5 DOF Robotic Arm – 3D Design

Mechanical Engineering

[Task 1]

Mohamad Abdulmoula,R. Rama

ramamohameed9@gmail.com

Department of Software Engineer

Under the Supervision Of:

Eng Asmaa Duramae

July 8, 2025

Contents

Glossary of Terms

Term	Definition
DOF (Degrees of Freedom)	The number of independent movements a mechanical system or joint can perform. A 5 DOF robotic arm means five joints, each allowing rotation in one direction.
Armature	A skeletal structure composed of bones used in 3D modeling and animation to control mesh deformations.
Bone	A unit within an armature that influences part of a mesh; bones define movement or rotation.
Rigging	The process of creating an armature and binding it to a mesh to allow articulated motion.
Skinning	Binding the mesh geometry to bones so that the mesh follows bone transformations during animation.
Keyframe	A specific frame in the animation timeline where a property (e.g., position or rotation) is explicitly defined. Blender interpolates values between keyframes.
Animation Timeline	A sequence of frames over time where keyframes are set to create motion in an animation.
Revolute Joint	A rotational joint allowing motion around a single axis. Common in robotic arms.
Serial Chain	A configuration where each joint or segment is connected in sequence, and each movement is relative to its parent.
Mesh	The 3D model made of vertices, edges, and faces that form the physical geometry of objects.
Auto Weight	A Blender feature that automatically calculates how much influence each bone has on surrounding mesh vertices.
Python API	Blender’s scripting interface that allows users to control and automate actions using Python.

Project Overview

As part of our exploration into 3D mechanical modeling and animation, we designed a fully rigged and animated **five degrees of freedom (5 DOF)** robotic arm using **Blender 3D** and **Python scripting**. This project aimed to simulate a realistic robotic manipulator that demonstrates fundamental principles of robot kinematics, rigging, and animation control.

