**UNIVERSIDAD MAYO REAL Y PONTIFICIA DE SAN FRANCISCO XAVIER DE CHUQUISACA**

**FACULTAD DE CIENCIAS Y TECNOLOGÍA**

CARRERA DE INGENIERÍA

EN DISEÑO Y ANIMACIÓN DIGITAL



Examen FInal

**Generación y Transferencia de Poses Humanas en Blender Mediante Redes Neuronales Preentrenadas**

**Universitario:** Enzo Tancara Maman

**Docente :** Ing. Carlos W. Pacheco Lora

Sucre, 6 de febrero de 2025

**Generación y Transferencia de Poses Humanas en Blender Mediante Redes Neuronales Preentrenadas**

**Objetivo**

El objetivo de este proyecto fue desarrollar un procedimiento multimedial que integre inteligencia artificial con Blender, utilizando redes neuronales preentrenadas para detectar poses humanas desde imágenes 2D y transferir dicha información a un personaje tridimensional en Blender mediante scripting en Python

**Flujo de trabajo del proyecto**

Se utilizó Google Colab para aplicar MediaPipe Pose (una red neuronal preentrenada) sobre imágenes de entrada.

Se extrajeron los puntos clave (landmarks) del cuerpo humano en 2D.

Los puntos fueron exportados en un archivo .json.

En Blender, se construyó un personaje con rig y se importaron los datos del .json.

A través de scripts en Python, se aplicaron las poses detectadas al rig del personaje

**Red neuronal utilizada**

Se empleó **MediaPipe Pose**, un modelo preentrenado desarrollado por Google, basado en redes neuronales convolucionales (CNN). Este modelo permite detectar hasta 33 puntos claves del cuerpo humano, incluyendo caderas, hombros, extremidades y cabeza, tanto en 2D como en 3D.

MediaPipe fue ejecutado desde Colab por ser un entorno optimizado para esa librería y por facilitar la visualización interactiva de los resultados.

**¿Cómo funciona y cómo fue entrenada MediaPipe Pose?**

MediaPipe Pose es un modelo de inteligencia artificial desarrollado por Google para detectar automáticamente poses humanas en imágenes o video. Está basado en una arquitectura de redes neuronales convolucionales (CNNs), específicamente adaptadas para detección de poses, y fue preentrenado con grandes conjuntos de datos de imágenes de personas realizando distintas actividades físicas.

. **Proceso de entrenamiento**

MediaPipe Pose fue entrenado con datasets de visión por computadora que contienen imágenes etiquetadas con las posiciones exactas de las articulaciones del cuerpo humano (por ejemplo: COCO dataset, BlazePose dataset).

La red neuronal aprende a detectar 33 puntos clave del cuerpo: cabeza, cuello, hombros, codos, muñecas, caderas, rodillas, tobillos, entre otros.

Utiliza dos etapas:

Detección del cuerpo completo en la imagen (con una red "detector").

Estimación precisa de los puntos clave usando una red refinadora ("pose tracker").

Repositorio principal: <https://github.com/google-ai-edge/mediapipe>

**Aplicación de la inteligencia artificial (modelo de pose humana)**

mp\_pose = mp.solutions.pose

pose = mp\_pose.Pose(static\_image\_mode=True)

results = pose.process(image\_rgb)

Estas tres líneas son el corazón del modelo de IA:

mp\_pose.Pose(...) carga el modelo MediaPipe Pose en modo estático.

.process(image\_rgb) aplica la red neuronal a la imagen cargada

El resultado contiene las coordenadas 2D normalizadas (entre 0 y 1) de 33 puntos clave del cuerpo humano.

