OpenMocap

Trabajo realizado por: Alfredo Villarroya Roldós y Javier Sánchez Godoy

Índice

Motivación/argumentación del trabajo

Nuestra intención como equipo era probar a hacer cosas con objetivos diferentes a lo que hubiésemos hecho en las prácticas de la asignatura. Como primera idea del proyecto habíamos pensado en controlar un peleador de un juego de lucha en 2D a través de las poses que hiciese el jugador delante de la cámara.

El único problema que presentaba esta idea es que no era completamente original, pues ya se nos habían mostrado ejemplos de proyectos con objetivos similares, de usar la cámara como los controles de un juego, que fueron desarrollados por otros alumnos en años anteriores.

Tras darle muchas vueltas se nos ocurrió unir los conocimientos que habíamos adquirido en la asignatura de Visión por Computador y juntarlo con aspectos de la asignatura de Informática Gráfica.

Con esta nueva idea, nos era posible hacer una síntesis de los conocimientos que habíamos adquirido en ambas asignaturas y aparte nos permitía enfrentarnos a un desafío al tener que juntar diferentes tecnologías que no se han desarrollado para trabajar de forma conjunta. Lo que presenta una forma de trabajar más cercana al desarrollo de proyectos en el mundo real y nos sirve para prepararnos al mundo laboral.

Objetivo de la propuesta

El objetivo principal que se tenía con este proyecto era desarrollar un programa interactivo capaz de animar y controlar un modelo 3D en tiempo real, haciendo uso de una cámara con la que obtener las poses realizadas por el usuario.

Asimismo, el sistema debía permitir que el modelo 3D se desplazase correctamente dentro de un entorno tridimensional, manteniéndose estable, con fluidez en los movimientos y con una representación visual consistente. Este objetivo implicó, no solo la correcta detección de las poses, sino también el procesamiento de esos datos, su transmisión y su aplicación sobre el modelo, introduciendo el menor retardo e inconsistencias posibles en los movimientos de la animación.

De forma complementaria, el proyecto tuvo como objetivo demostrar la viabilidad de integrar técnicas de visión por computador con herramientas de informática gráfica, demostrando su posible aplicación en escenarios interactivos en tiempo real. Sentando las bases para futuras ampliaciones en ámbitos como la simulación, la realidad aumentada o los sistemas de interacción con físicas.

Descripción técnica del trabajo realizado

Para que el proyecto sea funcional se deben cumplir 3 requisitos:

- La detección de la pose del usuario
- El traspaso de esos datos al backend
- El movimiento del modelo 3D

Una vez se cumplan esos requisitos la arquitectura del programa es la siguiente:

Cámara → Python (OpenCV + YOLOv8) → WebSocket → Navegador (Three.js + JavaScript)

A lo largo de esta sección se describen los métodos y tecnologías empleados para cumplir dichos requisitos, así como su integración dentro del sistema completo.

Detección de pose

La detección de la pose constituye la parte principal del sistema y es el contenido del proyecto más directamente relacionado con los contenidos explicados en la asignatura de Visión por Computador. Este módulo se encarga de capturar las imágenes del usuario y extraer, a partir de ellas, la información correspondiente a las articulaciones del cuerpo humano.

Para la obtención de las imágenes se emplea una cámara web, gestionada mediante la biblioteca OpenCV, que permite la captura de vídeo en tiempo real y la configuración de parámetros como la resolución y la tasa de fotogramas. En este proyecto, la cámara se configura para trabajar a una resolución de 640×480 píxeles y una frecuencia aproximada de 30 FPS, lo que representa un compromiso adecuado entre calidad visual y rendimiento computacional.

Para mejorar el tiempo de respuesta establecimos, que la resolución de la imagen ya que no necesitamos tanta resolución para detectar los huesos, hacemos que YOLO procese una imagen más pequeña internamente y solo dejamos una espera mínima para no bloquear al procesador.

Una vez capturado cada frame, se aplica el modelo YOLOv8 Pose, una red neuronal convolucional entrenada para la detección de poses humanas. Este modelo permite identificar un conjunto de puntos clave (keypoints) correspondientes a las principales articulaciones del cuerpo, tales como hombros, codos, muñecas, caderas, rodillas y tobillos. El modelo se ejecuta en tiempo real sobre cada imagen, utilizando un umbral de confianza para descartar detecciones poco fiables y reducir el ruido en los resultados.

