Mobile Mocap

Indice:

Introdução

Mobile Mocap é um projeto desenvolvido na plataforma Unity Engine para dispositivos Android. Esta aplicação recolhe imagens captadas pela câmera do dispositivo, procura pela presença de um corpo humano na imagem, analisa a posição do mesmo e mostra os resultados, construindo um boneco composto por vários pontos detectados e interconectados, que representa a pose atual do utilizador.

O projeto suporta o uso de um servidor python, disponibilizado à parte que, quando conectado com a aplicação, recolhe e guarda os dados por detrás da análise feita, gravados em ficheiros prontos a usar.

Este projeto foi construído por cima do projeto já existente BlazePoseBarracuda e utiliza o projeto adicional WebSocketSharp nas funcionalidades relacionadas ao servidor. O repositório deste projeto pode ser encontrado neste link.

Landmark Detection

Este projeto tem como base encontrar uma solução para “Motion capture em um dispositivo móvel com o uso de uma câmera”. Após pesquisas, foram encontradas as tecnologias Landmark Detection e Pose Estimation, que podem ser adaptadas para este efeito.

Das duas, Landmark Detection apresentava melhor suporte para o pretendido, não só com outros projetos externos sobre o tema, mas também através da framework pré-existente MediaPipe Pose, criada pela Google.

Um dos projetos externos encontrados e explorados foi o BlazePoseBarracuda, que implementa a framework MediaPipe Pose em Unity Engine. Este projeto também apresenta-se público e com cenas Unity já montadas e prontas a usar, o que facilita a compreensão desta tecnologia.

Como o projeto encontrado disponibiliza o código correspondente ao detector responsável por realizar a Landmark Detection e mostrou bons resultados, foi escolhido para ser trabalhado em cima.

Projeto Unity

Cena Main

A única cena presente no projeto, chamada Main, contém os seguintes objetos mais pertinentes para o funcionamento do programa:

OutputCanvas e UICanvas

Estes objeto incorporam os seguintes objetos de UI:

- WebCam Output é uma simples imagem vazia. A textura desta imagem é mais tarde substituída por uma obtida através da câmera do dispositivo, mostrando ao utilizador o que o programa está a ver.

  Black Screen é uma simples imagem preta que ofusca o ecrã quando nenhuma imagem é mostrada na WebCam Output.

  SettingsMenu: Contém os seguintes objetos de UI:

    CameraButton: Alterna a câmera a ser lida entre as várias disponíveis pelo dispositivo, chamando a função NextWebcamDevice da classe WebCamInput;

    IP Input é um objeto Input Field, que guarda o IP lá escrito, mais tarde lido pelo programa.

    ServerButton: envia o texto escrito em IP Input para o objeto WebsocketClient de forma a este realizar a ligação ao servidor, via o método TryServerConnection.

    Pose Save Button chama o método StartPoseSave da classe UIController, que inicia o processo de guardar a pose atual do utilizador.

  SettingsButton: Liga e desliga o elemento SettingsMenu.

  CountdownTimer: Temporizador geral.

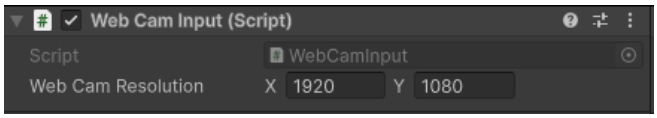
  PoseSimilarity: Mostra a pontuação atual calculada pela classe PoseSimilarityComparer.

  ServerButton: Chama o método StartServerLogCountdown.

WebCamInput

WebCamInput é o objeto responsável pela leitura da câmera. Este objeto têm um único componente do tipo WebCamInput. Esta classe lê a câmera e, tanto escreve a imagem numa textura, substitui a presente no objeto WebCam Output, como envia-a para o objeto Visualizer.

