```
"""
================================================================================
=====
                        NOME DO FICHEIRO:
VisionDebugger_PC_PT.py
================================================================================
=====

DESCRIÇÃO:
Este programa é uma versão para executar no Computador (PC).
Serve para testar a "visão" (inteligência artificial) sem precisar do braço
ligado.
Usa a webcam do computador para ver a mão e o braço e mostra no ecrã o que está a
perceber.

O que este programa faz:
1. Liga a webcam do computador.
2. Usa o MediaPipe (Nova API 'Tasks') para detetar o esqueleto do braço e da mão.
3. Calcula se o braço está fletido, para onde aponta, e se a mão está rodada.
4. Mostra tudo num vídeo no ecrã com desenhos coloridos.
5. Escreve os dados na janela preta (consola) para análise técnica.

Versão FINAL OTIMIZADA.
- Usa "Thread" de vídeo para eliminar atrasos (Lag/Delay).
- Resolução ajustada para 640x480 (Padrão para alta velocidade).
- GPU ativada (Usa a placa gráfica para ser mais rápido).
- Modelo Lite (Versão leve da inteligência artificial).

LINGUAGEM: Python
================================================================================
=====
"""

# --- BIBLIOTECAS (Ferramentas que o programa usa) ---
import cv2                  # "OpenCV": Controla a câmara e janelas de vídeo
import mediapipe as mp      # "MediaPipe": A Inteligência Artificial que deteta
pessoas
import math                 # Matemática (cálculo de ângulos)
import statistics           # Estatística (médias)
import time                 # Relógio (para medir o tempo e calcular FPS)
import os                   # Sistema Operativo (para encontrar ficheiros no PC)
import threading            # "Multitarefa": Permite ler a câmara e processar ao
mesmo tempo

# --- NOVA API MEDIAPIPE TASKS ---
```

```python
from mediapipe.tasks import python
from mediapipe.tasks.python import vision

#
================================================================================
=====
#                       >>> ÁREA DE CONFIGURAÇÃO (MEXER AQUI) <<<
#
================================================================================
=====

# 1. CÂMARA (Webcam)
INDICE_CAMERA = 1        # Qual câmara usar? (0 = Principal, 1 =
Secundária/Externa)
LARGURA_CAMERA = 1280    # Largura da imagem (Recomendado: 640 para rapidez, 1280
para qualidade)
ALTURA_CAMERA = 720      # Altura da imagem  (Recomendado: 480 para rapidez, 720
para qualidade)

# 2. LIMIARES DOS DEDOS (Ângulo mínimo para considerar "Aberto")
# Se o ângulo do dedo for maior que isto, o programa diz "ABERTO"
TH_POLEGAR = 150.0
TH_INDICADOR = 160.0
TH_MEDIO = 150.0
TH_ANELAR = 150.0
TH_MINDINHO = 140.0

# 3. LIMIARES DO BRAÇO
TH_BRACO_ESQUERDA = 70    # Graus para considerar virado à esquerda
TH_BRACO_CENTRO_MAX = 130 # Até quantos graus é centro?
TH_FLEXAO_Y = 0.1         # Quão levantado o braço precisa estar para ser
"fletido"

# 4. MAPA DOS DEDOS (Pontos do MediaPipe)
# Diz ao programa quais os "pontos" do esqueleto que formam cada dedo
DEDOS = {
    "polegar": [1, 2, 4],
    "indicador": [5, 6, 8],
    "medio": [9, 10, 12],
    "anelar": [13, 14, 16],
    "mindinho": [17, 18, 20]
}
```

```
#
================================================================================
=====
#                           >>> FIM DA CONFIGURAÇÃO <<<
#
================================================================================
=====

# --- CONSTANTES INTERNAS (Valores fixos que o programa usa) ---
LIMIARES_DEDOS = {
    "polegar": TH_POLEGAR,
    "indicador": TH_INDICADOR,
    "medio": TH_MEDIO,
    "anelar": TH_ANELAR,
    "mindinho": TH_MINDINHO
}

# Cores para os desenhos (Formato BGR: Azul, Verde, Vermelho)
COR_OSSO_POSE = (0, 255, 0)         # Verde
COR_ARTICULACAO_POSE = (0, 0, 255)  # Vermelho
COR_OSSO_MAO = (255, 255, 0)        # Ciano (Amarelo+Verde ?)
