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

Отчет по практической работе №4

По дисциплине: Имитационное моделирование робототехнических систем

На тему: «Моделирование системы с приводом в среде Mujoko»

Студент:

Рязанцев Д.Л.

Группа:

R4134c

Преподаватель:

Ракшин Е. А.

Санкт-Петербург, 2025

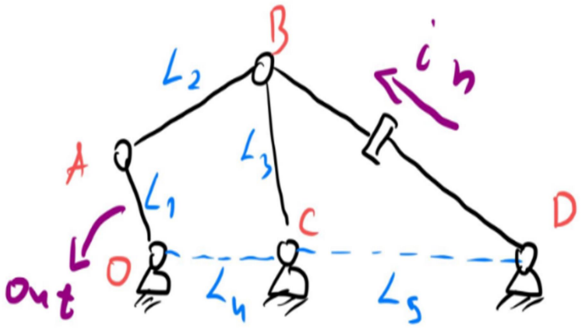
СОДЕРЖАНИЕ

Цель работы: промоделировать систему в Mujoko

Постановка задачи

- Необходимо добавить привод q1;
- Определить усилие управления с помощью ПИД-регулятора. q^des=AMP·sin(FREQ·t)+BIAS;

Исходные данные берутся в соответствии с номером ИСУ 507578. Вариант 2.




```
# Параметры длин
L1 = 0.079
L2 = 0.1027
L3 = 0.1185
L4 = 0.079
L5 = 0.395
```