El resultado del proceso de detección son un conjunto de coordenadas bidimensionales que representan la posición de cada articulación detectada en la imagen.

Traspaso de datos

Una vez obtenidos y estructurados los datos de la pose del usuario, resulta necesario un mecanismo que permita su transmisión en tiempo real hacia el sistema encargado de la animación del modelo 3D. Para este propósito, se implementó un servidor basado en el protocolo websocket, el cual actúa como intermediario entre el módulo de detección de poses y los clientes encargados de la representación gráfica.

El servidor WebSocket se desarrolló utilizando la biblioteca websockets de Python y se ejecuta de forma asíncrona mediante asyncio, lo que permite gestionar múltiples conexiones simultáneas sin bloquear la ejecución del sistema. Esta arquitectura resulta especialmente adecuada para aplicaciones interactivas en tiempo real, donde es necesario transmitir datos de manera continua con una latencia mínima.

El servidor mantiene un registro de los clientes conectados, almacenándolos en una estructura de tipo conjunto. Cada vez que un cliente se conecta, se añade automáticamente, y cuando se produce una desconexión, el cliente se elimina, garantizando una gestión correcta de los recursos y evitando conexiones inactivas.

El funcionamiento principal del servidor consiste en recibir mensajes provenientes del módulo de detección de poses y reenviarlos al resto de clientes conectados. De este modo, el sistema desacopla la generación de datos de su consumo, permitiendo que distintos módulos (el motor gráfico basado en Three.js) reciban la información de las poses sin necesidad de comunicarse directamente con el módulo de visión por computador.

Para evitar redundancias y bucles de comunicación innecesarios, el servidor implementa un mecanismo de filtrado que impide reenviar los mensajes al cliente que los ha originado.

Además, el sistema incorpora mecanismos básicos de control de errores y manejo de desconexiones, permitiendo que el servidor continúe funcionando correctamente incluso ante cierres inesperados de clientes o interrupciones temporales en la comunicación. Esta característica contribuye a aumentar la robustez y estabilidad del sistema global.

Movimiento modelo 3D

Para empezar, se configura una escena 3D compuesta por los elementos básicos necesarios para una correcta visualización: escena, cámara, iluminación y superficie de referencia. Se emplea una cámara en perspectiva, situada frente al modelo, con parámetros ajustados para ofrecer una visión natural del personaje dentro del entorno.

La iluminación se compone de una luz hemisférica y una luz direccional, lo que permite una correcta percepción del volumen del modelo y mejora la sensación de profundidad. También se incluye un plano horizontal que actúa como suelo, proporcionando una referencia espacial para el movimiento del personaje.

El modelo 3D utilizado corresponde a un personaje previamente riggeado, es decir, que presenta un esqueleto interno que permite la animación de sus distintas articulaciones. Este modelo se carga en formato GLTF y se escala y posiciona adecuadamente dentro de la escena. Durante el proceso de carga, se identifica el esqueleto asociado al modelo, el cual será utilizado después para aplicar las rotaciones derivadas de los datos obtenidas por las poses detectadas.

Los datos de las poses, que han sido transmitidos a través de WebSockets, son ese conjunto de keypoints bidimensionales mencionados anteriormente que representan la posición de las articulaciones del usuario en el plano de la imagen.

Dado que estos keypoints se encuentran expresados en coordenadas de imagen, es necesario realizar una normalización de los datos para adaptarlos al sistema de coordenadas utilizado en Three.js. Para ello, las coordenadas se transforman al rango normalizado $[-1,1]$, teniendo en cuenta la resolución de la cámara.

Este proceso permite establecer una correspondencia entre las posiciones detectadas en el espacio 2D de la imagen y los movimientos aplicados en el entorno tridimensional.

Una vez normalizados los puntos clave, se calcula la orientación de las distintas extremidades del cuerpo mediante relaciones geométricas entre las articulaciones consecutivas. Para ello, se emplean funciones trigonométricas que permiten obtener el ángulo relativo entre segmentos corporales, como hombro–codo, codo–muñeca o cadera–rodilla.

Estos ángulos se aplican directamente sobre las rotaciones de los huesos correspondientes del esqueleto del modelo 3D, causando así una animación que replica la postura del usuario. Para mejorar la estabilidad visual y evitar movimientos bruscos, se utilizan técnicas de interpolación lineal (LERP), que suavizan las transiciones entre estados.