En este paso, la red neuronal ya hizo su trabajo: analizar la imagen y estimar la posición de las articulaciones.

lm = results.pose\_landmarks.landmark Se calculan coordenadas medias (cuello y cadera) y se extraen puntos específicos relevantes para riggin

Aquí se convierten los resultados del modelo en coordenadas comprensibles. Se agrupan ciertos puntos para generar referencias centrales del cuerpo (como el cuello) y se filtran los puntos clave que se usarán para construir el esqueleto (stickman).

|  |
| --- |
| # 📥 Subir imagen  from google.colab import files  uploaded = files.upload()  image\_path = list(uploaded.keys())[0]  # 🔧 Librerías  import cv2  import mediapipe as mp  import json  from PIL import Image  from IPython.display import display  # 🖼️ Cargar imagen  image = cv2.imread(image\_path)  image\_rgb = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)  h, w, \_ = image.shape  # Mostrar imagen original  display(Image.open(image\_path))  # 🕺 Detección de pose  mp\_pose = mp.solutions.pose  pose = mp\_pose.Pose(static\_image\_mode=True)  results = pose.process(image\_rgb)  # 🎨 Visualización y extracción  image\_stick = image\_rgb.copy()  keypoints\_2d = {}  if results.pose\_landmarks:      lm = results.pose\_landmarks.landmark      # Coordenadas hombros y caderas      l\_shoulder, r\_shoulder = lm[11], lm[12]      l\_hip, r\_hip = lm[23], lm[24]      # Punto medio del cuello      neck\_x = (l\_shoulder.x + r\_shoulder.x) / 2      neck\_y = (l\_shoulder.y + r\_shoulder.y) / 2      keypoints\_2d["landmark\_100"] = {"x": neck\_x, "y": neck\_y}      # Punto medio de la cadera      hip\_x = (l\_hip.x + r\_hip.x) / 2      hip\_y = (l\_hip.y + r\_hip.y) / 2      keypoints\_2d["landmark\_101"] = {"x": hip\_x, "y": hip\_y}      # Guardar puntos reales necesarios      indices = [0, 11, 12, 13, 14, 15, 16, 21, 22,                 23, 24, 25, 26, 27, 28, 31, 32]      for idx in indices:          keypoints\_2d[f"landmark\_{idx}"] = {"x": lm[idx].x, "y": lm[idx].y}      # Conexiones stickman      connections = [          (0, 100),         # Cabeza a cuello          (100, 11), (100, 12),  # Cuello a hombros          (100, 101),       # Cuello a cadera central          (101, 23), (101, 24),  # Cadera central a caderas          (11, 13), (13, 15), (15, 21),  # Brazo izq          (12, 14), (14, 16), (16, 22),  # Brazo der          (23, 25), (25, 27), (27, 31),  # Pierna izq          (24, 26), (26, 28), (28, 32)   # Pierna der      ]      # Dibujar líneas      for start, end in connections:          def get\_coords(idx):              if idx == 100:                  return int(neck\_x \* w), int(neck\_y \* h)              elif idx == 101:                  return int(hip\_x \* w), int(hip\_y \* h)              else:                  return int(lm[idx].x \* w), int(lm[idx].y \* h)          p1 = get\_coords(start)          p2 = get\_coords(end)          cv2.line(image\_stick, p1, p2, (0, 255, 0), 3)      # Articulaciones visibles (incluye hombros y caderas)      articulaciones = [11, 12, 13, 14, 15, 16, 21, 22,                        23, 24, 25, 26, 27, 28, 31, 32]      for idx in articulaciones:          cx, cy = int(lm[idx].x \* w), int(lm[idx].y \* h)          cv2.circle(image\_stick, (cx, cy), 5, (255, 0, 0), -1)      # Cuello y cadera central en rojo      cv2.circle(image\_stick, (int(neck\_x \* w), int(neck\_y \* h)), 5, (0, 0, 255), -1)      cv2.circle(image\_stick, (int(hip\_x \* w), int(hip\_y \* h)), 5, (0, 0, 255), -1)  # 🖼️ Mostrar resultado final  display(Image.fromarray(image\_stick))  # 💾 Guardar JSON  with open("pose\_data\_2D.json", "w") as f:      json.dump(keypoints\_2d, f, indent=2)  # 📤 Descargar JSON  files.download("pose\_data\_2D.json") |

Este código demuestra cómo se puede integrar un modelo de IA preentrenado en un flujo de trabajo multimedial. La red neuronal realiza la detección sin necesidad de entrenamiento adicional, y los resultados se procesan para construir una visualización comprensible, así como un archivo estructurado que puede ser consumido por otros entornos como Blender.