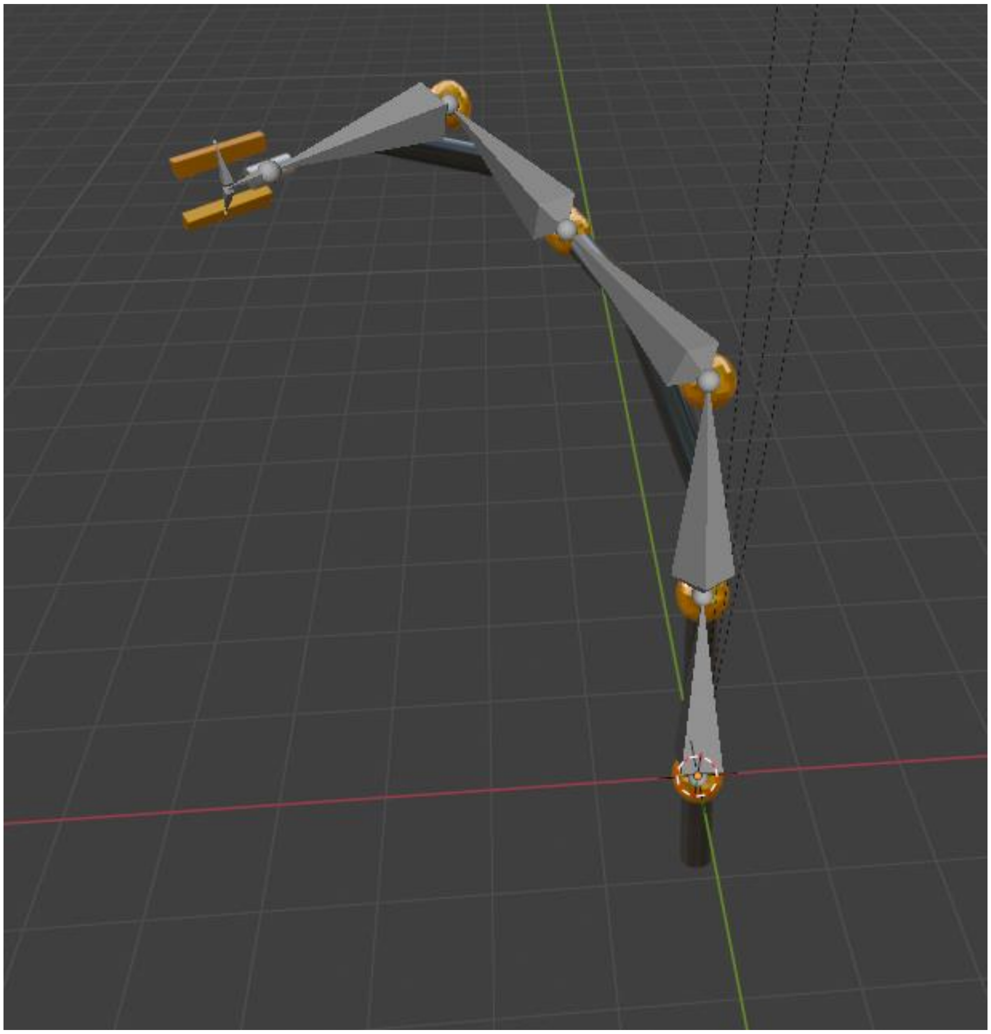
1. Planning

At the outset, we defined the primary objective: to construct a 3D robotic arm capable of performing rotational movements through a chain of five revolute joints, culminating in a functional two-finger gripper. Our goal was to produce a fully rigged model that could not only be visualized interactively in Blender but also exported for use in real-time environments such as game engines or robotics simulation platforms. We selected Blender 3.x as our development platform due to its advanced rigging tools, built-in rendering engine, and full scripting support through the Python API.

2. Requirements Analysis

During the analysis phase, we identified both the functional and non-functional requirements. Functionally, the robotic arm was required to include five independent joints, represented as bones in a skeletal structure (armature), with each joint contributing one rotational degree of freedom. The final segment was to include a gripper mechanism, composed of two opposing fingers. Non-functional requirements included code modularity, visual clarity via material shading, smooth animation between keyframes.

