JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 25, No. 11, pp. 3291-3299, Nov. 2024

Check for updates

자세 추정 알고리즘을 이용한 웹캠 기반 격투 게임 제어 시스템 연구

강 경 민[1] · 허 관[2] · 이 준 석[3*]
[1]동서대학교 영상콘텐츠학과 석사과정
[2]동서대학교 소프트웨어학과 학사과정
[3]동서대학교 영상애니메이션학과 교수

Webcam-Based Fighting Game Control System Using Pose Estimation Algorithm

Kyoung-Min Kang[1] · Gwan Heo[2] · David Junesok Lee[3*]

[1]Master's Course, Department of Visual Contents, Dongseo University Graduate School, Busan 47011 Korea
[2]Bachelor's Course, Department of Software Engineering, Dongseo University, Busan 47011, Korea
[3]Professor, Department of VFX & Animation, Dongseo University, Busan 47011, Korea

[요 약]

본 연구는 Google MediaPipe와 Unreal Engine을 통합한 새로운 실시간 몰입형 게임 컨트롤 시스템을 제안한다. 본 시스템은 특수 장비 대신 일반 웹캠을 사용하여 접근성을 높였다. MediaPipe와 LSTM 신경망 기반 포즈 추정 모듈이 실시간으로 플레이어의 동작을 인식하고, Unreal Engine과 연동하여 몰입감 있는 게임 플레이를 구현한다. 실험 결과, 평균 338.66ms의 지연 시간과 89.33%의 동작 인식 정확도를 달성하였으며 이는 개선이 필요하지만 전시형 콘텐츠에서 사용 가능한 수치이다. 이를 통해 본 논문은 접근성 높은 동작 인식 게임의 실현 가능성을 검증하였다. 제안된 시스템은 교육, 의료, 엔터테인먼트 분야에서 활용 잠재력을 보여준다. 향후 연구에서는 딥러닝 기술기반 정확도 향상과 실시간 성능 최적화를 통한 개인화를 목표로 한다.

[Abstract]

This study introduces a novel real-time immersive game control system that integrates Google MediaPipe with Unreal Engine. The system enhances accessibility by leveraging standard webcams instead of specialized equipment. A pose estimation module, based on MediaPipe and LSTM neural networks, recognizes player movements in real-time and interfaces with Unreal Engine to enable immersive gameplay. Experimental results show an average latency of 338.66 ms and motion recognition accuracy of 89.33%. While these figures suggest room for improvement, they are sufficient for exhibition-style content. This study demonstrates the feasibility of accessible motion recognition games. The proposed system has potential applications in education, healthcare, and entertainment. Future research will focus on improving accuracy through deep learning techniques and enabling personalization via real-time performance optimization.

색인어 : 동작인식, 자세추정, 몰입형 콘텐츠, 신경망모델, 언리얼엔진
Keyword : Motion Recognition, Pose Estimation, Immersive Contents, Neural Network Model, Unreal Engine

http://dx.doi.org/10.9728/dcs.2024.25.11.3291

Received 26 September 2024; **Revised** 28 October 2024
Accepted 14 November 2024

***Corresponding Author; David Junesok Lee**
Tel: ※ 개인정보 표시제한
E-mail: jsdavid88@g.dongseo.ac.kr

Ⅰ. 서 론

1-1 연구 배경

최근 게임 산업은 기술의 발전과 함께 급속도로 변화하고 있다. 특히, 사용자의 실제 동작을 게임 내 조작으로 연결하는 동작 인식 기술이 큰 주목을 받고 있다[1]. 이러한 기술은 게임의 몰입도를 높이고 사용자 경험을 향상시키는 데 크게 기여하고 있다. 동작 인식 기술을 게임에 도입한 대표적인 사례로는 Microsoft의 Kinect와 Nintendo의 Wii를 들 수 있다.

Kinect는 적외선 센서와 컬러 카메라를 이용해 사용자의 전신 동작을 3D로 인식할 수 있었고 얼굴인식 기능도 제공한다[2]. Wii 리모컨 플러스는 가속도 센서와 자이로스코프가 내장된 컨트롤러를 통해 사용자의 손동작을 정밀하게 추적할 수 있다[3]. 이러한 시스템들은 게임 시장에 혁신을 가져왔고, 기존에 게임에 관심이 없던 새로운 사용자층을 유입시키는 데 성공했다. 특히 가족 단위의 사용자들이 함께 즐길 수 있는 스포츠, 댄스 게임 등이 큰 인기를 얻었다.

그러나 이러한 시스템들은 대부분 특수한 하드웨어를 필요로 한다는 한계가 있다. Kinect나 Wii와 같은 전용 장비는 일반 사용자들에게 추가적인 비용 부담을 주었고, 이는 동작 인식 게임의 보급에 장애 요인이 되었다.

그러나 컴퓨터 비전과 머신러닝 기술의 발전으로 인해 일반 카메라만으로도 고성능의 동작 인식이 가능해지고 있다. 특히 딥러닝 기반의 자세 추정 알고리즘들이 큰 성과를 거두면서, 특수한 센서 없이도 정확한 동작 인식이 가능해졌다[4]. 이러한 기술적 진보는 동작 인식 게임의 새로운 가능성을 열어주고 있다.

Google의 MediaPipe와 같은 오픈소스 프레임워크의 등장은 이러한 첨단 컴퓨터 비전 기술을 더욱 접근성 있게 만들었다. MediaPipe는 복잡한 머신러닝 알고리즘을 추상화하여 제공함으로써, 개발자들이 쉽게 고성능의 동작 인식 시스템을 구현할 수 있게 해주었다. 또한, 모바일 기기에서도 실시간으로 동작할 수 있도록 최적화되어 있어, 스마트폰이나 태블릿에서도 동작 인식 애플리케이션을 구현할 수 있게 되었다[5].