**Integración en Blender**

Se utilizó el archivo .json generado en Colab para extraer las coordenadas de las articulaciones clave. Desde Blender se construyó un rig con los mismos nombres de huesos, y se escribieron scripts en Python que leen el archivo, calculan vectores de dirección y aplican la rotación correspondiente a cada hueso del personaje.

Debido a limitaciones técnicas, el modelo IA no pudo ser ejecutado directamente dentro de Blender, por lo que se recurrió a una integración externa desde Colab.

**1 Aplicación en Blender con Python: visualización de puntos clave**

Una vez obtenido el archivo pose\_data\_2D.json generado en Google Colab por la red neuronal preentrenada MediaPipe Pose, el siguiente script fue utilizado en Blender para leer ese archivo y visualizar los puntos clave (landmarks) como objetos Empty.

**Objetivo Este script:**

Abre el archivo .json con las coordenadas estimadas por la IA.

Convierte cada punto en un objeto visual en Blender (tipo Empty en forma de esfera).

Escala las posiciones para que puedan visualizarse correctamente dentro del espacio 3D

Opcionalmente borra cualquier marcador previo creado por ejecuciones anteriores.

MediaPipe entrega coordenadas normalizadas entre 0 y 1, donde (0,0) está en la esquina superior izquierda de la imagen.

Estas coordenadas se adaptan al sistema de Blender, donde se trabaja en una escala 3D centrada.

El eje vertical de la imagen (Y) se convierte al eje Z de Blender, por eso se invierte con (0.5 - y).

|  |
| --- |
| import bpy  import json  # ✅ Ruta a tu archivo JSON  json\_path = "C:/Users/Enzo/Desktop/IA\_Final/2/pose\_data\_2D3.json"  # 🔄 Escala para hacer la pose visible (ajustable)  scale = 10  # 📁 Leer JSON  with open(json\_path) as f:      data = json.load(f)  # 🧹 Eliminar empties anteriores (opcional)  for obj in bpy.data.objects:      if obj.name.startswith("LM\_"):          bpy.data.objects.remove(obj)  # 🧠 Crear un empty para cada punto del JSON  for lm\_name, coords in data.items():      x = (coords["x"] - 0.5) \* scale      z = (0.5 - coords["y"]) \* scale      pos = (x, 0, z)      empty = bpy.data.objects.new(f"LM\_{lm\_name}", None)      empty.location = pos      empty.empty\_display\_size = 0.05      empty.empty\_display\_type = 'SPHERE'      bpy.context.collection.objects.link(empty)  print("✅ Puntos creados") |

**2 Generación del stickman en Blender a partir de los puntos IA**

Este script lee el archivo JSON generado por MediaPipe Pose y reconstruye una figura humana simplificada en 3D, conectando puntos clave mediante líneas (segmentos). Así se genera un “stickman” que sirve como guía visual y base para animación o rigging.

¿Qué hace el script? Define las conexiones entre articulaciones (por ejemplo: hombro → codo → muñeca).

Lee las coordenadas desde pose\_data\_2D.json (previamente generado).

Convierte las coordenadas normalizadas a posiciones 3D escaladas.

Crea líneas 3D (segmentos) usando from\_pydata entre cada par de puntos.

Muestra todo como objeto Wireframe, visible en modo frontal para mayor claridad.

Los puntos centrales extra como cuello (100) y cadera central (101) fueron añadidos manualmente en la detección para permitir una mejor jerarquía anatómica.

Se usa el eje Z como altura, ya que los valores y de imagen se invierten para coincidir con el espacio de Blender.