Este componente apresenta os seguintes parâmetros públicos:



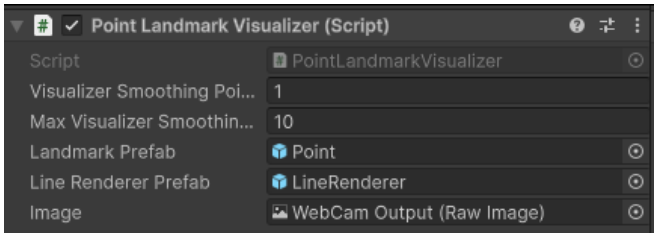
- WebCamResolution define a resolução a usar na textura criada e enviada;

Visualizer

Visualizer é o objeto responsável por enviar a textura obtida pelo objeto WebCamInput a um objeto gerado internamente do tipo BlazePoseDetecter, como por analisar os dados obtidos por esse objeto e construir o esqueleto de pontos e ligações obtidos.

Tem dois componentes, o primeiro um componente do tipo PointLandmarkVisualizer , uma versão modificada da original presente no projeto BlazePoseBarracuda.

Este componente apresenta os seguintes parâmetros públicos:

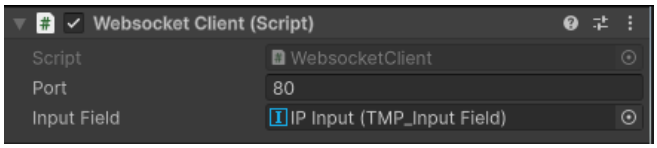


- LandmarkPrefab define que *prefab* usar na criação dos pontos do esqueleto;
- LineRendererPrefab define que *prefab* usar na criação das ligações entre pontos do esqueleto;
- Image define qual a imagem de UI usada para mostrar ao utilizador a imagem captada pela câmera e enviada ao programa.
- Visualizer Smoothing Points define quantos pontos a usar no efeito de smoothing. Este valor pode ser modificado pelo utilizador através do Smoothing Controller.
- Max Visualizer Smoothing Points indica o máximo de pontos permitidos para este efeito.

O segundo componente é do tipo PoseSimilarityComparer. Não apresenta nenhum parâmetro público, mas é responsável por avaliar o quão similar a pose atual e a pose guardada, esta segunda internamente obtida pelo PoseLandmarkVisualizer.

WebsocketClient

WebsocketClient é o objeto responsável por estabelecer a comunicação com o servidor remoto, recolher e comprimir a informação a enviar e fazer esse envio. Tem um único componente do tipo WebSocketClient.



Este componente apresenta os seguintes parâmetros públicos:

- Port define qual porta usar na conexão com o servidor;
- InputField define qual objeto deste tipo recolher o IP necessário na conexão com o servidor

Prefabs

Este projeto utiliza os seguintes objetos pré-definidos para um correto funcionamento:

Point

Este objeto representa um ponto do esqueleto usado para a representação visual da pose do utilizador. É instanciado um Point por ponto detectado pelo programa.

LineRenderer

Este objeto representa uma ligação entre pontos do esqueleto usado para a representação visual da pose do utilizador. É instanciado um LineRenderer para cada ligação entre pontos, definida pelo programa para a construção visual da pose.

Classes C# internas

Nota: O código do projeto é disponibilizado pelo repositório do projeto: Repositório.

BodyLandmarks

BodyLandmarks é uma classe *static* com duas listas públicas chamadas PoseLandmarks e PoseLandmarkPairs.

A primeira enumera cada ponto detectado pela aplicação e a segunda guarda cada ligação entre pontos, necessárias para a criação do esqueleto simulado. Esta classe dá acesso destas informações a outras classes e permite fazer modificações aos pontos a criar e ligações a realizar de forma fácil.

Adicionalmente, disponibiliza o metodo SortPoseLandmarks, que ordena um conjunto de PoseLandmarks, de forma a estarem na ordem correta.

Landmark

Landmark é a classe que representa e atualiza os pontos do esqueleto e as suas ligações. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

UpdateValues: Usa um Vector4 que recebe para atualizar este ponto. Chama os metodos AddSmoothingPoint e UpdatePointSmooth ou UpdatePoint consoante se for para usar o efeito de Smoothing. Por final atualiza o material do ponto, representando o quão visível este está para o sistema.

UpdateLineRenderers: Atualiza cada linerenderer com os dados mais recentemente recebidos.