이와 함께, 게임 엔진 기술도 크게 발전하여 고품질의 그래픽과 복잡한 게임 로직을 쉽게 구현할 수 있게 되었다. 특히 Unreal Engine과 같은 현대적인 게임 엔진은 물리 기반 렌더링, 고급 애니메이션 시스템, 강력한 물리 엔진 등을 제공하여, 현실감 있는 게임 환경을 구축할 수 있게 해준다. 또한, 비주얼 스크립팅 도구인 블루프린트 시스템을 통해 프로그래밍 지식이 없는 개발자들도 복잡한 게임 로직을 구현할 수 있게 되었다.

1-2 연구 목적

본 연구의 주요 목적은 다음과 같다. 일반 웹캠과 MediaPipe를 이용하여 실시간 동작 인식 시스템을 구현한다. 인식된 동작을 격투 게임에 적합한 명령으로 변환하는 알고리즘을 제작한다. Unreal Engine을 활용하여 동작 인식 결과를 실시간으로 반영하는 격투 게임을 구현하여 이를 바탕으로 일반 웹캠 기반 동작 인식 격투 게임 컨트롤 시스템의 가능성과 한계를 탐구한다.

본 연구는 접근성 높은 동작 인식 게임 시스템의 실현 가능성을 입증하고, 다양한 게임 장르 및 인터랙티브 애플리케이션으로의 확장 가능성을 제시하고자 한다. 이를 위해 동작 인식 기술의 동향과 본 시스템에 사용된 기술을 정리하고, 시스템 구동 시나리오를 기반으로 동작 인식 정확도 평가, 지연 시간 측정, 성능 분석을 수행하였다. 이 과정을 통해 일반 웹캠 기반 동작 인식 기술의 가능성과 개선 방향을 제시하고자 한다.

Ⅱ. 동작 인식 기술과 게임 개발 환경

2-1 동작 인식 기술 동향

동작 인식 기술은 컴퓨터 비전과 기계학습의 발전에 힘입어 빠르게 진화하고 있다. 초기의 동작 인식 기술은 주로 특징점 추출과 규칙 기반 분류 방법을 사용했으나[1], 최근에는 딥러닝 기반의 방식이 주류를 이루고 있다[6].

합성곱 신경망(CNN; convolution neural network)과 순환 신경망(RNN; recurrent neural network)을 결합한 모델들이 높은 성능을 보이고 있으며, 3D CNN을 이용한 방식도 제안되었다[7]. 특히 포즈 추정 분야에서는 OpenPose, DeepPose 등의 알고리즘이 큰 성과를 거두었다.

최근에는 실시간 처리 능력이 크게 향상되어, 모바일 기기에서도 동작 인식이 가능한 수준에 이르렀다. Google의 MediaPipe는 이러한 트렌드를 반영한 대표적인 프레임워크이다[5].

2-2 게임에서의 동작 인식 적용 사례

게임 산업에서 동작 인식 기술의 도입은 새로운 형태의 사용자 경험을 제공하며 큰 반향을 일으켰다. 대표적인 사례로는 다음과 같은 것들이 있다. Microsoft Kinect는 적외선 센서와 RGB (red green blue) 카메라를 이용해 사용자의 전신 동작을 인식한다. Xbox 게임 콘솔과 함께 사용되어 다양한 스포츠, 댄스 게임 등에 활용되었다[2]. Nintendo Wii는 가속도 센서와 자이로스코프가 내장된 컨트롤러를 이용해 사용자의 손동작을 인식한다. 테니스, 복싱 등의 스포츠 게임에서 높은 인기를 얻었다[3]. PlayStation Move는 Sony에서 개발한 모션 컨트롤러로, Wii와 유사하지만 카메라를 추가로 사용하여 더 정확한 3D 위치 추적이 가능하다[8]. Just Dance

은 Ubisoft에서 개발한 댄스 게임 시리즈로, 초기에는 Wii 리모컨을 사용했지만 이후 Kinect, 스마트폰 카메라 등 다양한 입력 방식을 지원하게 되었다[9].

이러한 시스템들은 게임의 몰입도를 크게 높이고 새로운 사용자층을 유입시키는 데 성공했지만, 대부분 특수한 하드웨어가 필요하다. 또한, 이러한 시스템들은 주로 콘솔 게임 플랫폼에 한정되어 있어, PC (personal computer)나 모바일 게임에서는 상대적으로 적용이 제한적이었다.

2-3 포즈 추정 알고리즘 방식 비교

포즈 추정 알고리즘은 상향식(Bottom-Up) 방식과 하향식(Top-Down) 방식으로 구분할 수 있다[10]. 상향식 방식에서는 이미지 내의 모든 관절 포인트를 추출한 후 이를 바탕으로 사람의 포즈를 구성하며, 하향식 방식에서는 사람의 존재를 우선으로 탐지한 후 각 사람의 신체 관절을 개별적으로 추정한다. 예를 들어, 상향식 방식을 사용하는 OpenPose는 여러 사람을 동시에 탐지하는 데 효율적이나 이미지가 복잡해질 경우 노이즈가 발생할 가능성이 있다. 반면, MediaPipe는 하향식 방식을 채택하여 복잡한 자세나 겹친 인물에 대해서도 상대적으로 정확한 포즈 추정을 제공하나, 이미지 내 인물 수가 많아질수록 처리 속도가 저하되는 단점이 있다.

본 연구에서는 한 사람의 포즈 추정을 통해 게임에 활용할 시스템을 구축하기 위해 복잡한 자세에서도 높은 정확도를 제공하는 하향식 방식의 MediaPipe를 채택하였다.