Los objetos antiguos que empiezan con "Segment\_" se eliminan automáticamente para evitar duplicados.

|  |
| --- |
| import bpy  import json  import mathutils  # ✅ Ruta al archivo JSON (actualizá con tu propia ruta si cambia)  json\_path = "C:/Users/Enzo/Desktop/IA\_Final/2/pose\_data\_2D3.json"  scale = 10  # Escala de visualización  # 🧱 Conexiones del stickman, incluyendo cuello (100) y cadera central (101)  connections = [      (0, 100),         # Cabeza a cuello      (100, 11), (100, 12),  # Cuello a hombros      (100, 101),            # Cuello a cadera central      (101, 23), (101, 24),  # Cadera central a caderas      (11, 13), (13, 15), (15, 21),  # Brazo izquierdo completo      (12, 14), (14, 16), (16, 22),  # Brazo derecho completo      (23, 25), (25, 27), (27, 31),  # Pierna izquierda      (24, 26), (26, 28), (28, 32)   # Pierna derecha  ]  # 📂 Leer archivo JSON  with open(json\_path) as f:      data = json.load(f)  # 📌 Función para obtener posición en 3D  def get\_pos(idx):      key = f"landmark\_{idx}"      if key not in data:          raise ValueError(f"❗ PUNTO FALTANTE: {key}")      x = (data[key]["x"] - 0.5) \* scale      z = (0.5 - data[key]["y"]) \* scale      return mathutils.Vector((x, 0, z))  # 🧹 Limpiar objetos anteriores del tipo "Segment\_"  for obj in bpy.data.objects:      if obj.name.startswith("Segment\_"):          bpy.data.objects.remove(obj, do\_unlink=True)  # 🪢 Crear segmentos como líneas entre puntos  for i, (a, b) in enumerate(connections):      try:          p1 = get\_pos(a)          p2 = get\_pos(b)      except ValueError as e:          print(e)          continue      mesh = bpy.data.meshes.new(f"SegmentMesh\_{i}")      obj = bpy.data.objects.new(f"Segment\_{i}", mesh)      bpy.context.collection.objects.link(obj)      mesh.from\_pydata([p1, p2], [(0, 1)], [])      mesh.update()      obj.show\_in\_front = True      obj.display\_type = 'WIRE'  print("✅ Stickman 3D generado con cuello y cadera incluidos") |

**Creación automática de rig en Blender desde datos IA**

A partir de los keypoints obtenidos con MediaPipe Pose, este script construye un esqueleto completo y articulado en Blender. La estructura resultante es un rig funcional que representa fielmente la postura estimada en 2D, proyectada en un espacio tridimensional.

Funciones del script Lee un archivo .json con las coordenadas generadas por la IA.

Calcula un punto intermedio virtual (landmark\_102) entre el cuello y la cadera para dividir el torso en dos partes.

Define la estructura ósea en términos de nombres, jerarquía y posiciones absolutas.

Crea automáticamente un Armature en modo Edit y posiciona cada hueso según sus puntos de origen y destino en el plano frontal (X-Z).

Asigna relaciones de padre–hijo a cada sección del cuerpo, lo que permite animaciones completas.