Рисунок 1. Схема системы

Таблица 1 - параметры синусоидальной волны

AMP, deg	FREQ, Hz	BIAS, deg
59.02	2.58	-15.4

МОДЕЛИРОВАНИЕ

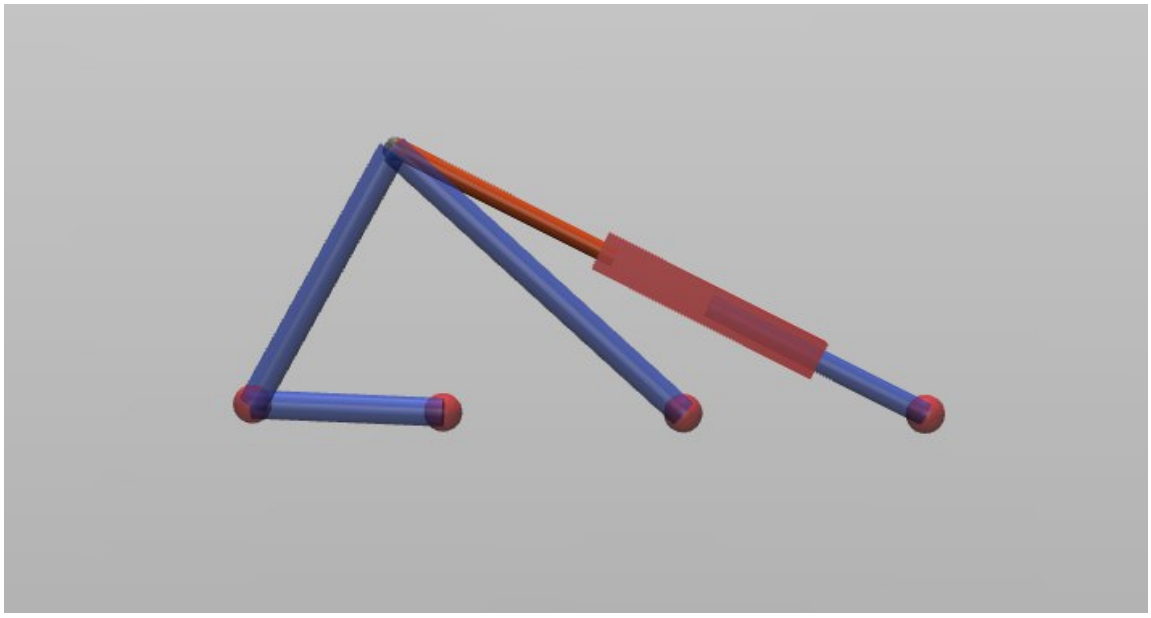


Рисунок 2. Модель коленного механизма с замкнутой цепью Оптимуса

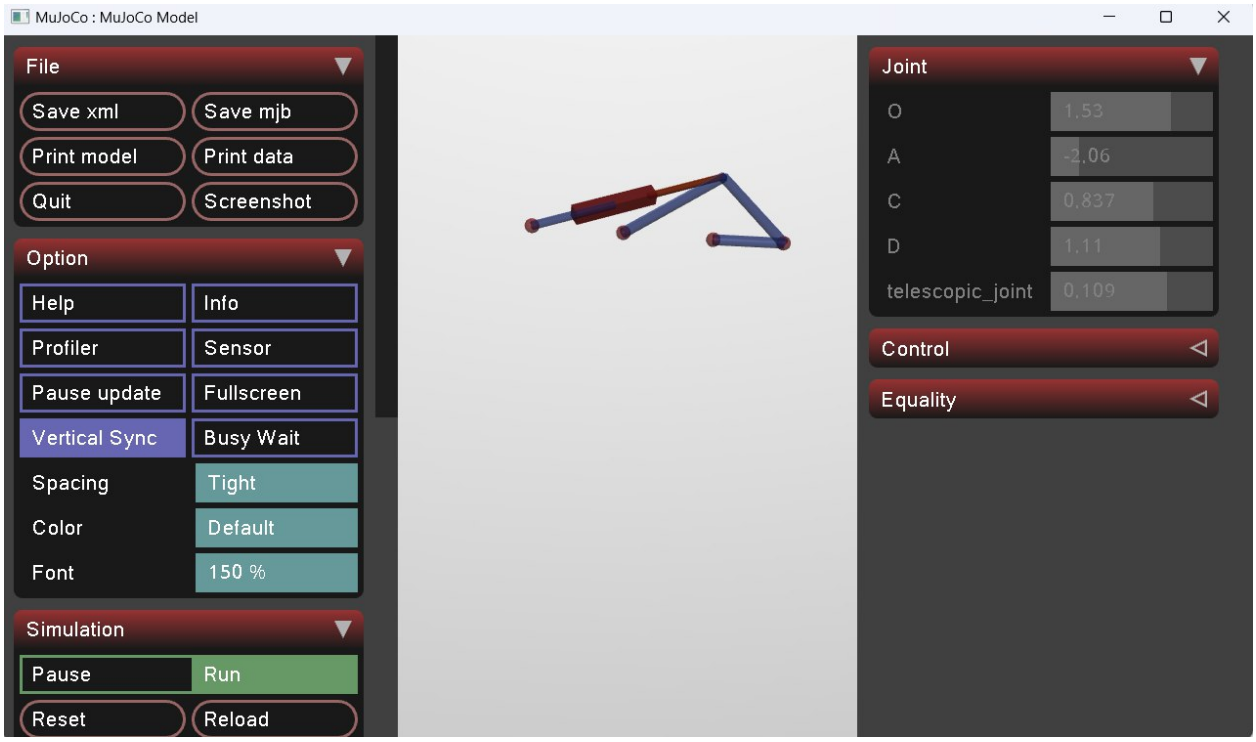


Рисунок 3. Схема системы

КОД НА python

```
import mujoco
import mujoco.viewer
import matplotlib
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import numpy as np
import os
from lxml import etree
import time

f1 = "4bar.xml"
f2 = "4bar_optimized.xml"

def swap_par(tree, element_type, element_name, attribute_name, new_value):
    element = tree.find(f'.//{element_type}[@name="{element_name}"]')
    if element is not None:
        element.set(attribute_name, new_value)
    else:
        print(f"Предупреждение: Элемент {element_type}[@name='{element_name}'] не
найден")

# Параметры длин
L1 = 0.079
L2 = 0.1027
L3 = 0.1185
L4 = 0.079
L5 = 0.395
L_BD_fixed = 0.20
L_BD_telescopic = 0.12
connector_length = 0.10
h = 1.5

# Параметры синусоидальной волны из таблицы
AMP = np.deg2rad(59.02)  # Амплитуда в радианах
FREQ = 2.58              # Частота в Гц
BIAS = np.deg2rad(-15.4) # Смещение в радианах

tree = etree.parse(f1)
swap_par(tree, 'geom', 'link OA', 'pos', f"0 -{L1/2} 0")
swap_par(tree, 'geom', 'link OA', 'size', f"0.015 {L1/2}")
swap_par(tree, 'body', 'AB', 'pos', f"0 -{L1} 0")
swap_par(tree, 'geom', 'link AB', 'pos', f"0 -{L2/2} 0")
swap_par(tree, 'geom', 'link AB', 'size', f"0.015 {L2/2}")
swap_par(tree, 'site', 'sB1', 'pos', f"0 -{L2} 0")
swap_par(tree, 'body', 'CB2', 'pos', f"{L4} 0 {h}")
swap_par(tree, 'geom', 'link CB', 'pos', f"0 -{L3/2} 0")
swap_par(tree, 'geom', 'link CB', 'size', f"0.015 {L3/2}")
swap_par(tree, 'site', 'sB2', 'pos', f"0 -{L3} 0")
swap_par(tree, 'body', 'DB3', 'pos', f"{L4 + L5} 0 {h}")
```

```
swap_par(tree, 'geom', 'link DB_fixed', 'pos', f"0 -{L_BD_fixed/2} 0")
swap_par(tree, 'geom', 'link DB_fixed', 'size', f"0.012 {L_BD_fixed/2}")


swap_par(tree, 'geom', 'connector', 'pos', f"0 -{L_BD_fixed + connector_length/2}
0")
swap_par(tree, 'geom', 'connector', 'size', f"0.02 {connector_length/2} 0.02")
telescopic_body = tree.find('.//body[@name="BD_telescopic"]')
if telescopic_body is not None:
    telescopic_body.set('pos', f"0 -{L_BD_fixed + connector_length} 0")

telescopic_geom = tree.find('.//geom[@name="link DB_telescopic"]')
if telescopic_geom is not None:
    telescopic_geom.set('pos', f"0 -{L_BD_telescopic/2} 0")
    telescopic_geom.set('size', f"0.01 {L_BD_telescopic/2}")