MediaPipe는 Google에서 개발한 오픈소스 머신러닝 파이프라인으로, 다양한 컴퓨터 비전 태스크를 쉽게 구현할 수 있게 해준다[5]. MediaPipe의 주요 특징은 다음과 같다. Android, iOS, 웹 브라우저 등 다양한 플랫폼에서 동작한다. 모바일 기기에서도 실시간으로 동작할 수 있도록 최적화되어 있다. 얼굴 인식, 손 추적, 포즈 추정, 객체 검출 등 다양한 태스크를 제공한다. 사용자 정의 모델을 쉽게 통합할 수 있는 구조를 가지고 있다. 특히 MediaPipe의 포즈 추정 기능은 실시간 동작 인식에 매우 적합하다. 33개의 신체 랜드마크를 추정하며, 2D와 3D 좌표를 모두 제공한다. 이는 본 연구에서 제안하는 격투 게임 시스템의 동작 인식에 핵심적인 역할을 한다.

2-4 장단기기억(LTMS) 기술 소개

딥러닝을 이용해 시계열 데이터를 학습 시키는 대표적인 모델은 순환신경망(Recurrent Neural Network)이다. 순환신경망은 상위 층과 하위 층의 뉴런들 사이의 순환 연결과 선택적 자가 피드백 연결을 가지고 있어 이전 단계에서 현 단계로 데이터를 전파하는 방식으로 시계열 데이터의 메모리를 구축하게 된다. 그러나 순환신경망은 역전파되는 오류가 점점 줄어들어 학습에 거의 영향을 미치지 못하는 기울기 소멸 문제가 있다. 이로 인해 5~10개 이상의 시간 단계를 연결할 수 없는데 이런 문제를 해결하기 위해 게이트 기능을 도입한 장단기기억(LSTM; long short-term memory)) 모델이 제안되었다. 장단기기억 모델은 1,000개 이상의 개별 시간 단계의 최소 시간 지연을 연결하는 방법을 학습하는 것이 가능하다[11]. 이는 본 연구에서 제안하는 격투 게임 시스템의 커맨드를 입력하기 위한 동작을 판별하는데 중요한 역할을 한다.

2-5 Unreal Engine의 게임 개발 환경

Unreal Engine은 Epic Games에서 개발한 게임 엔진으로, 고품질의 3D 그래픽과 물리 기반 렌더링을 제공한다[12]. Unreal Engine의 주요 특징은 다음과 같다. 실시간 레이 트레이싱, 동적 조명 등 고품질 그래픽 기술을 지원한다. 애니메이션 블렌딩, 컨트롤 릭 시스템을 통한 캐릭터에 대한 복잡한 애니메이션을 손쉽게 제어할 수 있다. 블루프린트 시스템은 비주얼 스크립팅 도구로, 프로그래밍 지식 없이도 게임 로직을 구현할 수 있다[13]. PC, 콘솔, 모바일 등 다양한 플랫폼에 대한 개발과 크로스 플랫폼 개발을 지원한다. 마켓플레이스는 다양한 에셋과 플러그인을 제공하여 개발 효율을 높인다. Unreal Engine은 특히 C++ 코드와 블루프린트를 혼용할 수 있어, 성능과 개발 효율성을 모두 확보할 수 있다는 장점이 있다. 이는 본 연구에서 제안하는 시스템의 게임 로직 구현과 렌더링에 중요한 역할을 한다.

Ⅲ. 시스템 설계 및 구현

3-1 전체 시스템 구조

제안하는 시스템의 전체 구조는 그림 1과 같다. 시스템은 MediaPipe 기반 포즈 추정 모듈, 동작 데이터를 이용한 동작 커맨드 학습, 학습된 동작 판별 모델을 이용한 동작 판별 모듈, Unreal Engine 기반 게임 로직 및 렌더링 모듈로 크게 네 부분으로 구성된다.

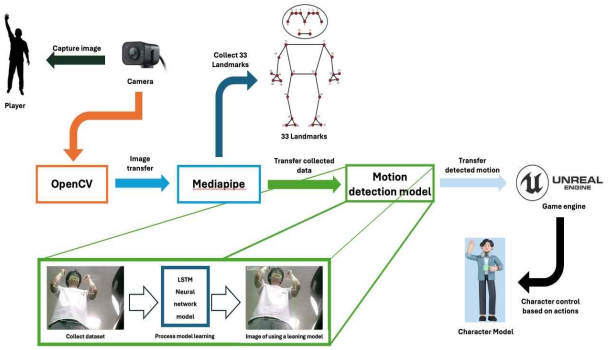

그림 1. 자세 추정 기반 몰입형 격투 게임의 개념도
Fig. 1. Conceptual diagram of an immersive fighting game based on posture estimation

웹캠으로부터 입력받은 영상은 MediaPipe를 통해 처리되어 포즈 데이터로 변환된다. 이 데이터는 LSTM 신경망 모델을 통해 매칭되는 동작을 학습하고, 학습이 완료된 모델을 이용하여 동작을 분석한다. 분석된 동작은 게임 커맨드로 변환되고, 시리얼 통신을 통해 Unreal Engine으로 전송된다. Unreal Engine은 받은 커맨드를 게임 로직에 반영하여 화면에 출력한다.

본 시스템에서 사용하는 커맨드는 Guard, Attack, Counter, Uppercut, Finish 총 5가지이며 각 커맨드를 발동시키기 위한 포즈는 그림 2와 같다.

그림 2. 커맨드를 발동 시키기 위한 포즈
Fig. 2. Pose for activating commands

3-2 MediaPipe를 이용한 포즈 추출

MediaPipe의 Pose 모듈을 사용하여 웹캠 영상에서 실시간으로 사용자의 포즈를 추출한다. 추출된 포즈 데이터는 33개의 신체 랜드마크 좌표로 구성된다. 각 랜드마크는 x, y, z 좌표와 가시성(visibility) 점수를 가진다.

