THE MINISTRY OF SCIENCE AND HIGHER EDUCATION OF THE RUSSIAN FEDERATION

ITMO University

Report for the task 4 " Optimus knee closed-chain mechanism"

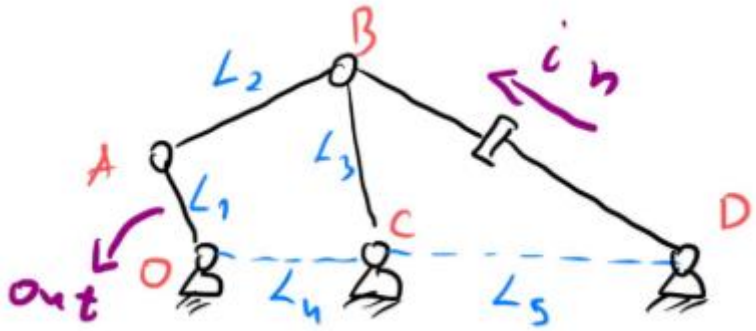
for the subject

SIMULATION OF ROBOTIC SYSYTEMS

Student:

ID_Number 503259 *Afif, Hazem*

Saint Petersburg 24/11/2025



Optimus' knee mechanism overview

A closed-loop linkage system that can convert the linear movement of a slider joint into rotational movement for a hinge joint and vice versa. This system operates on the principle that when the link length follows a specific ratio, half of the slider's movement approximates a straight line. This distinctive feature makes it suitable for use in the knee joints of walking robots.

In this task, we will generate a periodic rotational motion at the output angle OA, such that it follows a prescribed sinusoidal trajectory, using only a linear actuator placed between points B and D.

XML file

The parameters of the figure are shown in the following table:

Parameter	L_1	L_2	L_3	L_4	L_5
Value (m)	0.044	0.0572	0.066	0.044	0.22

We will add to our xml file from the last task the actuator and sensor containers as follows:

```
<mujoco model="Optimus">
 <option gravity="0 -9.81 0" timestep="0.0001" solver="Newton" iterations="200" tolerance="1e-8"/>
 <default>
    <site rgba="0 0 1 1"/>
    <geom rgba="1 0 0 1"/>
 </default>
 <worldbody>
    <light pos="0 0 3"/>
    <geom type="plane" size="5 5 0.1" rgba=".5 .5 .5 .7" />
    <body name="base" pos="0 0 0.5">
        <geom type="sphere" size="0.015" rgba="0 0 1 1"/>
    </body>
    <body name="L1" pos="0 0 0.5">
        <joint name="O" type="hinge" axis="0 1 0" range="-90 90"/>
        <geom type="cylinder" fromto="0 0 0 0 0 0.044" size="0.01"/>
        <body name="L2" pos="0 0 0.044">
            <joint name="A" type="hinge" axis="0 1 0" range="-90 90"/>
            <geom name="A" type="sphere" size="0.015" rgba="1 1 1 1"/>
            <geom name="L2" type="cylinder" fromto="0 0 0 0.044 0 0.022" size="0.01" rgba="1 0 0 1"/>
            <site name="s1" pos="0.044 0 0.022" size="0.005" rgba="0 1 0 1"/>
        </body>
```

```
    </body>
    <body name="L3_base" pos="0.044 0 0.5">
        <geom type="sphere" size="0.015" rgba="0 0 1 1"/>
    </body>
    <body name="L3" pos="0.044 0 0.5">
        <joint name="C" type="hinge" axis="0 1 0" range="-90 90"/>
        <geom type="cylinder" fromto="0 0 0 0 0 0.066" size="0.01"/>
        <site name="s2" pos="0 0 0.066" size="0.005" rgba="1 0 1 1"/>
        <body pos="0 0 0.066">
            <geom name="B" type="sphere" size="0.015" rgba="1 1 1 1"/>
        </body>
    </body>
    <body name="L5" pos="0.264 0 0.5">
        <joint name="D" type="hinge" axis="0 1 0" range="-90 180"/>
        <geom type="sphere" size="0.015"/>
        <site name="D_point" pos="0 0 0" size="0.005"/>
    </body>
 </worldbody>
 <equality>
    <connect site1="s1" site2="s2"/>
 </equality>
 <actuator >
    <general name="motor" cranksite="s2" slidersite="D_point" ctrlrange="0 2" cranklength="0.1"/>
 </actuator >
 <sensor>
    <framepos name="sensor_A_pos" objtype="site" objname="A_point"/>
 </sensor>
</mujoco>
```