|  |
| --- |
| import bpy  import json  import mathutils  # 📂 Ruta al archivo JSON  json\_path = "C:/Users/Enzo/Desktop/IA\_Final/2/pose\_data\_2D3.json"  scale = 10  arm\_name = "StickmanRig"  # 📖 Leer JSON  with open(json\_path) as f:      data = json.load(f)  # 🧠 Calcular landmark virtual 102 (punto medio entre cuello 100 y cadera 101)  def calc\_midpoint(id1, id2, new\_id):      p1 = data[f"landmark\_{id1}"]      p2 = data[f"landmark\_{id2}"]      mx = (p1["x"] + p2["x"]) / 2      my = (p1["y"] + p2["y"]) / 2      data[f"landmark\_{new\_id}"] = {"x": mx, "y": my}  calc\_midpoint(100, 101, 102)  # 🧱 Definición de huesos  bone\_connections = [      ("head", 0, 100),      ("neck", 100, 102),      ("spine\_lower", 102, 101),      ("shoulder\_L", 100, 11), ("upper\_arm\_L", 11, 13), ("forearm\_L", 13, 15), ("hand\_L", 15, 21),      ("shoulder\_R", 100, 12), ("upper\_arm\_R", 12, 14), ("forearm\_R", 14, 16), ("hand\_R", 16, 22),      ("hip\_L", 101, 23), ("thigh\_L", 23, 25), ("shin\_L", 25, 27), ("foot\_L", 27, 31),      ("hip\_R", 101, 24), ("thigh\_R", 24, 26), ("shin\_R", 26, 28), ("foot\_R", 28, 32)  ]  # 🔍 Convertir puntos a coordenadas 3D  def get\_point(idx):      key = f"landmark\_{idx}"      if key not in data:          raise ValueError(f"⚠️ Faltante: {key}")      x = (data[key]["x"] - 0.5) \* scale      z = (0.5 - data[key]["y"]) \* scale      return mathutils.Vector((x, 0, z))  # 🧹 Limpiar armature previa  for obj in bpy.data.objects:      if obj.type == 'ARMATURE' and obj.name == arm\_name:          bpy.data.objects.remove(obj, do\_unlink=True)  # 🦴 Crear nuevo rig  bpy.ops.object.add(type='ARMATURE', enter\_editmode=True)import bpy  import mathutils  # ⚙️ Nombres de los rigs (ajustalos según tu escena)  source\_rig = bpy.data.objects["StickmanRig"]  target\_rig = bpy.data.objects["StickmanRig22"]  # Leer dirección ósea de StickmanRig en Edit Mode (pose en reposo)  bpy.context.view\_layer.objects.active = source\_rig  bpy.ops.object.mode\_set(mode='EDIT')  source\_dirs = {}  for b in source\_rig.data.edit\_bones:      vec = b.tail - b.head      if vec.length > 0:          source\_dirs[b.name] = vec.normalized()  # Leer dirección ósea de StickmanRig22 en Edit Mode (para comparar espacio)  bpy.context.view\_layer.objects.active = target\_rig  bpy.ops.object.mode\_set(mode='EDIT')  target\_dirs = {}  for b in target\_rig.data.edit\_bones:      vec = b.tail - b.head      if vec.length > 0:          target\_dirs[b.name] = vec.normalized()  # Volver a Pose Mode para aplicar rotaciones  bpy.ops.object.mode\_set(mode='POSE')  for bone in target\_rig.pose.bones:      name = bone.name      if name in source\_dirs and name in target\_dirs:          from\_dir = target\_dirs[name]          to\_dir = source\_dirs[name]          if from\_dir.length > 0 and to\_dir.length > 0:              rot = from\_dir.rotation\_difference(to\_dir)              bone.rotation\_mode = 'QUATERNION'              bone.rotation\_quaternion = rot  # Aplicar la pose visualmente  bpy.ops.pose.select\_all(action='SELECT')  bpy.ops.pose.visual\_transform\_apply()  print("✅ La Rest Pose del StickmanRig fue aplicada como pose activa en StickmanRig22.")  arm = bpy.context.object  arm.name = arm\_name  arm.show\_in\_front = True  edit\_bones = arm.data.edit\_bones  created\_bones = {}  # 🔧 Crear huesos  for name, start\_idx, end\_idx in bone\_connections:      bone = edit\_bones.new(name)      bone.head = get\_point(start\_idx)      bone.tail = get\_point(end\_idx)      bone.roll = 0      created\_bones[name] = bone  # 🧷 Definir jerarquía  created\_bones["neck"].parent = created\_bones["head"]  created\_bones["spine\_lower"].parent = created\_bones["neck"]  # Brazo izquierdo  created\_bones["shoulder\_L"].parent = created\_bones["neck"]  created\_bones["upper\_arm\_L"].parent = created\_bones["shoulder\_L"]  created\_bones["forearm\_L"].parent = created\_bones["upper\_arm\_L"]  created\_bones["hand\_L"].parent = created\_bones["forearm\_L"]  # Brazo derecho  created\_bones["shoulder\_R"].parent = created\_bones["neck"]  created\_bones["upper\_arm\_R"].parent = created\_bones["shoulder\_R"]  created\_bones["forearm\_R"].parent = created\_bones["upper\_arm\_R"]  created\_bones["hand\_R"].parent = created\_bones["forearm\_R"]  # Pierna izquierda  created\_bones["hip\_L"].parent = created\_bones["spine\_lower"]  created\_bones["thigh\_L"].parent = created\_bones["hip\_L"]  created\_bones["shin\_L"].parent = created\_bones["thigh\_L"]  created\_bones["foot\_L"].parent = created\_bones["shin\_L"]  # Pierna derecha  created\_bones["hip\_R"].parent = created\_bones["spine\_lower"]  created\_bones["thigh\_R"].parent = created\_bones["hip\_R"]  created\_bones["shin\_R"].parent = created\_bones["thigh\_R"]  created\_bones["foot\_R"].parent = created\_bones["shin\_R"]  bpy.ops.object.mode\_set(mode='OBJECT')  print("✅ Rig creado con torso dividido: cuello → centro → cadera ✅") |