A parte se ha implementado un modo espejo, que permite alternar entre una vista directa o invertida de los movimientos del usuario y el modelo 3D.

Como los keypoints proporcionados por YOLOv8 Pose se encuentran en un espacio bidimensional, no se dispone de información directa sobre su posición en la profundidad. Para resolver esta carencia, se implementó una estimación del movimiento en el eje Z, basada en la distancia aparente entre los hombros del usuario.

De tal forma que un mayor ancho relativo entre los hombros indica que el usuario se encuentra más cerca de la cámara, mientras que un ancho menor sugiere una mayor distancia. A partir de esta medida, se calcula un desplazamiento del modelo completo a lo largo del eje Z, limitado mediante umbrales para evitar comportamientos no deseados. Este enfoque proporciona una sensación básica de profundidad y mejora la inmersión del sistema, aun cuando no se dispone de información tridimensional real.

Pese a que se ha conseguido una base sólida para las rotaciones de las articulaciones superiores, el cálculo de las rotaciones de los huesos que componen las articulaciones inferiores supuso una dificultad técnica relevante dentro del proyecto. Las piernas se mueven menos de forma visible en las imágenes y muchos de sus movimientos dependen de la profundidad, por ejemplo, doblar la rodilla o estirar la pierna, algo que no se puede obtener directamente a partir de una cámara normal y keypoints en 2D.

Para animar las piernas, las rotaciones se calculan usando la posición de la cadera y la rodilla. El problema es que pequeños errores en la detección de estos puntos hacen que los ángulos calculados cambien mucho, lo que puede provocar movimientos poco naturales o inestables en el modelo 3D.

Además, cada modelo 3D tiene su propio esqueleto, por lo que fue necesario ajustar manualmente cómo se aplican las rotaciones a los huesos de las piernas. Para evitar movimientos extraños, se limitaron los ángulos y se suavizaron las transiciones, aunque esto reduce el rango de movimiento que puede imitar el modelo 3D en las piernas.

Ejecución del programa

Además de la implementación de los diferentes módulos funcionales del programa, se desarrolló el método de ejecución del proyecto. Inicialmente, la intención era ejecutar el proyecto de forma monolítica mediante un único archivo de Python principal. Este programa habría sido responsable de:

- Arrancar un servidor WebSocket para permitir la comunicación en tiempo real entre los diferentes módulos del sistema.
- Gestionar la captura de vídeo desde la cámara web mediante OpenCV, aplicando el modelo de detección de poses YOLOv8 Pose sobre las imágenes.
- Levantar, de forma automática, un servidor HTTP que sirviera el cliente en Three.js y abrirlo en una nueva pestaña del navegador.

Aunque esta aproximación parecía razonable, en la práctica se encontraron limitaciones técnicas que impidieron su implementación.

En Python, cuando varias tareas comparten un único hilo principal, puede producirse bloqueo de la ejecución si alguna de ellas realiza operaciones de E/S o procesamiento que no son asíncronas. Por ejemplo, la llamada cap.read() de OpenCV para capturar fotogramas desde la cámara no está diseñada como una operación no bloqueante, y puede detener la ejecución del programa mientras espera a que se lea un nuevo frame.

Del mismo modo, la gestión de un servidor WebSocket mediante asyncio y la biblioteca websockets requiere un event loop cooperativo, donde cada tarea debe ceder explícitamente el control para permitir que otras corutinas se ejecuten. Si una tarea (como la captura de cámara o un procesamiento intensivo de cada frame) no cede el control, el event loop puede quedar bloqueado, provocando congelamientos o respuestas tardías en la comunicación.

A pesar de múltiples intentos con diferentes enfoques concurrentes (como el uso de threads de Python o intentar ejecutar tareas asíncronas mediante asyncio simultáneamente), ambos módulos chocaban continuamente en el hilo principal de forma que se bloqueaban mutuamente. Esto resultaba en:

- Congelaciones temporales cuando el módulo de WebSockets esperaba eventos de red mientras la captura de vídeo ocupaba el hilo.
- Caídas o interrupciones del proceso cuando las tareas de E/S se superponían, debido a que ni threading ni asyncio resolvían completamente el problema de concurrencia sin una arquitectura de procesos más compleja.
- Cierres e imposibilidad de reconexión para los clientes hacia el servidor de WebSockets.