UpdateSingleLineRenderer: Permite atualizar manualmente um linerenderer em especifico, especificando o seu index e o seu visibility score.

AddSmoothingPoint: Adiciona o ponto recebido à lista de pontos usados para o efeito de Smoothing, removendo o mais antigo se não houver espaço.

UpdatePointSmooth: calcula a média das posições e visibilidades dos pontos da lista para este efeito e atualiza os valores a usar pelo método UpdatePositionVectorNanCheck;

UpdatePoint: atualiza a visibilidade do ponto diretamente, e a posição através do método UpdatePositionVectorNanCheck;

UpdatePositionVectorNanCheck: Faz um pequeno check de erro antes de atualizar diretamente a posição do ponto.

InitializeSmoothing: Cria uma lista de pontos a usar para o efeito de smoothing.

SetNext: adiciona um ponto à lista dos pontos ligados e instancia um objeto LineRenderer para fazer essa representação.

PointLandmarkVisualizer

PointLandmarkVisualizer é a classe responsável por criar um esqueleto que simula a pose do utilizador através dos dados obtidos pela classe BlazePoseDetecter, de guardar a pose num instante e de controlar o efeito de Smoothing. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

Start: Prepara o funcionamento desta classe, cria um novo BlazePoseDetecter, a classe onde a detecção da pose do utilizador é feita, e chama o método InitializePose;

LateUpdate: envia as informações de câmera recebidas por WebCamInput ao BlazePoseDetecter, através do método ProcessImage, e de chamar o método UpdatePoints da classe Pose.

InitializePose: inicia a criação de uma nova pose. Primeiro cria um novo GameObject, dá-lhe um novo componente do tipo Pose e chama o método Init da classe Pose de forma a preparar a criação de uma nova pose, com as informações que lhe forem dadas.

GetLandmarkPointData: prepara e envia as informações de cada ponto de uma pose, chamada por WebsocketClient, objeto responsável pela conexão entre a aplicação e o servidor remoto.

SaveCurrentPose: Cria uma cópia exata do objeto que guarda e que representa a pose detectada, efetivamente gravando a pose nesse instante. De seguida chama o metodo StartComparer, iniciando o comparador de poses e UpdateSavedPointsAlpha.

WebCamInput

WebCamInput é a classe responsável por inicializar e atualizar a câmera ativa, e de enviar o que é lido pela câmera ao resto do programa. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

Start: Prepara o funcionamento desta classe e inicializa a câmera, usando o método InitializeWebcam.

InitializeWebcam: inicia o dispositivo e a textura a usar e força a primeira atualização da câmera pelo método UpdateWebcam.

UpdateWebcam: Começa a leitura da câmera e roda o PointLandmarkVisualizer, alinhando o esqueleto representado à rotação da câmera.

Update: atualiza a imagem da aplicação com a textura lida pela câmera.

Pose

Pose é a classe base de cada pose, contendo os seguintes métodos necessários para a criação e atualização dos pontos desta mesma:

Init: Segundo as informações recebidas, instancia um novo objeto para cada ponto a criar, envia-lhes os seus pontos par e cria ligações entre eles.

UpdatePoints: este método trata de atualizar os pontos e os linerenderers, criados em Init, usando o método UpdateValues e UpdateLineRenderers da classe Landmark, com as informações que o BlazePoseDetecter disponibiliza, através do método GetWorldPoseLandmark.

WebsocketClient

WebsocketClient é a classe responsável por estabelecer a conexão cliente-servidor, tratar da informação e fazer o envio dessa mesma. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

Update: Chama o método ConvertAndSendMessage. Só o consegue fazer depois do método StartServerLog ser chamado.

ConvertAndSendMessage: Chamado pelo Update, quando conectado ao servidor, envia primeiramente a informação do frame e tempo atuais desde o início da conexão e depois a informação de cada ponto, através do método SendMessage.

SendMessage: Converte a mensagem a enviar em Bytes e envia ao servidor via o método Send da classe Websocket.

InitWebsocketClient: Usa o IP recebido e o port definido para estabelecer a conexão e adiciona o método OnServerOpen como observador do evento OnOpen da classe Websocket.