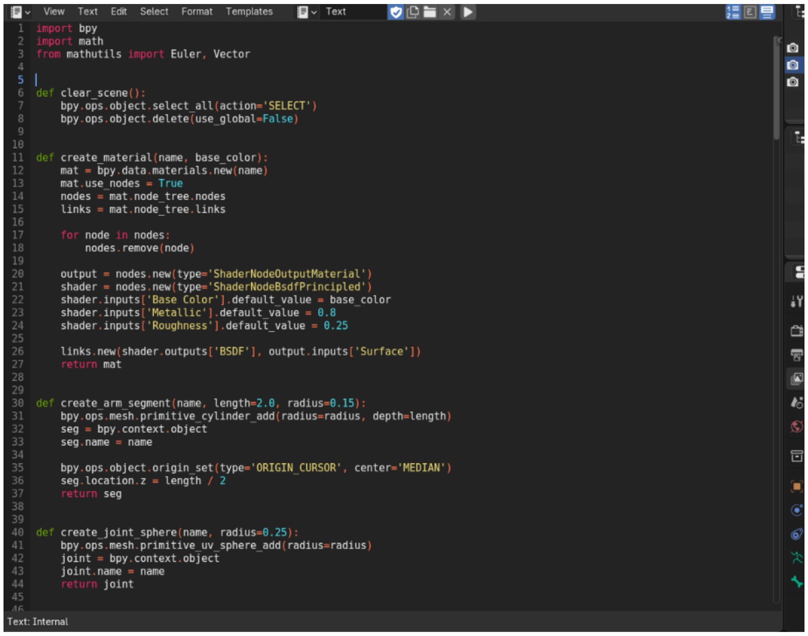
3. Design



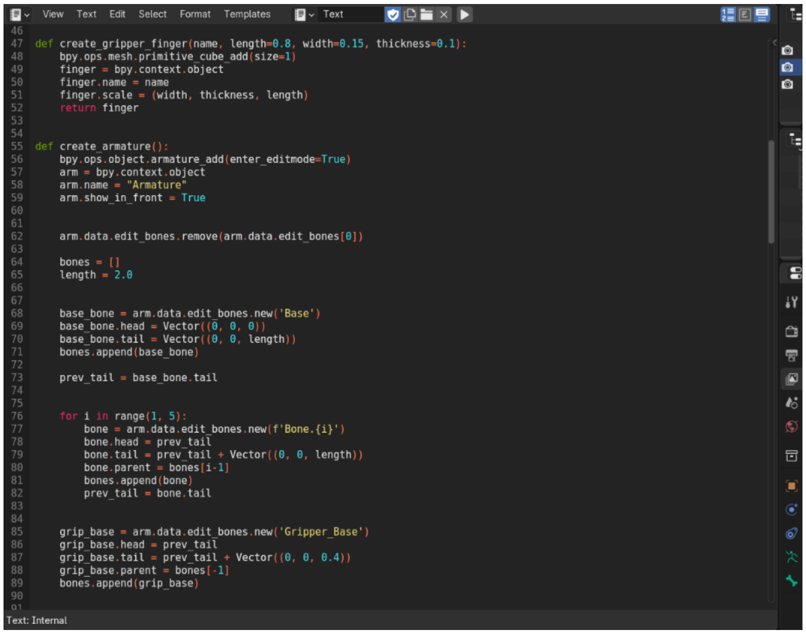
We designed the robotic arm following a **serial kinematic chain**, where each segment connects sequentially to the previous one. The structure includes a base joint, followed by four rotational joints, and terminates in a gripper assembly. Each link was represented by a cylinder, and the joints were visualized using spheres for clarity. The gripper mechanism was designed using scaled cubes, one for each finger, attached to the final bone in the hierarchy. This modular design allowed us to simulate realistic motion by rotating each joint in a hierarchical manner, where movement of a parent joint affects all its children.

3. Coding

This section contains the Python script that constructs, rigs, and animates a 3D robotic arm with five degrees of freedom and a functional gripper using Blender's scripting API.



```python
import bpy
import math
from mathutils import Euler, Vector


def clear_scene():
    bpy.ops.object.select_all(action='SELECT')
    bpy.ops.object.delete(use_global=False)


def create_material(name, base_color):
    mat = bpy.data.materials.new(name)
    mat.use_nodes = True
    nodes = mat.node_tree.nodes
    links = mat.node_tree.links

    for node in nodes:
        nodes.remove(node)

    output = nodes.new(type='ShaderNodeOutputMaterial')
    shader = nodes.new(type='ShaderNodeBsdfPrincipled')
    shader.inputs['Base Color'].default_value = base_color
    shader.inputs['Metallic'].default_value = 0.8
    shader.inputs['Roughness'].default_value = 0.25

    links.new(shader.outputs['BSDF'], output.inputs['Surface'])
    return mat


def create_arm_segment(name, length=2.0, radius=0.15):
    bpy.ops.mesh.primitive_cylinder_add(radius=radius, depth=length)
    seg = bpy.context.object
    seg.name = name

    bpy.ops.object.origin_set(type='ORIGIN_CURSOR', center='MEDIAN')
    seg.location.z = length / 2
    return seg


def create_joint_sphere(name, radius=0.25):
    bpy.ops.mesh.primitive_uv_sphere_add(radius=radius)
    joint = bpy.context.object
    joint.name = name
    return joint
```