telescopic_site = tree.find('.//site[@name="sB3"]')
if telescopic_site is not None:
    telescopic_site.set('pos', f"0 -{L_BD_telescopic} 0")


telescopic_joint = tree.find('.//joint[@name="telescopic_joint"]')
if telescopic_joint is not None:
    telescopic_joint.set('range', f"{-L_BD_telescopic/2} {L_BD_telescopic/2}")
    telescopic_joint.set('damping', '0.02')


actuator = tree.find('.//actuator')
if actuator is not None:
    actuator.clear()

# Добавляем привод для q1 (сустава A)
position_actuator = etree.SubElement(actuator, 'position')
position_actuator.set('name', 'q1_control')
position_actuator.set('joint', 'A')
position_actuator.set('ctrllimited', 'true')
position_actuator.set('ctrlrange', f'{-np.pi} {np.pi}')  # Ограничение ±180
градусов
position_actuator.set('kp', '500')
position_actuator.set('kv', '20')

# Добавляем привод для телескопического звена
position_actuator_telescopic = etree.SubElement(actuator, 'position')
position_actuator_telescopic.set('name', 'telescopic_control')
position_actuator_telescopic.set('joint', 'telescopic_joint')
position_actuator_telescopic.set('ctrllimited', 'true')
position_actuator_telescopic.set('ctrlrange', f'{-1.0} {1.0}')
position_actuator_telescopic.set('kp', '800')
position_actuator_telescopic.set('kv', '15')

# Обновляем сенсоры
```

```
sensor = tree.find('.//sensor')
if sensor is not None:
    sensor.clear()

# Добавляем сенсоры для контроля состояния
jointpos_A = etree.SubElement(sensor, 'jointpos')
jointpos_A.set('joint', 'A')
jointpos_A.set('name', 'joint_A_pos')

jointpos_telescopic = etree.SubElement(sensor, 'jointpos')
jointpos_telescopic.set('joint', 'telescopic_joint')
jointpos_telescopic.set('name', 'telescopic_joint_pos')

# Сохраняем обновленную модель
tree.write(f2, pretty_print=True, xml_declaration=True, encoding='UTF-8')

# Загружаем модель в MuJoCo
model = mujoco.MjModel.from_xml_path(f2)
data = mujoco.MjData(model)

class PIDController:
    def __init__(self, kp, ki, kd, output_limits=(-np.inf, np.inf)):
        self.kp = kp
        self.ki = ki
        self.kd = kd
        self.output_limits = output_limits
        self.reset()

    def reset(self):
        self.integral = 0.0
        self.previous_error = 0.0

    def compute(self, error, dt):
        # Пропорциональная составляющая
        proportional = self.kp * error

        # Интегральная составляющая с ограничением
        self.integral += error * dt
        self.integral = np.clip(self.integral, -1.0, 1.0)
        integral = self.ki * self.integral

        # Дифференциальная составляющая
        if dt > 0:
