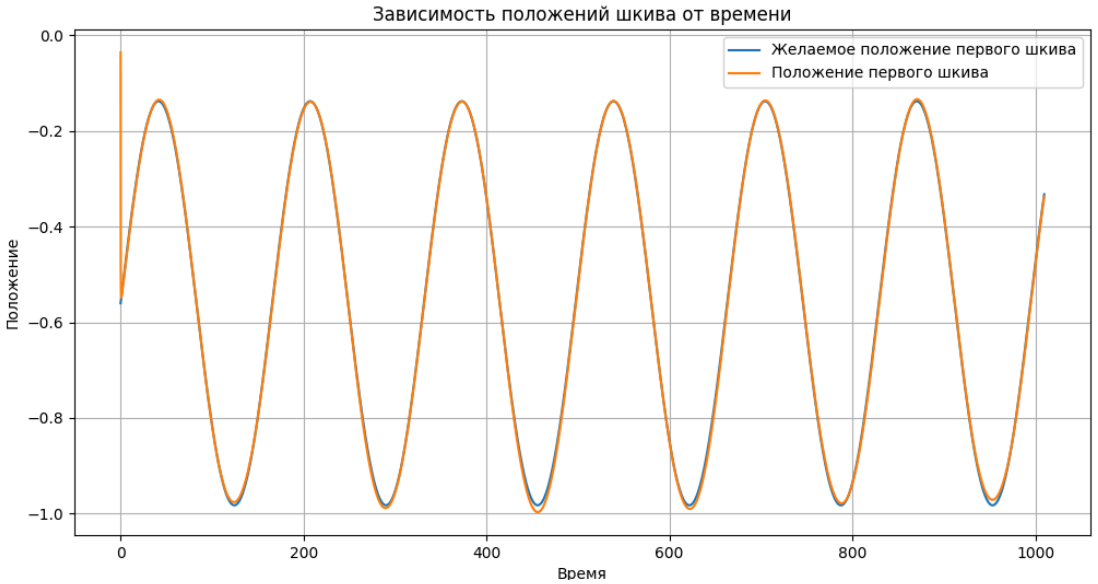

Рисунок 5 - График положения первого шкива

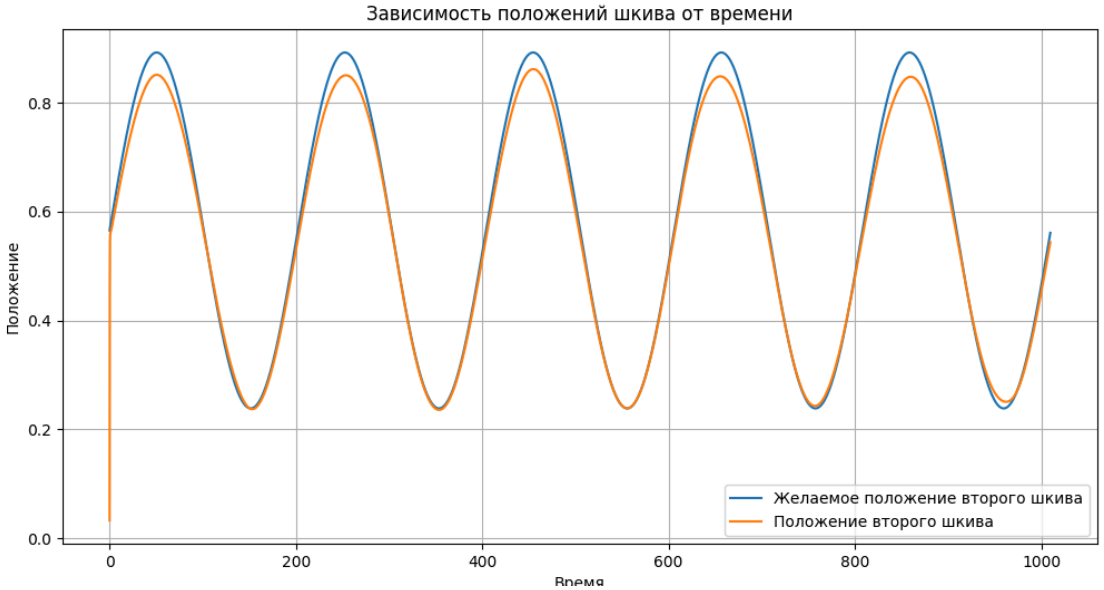

Рисунок 6 - График положения второго шкива

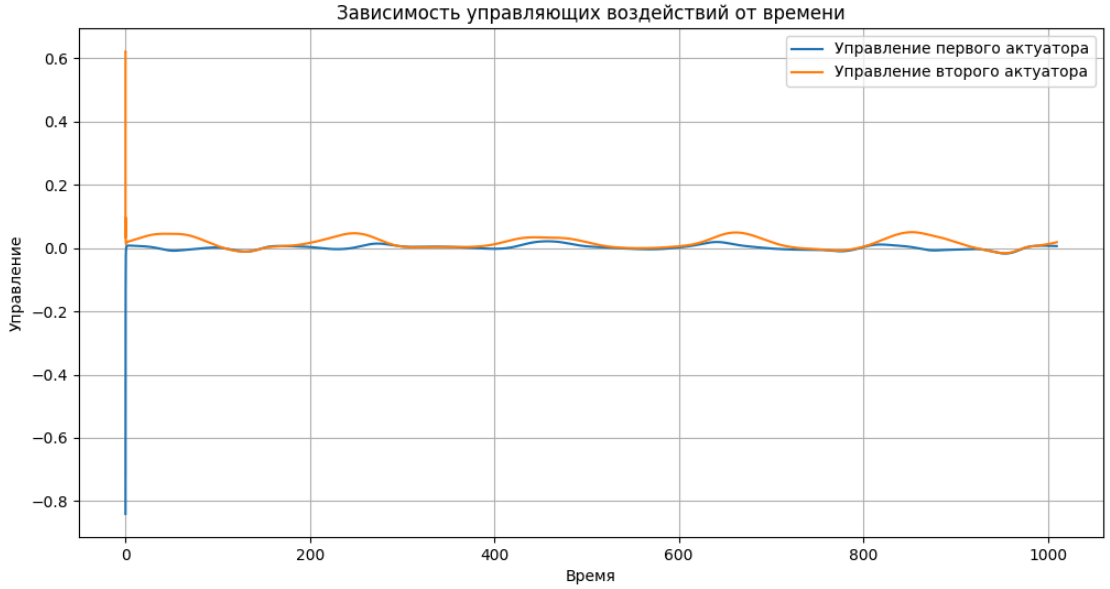

Рисунок 7 - Графики управляющих воздействий

Выводы

В ходе выполнения данного практического задания в созданный ранее файл XML были добавлены сенсоры и актуаторы, затем был реализован ПД-регулятор, осуществляющий требуемое управление. Если посмотреть на графики, то можно увидеть, что регулирование, безусловно, не идеальное. Однако, никаких требований на качество регулирования наложено не было, также следует отметить, что при иных коэффициентах механизм начинало страшно трясти, поэтому я выбрал некий компромисс: более спокойный выход на траекторию, но да, второй шкив чуть-чуть не доходит до высших точек синусоиды.