본 시스템에서는 웹캠의 좁은 화각과 화질 특성상 상체만 포즈를 추출한다.

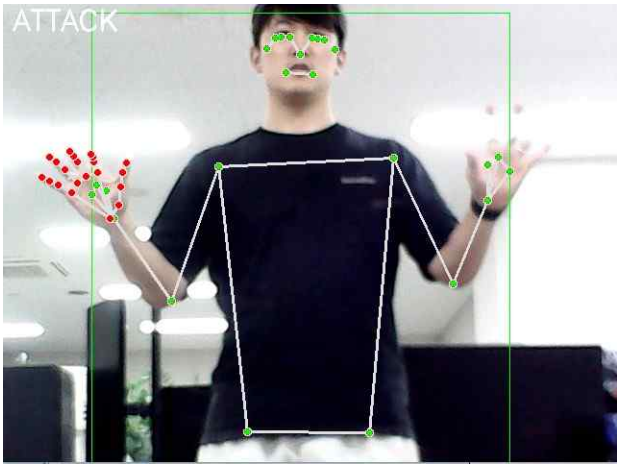

그림 3. MediaPipe를 사용한 포즈 추출
Fig. 3. Pose extraction using MediaPipe

본 시스템에서는 웹캠의 좁은 화각과 화질 특성상 상체만 포즈를 추출한다. 포즈 추출 과정은 웹캠으로부터 프레임 획득, MediaPipe Pose 모델에 프레임 입력, 포즈 추정 결과(랜드마크) 획득, 결과를 정규화된 좌표로 변환, 변환된 좌표의 각도 계산, 계산한 각도를 이용해 포즈 판단의 순서로 이루어진 다 .

Algorithm 1 Pose Estimation Using MediaPipe in Python

Require: Video capture stream 'cap' is opened
Ensure: Pose estimation is performed on each frame of the video stream
Initialize MediaPipe pose solution with the following parameters:
    static_image_mode ← False
    model_complexity ← 1
    smooth_landmarks ← True
    min_detection_confidence ← 0.5
    min_tracking_confidence ← 0.5
Initialize 'pose' ← MediaPipe Pose Solution
while video capture stream 'cap' is opened **do**
    Read a frame from 'cap', store in 'frame'
    Get dimensions of 'frame' as *height (h), width (w), channels (c)*
    if not *ret* **then**
        Break the loop
    end if
    Convert the image from BGR to RGB color space: *image* ← **cvtColor**(*frame*, **COLOR_BGR2RGB**)
    Set image flags to *not writeable* for efficiency: *image.flags.writeable* ← False
    Process the image for pose detection: *results* ← **pose.process**(*image*)
    Set image flags to *writeable*: *image.flags.writeable* ← True
    Convert the image back to BGR color space: *image* ← **cvtColor**(*image*, **COLOR_RGB2BGR**)
    Extract landmarks from pose detection results: *landmarks* ← *results.pose_landmarks.landmark*
end while

그림 4. Python에서 MediaPipe를 사용한 자세 추정
Fig. 4. Pose estimation using MediaPipe in Python

포즈 추출 코드의 핵심 부분은 그림 4와 같다.

그림 4는 MediaPipe의 Pose 모듈을 초기화하고, 각 프레임에서 포즈를 추출하는 알고리즘을 나타낸다. 추출된 랜드마크 데이터는 후속 처리를 위해 리스트 형태로 반환된다.

3-3 동작 데이터 학습

추출된 포즈 데이터는 LSTM 신경망 모델을 이용하여 공격, 방어, 카운터, 어퍼컷, 마무리 동작으로 학습된다. 학습 코드의 예시는 그림 5과 같다.

Algorithm 2 Pose Estimation Model Training

Require: CSV files for various pose data
Ensure: Trained LSTM model for pose classification

function TrainPoseEstimationModel
    Load the CSV files containing pose data: POSE_NONE, POSE_GUARD, POSE_ATTACK, POSE_COUNTER, POSE_UPPERCUT, POSE_FINISH.
    ▷ Prepare Data
    Initialize empty arrays X and y for storing sequences and labels
    Set the number of timesteps: no_of_timesteps ← 20
    pose dataset (e.g., none_df, guard_df, . . .)
    Extract dataset values from all columns except the first
    for i ← no_of_timesteps to *n_samples* **do**
        Append sequence of 20 timesteps from dataset to X
        Append corresponding label to y
    end for
    Convert X and y to NumPy arrays
    ▷ Split Data
    Split the dataset into training and testing sets: $X_{\text{train}}, X_{\text{test}}, y_{\text{train}}, y_{\text{test}}$
    ▷ Model Definition
    Initialize a Sequential model
    Add 4 LSTM layers with 50 units each and Dropout of 0.2 after each LSTM layer
    Add a Dense layer with 17 units and softmax activation for multi-class classification
    ▷ Compile and Train Model
    Compile the model with the Adam optimizer, sparse categorical cross-entropy loss, and accuracy metric
    Train the model on X_{train} and y_{train}, validating on X_{test} and y_{test}, with 100 epochs and a batch size of 32
    ▷ Save the trained model
    Save the model as "pose-estimation-model11.h5"
end function

그림 5. 자세 추정 모델 훈련
Fig. 5. Pose estimation model training

3-4 포즈 데이터 분석 및 게임 명령 변환

추출된 포즈 데이터는 Python으로 구현된 분석 모듈에서 처리된다. 이 모듈은 연속된 프레임의 포즈 변화를 분석하여 공격, 방어, 카운터 등의 격투 동작을 식별한다. 동작 식별 알고리즘은 추출된 랜드마크 중 인식을 위한 랜드마크 저장, 학

습된 모델을 이용하여 추출된 랜드마크로부터 동작 판별, 판별된 동작 게임 엔진으로 전달한다. 그 중 랜드마크 저장 코드의 예시는 그림 6과 같다.

