Hill Climb Racing

Game Using OpenCV

Introduction :

- Hill Climb Racing is a physics-based driving game where the player controls a vehicle navigating through uneven terrains.
- In this project, we integrate OpenCV and MediaPipe to control the game using hand gestures, providing a unique and interactive way to play without using traditional keyboard inputs.
- The system relies on computer vision techniques to track hand movements and translate them into game controls.

Key Concepts :

1. **OpenCV :** Used for real-time image processing and video capture
2. **MediaPipe Hands :** A deep-learning-based hand-tracking framework that detects hand landmarks.
3. **Hand Landmark Detection :** Identifying key points on the hand, such as fingertips and joints, to determine gestures.
4. **Gesture Recognition :** Using the position of fingers to classify actions such as pressing the gas pedal or braking.
5. **Keyboard Simulation (pydirectinput):** Sending virtual keypresses to the game based on recognized hand gestures.

Libraries Used :

- opencv-python: For video capture and image processing.
- mediapipe: For hand tracking and gesture recognition.
- pydirectinput: To simulate keyboard inputs.
- time: To introduce delays in execution where needed.

Implementation Details :

1. Setting Up Video Capture

A webcam is used to capture real-time hand movements. OpenCV's `VideoCapture` is utilized to access the video stream.

```
video = cv2.VideoCapture(0)
```

2. Hand Tracking with MediaPipe

MediaPipe's hand detection module is employed with a confidence threshold of 0.8 to ensure reliable hand tracking.

```
mp_hand = mp.solutions.hands
with mp_hand.Hands(min_detection_confidence=0.8, min_tracking_confidence=0.8) as hands:
```

3. Extracting Hand Landmarks

The hand key points are extracted, and their coordinates are used to determine finger states.

```
lmList = []
if results.multi_hand_landmarks:
    for hand_landmark in results.multi_hand_landmarks:
        for id, lm in enumerate(hand_landmark.landmark):
            h, w, c = image.shape
            cx, cy = int(lm.x * w), int(lm.y * h)
            lmList.append([id, cx, cy])
```

4. Gesture Recognition

The number of extended fingers is counted to determine the action:

- **No fingers extended:** Apply brakes
- **All fingers extended:** Accelerate

```
if total == 0:
    new_key = BREAK_KEY
elif total == 5:
    new_key = GAS_KEY
```

5. Sending Keystrokes to the Game

Using `pydirectinput`, the system simulates key presses based on detected gestures.

```
if new_key != current_key:
    if current_key is not None:
        pydirectinput.keyUp(current_key)
    if new_key is not None:
        pydirectinput.keyDown(new_key)
    current_key = new_key
```

6. Displaying Feedback

Visual indicators are provided on the screen to confirm detected gestures.

```
cv2.rectangle(image, (20, 300), (270, 425), (0, 255, 0), cv2.FILLED)
cv2.putText(image, "BRAKE" if new_key == BREAK_KEY else "GAS", (45, 375),
cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 0, 0), 5)
```

Conclusion :

This project highlights the potential of using computer vision for game control, offering an intuitive and immersive experience. The approach can also be extended to other games or interactive applications that require hands-free control.