Python Code

The next step is to create a Python script to load the mechanism model from an XML file that defines all the bodies and joints and then animates and analyzes the motion of the Optimus closed-chain mechanism in MuJoCo while applying a time-varying control input.

```
import mujoco
from mujoco.viewer import launch_passive
import mujoco_viewer
import numpy as np
import matplotlib.pyplot as plt
import time

model_path=r"C:\users\user\OneDrive\Desktop\MuJoCo\hazem2.xml"
model = mujoco.MjModel.from_xml_path(model_path)
data = mujoco.MjData(model)

def set_torque(mj_data, KP, KV, theta):
    data.ctrl[0] = KP * (-mj_data.qpos[0] + theta) + KV * (0 - mj_data.qvel[0])

SIMEND = 5
TIMESTEP = 0.01
STEP_NUM = int(SIMEND / TIMESTEP)
timeseries = np.linspace(0, SIMEND, STEP_NUM)
```

```
FREQ = 1.59 # [Hz]
AMP = np.deg2rad(21.47) # [rad]
BIAS = np.deg2rad(-2.1) # [rad]

theta_des = AMP * np.sin(FREQ * timeseries) + BIAS

EE_position_x = []
EE_position_z = []

viewer = mujoco_viewer.MujocoViewer(model,
                                    data,
                                    title="Optimus",
                                    width=1920,
                                    height=1080)

for i in range(STEP_NUM):
    if viewer.is_alive:
        set_torque(data, 20, 5, theta_des[i])

        position_EE = data.site_xpos[1]
        EE_position_x.append(position_EE[0])
        EE_position_z.append(position_EE[2])

        mujoco.mj_step(model, data)
        viewer.render()

    else:
        break
viewer.close()

midlength = int(STEP_NUM/2)


plt.clf()
plt.plot(timeseries[midlength:], EE_position_z[midlength:], '-', linewidth=2, label='P')
plt.legend()
plt.xlabel("time (s)")
plt.ylabel("q1 (rad)")
plt.axis('equal')
plt.grid()
plt.show()
```

The desired output is the rotation of segment OA which must follow:

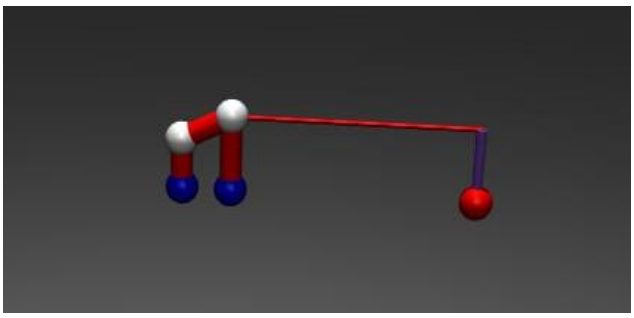
$$\theta_{OA}(t) = AMP \sin Freq.t + BIAS$$

Where:

Parameter	$AMP\ (deg)$	$Freq\ (Hz)$	$BIAS\ (deg)$
Value	21.47	1.59	-2.1

Results

The script begins by loading the MuJoCo model of the mechanism and initializing the simulation data structure.

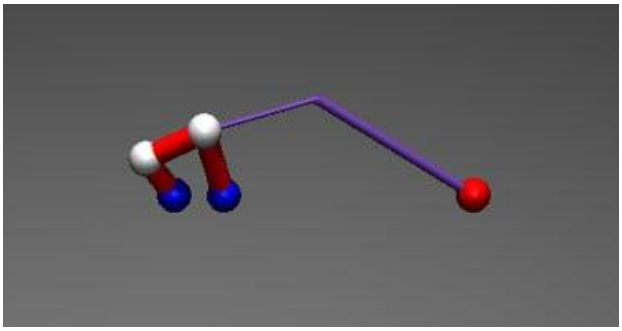


Optimus closed-chain mechanism in MuJoCo with linear actuator

A sinusoidal reference signal is then generated, representing the desired angular motion the system is intended to follow. This signal has a prescribed amplitude, frequency, and bias shift, and it defines the target motion over the full simulation period.

A simple proportional-derivative (PD) controller is implemented to create torque commands based on the difference between the desired motion and the current joint state. This controller attempts to reduce both position error and velocity error, producing a corrective torque that drives the mechanism toward the target trajectory.