Algorithm 3 Save Extracted Landmark

Require: Initialized video capture object cap and pose/hand processing models
Ensure: Process each frame from video stream to detect pose and hand landmarks

```
function DetectLandmarks
    while cap.isOpened() do
        ret, frame ← cap.read()                    ▷ Capture a frame from video stream
        frame_RGB ← cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)    ▷ Convert frame to RGB
        if ¬ret then                               ▷ If the frame is not captured, exit the loop
            break
        end if
        results ← pose.process(frame_RGB)          ▷ Process pose landmarks
        hand_results ← hands.process(frame_RGB)    ▷ Process hand landmarks
        if results.pose_landmarks then             ▷ If pose landmarks are detected
            send_end_time ← time.time()            ▷ Record end time
            lm ← make_landmark_timestep(results)   ▷ Generate landmark data
            lm_list.append(lm)                     ▷ Append landmark data to list
        end if
    end while
end function
```

그림 6. 추출된 랜드마크 저장
Fig. 6. Save extracted landmark

동작 판별 코드의 예시는 그림 7과 같다.

Algorithm 4 Landmark Detection and Pose Classification

Require: Pre-trained pose classification model, list of pose landmarks (lm_list)
Ensure: Predict the current pose based on landmarks

```
function LandmarkDetectionAndClassification
    if len(lm_list) == 20 then                     ▷ If 20 landmarks are collected
        t1 ← threading.Thread(target=detect, args=(model, lm_list))    ▷ Start a new thread for detection
        t1.start()
        lm_list.clear()        ▷ Clear the landmark list after starting detection
    end if
    Initialize empty lists: x_coordinate, y_coordinate
     lm ∈ results.pose_landmarks.landmark
    h, w, _ ← frame.shape                          ▷ Get frame dimensions
    cx ← int(lm.x * w), cy ← int(lm.y * h)  ▷ Compute x and y coordinates of landmark
    Append cx to x_coordinate, and cy to y_coordinate

end function

function DETECT(model, lm_list)
    Convert lm_list to NumPy array
    Expand the dimensions of lm_list: lm_list ← np.expand_dims(lm_list, axis=0)
    Predict pose using model: result ← model.predict(lm_list)
    if result[0][0] > 0.5 then
        label ← NONE, current_pose ← POSE.NONE
    else if result[0][1] > 0.5 then
        label ← GUARD, current_pose ← POSE.GUARD
    else if result[0][5] > 0.5 then
        label ← ATTACK, current_pose ← POSE.ATTACK
    else if result[0][9] > 0.5 then
        label ← COUNTER, current_pose ← POSE.COUNTER
    else if result[0][10] > 0.5 then
        label ← UPPERCUT, current_pose ← POSE.UPPERCUT
    else if result[0][13] > 0.5 then
        label ← FINISH, current_pose ← POSE.FINISH
    else
        label ← NONE, current_pose ← POSE.NONE
    end if
    return label
end function
```

그림 7. 랜드마크 감지 및 포즈 분류
Fig. 7. Landmark detection and pose classification

3-5 Unreal Engine으로 데이터 전송

동작 분석 모듈에서 식별된 동작은 String 값으로 생성되어 시리얼 통신을 통해 Unreal Engine으로 전송된다. Unreal Engine 측에서는 C++로 구현된 시리얼 리스너가 이 데이터를 수신하여 게임 로직에 전달한다. 시리얼 통신 코드의 예시는 그림 8과 같다.

Algorithm 5 Serial Communication Example

Require: Serial communication setup and data transmission
Ensure: Data sent over serial port

```
Import Serial
                                                ▷ Initialize serial communication
ser ← Serial('COM7', 9600, timeout = 1)
                                                ▷ Encode and send estimated pose data
estimated_pose ← data to be sent
ser.write(str.encode(estimated_pose))
```

그림 8. 직렬 통신 예
Fig. 8. Serial communication example

그림 9는 Unreal Engine에서 시리얼을 통해 데이터를 수신하고, 이를 게임 로직에 전달하는 과정을 보여준다.

Algorithm 6 Receive Action Command Function

Require: Listener socket with pending data
Ensure: Process received data and pass commands to game logic

```
function ReceiveActionCommand
    Size ← uint32                                  ▷ Size of received data
    ReceivedData ← TArray¡uint8¿                   ▷ Array to store received data
    if ListenerSocket− > HasPendingData(Size) then
        ReceivedData.SetNumUninitialized(min(Size, 65507u))    ▷ Allocate memory for received data
        Read ← 0                                   ▷ Variable to store bytes read
        ListenerSocket− > Recv(ReceivedData.GetData(), ReceivedData.Num(), Read)    ▷ Receive data from socket
                                                   ▷ Convert received data to string
        ReceivedString ← BytesToString(ReceivedData.GetData(), ReceivedData.Num())
                                                   ▷ Process action command in game logic
        ProcessActionCommand(ReceivedString)
    end if
end function
```

그림 9. 액션 명령 수신 기능
Fig. 9. Receive action command function

3-6 게임 로직 구현

게임 로직은 Unreal Engine의 블루프린트 시스템을 이용하여 구현되었다. 각 동작에 대응하는 캐릭터 애니메이션과 게임 효과를 설정하고, 실시간으로 수신되는 동작 명령에 따라 이를 실행한다.