            derivative = self.kd * (error - self.previous_error) / dt
        else:
            derivative = 0.0

        # Суммируем все составляющие
        output = proportional + integral + derivative

        # Ограничиваем выходной сигнал
```

```
        output = np.clip(output, self.output_limits[0], self.output_limits[1])

        self.previous_error = error
        return output

# Получаем индексы для управления
q1_actuator_idx = mujoco.mj_name2id(model, mujoco.mjtObj.mjOBJ_ACTUATOR,
"q1_control")
q1_joint_idx = mujoco.mj_name2id(model, mujoco.mjtObj.mjOBJ_JOINT, "A")

telescopic_actuator_idx = mujoco.mj_name2id(model, mujoco.mjtObj.mjOBJ_ACTUATOR,
"telescopic_control")
telescopic_joint_idx = mujoco.mj_name2id(model, mujoco.mjtObj.mjOBJ_JOINT,
"telescopic_joint")

# Создаем ПИД-регулятор для q1
pid_controller = PIDController(
    kp=300.0,
    ki=10.0,
    kd=8.0,
    output_limits=(-50.0, 50.0)
)

# Функция для вычисления желаемого угла
def desired_angle(t):
    return AMP * np.sin(2 * np.pi * FREQ * t) + BIAS

# Основной цикл симуляции
with mujoco.viewer.launch_passive(model, data) as viewer:
    viewer.cam.distance = 2.5
    viewer.cam.azimuth = 90
    viewer.cam.elevation = -15
    viewer.cam.lookat[:] = [0.2, -0.2, 1.3]

    print("=== СИМУЛЯЦИЯ УПРАВЛЕНИЯ ПРИВОДОМ q1 ===")
    print(f"Параметры синусоиды: AMP={np.rad2deg(AMP):.2f}°, FREQ={FREQ} Гц,
BIAS={np.rad2deg(BIAS):.2f}°")

    # Переменные для сбора данных
    time_history = []
    desired_angle_history = []
    actual_angle_history = []
    control_signal_history = []
    error_history = []

    start_time = time.time()
    simulation_duration = 5.0  # Длительность симуляции в секундах

    pid_controller.reset()

    while time.time() - start_time < simulation_duration:
```

```
        current_time = time.time() - start_time

        # Вычисляем желаемый угол
        q_des = desired_angle(current_time)

        # Получаем текущий угол
        if q1_joint_idx != -1:
            q_actual = data.qpos[model.jnt_qposadr[q1_joint_idx]]
        else:
            q_actual = 0.0

        # Вычисляем ошибку
        error = q_des - q_actual

        # Вычисляем управляющее воздействие через ПИД-регулятор
        dt = model.opt.timestep
        control_signal = pid_controller.compute(error, dt)

        # Применяем управляющее воздействие
        if q1_actuator_idx != -1:
            data.ctrl[q1_actuator_idx] = control_signal

        # Шаг симуляции
        mujoco.mj_step(model, data)
        viewer.sync()

        # Данные для анализа