COR_ARTICULACAO_MAO = (255, 0, 0)   # Azul
ESPESSURA_LINHAS = 2                # Espessura da linha para os desenhos
ESPESSURA_PONTOS = 6                # Espessura dos pontos

# --- CONFIGURAÇÃO DOS MODELOS (Onde estão os ficheiros da IA) ---
# O programa procura os ficheiros ".task" na mesma pasta onde este ficheiro .py
está
diretorio_script = os.path.dirname(os.path.abspath(__file__))
CAMINHO_MODELO_POSE = os.path.join(diretorio_script, 'pose_landmarker_lite.task')
# 'pose_landmarker_heavy.task' para melhor performance
CAMINHO_MODELO_MAO = os.path.join(diretorio_script, 'hand_landmarker.task')

# --- DEFINIÇÃO DE CLASSES (Ferramentas personalizadas) ---

class FluxoCamera:
    """
    CLASSE ESPECIAL: LEITOR DE CÂMARA RÁPIDO
    Esta ferramenta lê a câmara numa 'linha separada' (Thread).
    Isto serve para que o programa nunca fique à espera da câmara.
    Elimina o 'Lag' ou 'Delay' da imagem.
    """
    def __init__(self, src=0, width=640, height=480):
        # Inicia a ligação à câmara
        self.stream = cv2.VideoCapture(src)
```

```
        self.stream.set(cv2.CAP_PROP_FRAME_WIDTH, width)
        self.stream.set(cv2.CAP_PROP_FRAME_HEIGHT, height)
        # Tenta dizer ao Windows para não guardar imagens velhas em memória
        self.stream.set(cv2.CAP_PROP_BUFFERSIZE, 1)

        # Lê a primeira imagem
        (self.capturado, self.quadro) = self.stream.read()
        self.parado = False

    def iniciar(self):
        # Inicia a tarefa em segundo plano (background)
        threading.Thread(target=self.atualizar, args=()).start()
        return self

    def atualizar(self):
        # Loop infinito que só faz uma coisa: Ler a imagem mais recente
        while True:
            if self.parado:
                return
            (self.capturado, self.quadro) = self.stream.read()

    def ler(self):
        # Devolve a última imagem capturada
        return self.quadro

    def parar(self):
        # Pára a câmara
        self.parado = True
        self.stream.release()

# --- FUNÇÕES AUXILIARES (Pequenos comandos úteis) ---

def desenhar_marcos_personalizado(imagem, marcos, conexoes, cor_articulacao,
cor_osso):
    """
    Função para desenhar o esqueleto (bolinhas e riscos) na imagem.
    """
    h, w, _ = imagem.shape # Altura e Largura da imagem
    # Desenhar Pontos (Bolinhas)
    for mc in marcos:
        cx, cy = int(mc.x * w), int(mc.y * h)
        cv2.circle(imagem, (cx, cy), ESPESSURA_PONTOS, cor_articulacao, -1)
    # Desenhar Conexões (Riscos)
    if conexoes:
        for s, e in conexoes:
```

```
            if s < len(marcos) and e < len(marcos):
                cv2.line(imagem, (int(marcos[s].x*w), int(marcos[s].y*h)),
(int(marcos[e].x*w), int(marcos[e].y*h)), cor_osso, ESPESSURA_LINHAS)

def calcular_angulo_3pts(a, b, c):
    """
    Matemática: Calcula o ângulo entre 3 pontos.
    (Exemplo: Ombro -> Cotovelo -> Pulso para saber se o braço está esticado)
    """
    ax, ay = a[0]-b[0], a[1]-b[1]
    cx, cy = c[0]-b[0], c[1]-b[1]
    num = ax*cx + ay*cy
    den = math.hypot(ax, ay) * math.hypot(cx, cy) + 1e-8
    return math.degrees(math.acos(max(-1.0, min(1.0, num/den))))

def calcular_angulo_dedo(lista_marcos_mao, ids):
    """Verifica quanto um dedo está dobrado."""
    a = lista_marcos_mao[ids[0]]
    b = lista_marcos_mao[ids[1]]
    c = lista_marcos_mao[ids[2]]
    return calcular_angulo_3pts((a.x,a.y),(b.x,b.y),(c.x,c.y))

def obter_centro_mao(lista_mc):
    """Encontra o centro da mão (média de todos os pontos)."""
    return (statistics.mean([l.x for l in lista_mc]), statistics.mean([l.y for l
in lista_mc]))

def determinar_lado_mao(cm_x, cm_y, marcos_pose):
    """
    Tenta adivinhar se a mão é Esquerda ou Direita.