주요 게임 로직은 플레이 난이도 설정, 플레이어 캐릭터 및 적 캐릭터 생성, 동작 명령에 따른 공격 성공 여부, 캐릭터 애니메이션 재생, 체력 계산, 게임 상태 관리, 게임 종료 조건 체크로 구성하였다. 이러한 로직은 Unreal Engine의 블루프린트 시스템을 통해 시각적으로 구현되었으며, 필요한 경우 C++ 코드와 연동하여 성능을 최적화하였다. 특히 커맨드 간 상성을 부여하여 단순한 게임 플레이를 지양하고자 하였다.

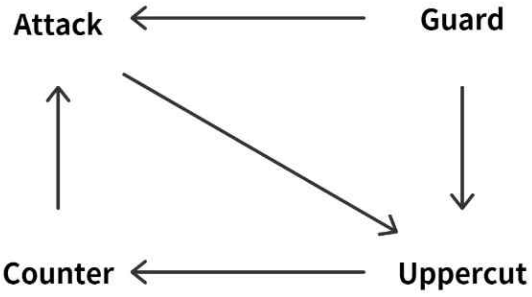

그림 10. 커맨드 간 상성
Fig. 10. Compatibility between commands

3-7 게임 그래픽 구현

본 시스템은 Unreal Engine의 물리기반 렌더링(PBR; physically based rendering)을 적용하여 사실적인 조명과 재질을 구현하였으며, 공격 방어에 따른 이펙트는 나이아가라 시스템을 사용해 제작하였다. 캐릭터의 경우 Unreal Engine에서 기본적으로 제공하는 고품질의 캐릭터 모델링을 사용하여 현실감을 높였다.

그림 11. 언리얼 엔진에서 기본 제공하는 캐릭터
Fig. 11. Characters provided by default in Unreal Engine

격투 게임이라는 특성상 빠른 공격 애니메이션이 많고 서로 겹치는 경우도 많아 Anim Montage와 Blend Space기능을 사용하여 Animation Blueprint에서 재생하였다. Anim Montage는 여러 애니메이션을 단일 에셋으로 결합하고 런타임에서 로직을 통해 순서에 구애받지 않고 동적으로 재생할 수 있는 기능이다[14]. 이는 여러 애니메이션을 몽타주 섹션 기능을 통해 사운드, 이펙트 재생을 효율적으로 제어 할 수 있게 한다. Blend Space는 다수의 애니메이션 또는 포즈를 1차원 또는 2차원 그래프로 구성하여 블렌딩할 수 있는 에셋이다. 이를 사용하면 거의 모든 유형의 블렌딩 배열을 애니메이션에 사용할 수 있다[15]. 본 연구에서는 Anim Motage로 재생된 애니메이션과 Blend Space기능으로 적 캐릭터의 현재 체력 변수를 Idle 애니메이션에 적용하여 체력이 저하되면 자세가 무너지는 애니메이션을 블렌딩하여 격투게임의 몰입도를 더욱 높였다.

그림 12. 체력에 따른 애니메이션 변화
Fig. 12. Animation changes according to physical strength

3-8 시스템 구동 시나리오

본 시스템의 구동 시나리오는 다음과 같다.

게임이 시작되고 플레이어는 웹캠 앞 지정된 장소에 서서 위치를 잡는다. 이후 게임을 시작하고 난이도를 선택하고 본 게임이 시작된다.

동작 분석 모듈에서는 새로운 동작이 분석될 때 마다 Unreal Engine으로 현재 플레이어의 동작에 맞는 커맨드를 전송한다.

Unreal Engine에서는 Serial Communication 플러그인의 리스너를 사용하여 데이터를 수신하여 현재 플레이어의 커맨드를 Player Controller에 저장한다. 플레이어 캐릭터는 0.05초 마다 현재 커맨드를 확인하고 현재 커맨드에 맞는 공격 로직을 실행한다. 공격 로직이 진행되는 동안에는 커맨드 확인이 무효화 되며 로직이 종료되면 다시 커맨드를 확인한다. 적 캐릭터는 난이도에 따라 Max Attack Delay와 Min Attack Delay의 변수로 무작위 길이의 타이머를 설정하여 플레이어가 예측할 수 없는 순간에 공격 로직이 실행된다. 적 캐릭터의 공격 로직이 실행되는 순간 플레이어 캐릭터에서는 현재 커맨드를 확인하여 커맨드 간 상성을 계산한다. 이후 계산 결과에 맞는 애니메이션을 재생하고 플레이어와 적 캐릭터의 현재 체력을 조정한다. 플레이어가 공격 중이라면 타이머는 작동을 정지하였다가 공격 로직이 끝나는 순간 다시 활성화 된다.

적 캐릭터의 현재 체력이 0이하로 떨어지게 되면 Finish 커맨드를 입력할 수 있게 되며 커맨드가 올바르게 입력되면 최종 공격 애니메이션이 재생되고 Victory 영상이 재생되며 승리한다. 플레이어 캐릭터의 현재 체력이 0이하로 떨어지게 되면 적 캐릭터의 최종 공격 애니메이션이 Defeat 영상이 재생되고 패배하게 된다. 이후 승패가 결정나면 모든 변수가 초기화 되며 첫 화면으로 돌아간다.

Ⅳ. 실험 및 결과

4-1 실험 환경 설정

자세 추정 모델의 동작인식 정확도 평가 실험은 Apple M3 Pro CPU (central processing unit), 18GB RAM (random access memory), Apple M3 Pro GPU (graphic processing unit)가 장착된 Mac 운영체제의 노트북에서 진행되었다. 웹캠은 GD-C100 모델을 사용하였으며 소프트웨어 환경으로는 Python 3.8, MediaPipe 0.8.9을 사용하였다.