        time_history.append(current_time)
        desired_angle_history.append(q_des)
        actual_angle_history.append(q_actual)
        control_signal_history.append(control_signal)
        error_history.append(error)

        # Вывод информации каждые 0.5 секунды
        if len(time_history) % 500 == 0:
            print(f"Время: {current_time:.2f}с, Желаемый угол:
{np.rad2deg(q_des):.2f}°, "
                  f"Фактический угол: {np.rad2deg(q_actual):.2f}°, Ошибка:
{np.rad2deg(error):.2f}°")

        time.sleep(0.001)

    print("Симуляция завершена")

    # Анализ результатов и построение графиков
    if time_history:
        plt.figure(figsize=(12, 10))

        # График 1: Желаемый и фактический углы
        plt.subplot(3, 1, 1)
```

```
        plt.plot(time_history, np.rad2deg(desired_angle_history), 'r-',
linewidth=2, label='Желаемый угол')
        plt.plot(time_history, np.rad2deg(actual_angle_history), 'b-',
linewidth=2, label='Фактический угол')
        plt.ylabel('Угол [°]')
        plt.title('Управление приводом q1 с ПИД-регулятором')
        plt.legend()
        plt.grid(True)

        # График 2: Ошибка управления
        plt.subplot(3, 1, 2)
        plt.plot(time_history, np.rad2deg(error_history), 'g-', linewidth=1,
label='Ошибка')
        plt.ylabel('Ошибка [°]')
        plt.xlabel('Время [с]')
        plt.title('Ошибка управления')
        plt.legend()
        plt.grid(True)

        # График 3: Сигнал управления
        plt.subplot(3, 1, 3)
        plt.plot(time_history, control_signal_history, 'orange', linewidth=1,
label='Сигнал управления')
        plt.ylabel('Управление')
        plt.xlabel('Время [с]')
        plt.title('Сигнал управления ПИД-регулятора')
        plt.legend()
        plt.grid(True)

        plt.tight_layout()
        plt.savefig('q1_pid_control.png', dpi=300, bbox_inches='tight')
        print("Графики сохранены в 'q1_pid_control.png'")

        # Статистика
        max_error = np.max(np.abs(np.rad2deg(error_history)))
        rmse = np.sqrt(np.mean(np.array(np.rad2deg(error_history))**2))

        print(f"\nСТАТИСТИКА:")
        print(f"Максимальная ошибка: {max_error:.2f}°")
        print(f"Среднеквадратичная ошибка: {rmse:.2f}°")
        print(f"Диапазон желаемого угла: от
{np.min(np.rad2deg(desired_angle_history)):.2f}° до
{np.max(np.rad2deg(desired_angle_history)):.2f}°")
        print(f"Диапазон фактического угла: от
{np.min(np.rad2deg(actual_angle_history)):.2f}° до
{np.max(np.rad2deg(actual_angle_history)):.2f}°")

print("=== СИМУЛЯЦИЯ ЗАВЕРШЕНА ===")
```