OnServerOpen: Guarda o tempo de conexão ao servidor e começa o envio do log inicial, pelo método InitialLog.

InitialLog: Envia as informações da quantidade de pontos e das conexões detectadas pelo programa ao servidor, pelo método SendMessage.

StartServerLog: Permite o método Update de chamar o método ConvertAndSendMessage.

PoseSaver

Responsável por guardar a pose atual. A funcionalidade planeada de guardar a pose num ficheiro nunca foi finalizada. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

Start: Inicializa esta classe e quaisquer referências necessárias ao seu funcionamento.

SaveCurrentPose: inicializa uma pose nova, chamando o método InitializePose da classe PontLandmarkVisualizer, usando os dados da pose atual, efetivamente criando uma cópia, e de seguida inicia o PoseSimilarityComparer, chamando o método StartComparer com a nova pose criada.

PoseSimilarityComparer

Responsável por avaliar o quão similar é a pose atual e uma pose guardada. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

Update: Se houver uma pose guardada, calcula as direções das conexões da pose atual, pelo método GetPoseLineDirections, calcula a similaridade entre a pose guardada e a pose atual, pelo método CalculatePoseSimilarity, e finalmente atualiza o UIController com o valor dessa similaridade, via o método UpdatePoseSimilarityScore.

StartComparer: Recebe e guarda a pose a guardar, e calcula as direções das suas ligações, usando o método GetPoseLineDirections.

GetPoseLineDirections: Para cada ponto da pose guardada, encontra o vetor direção entre o mesmo e os pontos seguintes que formam as ligações.

DisplayIndividualSimilarityScores: Atualiza cada segmento da pose guardada pela classe PoseSaver com as suas pontuações obtidas pelo método CalculatePoseSimilarity.

CalculatePoseSimilarity: calcula e guarda a pontuação média da similaridade entre todas as ligações correspondentes das duas poses, cada uma obtida através do método CosineSimilarity. Guarda também uma lista com cada pontuação individual.

CosineSimilarity: aplica o método matemático com o mesmo nome para comparar a direção entre dois vetores. 1 representa a mesma direção, -1 representa direções opostas e 0 representa direções oblíquas.

UIController

Responsável por dar funcionalidade e controlar diversos elementos de UI. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

Update: Se a o temporizador para guardar a pose atual ou de ligação ao servidor estiver ativo, atualiza estes mesmos pelos métodos UpdatePoseCountdown e UpdateServerCountDown, respectivamente.

StartPoseSave: inicia o temporizador para guardar a pose atual.

UpdatePoseCountdown: atualiza o temporizador decrescentemente. Quando este chega a 0, desliga-o, e chama o método SaveCurrentPose da classe PoseSaver.

UpdatePoseSimilarityScore: atualiza o elemento de UI que representa esta pontuação.

StartServerLogCountDown: inicia o temporizador para enviar dados ao servidor.

UpdateServerCountdown: atualiza o temporizador decrescentemente. Quando este chega a 0, desliga-o, e chama o método StartServerLog da classe WebsocketClient.

TryServerConnection: Chama o método InitWebsocketClient. Se este devolver True, fecha o UI das opções e liga o UI da ligação ao servidor

Classes C# externas

Nota: Apenas as secções diretamente relacionadas com este projeto estão incluídas. Os repositórios dos dois projetos estão disponíveis aqui para uma consulta mais aprofundada.

BlazePoseDetecter

Responsável por controlar o modelo ML, de enviar a imagem para análise e de disponibilizar os dados obtidos ao resto do programa. Adicionalmente também realiza a rotação da imagem obtida pela câmera, caso esta esteja rotacionada. Esta classe define os seguintes métodos essenciais para o seu funcionamento:

ProcessImage: Recebe a imagem obtida pela câmera, a textura onde mostrar a imagem e o ângulo de rotação da imagem recebida. Contém um conjunto de instruções para o GPU, incluindo:

- Rotação da imagem obtida, para corrigir quaisquer rotações que o dispositivo possa ter pré definido na câmera;
- Previsão da pose segundo a imagem enviada.