격투 게임 컨트롤 시스템의 응답시간, 성능 분석 실험의 경우 본 시스템을 정확도 평가 실험을 진행한 노트북으로 올바른 구동이 불가하여 AMD Ryzen 9 5950X CPU, 64GB RAM, NVDIA GeForce RTX 3080가 장착된 Window 운영체제 PC에서 진행되었다.

4-2 동작 인식 정확도 평가

각 동작을 3회씩 수행하는 영상 5개를 웹캠으로 녹화한 영상을 데이터화하여 학습률 70%, Epochs 100, Batch Size 32으로 설정하여 학습을 진행하고 인식 정확도를 측정하였다. 측정한 동작은 Guard, Attack, Counter, Uppercut, Finish 총 5개이며 게임 로직에서 사용되는 5가지의 커맨드를 입력하기 위한 동작이다. 그 결과는 표 1과 같다.

표 1. 동작 인식 정확도 측정 결과
Table 1. Motion recognition accuracy measurement results

Command	Accuracy Count	Percentage
Guard	10/15	66.67%
Attack	13/15	86.67%
Counter	14/15	93.33%
Uppercut	15/15	100%
Finish	15/15	100%
Total Average	67/75	89.33%

본 연구에서 구현한 시스템은 전체적으로 89.33%의 정확도를 보였으며, 특히 Uppercut과 Finish동작의 인식률이 가장 높았다. Guard 동작의 정확도가 상대적으로 낮은 것은 동작의 복잡성과 각 시도별 수행 방식의 차이 때문으로 분석된다.

4-3 시스템 지연 시간 측정

동작 수행부터 게임 내 반영까지의 지연 시간을 측정을 위해 Iphone 13 Mini의 슬로 모션 녹화 기능을 활용하여 시각적으로 분석하는 방법을 채택했다. 실험 설계는 커맨드 동작을 50회 무작위로 반복 수행하며 모듈의 인식 화면과 게임화면이 동시에 프레임 내에 포착될 수 있도록 구성하였다.

데이터 분석은 비디오 편집 소프트웨어를 사용한 프레임 단위 분석을 통해 수행되었으며, 각 동작의 시작 프레임과 게임 화면에 반영된 종료프레임 간의 차이를 계산하여 지연시간을 산출하였다. 촬영된 영상의 FPS (frame per second)는 185 fps이며 각 프레임은 약 5.41 ms로 계산했다. 결과는 표 2와 같다.

표 2. 시스템 지연 시간 측정 결과
Table. 2. System latency measurement results

Measurement	Minimum Delay (ms)	Maximum Delay (ms)	Average Delay (ms)
50	54.1	562.64	338.66

평균 지연 시간은 338.66ms로 측정되었으며 최소 54.1ms, 최대 562.64ms로 측정되었다. 이는 게임 경험 중 사용자가 지연 시간을 확실히 인식할 수 있는 수준이며 격투 게임을 구현하기에 모듈과 Unreal Engine 간의 통신 최적화, 동작 인식 모듈의 인식 속도 개선이 필요하다고 분석된다. 본 실험 중 웹캠의 성능으로 인한 빛번짐과 좁은 화각이 가장 큰 장애요인으로 작용했다. 특정 커맨드로 인해 빛을 가려 빛번짐 현상이 줄어들면 지연 시간이 감소했으며 커맨드 동작의 크기에 따라 화각을 벗어나는 경우에 지연 시간이 증가했다.

4-4 시스템 성능 분석

시스템의 전반적인 성능을 평가하기 위해 본 시스템 실행 중 1분간의 CPU 점유율, GPU 점유율, 메모리 점유율과 언리얼 엔진 내부의 프레임레이트를 측정하였다. 측정 결과 CPU는 최대 6%의 점유율을 기록했으며 GPU의 경우 최대 15.3%의 점유율, 메모리는 16.7%의 점유율을 지속적으로 유지했다. 시스템은 안정적인 성능을 보여주었으며, 특히 평균 90 fps의 프레임레이트를 유지하며 공격 로직, 방어 로직 진행 중에는 평균 85 fps를 유지하며 부드러운 게임 경험을 제공하였다. GPU 점유율과 메모리 점유율이 상대적으로 높은 것은 Unreal Engine의 고품질 그래픽 렌더링 때문으로 분석된다.

Ⅴ. 결론 및 향후 연구

5-1 결론

본 논문에서는 MediaPipe와 Unreal Engine을 연동한 웹캠 기반 동작 인식 격투 게임 컨트롤 시스템을 제안 및 구현하였다. 이는 고가의 특수 하드웨어 없이도 일반 사용자들이 쉽게 접근할 수 있는 동작 인식 게임 시스템을 실현하는 새로운 접근 방식이다. 실험 결과, 제안된 시스템은 89.33%의 정확도와 평균 338.66ms의 지연 시간을 기록하였다. 이는 상

업적으로 발매된 Kinect에 비해 다소 부족하나, 전시, 체험형 콘텐츠 제작에는 충분히 사용될 수 있는 수준이며 스타트업, 소규모 팀의 제작 접근성을 높일 수 있을 것이다.

본 연구는 일반 웹캠을 기반으로 MediaPipe의 포즈 추정 기술과 Unreal Engine의 게임 개발 환경을 효과적으로 통합하는 방법을 제시하고, 격투 게임에 특화된 동작 인식 알고리즘을 개발하였다. 이는 일반 웹캠을 사용함으로써 접근성을 향상시킬 수 있다는 가능성을 확인했다. 또한, 웹캠 기반 동작 인식 시스템의 성능에 대한 실증적 데이터를 제공함으로써, 접근성 높은 동작 인식 게임 시스템의 실현 가능성을 명확히 보여주었으며 향후 AI (artificial intelligence) 기반 게임 디자인 연구에 유용한 기반 자료가 될 수 있다. 또한, MediaPipe와 Unreal Engine의 통합 과정에서 얻은 결과는 다른 연구자들이 동작 인식 알고리즘을 다양한 개발 환경에 적용할 수 있는 모델을 제공한다.