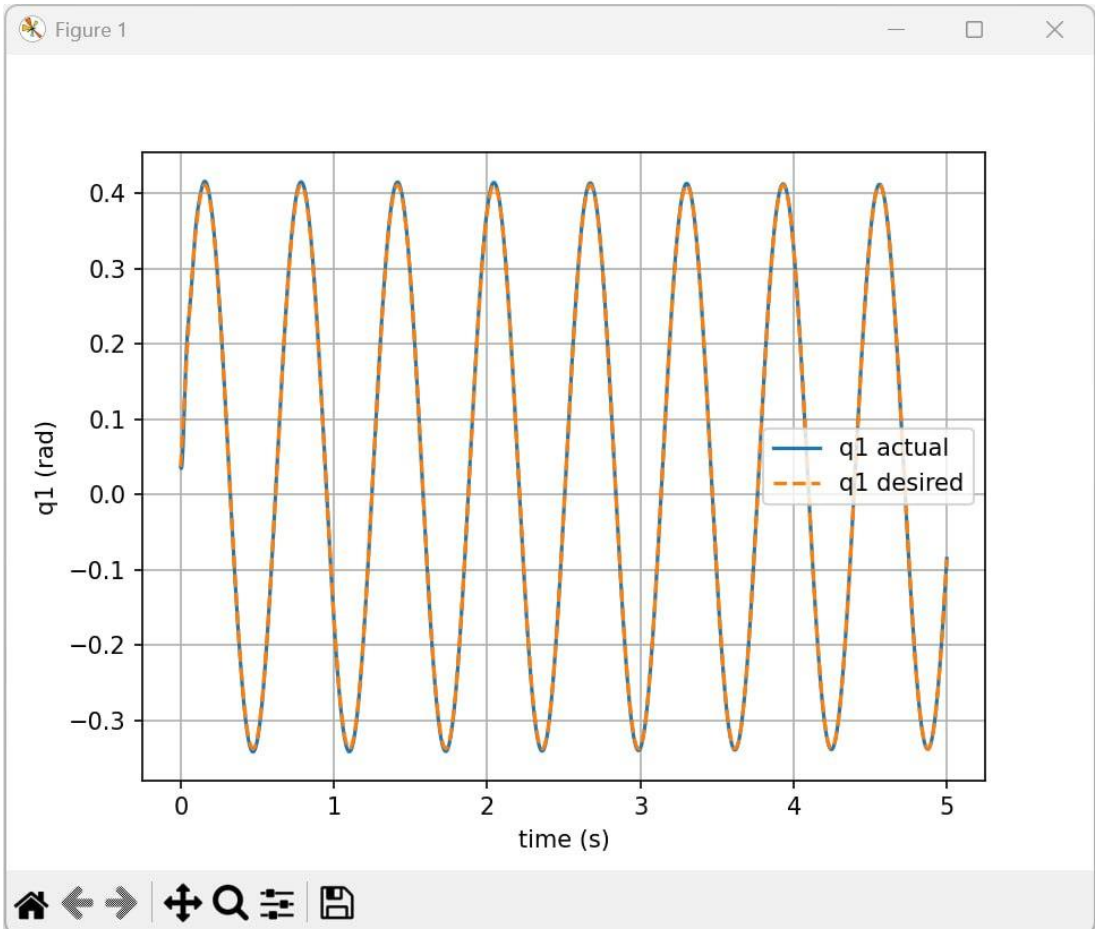
A visualization window is opened so that the motion of the system can be observed in real time as the simulation progresses.



Optimus closed-chain mechanism in MuJoCo during the motion

At every simulation step, the controller computes the torque required for the current desired angle, MuJoCo advances the dynamics, and the viewer renders the updated configuration. At the same time, the script records the position of the end-effector point on the mechanism throughout the entire simulation, allowing its movement to be analyzed afterward.

Once the simulation ends, the recorded motion is plotted to illustrate how the selected point moves over time.



The plot compares two signals:

- The desired OA rotational angle, defined as a sinusoidal function.
- The actual OA rotational angle, produced by the mechanism as it is driven by the PD controller.

The results demonstrate that the mechanism is able to closely follow the prescribed sinusoidal trajectory. The actual angle overlaps almost perfectly with the desired signal throughout the entire simulation period. This indicates that the control input applied to the linear actuator is effectively transmitted through the closed-chain mechanical structure, generating the intended periodic motion at point A.

Overall, the comparison confirms that the actuation strategy and the mechanical geometry of the linkage are well-suited to generate a periodic rotational output from a linear actuation input.