    Compara a distância da mão aos pulsos do corpo.
    """
    return "Esquerda" if math.hypot(cm_x - marcos_pose[15].x, cm_y -
marcos_pose[15].y) < math.hypot(cm_x - marcos_pose[16].x, cm_y -
marcos_pose[16].y) else "Direita"

def determinar_orientacao_braco(angulo):
    """Diz para onde o braço aponta baseado no ângulo do ombro."""
    if angulo <= TH_BRACO_ESQUERDA:
        return "Esquerda (Corpo)"
    elif angulo <= TH_BRACO_CENTRO_MAX:
        return "Centro (Frente)"
    else:
        return "Direita (Fora)"
```

```python
def determinar_flexao_braco(ombro_y, pulso_y):
    """
    Diz se o braço está levantado (fletido).
    Compara a altura (Y) do ombro e do pulso.
    """
    diferenca_y = abs(ombro_y - pulso_y)
    if diferenca_y > TH_FLEXAO_Y:
        return "Fletido", diferenca_y
    else:
        return "Estendido", diferenca_y

def calcular_rotacao_mao(marcos_mao, status_flexao, orientacao_braco,
lado="Direita"):
    """
    Calcula se a mão está de Palma ou de Costas (Dorso).
    Mede a distância entre o Polegar e o Mindinho.
    Se a distância for positiva é palma, negativa é dorso (ou vice-versa).
    """
    polegar = marcos_mao[4]  # Ponto na ponta do polegar
    mindinho = marcos_mao[20] # Ponto na ponta do mindinho

    diferenca = 0

    # A lógica muda dependendo se o braço está dobrado ou esticado.
    esta_fletido = (status_flexao == "Fletido")

    if esta_fletido:
        # CASO 1: Braço Fletido (Dobrado)
        # Usamos diferença no eixo X (Horizontal)
        val_p = polegar.x
        val_m = mindinho.x

        # LÓGICA CORRETA PARA IMAGEM ESPELHADA (Modo Selfie):
        if lado == "Direita":
            # Mão Direita (no ecrã Direita): Palma significa Polegar(Esq) <
Mindinho(Dir) -> Diferença é Negativa
            diferenca = -(val_p - val_m)
        else:
            # Mão Esquerda (no ecrã Esquerda): Palma significa Polegar(Dir) >
Mindinho(Esq) -> Diferença é Positiva -> Inverter
            diferenca = (val_p - val_m)

        eixo_usado = "X (Fletido)"

    else:
```

```
        # Braço Estendido
        if "Direita" in orientacao_braco:
            # CASO 3: Estendido para Direita -> Usar eixo Y (Vertical)
            val_p = polegar.y
            val_m = mindinho.y
            diferenca = (val_p - val_m)
            eixo_usado = "Y (Est-Dir)"

        else:
            # CASO 2: Estendido para Centro/Corpo -> Usar eixo Y Invertido
            val_p = polegar.y
            val_m = mindinho.y
            diferenca = -(val_p - val_m)
            eixo_usado = "Y-Inv (Est-Corpo)"

    # Normalização: Converte a diferença bruta de pixels num valor padrão entre -
1 e 1
    DIF_MAXIMA = 0.22
    valor_rotacao = max(-1.0, min(1.0, diferenca / DIF_MAXIMA))

    # Classificar por texto para Depuração (Debugging)
    orientacao = "Indefinido"
    if valor_rotacao < -0.3:
        orientacao = "Palma"
    elif valor_rotacao > 0.3:
        orientacao = "Dorso"
    else:
        orientacao = "A Rodar / Canto"

    return diferenca, valor_rotacao, orientacao, eixo_usado


# --- INICIALIZAÇÃO DE MODELOS E CÂMARA (Preparar tudo antes de começar) ---

# 1. Carregar a Inteligência Artificial (MediaPipe)
BaseOptions = mp.tasks.BaseOptions
PoseLandmarker, PoseLandmarkerOptions = mp.tasks.vision.PoseLandmarker,
mp.tasks.vision.PoseLandmarkerOptions
HandLandmarker, HandLandmarkerOptions = mp.tasks.vision.HandLandmarker,
mp.tasks.vision.HandLandmarkerOptions
VisionRunningMode = mp.tasks.vision.RunningMode

try:
    print("A Tentar ativar a Placa Gráfica (GPU) para ser mais rápido...")