Esta classe também disponibiliza dois métodos, GetPoseLandmark e GetPoseWorldLandmark, onde são disponibilizados os pontos obtidos. O primeiro conjunto guarda as posições relativamente à imagem, não disponibilizando a terceira coordenada de profundidade, e o segundo relativamente ao espaço 3D.

Websocket

Responsável por controlar a ligação e comunicação com o servidor.

O método Connect tenta realizar a conexão entre cliente e servidor, usando o URL previamente enviado via contructor.

Servidor Python

O servidor, é composto por um único ficheiro server.py para o seu funcionamento. É responsável por criar um servidor, receber mensagens enviadas e escrever essas mesmas num ficheiro devidamente identificado. Este servidor permite guardar as informações captadas pelo programa num único ficheiro, permitindo fazer uma análise externa dos dados obtidos.

Server.py

Este ficheiro é responsável pelo funcionamento completo do servidor e define as seguintes funções:

main: inicializa o servidor e dá ao mesmo o método handle_websocket para este chamar assim que e enquanto a conexão se realiza e mantém.

Handle_message: primeiro transforma o bytes recebidos pela mensagem de volta em floats e de seguida abre e escreve no ficheiro log o recebido.

Handle_websocket: para cada mensagem recebida, chama o método handle_message.

Além destas funções, o ficheiro define as seguintes instruções a realizar assim que é corrido:

- Define o local e nome do ficheiro onde gravar as mensagens recebidas;
- Encontra e escreve na consola o IP do wifi conectado;
- Corre a função main.

Validação com VICON

Durante o desenvolvimento deste projeto foi feita uma sessão de colheita de dados feita de forma sincronizada com o sistema VICON. Esta sessão foi realizada para comparar os dados obtidos das duas plataformas, analisar os resultados obtidos e determinar se este projeto tem a precisão necessária para algumas utilidades e funções onde o sistema VICON é utilizado.

Durante a sessão, foram gravados os dados de quatro movimentos com a aplicação e com VICON, tentando começar e parar a gravação de forma mais sincronizada possível. Tendo em conta que os únicos membros analisados foram do lado direito do corpo, os movimentos gravados foram os seguintes:

Membros superiores frontal (MS Frontal):

- Modelo em pé de frente para a câmera, braço e antebraço estendidos junto ao corpo;
- Flexão contínua do antebraço até ao máximo, mantendo a flexão do braço no mínimo;
- Extensão do membro de volta à pose inicial;
- Flexão do braço até formar um ângulo de 90º, mantendo o antebraço estendido.
- Extensão do membro de volta à pose inicial.

Membros superiores sagittal (MS Sagital):

- O movimento foi o mesmo do descrito acima, mas realizado com um ângulo de 90º para a câmera, mantendo o membro a analisar visível.

Membros inferiores frontal (MI Frontal):

- Modelo em pé de frente para a câmera, coxa e perna estendidas.
- Flexão contínua da perna até ao máximo, mantendo a flexão da coxa no mínimo;
- Extensão do membro de volta à pose inicial;
- Flexão da coxa até formar um ângulo de 90º, flexão da perna até formar um ângulo de 90º;
- Extensão do membro de volta à pose inicial.

Membros superiores sagittal (MI Sagital):

- O movimento foi o mesmo do descrito acima, mas realizado com um ângulo de 90º para a câmera, mantendo o membro a analisar visível.

Foram gravados estes quatro movimentos para analisar a precisão da aplicação tanto com membros superiores e inferiores, como com diferentes ângulos relativamente é câmera. Um documento foi criado de forma a incluir os dados obtidos para poder criar gráficos com os mesmos, para uma melhor análise e comparação de resultados. Este documento pode ser encontrado no seguinte link: Excel VICON.

Links

Repositório Github: https://github.com/ArKynn/Mobile-MoCap
BlazePoseBarracuda: https://github.com/creativeIKEP/BlazePoseBarracuda
MediaPipe Pose: https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker
WebSocketSharp: https://github.com/sta/websocket-sharp
Excel VICON: Vicon - APK data comparison