Debido a estas limitaciones para la correcta ejecución del programa actualmente se deben abrir tres terminales diferentes, en cada uno de ellos se ejecutará una parte del programa, uno abrirá el servidor de WebSockets, otro abrirá el http con el archivo javascript y el último es el que encenderá la cámara y enviará los datos de los keypoints a través de WebSockets.

Fuentes y tecnologías utilizadas

Como se ha ido mencionando a lo largo del documento, para poder combinar ambas disciplinas de Visión por Computador e Informática Gráfica han sido necesarias diversas tecnologías que permitieran desarrollar el proyecto.

Algunas de estas tecnologías ya se habían utilizado previamente en otras asignaturas, aunque en este proyecto se han tenido que aprender aspectos más avanzados y específicos de cada una de ellas para adaptarlas al trabajo requerido.

WebSockets

Comenzando con la tecnología nueva, no perteneciente a ninguna de las asignaturas que han inspirado este proyecto, se encuentra la biblioteca websockets de Python.

Gracias a esta herramienta, se ha podido implementar un servidor que establece comunicación entre distintos módulos del proyecto, conectando la captura de poses realizada a través de la cámara con el cliente en Three.js que gestiona el modelo 3D.

El uso de WebSockets permite transmitir los datos de las poses de forma continua y en tiempo real, manteniendo una comunicación bidireccional eficiente entre el backend y el frontend, lo que es esencial para que el modelo 3D responda inmediatamente a los movimientos del usuario.

Three.js

El cliente en el navegador se encarga de mostrar la escena tridimensional y de representar el modelo 3D reaccionando a los movimientos detectados.

Además de los conceptos generales de Three.js ya vistos en la asignatura de Informática Gráfica, durante el desarrollo se profundizó en aspectos más avanzados, como la gestión de esqueletos, la correcta rotación de los huesos y la aplicación de movimientos a las articulaciones de forma suave. Esto permitió que el modelo 3D imitara las poses del usuario de manera más realista y estable.

Opencv

Ha sido la biblioteca encargada de gestionar la captura de vídeo desde la cámara web, así como de realizar las operaciones básicas de preprocesamiento necesarias para que las imágenes pudieran ser analizadas por el modelo de detección de poses.

Entre estas operaciones se incluyen la configuración de la resolución, la tasa de fotogramas, y la adaptación del formato de la imagen para que pudiera ser procesada por YOLOv8 Pose.

YOLOv8 Pose

El modelo de detección de poses YOLOv8 Pose ha sido el componente encargado de obtener los datos de las articulaciones del usuario a partir de las imágenes capturadas por la cámara.

Gracias a este modelo, se pudieron identificar de manera precisa los keypoints de las distintas articulaciones del cuerpo. Además, la utilización de YOLOv8 Pose influyó directamente en la elección del modelo 3D riggeado, ya que se seleccionó un personaje que contara con un esqueleto cuyas articulaciones coincidieran, como mínimo, con los keypoints que el modelo era capaz de detectar. Esto aseguró que la animación del personaje pudiera ser controlada de forma coherente con los datos de la pose.

Conclusiones y propuestas de ampliación

Creemos fervientemente que este proyecto demuestra el gran potencial que tiene la combinación de visión por computador y gráficos 3D en tiempo real para crear sistemas interactivos. El programa desarrollado permite capturar la pose de una persona a partir de una cámara convencional y trasladar dicha información a un modelo 3D animado, logrando una representación inmediata del movimiento. A pesar de los buenos resultados obtenidos, el proyecto presenta margen de mejora en varios aspectos clave, principalmente mejorar el movimiento de las piernas para que se pueda aumentar el rango de movimiento de estas, y mejorar la forma de ejecutar el programa a una versión más cómoda.

Aparte de estos detalles ha pulir, hay gran cantidad de ideas para mejorar este proyecto base en diferentes direcciones:

- Una de las posibles líneas de ampliación del proyecto podría ser implementar visión estereoscópica mediante el uso de dos cámaras. Esto permitiría calcular las posiciones de las articulaciones en tres dimensiones de manera precisa, eliminando la necesidad de heurísticas aproximadas de profundidad, como lo es el cálculo del ancho de hombros para estimar la distancia al modelo. Con esta información, sería posible rotar el modelo 3D y sus articulaciones en los tres ejes (X, Y y Z), logrando animaciones más naturales y coherentes con los movimientos reales del usuario. Esta mejora abriría la puerta a aplicaciones más sofisticadas, como simulaciones en tiempo real, entrenamiento físico o interacciones inmersivas, donde la fidelidad del movimiento es crucial.