МОДЕЛЬ В XML

```
<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"/>
<material name="connector_red" rgba="0.9 0.2 0.2 0.8"/>
</asset>
<worldbody>
<light pos="0 0 10"/>
<geom type="plane" size="0.5 0.5 0.1" material="grid"/>
<body name="OAB" pos="0 0 1.5" euler="90 0 180">
<joint name="O" type="hinge" axis="0 0 1" stiffness="0" damping="0.05"/>
<geom name="point O" type="sphere" pos="0 0 0" size="0.02" rgba="0.89 0.14 0.16
0.5"/>
<geom name="link OA" type="cylinder" pos="0 -0.0395 0" size="0.015 0.0395"
rgba="0.21 0.32 0.82 0.5" euler="90 0 0"/>
<body name="AB" pos="0 -0.079 0" euler="0 0 0">
<joint name="A" type="hinge" axis="0 0 1" stiffness="0" damping="0.05"/>
<geom name="point A" type="sphere" pos="0 0 0" size="0.02" rgba="0.89 0.14 0.16
0.5"/>
<geom name="link AB" type="cylinder" pos="0 -0.05135 0" size="0.015 0.05135"
rgba="0.21 0.32 0.82 0.5" euler="90 0 0"/>
<site name="sB1" size="0.01" pos="0 -0.1027 0"/>
</body>
</body>
<body name="CB2" pos="0.079 0 1.5" euler="90 0 180">
<joint name="C" type="hinge" axis="0 0 1" stiffness="0" damping="0.05"/>
<geom name="point C" type="sphere" pos="0 0 0" size="0.02" rgba="0.89 0.14 0.16
0.5"/>
<geom name="link CB" type="cylinder" pos="0 -0.05925 0" size="0.015 0.05925"
rgba="0.21 0.32 0.82 0.5" euler="90 0 0"/>
<site name="sB2" size="0.01" pos="0 -0.1185 0"/>
</body>
<body name="DB3" pos="0.474 0 1.5" euler="90 0 180">
<joint name="D" type="hinge" axis="0 0 1" stiffness="0" damping="0.05"/>
<geom name="point D" type="sphere" pos="0 0 0" size="0.02" rgba="0.89 0.14 0.16
0.5"/>
<geom name="link DB_fixed" type="cylinder" pos="0 -0.1 0" size="0.012 0.1"
rgba="0.21 0.32 0.82 0.5" euler="90 0 0"/>
<geom name="connector" type="box" size="0.02 0.05 0.02" rgba="0.9 0.2 0.2 0.8"
pos="0 -0.2 0"/>
<body name="BD_telescopic" pos="0 -0.2 0">
<joint name="telescopic_joint" type="slide" axis="0 -1 0" limited="true" range="-
0.06 0.06" stiffness="0" damping="0.02"/>
```

```
<geom name="link DB_telescopic" type="cylinder" pos="0 -0.06 0" size="0.01 0.06"
rgba="0.9 0.3 0.1 0.8" euler="90 0 0"/>
<site name="sB3" size="0.01" pos="0 -0.12 0"/>
<site name="connector_start" size="0.005" pos="0 0.06 0" rgba="1 0 0 1"/>
</body>
</body>
</worldbody>
<equality>
<connect site1="sB1" site2="sB2" solimp="0.8 0.9 0.01" solref="0.01 0.5"/>
<connect site1="sB2" site2="sB3" solimp="0.8 0.9 0.01" solref="0.01 0.5"/>
</equality>
<actuator>
<position name="q1_control" joint="A" ctrllimited="true" ctrlrange="-3.14 3.14"
kp="500" kv="20"/>
<position name="telescopic_control" joint="telescopic_joint" ctrllimited="true"
ctrlrange="-1.0 1.0" kp="800" kv="15"/>
</actuator>
<sensor>
<jointpos joint="A" name="joint_A_pos"/>
<jointpos joint="telescopic_joint" name="telescopic_joint_pos"/>
</sensor>
</mujoco>
```