    # Tenta configurar para usar a GPU
```

```
    base_p, base_h = BaseOptions(model_asset_path=CAMINHO_MODELO_POSE,
delegate=BaseOptions.Delegate.GPU),
BaseOptions(model_asset_path=CAMINHO_MODELO_MAO,
delegate=BaseOptions.Delegate.GPU)
    detetor_pose =
PoseLandmarker.create_from_options(PoseLandmarkerOptions(base_options=base_p,
running_mode=VisionRunningMode.VIDEO, num_poses=1))
    detetor_mao =
HandLandmarker.create_from_options(HandLandmarkerOptions(base_options=base_h,
running_mode=VisionRunningMode.VIDEO, num_hands=2))
    print(">>> SUCESSO! GPU ATIVADA! <<<")
except Exception as e:
    print(f"AVISO: GPU falhou ({e}). O Computador vai usar o Processador
(CPU)...")
    base_p, base_h = BaseOptions(model_asset_path=CAMINHO_MODELO_POSE,
delegate=BaseOptions.Delegate.CPU),
BaseOptions(model_asset_path=CAMINHO_MODELO_MAO,
delegate=BaseOptions.Delegate.CPU)
    detetor_pose =
PoseLandmarker.create_from_options(PoseLandmarkerOptions(base_options=base_p,
running_mode=VisionRunningMode.VIDEO, num_poses=1))
    detetor_mao =
HandLandmarker.create_from_options(HandLandmarkerOptions(base_options=base_h,
running_mode=VisionRunningMode.VIDEO, num_hands=2))

# 2. Iniciar Câmara (Usa o nosso 'Leitor Rápido')
fluxo = FluxoCamera(src=INDICE_CAMERA, width=LARGURA_CAMERA,
height=ALTURA_CAMERA).iniciar()
time.sleep(2.0) # Espera 2 segundos para a câmara "aquecer" (ajustar luz)

# Criar a janela onde vamos ver o vídeo
cv2.namedWindow("Detecao OTIMIZADA", cv2.WINDOW_NORMAL)
cv2.resizeWindow("Detecao OTIMIZADA", 1280, 720)

print("\n" + "="*60)
print("A INICIAR - MODO SEM DELAY")
print("="*60 + "\n")

# --- MAPA DE CONEXÕES (Quais pontos ligam a quais) ---
CONEXOES_POSE_BRACOS = [
    (11, 13), (13, 15), (12, 14), (14, 16), # Braços
    (11, 12), (23, 24), (11, 23), (12, 24)  # Tronco
]

# Mapa da mão (se o MediaPipe não tiver, usamos este manual)
```

```
CONEXOES_MAO = mp.solutions.hands.HAND_CONNECTIONS if hasattr(mp.solutions,
'hands') else [
    (0, 1), (1, 2), (2, 3), (3, 4), (0, 5), (5, 6),
    (6, 7), (7, 8), (0, 9), (9, 10), (10, 11), (11, 12),
    (0, 13), (13, 14), (14, 15), (15, 16), (0, 17), (17, 18),
    (18, 19), (19, 20), (5, 9), (9, 13), (13, 17)
]

# --- LOOP PRINCIPAL (Onde a magia acontece) ---
# O programa fica preso aqui dentro a repetir isto infinitamente
tempo_ant = time.time()
contagem_quadros = 0

while True:
    # 1. Ler Nova Imagem
    quadro = fluxo.ler()
    if quadro is None: continue

    # 2. Calcular FPS (Velocidade do programa)
    tempo_atual = time.time()
    dt = tempo_atual - tempo_ant
    if dt > 0:
        fps = 1 / dt
    else:
        fps = 0
    tempo_ant = tempo_atual
    contagem_quadros += 1

    # 3. Preparar Imagem para a IA (Converter cores)
    rgb = cv2.cvtColor(quadro, cv2.COLOR_BGR2RGB)
    imagem_mp = mp.Image(image_format=mp.ImageFormat.SRGB, data=rgb)
    timestamp_ms = int(time.time() * 1000)

    # 4. EXECUTAR DETEÇÃO (Perguntar à IA onde está o corpo e a mão)
    resultado_pose = detetor_pose.detect_for_video(imagem_mp, timestamp_ms)
    resultado_mao = detetor_mao.detect_for_video(imagem_mp, timestamp_ms)