**Explicación del flujo general**

Importación del JSON y cálculo de coordenadas Se cargan los puntos clave detectados por MediaPipe, incluyendo articulaciones reales y puntos virtuales estratégicos (como el centro del torso).

Definición estructural del cuerpo humano Las conexiones óseas se definen en la lista bone\_connections, especificando cómo cada parte se conecta (por ejemplo: "upper\_arm\_L" va de 11 a 13).

Creación y jerarquización del rig En modo EDIT, se crean los huesos con edit\_bones.new() y se vinculan unos a otros para reflejar la anatomía humana. Finalmente, se sale a modo OBJECT para aplicar los cambios.

Aplicación de pose entre rigs (opcional, incluido al final del script) Un segundo bloque de código compara las direcciones de huesos entre el StickmanRig y el rig objetivo (StickmanRig22) y transfiere la orientación de manera automática utilizando quaternions.

**Dificultades encontradas y aprendizajes**

A pesar de lograr construir un esqueleto básico (stickman) con los puntos detectados por la inteligencia artificial y generar un rig dentro de Blender, no se logro que ese esqueleto se conecte correctamente al modelo 3D del personaje.

Durante el proceso se intentaron varias cosas:

Crear un esqueleto automático basado en los datos del archivo .json.

Copiar la postura detectada desde el stickman al rig del personaje.

Ajustar nombres, estructuras y posiciones para que coincidieran.

Sin embargo, la conexión no funcionó como se esperaba. El personaje no adoptó la pose correctamente, o no reaccionaba bien al rig generado.

Las posibles razones que se identificaron fueron:

El esqueleto automático no coincidía del todo con el del personaje.

Blender requiere configuraciones adicionales para que un rig mueva una malla (como pesos de deformación y enlaces internos).

La integración completa puede llevar más tiempo y pruebas de las que tuvimos disponibles.

Aun así, el proyecto permitió comprobar que es posible generar una pose humana detectada por IA, representarla gráficamente en 3D, y acercarse bastante al rigging automático. El resto del trabajo dejó una buena base para seguir desarrollándolo en el futuro.

**Conclusión**

Este proyecto demuestra la posibilidad de utilizar modelos de IA preentrenados para generar estructuras humanas interpretables y adaptarlas a entornos 3D como Blender. Aunque la integración completa dentro de Blender no fue alcanzada, se validó un flujo funcional de ida y vuelta entre plataformas, y se consolidó el uso de Python como puente entre visión por computadora e infografía.

**Recursos**

Google Colab

MediaPipe Pose

Blender 4.x

Python

JSON (exportación de poses)