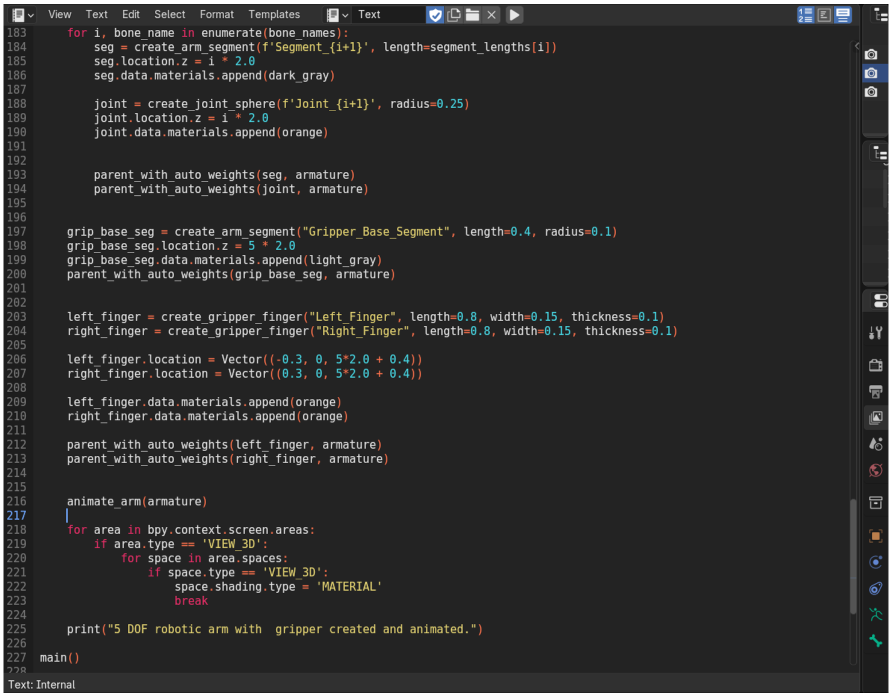


```python
def create_gripper_finger(name, length=0.8, width=0.15, thickness=0.1):
    bpy.ops.mesh.primitive_cube_add(size=1)
    finger = bpy.context.object
    finger.name = name
    finger.scale = (width, thickness, length)
    return finger


def create_armature():
    bpy.ops.object.armature_add(enter_editmode=True)
    arm = bpy.context.object
    arm.name = "Armature"
    arm.show_in_front = True


    arm.data.edit_bones.remove(arm.data.edit_bones[0])

    bones = []
    length = 2.0


    base_bone = arm.data.edit_bones.new('Base')
    base_bone.head = Vector((0, 0, 0))
    base_bone.tail = Vector((0, 0, length))
    bones.append(base_bone)

    prev_tail = base_bone.tail


    for i in range(1, 5):
        bone = arm.data.edit_bones.new(f'Bone.{i}')
        bone.head = prev_tail
        bone.tail = prev_tail + Vector((0, 0, length))
        bone.parent = bones[i-1]
        bones.append(bone)
        prev_tail = bone.tail


    grip_base = arm.data.edit_bones.new('Gripper_Base')
    grip_base.head = prev_tail
    grip_base.tail = prev_tail + Vector((0, 0, 0.4))
    grip_base.parent = bones[-1]
    bones.append(grip_base)
```

```

    left_finger = arm.data.edit_bones.new('Gripper_Left')
    left_finger.head = grip_base.tail
    left_finger.tail = left_finger.head + Vector((-0.4, 0, 0))
    left_finger.parent = grip_base
    bones.append(left_finger)

    right_finger = arm.data.edit_bones.new('Gripper_Right')
    right_finger.head = grip_base.tail
    right_finger.tail = right_finger.head + Vector((0.4, 0, 0))
    right_finger.parent = grip_base
    bones.append(right_finger)

    bpy.ops.object.mode_set(mode='OBJECT')
    return arm


def parent_with_auto_weights(obj, armature):
    bpy.context.view_layer.objects.active = armature
    obj.select_set(True)
    armature.select_set(True)
    bpy.context.view_layer.objects.active = armature
    bpy.ops.object.parent_set(type='ARMATURE_AUTO')
    obj.select_set(False)
    armature.select_set(False)


def animate_arm(armature):
    bpy.context.scene.frame_start = 1
    bpy.context.scene.frame_end = 120

    p_bones = armature.pose.bones


    for i in range(5):
        bone_name = 'Base' if i == 0 else f'Bone.{i}'
        bone = p_bones[bone_name]
        bone.rotation_mode = 'XYZ'

        bone.rotation_euler = Euler((0, 0, 0))
        bone.keyframe_insert(data_path="rotation_euler", frame=1)

        angle = math.radians(20 * (i+1))
        if i % 2 == 0:
            bone.rotation_euler = Euler((angle, 0, 0))
```

```
        else:
            bone.rotation_euler = Euler((0, angle, 0))
        bone.keyframe_insert(data_path="rotation_euler", frame=60)

        bone.rotation_euler = Euler((0, 0, 0))
        bone.keyframe_insert(data_path="rotation_euler", frame=120)


    left_finger = p_bones['Gripper_Left']
    right_finger = p_bones['Gripper_Right']
    left_finger.rotation_mode = 'XYZ'
    right_finger.rotation_mode = 'XYZ'


    left_finger.rotation_euler = Euler((0, 0, math.radians(20)))
    right_finger.rotation_euler = Euler((0, 0, math.radians(-20)))
    left_finger.keyframe_insert(data_path="rotation_euler", frame=1)
    right_finger.keyframe_insert(data_path="rotation_euler", frame=1)


    left_finger.rotation_euler = Euler((0, 0, math.radians(-20)))
    right_finger.rotation_euler = Euler((0, 0, math.radians(20)))
    left_finger.keyframe_insert(data_path="rotation_euler", frame=60)
    right_finger.keyframe_insert(data_path="rotation_euler", frame=60)


    left_finger.rotation_euler = Euler((0, 0, math.radians(20)))
    right_finger.rotation_euler = Euler((0, 0, math.radians(-20)))
    left_finger.keyframe_insert(data_path="rotation_euler", frame=120)
    right_finger.keyframe_insert(data_path="rotation_euler", frame=120)


def main():
    clear_scene()


    dark_gray = create_material("DarkGray", (0.1, 0.1, 0.1, 1))
    orange = create_material("Orange", (1.0, 0.4, 0.0, 1))
    light_gray = create_material("LightGray", (0.5, 0.5, 0.5, 1))


    armature = create_armature()

    bone_names = ['Base'] + [f'Bone.{i}' for i in range(1,5)]
    segment_lengths = [2.0] * 5
```