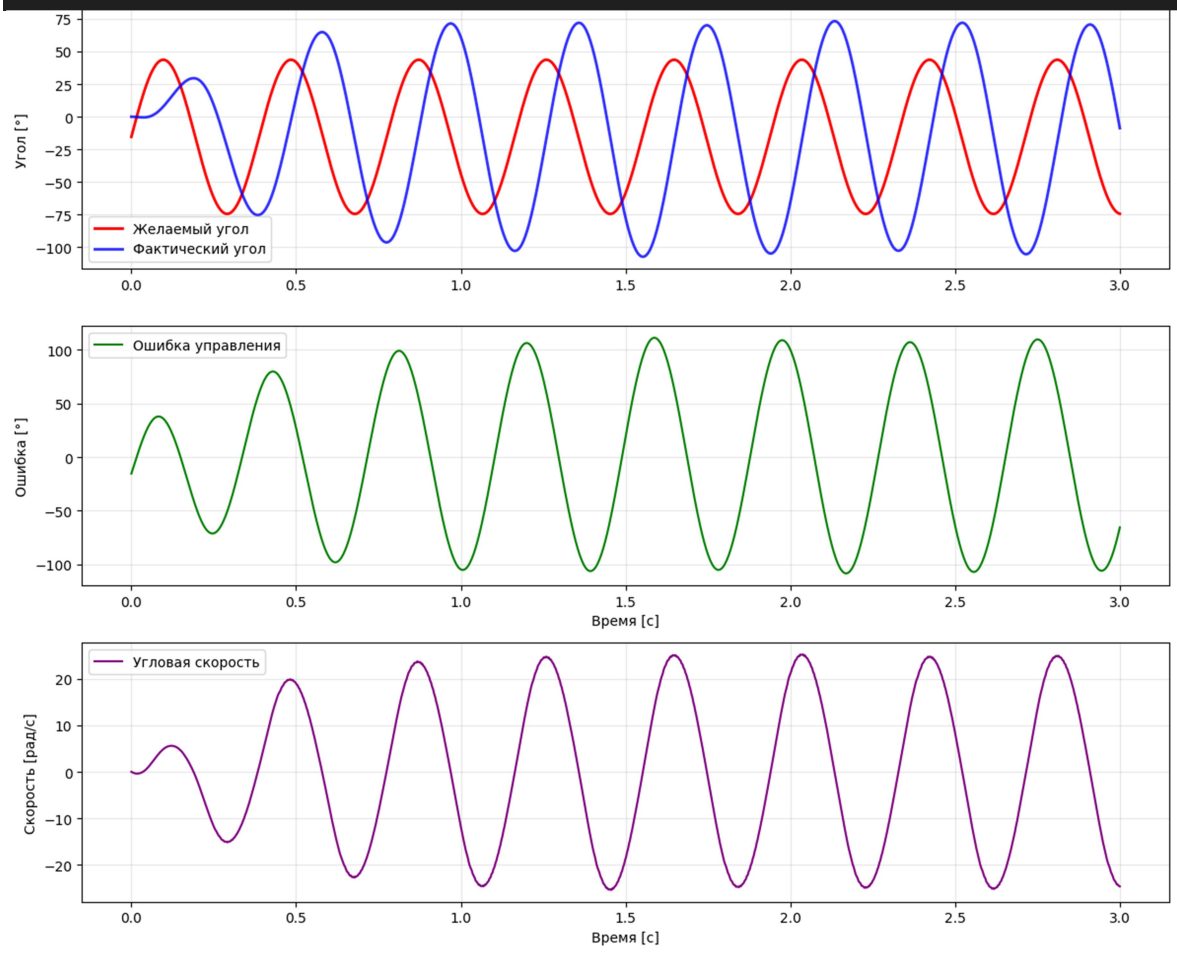


Рисунок 4. Управление механизмом «Оптимус» с ПИД-регулятором

Заключение

Данная система представляет собой комплекс, реализующий управление четырехзвенным механизмом с телескопическим звеном через физический движок MuJoCo. По своей сути, это виртуальный прототип реального механического устройства, где математическая модель точно воспроизводит физические свойства и динамические процессы.

Основная идея системы заключается в создании адаптивного механизма, способного изменять свою геометрию в реальном времени за счет телескопического элемента. Это не просто статическая конструкция, а динамическая система, где изменение длины одного звена вызывает перераспределение усилий и перемещений во всей кинематической цепи. Особенность подхода заключается в балансировке параметров управления — демпфирования, жесткости и управляющих воздействий, что обеспечивает плавное и предсказуемое поведение системы.