- Continuando con las mejoras en las detecciones y el movimiento del modelo, una posible ampliación es la detección y gestión de múltiples modelos 3D en la escena, en función de la cantidad de personas captadas por YOLO. Esto permitiría que el sistema escale automáticamente, añadiendo o eliminando avatares según la presencia de los usuarios, y asignando de forma dinámica los datos de movimiento a cada modelo. La integración de múltiples usuarios abriría un gran abanico de posibilidades para aplicaciones colaborativas, performances artísticas o juegos, aumentando significativamente la versatilidad del proyecto.

- Otra posibilidad es la grabación de los diferentes keypoints durante la ejecución del programa en archivos para poder reutilizar esos movimientos que han sido grabados en el futuro. Esto permitiría reproducir los movimientos capturados previamente sin necesidad de grabar el vídeo y pasarle cada frame de nuevo a

YOLO. Dandole al proyecto una funcionalidad de grabación de captura de movimiento (motion capture) de bajo coste.

- Finalmente sería interesante incorporar otros módulos interactivos adicionales, como físicas avanzadas (ammo.js), interacciones con objetos virtuales, elementos de gamificación o minijuegos. La combinación de modelos 3D animados en tiempo real con componentes de interacción física permitiría experiencias más interesantes, donde los movimientos del usuario tengan consecuencias directas en el entorno virtual. Esta expansión no solo aumentaría el potencial recreativo del sistema, sino que también sentaría las bases para futuros desarrollos en realidad aumentada o realidad virtual, integrando animación de personajes, interacción y simulación física en un solo flujo continuo.

Indicación de herramientas/tecnologías con las que les hubiera gustado contar

Una tecnología que habría resultado útil durante el desarrollo del proyecto es el uso de un motor gráfico con herramientas avanzadas de animación, como Unity o Blender.

Este tipo de motores proporcionan sistemas integrados para el control de esqueletos, tales como Inverse Kinematics (IK), que permiten calcular automáticamente las rotaciones de los huesos a partir de la posición final de las extremidades. En el caso de las piernas, la cinemática inversa hubiese facilitado que el modelo adopte posturas coherentes sin necesidad de calcular manualmente cada rotación de cadera y rodilla.

El uso de IK habría reducido la complejidad técnica asociada al control de las extremidades inferiores, evitando muchos de los ajustes manuales de ángulos, limitaciones y direcciones que fue necesario realizar con Three.js.

En conclusión, disponer de un motor gráfico y aprender a usarlo correctamente habría permitido centrar el esfuerzo en la detección de poses, en lugar de en la gestión manual de las rotaciones del esqueleto, mejorando así la calidad final de la animación.

Diario de reuniones del grupo

A lo largo del desarrollo del proyecto los integrantes del equipo se comunicaron para coordinar el trabajo y evaluar el progreso de cada una de las partes. En una primera fase, el grupo en conjunto se centró en la búsqueda de documentación relacionada con WebSockets y en el estudio de las tecnologías más adecuadas para integrar los requisitos para desarrollar el trabajo.

Durante las primeras semanas, Javier se encargó de preparar y configurar el servidor de WebSockets, mientras que Alfredo trabajó en la captura de imágenes desde la cámara y en la integración inicial del modelo de detección de poses de YOLO. Posteriormente, Javier continuó desarrollando el envío de datos entre los distintos módulos conectándolos al servidor de WebSockets, y Alfredo se centró en la búsqueda de un modelo 3D riggeado que fuese compatible con los keypoints proporcionados por el modelo de pose.

Más adelante, Javier preparó el frontend del proyecto, creando la escena básica en Three.js e incorporando las librerías necesarias para la carga y visualización del modelo 3D. Paralelamente, Alfredo realizó distintas pruebas de ejecución, intentando unificar todo el proyecto en un único archivo principal de Python y analizando las limitaciones de esta aproximación.

Finalmente, en la etapa previa a la entrega, ambos miembros trabajaron conjuntamente en mejorar el movimiento del modelo 3D en tiempo real, ajustando rotaciones, suavizando movimientos y corrigiendo errores detectados durante las pruebas, con el objetivo de lograr un comportamiento lo más estable y realista posible.

Créditos materiales no originales del grupo

Modelo 3D utilizado: https://poly.pizza/m/yiQDOLP4Ry