Text: Internal



```python
    for i, bone_name in enumerate(bone_names):
        seg = create_arm_segment(f'Segment_{i+1}', length=segment_lengths[i])
        seg.location.z = i * 2.0
        seg.data.materials.append(dark_gray)

        joint = create_joint_sphere(f'Joint_{i+1}', radius=0.25)
        joint.location.z = i * 2.0
        joint.data.materials.append(orange)


        parent_with_auto_weights(seg, armature)
        parent_with_auto_weights(joint, armature)


    grip_base_seg = create_arm_segment("Gripper_Base_Segment", length=0.4, radius=0.1)
    grip_base_seg.location.z = 5 * 2.0
    grip_base_seg.data.materials.append(light_gray)
    parent_with_auto_weights(grip_base_seg, armature)


    left_finger = create_gripper_finger("Left_Finger", length=0.8, width=0.15, thickness=0.1)
    right_finger = create_gripper_finger("Right_Finger", length=0.8, width=0.15, thickness=0.1)

    left_finger.location = Vector((-0.3, 0, 5*2.0 + 0.4))
    right_finger.location = Vector((0.3, 0, 5*2.0 + 0.4))

    left_finger.data.materials.append(orange)
    right_finger.data.materials.append(orange)

    parent_with_auto_weights(left_finger, armature)
    parent_with_auto_weights(right_finger, armature)


    animate_arm(armature)

    for area in bpy.context.screen.areas:
        if area.type == 'VIEW_3D':
            for space in area.spaces:
                if space.type == 'VIEW_3D':
                    space.shading.type = 'MATERIAL'
                    break

    print("5 DOF robotic arm with  gripper created and animated.")

main()
```

This Blender Python script creates and animates a 5 DOF robotic arm with a gripper. It starts by clearing the scene, then generates arm segments, joints, and materials. An armature (skeleton) with five linked bones and a two-finger gripper is built. The mesh parts are parented to the bones using auto skinning. The script then animates each joint in sequence and controls the gripper. Finally, it sets the viewport for material preview and confirms successful creation and animation of the robotic arm.

4. Testing

Following implementation, we conducted comprehensive testing to validate both functionality and visual output. We confirmed that the full 5 DOF arm moved sequentially with smooth joint rotation, and the gripper operated in a synchronized manner. Mesh deformation and skinning were evaluated to ensure that each part of the model followed the underlying bone structure without distortion. The animation loop was tested for accuracy and fluidity across 120 frames. All materials were rendered correctly under material preview shading, and the structure maintained proper alignment and scale.

Conclusion

Through this project, we successfully demonstrated the procedural creation of a 3D robotic arm with 5 degrees of freedom, incorporating advanced rigging and animation techniques within Blender. By applying structured development phases including planning, design, implementation, and testing we produced a modular, animated model that reflects real world robotic mechanics.

References

1. Blender Foundation Documentation
   Blender Manual – Rigging & Armatures
   https://docs.blender.org/manual/en/latest/animation/armatures/index.html

2. Blender Python API Reference
   bpy API Documentation
   https://docs.blender.org/api/current/bpy.types.html