    saida = quadro.copy() # Fazer uma cópia para desenhar por cima
    cv2.putText(saida, f"FPS: {int(fps)}", (saida.shape[1]-150, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

    angulo_braco, orientacao_braco, estado_flexao = None, "Indefinido",
"Estendido"

    # 5. Se detetou CORPO:
```

```
    if resultado_pose.pose_landmarks:
        plm = resultado_pose.pose_landmarks[0]
        # Tenta descobrir o lado
        lado = "Direita"
        if resultado_mao.hand_landmarks:
            lado =
determinar_lado_mao(*obter_centro_mao(resultado_mao.hand_landmarks[0]), plm) if
resultado_mao.hand_landmarks else "Direita"

        # Escolhe os pontos certos para Esquerda ou Direita
        idx_s, idx_e, idx_w = (11, 13, 15) if lado == "Esquerda" else (12, 14,
16)
        angulo_braco = calcular_angulo_3pts((plm[idx_s].x, plm[idx_s].y),
(plm[idx_e].x, plm[idx_e].y), (plm[idx_w].x, plm[idx_w].y))

        orientacao_braco = determinar_orientacao_braco(angulo_braco)
        # Vê se o cotovelo está dobrado
        estado_flexao, diff_y = determinar_flexao_braco(plm[idx_s].y,
plm[idx_w].y)

        # Escreve no ecrã e desenha (TRADUZIR OUTPUT VISUAL)
        cv2.putText(saida, f"Braco: {int(angulo_braco)} graus", (10, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
        cv2.putText(saida, f"Pos: {orientacao_braco}", (250, 30),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
        cv2.putText(saida, f"Flex: {estado_flexao}", (10, 55),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 255), 2)
        desenhar_marcos_personalizado(saida, plm, CONEXOES_POSE_BRACOS,
COR_ARTICULACAO_POSE, COR_OSSO_POSE)

    # 6. Se detetou MÃO:
    if resultado_mao.hand_landmarks:
        marcos_mao = resultado_mao.hand_landmarks[0]
        # Calcula Rotação da palma
        _, val_rotacao_mao, str_orientacao_mao, _ = calcular_rotacao_mao(
            marcos_mao, estado_flexao, orientacao_braco, lado)

        # Verifica cada dedo
        y0 = 80
        for nome, ids in DEDOS.items():
            esta_aberto = calcular_angulo_dedo(marcos_mao, ids) >
LIMIARES_DEDOS[nome]
            # TRADUZIR O NOME DOS DEDOS NO ECRÃ (3 LETRAS)
            nome_display = nome[:3].upper() # POL, IND, MED...
            status_display = "ABE" if esta_aberto else "FEC"
```

```
            cv2.putText(saida, f"{nome_display}: {status_display}", (10, y0),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0) if esta_aberto else (0, 0, 255), 2)
            y0 += 20

        # Desenha a barra de rotação
        cv2.rectangle(saida, (10, y0+10), (160, y0+25), (100, 100, 100), 2)
        cv2.circle(saida, (10 + int((val_rotacao_mao + 1) / 2 * 150), y0+17), 6,
(0, 255, 255), -1)
        cv2.putText(saida, f"{str_orientacao_mao}", (10, y0+45),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 255), 2)

        desenhar_marcos_personalizado(saida, marcos_mao, CONEXOES_MAO,
COR_ARTICULACAO_MAO, COR_OSSO_MAO)

    # 7. Mostrar a imagem final
    cv2.imshow("Detecao OTIMIZADA", saida)

    # Se carregar no ESC (tecla 27), sai do programa
    if (cv2.waitKey(1) & 0xFF) == 27: break

# Limpeza final
fluxo.parar()
cv2.destroyAllWindows()
```