Vision par ordinateur et détections d'objets

CHOU Xiaoyu et WANG Xuejiao

April 12, 2025

1 Introduction

Le projet consiste à créer un contrôleur basé sur la vision par ordinateur pour le jeu Space Invaders. On a utilisé un modèle pré-entraîné de MediaPipe capable de reconnaître les gestes et créé un nouveau module de contrôle basé sur control.module.py: en utilisant cv2 pour capturer la vidéo de la caméra, en utilisant mediapipe pour reconnaître les gestes, en convertissant les actions détectées en commandes de jeu et en envoyant ces commandes au serveur WebSocket.

2 Explications

```
import cv2
import asyncio
import websockets
import mediapipe as mp
import numpy as np
```

`cv2` : gère la caméra et l'affichage vidéo.
`asyncio` : permet d'exécuter des fonctions asynchrones.
`websockets` : permet de se connecter à un serveur WebSocket.
`mediapipe` : détecte les mains et les gestes.

```
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(static_image_mode=False,
                       max_num_hands=1,
                       min_detection_confidence=0.7)
```

`mp.solutions.hands` : le module de détection de main :
`max_num_hands = 1` : détecter une seule main.
`min_detection_confidence = 0.7` : seuil de confiance minimum pour accepter une détection.

```
def detect_command_from_landmarks(landmarks):
    if landmarks is None:
        return None
```

Cette fonction est définie pour recevoir les 21 landmarks de la main renvoyés par mediapipe et les commandes de contrôle de sortie.

```
def is_finger_folded(tip_id, pip_id):
        return landmarks[tip_id].y > landmarks[pip_id].y

    finger_tips = [8, 12, 16, 20]
    finger_pips = [6, 10, 14, 18]

    folded = 0
    for tip, pip in zip(finger_tips, finger_pips):
        if is_finger_folded(tip, pip):
            folded += 1

    print(f"[DEBUG] Folded fingers: {folded}")
```

Cette fonction est définie pour déterminer si le doigt est plié. Si le bout du doigt (tips) est en dessous de l'articulation médiane(pips), le doigt est plié.

Ensuite, une boucle est pour comparer la position de la pointe et de l'articulation de chaque doigt, et compter le nombre de plis des doigts.

```
is_index_up = not is_finger_folded(8, 6)
    is_middle_up = not is_finger_folded(12, 10)
    is_ring_folded = is_finger_folded(16, 14)
    is_pinky_folded = is_finger_folded(20, 18)

    if is_index_up and is_middle_up and is_ring_folded and is_pinky_folded:
        print("[DEBUG]    EXIT")
        return "EXIT"
    if folded == 0:
        print("[DEBUG]    FIRE")
        return "FIRE"
    if folded >= 4:
        print("[DEBUG]    ENTER")
        return "ENTER"
```

Si seuls l'index et le majeur sont tendus tandis que les autres doigts sont repliés, alors l'action déclenchée est la sortie du jeu.

Si au moins 4 doigts sont pliés : la commande "ENTER" est retournée.

Si 0 doigts sont pliés : la commande "FIRE" est retournée.

```
    wrist = landmarks[0]
    index_tip = landmarks[8]
    dx = index_tip.x - wrist.x

    print(f"[DEBUG]dx={dx:.3f}")

    if dx < -0.1:
        return "LEFT"
    elif dx > 0.1:
        return "RIGHT"
  return None
```

Ce segment de code compare la position de l'extrémité de l'index avec celle du poignet sur l'axe x, afin de déterminer si le geste de l'utilisateur indique un mouvement vers la gauche ou vers la droite.

```
async def send_commands():
    uri = "ws://localhost:8765"
    print("Connecting to WebSocket server...")

    async with websockets.connect(uri) as websocket:
        print("Connected to WebSocket server!")
```

Ce segment de code définit une fonction asynchrone nommée `send_commands`, destinée à communiquer avec un serveur `WebSocket`.

Il commence par définir l'adresse du serveur, en spécifiant que la connexion doit cibler un service `WebSocket` local fonctionnant sur le port `8765`.

Ensuite, le programme tente d'établir une connexion asynchrone à l'aide de `websockets.connect`, puis utilise l'instruction `async with` pour obtenir un objet de session `WebSocket` (`websocket`) qui servira à l'envoi ou à la réception de données.

Une fois la connexion établie avec succès, un message de confirmation est affiché dans la console pour indiquer que la liaison a bien été établie.

```
        cap = cv2.VideoCapture(0)

        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                continue

            frame = cv2.flip(frame, 1)
            rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            results = hands.process(rgb)
```

On utilise `OpenCV` pour accéder à la caméra de l'ordinateur et exécuter une boucle de capture d'image en temps réel.

À chaque itération, l'image est prétraitée, notamment par une symétrie horizontale (retournement gauche-droite), puis convertie du format `BGR` (par défaut dans `OpenCV`) au format `RGB`, requis par `MediaPipe`.

Une fois prétraitée, chaque image est transmise au module `MediaPipe Hands`, qui effectue automatiquement la détection de la main, et extrait 21 points clés correspondant aux articulations et extrémités de la main (*landmarks*).

```
            command = None
            if results.multi_hand_landmarks:
                hand_landmarks = results.multi_hand_landmarks[0]
                command =
                    detect_command_from_landmarks(hand_landmarks.landmark)
```

Si une main est détectée, on appelle la fonction `detect_command_from_landmarks()` afin de déterminer la commande correspondante en fonction du geste effectué (par exemple `"FIRE"`, `"LEFT"`, etc.).

```
            for hand_landmarks in results.multi_hand_landmarks:
                    mp.solutions.drawing_utils.draw_landmarks(
                        frame, hand_landmarks, mp_hands.HAND_CONNECTIONS)
```

Si une main est détectée, les fonctions utilitaires fournies par `MediaPipe` sont utilisées pour afficher les points clés et les connexions squelettiques directement sur l'image.

```
            if command:
                print(f"Detected: {command}")
                 if command == "EXIT":
                    print(" Exit gesture detected. Exiting...")
                    break
                try:
                    await websocket.send(command)
                    print(f"Sent: {command}")
                except Exception as e:
                    print(f"[ERROR] WebSocket send failed: {e}")


            cv2.imshow("Camera Control", frame)
```

Ce segment de code analyse le résultat de la reconnaissance : si la commande détectée est `"EXIT"`, le programme se termine immédiatement ; sinon, la commande est envoyée au jeu via `WebSocket`, tout en affichant l'image annotée à l'écran.

```
        cap.release()
        cv2.destroyAllWindows()

if __name__ == "__main__":
    asyncio.run(send_commands())
```

Libérer les ressources de la caméra et fermer toutes les fenêtres ouvertes par `OpenCV`.