본 시스템은 게임 산업뿐만 아니라 교육, 의료 재활, 스포츠 트레이닝 산업에서 비대면 체육 수업, 재활 운동 프로그램, 운동 자세 교정과 분석, 일반인 대상 피트니스 프로그램 등의 다양한 확장 가능성을 제시한다. 특히, 경제적 부담이 적고 쉽게 접근할 수 있는 동작 인식 시스템은 더 많은 사용자의 접근성을 향상시키고, 새로운 상호작용 경험을 제공하여 새로운 형태의 기술 적용 사례를 확장하는데 기여할 것으로 기대된다.

향후 연구에서는 특수 하드웨어보다 부족한 정확도와 사용자가 지연을 확실히 인식할 수 있는 지연 시간의 한계로 특정 콘텐츠에서만 운용될 수 있다는 한계를 개선하고자 한다. 이를 위해, 더 큰 규모의 학습 데이터세트를 구축하고, 3D CNN이나 Transformer 구조 등 최신 딥러닝 모델을 적용하여 동작 인식의 정확도를 95% 이상으로 개선하고 지연 시간을 사용자가 체감하기 어려운 100ms 이하로 개선할 계획이다. 또한, 다양한 환경에서의 강건성 향상과 실시간 처리 최적화를 통해 특수 하드웨어와의 성능 격차를 더욱 줄여 시스템 적용 가능 범위를 늘리고자 한다.

결론적으로, 본 연구는 접근성 높은 동작 인식 기술의 게임 산업 적용 가능성을 실증적으로 보여주었으며, 이를 통해 새로운 게임 경험 창출에 기여할 것으로 전망되며 향후 연구를 통해 그 가능성을 더욱 확장해 나갈 것이다.

감사의 글

본 연구는 2024년 과학기술정보통신부 및 정보통신기획평가원의 SW중심대학사업 지원을 받아 수행되었음(2019-0-1817)

본 연구는 동서대학교 4단계 BK21 미래영상콘텐츠 창의융합 교육연구단에 의해 지원되었습니다.

참고문헌

[1] J. K. Aggarwal and M. S. Ryoo, “Human Activity Analysis: A Review,” *ACM Computing Surveys (CSUR)*, Vol. 43, No. 3, 16, April 2011. https://doi.org/10.1145/1922649.1922653

[2] Z. Zhang, “Microsoft Kinect Sensor and Its Effect,” *IEEE MultiMedia*, Vol. 19, No. 2, pp. 4-10, February 2012. https://doi.org/10.1109/MMUL.2012.24

[3] Nintendo. Official Website [Internet]. Available: https://www.nintendo.co.kr/.

[4] Y. LeCun, Y. Bengio, and G. Hinton, “Deep Learning,” *Nature*, Vol. 521, No. 7553, pp. 436-444, May 2015. https://doi.org/10.1038/nature14539

[5] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, ... and M. Grundmann, “MediaPipe: A Framework for Building Perception Pipelines,” arXiv:1906.08172, June 2019. https://doi.org/10.48550/arXiv.1906.08172

[6] S. Herath, M. Harandi, and F. Porikli, “Going Deeper into Action Recognition: A Survey,” *Image and Vision Computing*, Vol. 60, pp. 4-21, April 2017. https://doi.org/10.1016/j.imavis.2017.01.010

[7] S. Ji, W. Xu, M. Yang, and K. Yu, “3D Convolutional Neural Networks for Human Action Recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 35, No. 1, pp. 221-231, January 2013. https://doi.org/10.1109/TPAMI.2012.59

[8] PlayStation. Official Website [Internet]. Available: https://www.playstation.com/ko-kr/.

[9] Wikipedia. Just Dance (Video Game Series) [Internet]. Available: https://en.wikipedia.org/wiki/Just_Dance_(video_game_series).

[10] J.-H. Jang and Y.-J. Choi, “Pose Estimation-Based 3D Model Motion Control Using Low-Performance Devices,” in *Proceedings of Annual Conference of KIPS 2023 (ACK 2023)*, Busan, pp. 763-765, November 2023.

[11] S.-Y. Kim, S.-J. Urm, S.-Y. Yoo, S.-J. Kim, and K.-M. Lee, “Application of Sign Language Gesture Recognition Using Mediapipe and LSTM,” *Journal of Digital Contents Society*, Vol. 24, No. 1, pp. 111-119, January 2023. http://dx.doi.org/10.9728/dcs.2023.24.1.111

[12] Epic Games Developers. Physically Based Materials in Unreal Engine [Internet]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/physically-based-materials-in-unreal-engine.

[13] Epic Games Developers. Overview of Blueprints Visual Scripting in Unreal Engine [Internet]. Available: https://dev.epicgames.com/documentation/en-us/unreal-engine/overview-of-blueprints-visual-scripting-in-unreal-engi

ne.
[14] Epic Games Developers. Animation Montage in Unreal Engine [Internet]. Available: https://dev.epicgames.com/documentation/ko-kr/unreal-engine/animation-montage-in-unreal-engine.
[15] Epic Games Developers. Blend Spaces in Unreal Engine [Internet]. Available: https://dev.epicgames.com/documentation/ko-kr/unreal-engine/blend-spaces-in-unreal-engine.

강경민(Kyoung-Min Kang)

2022년 : 동서대학교 게임학과
디지털콘텐츠학사

2022년~2023년: 한 번 더 스튜디오
2024년~현 재: 동서대학교 일반대학원 영상콘텐츠학과 (석사과정)
※관심분야 : 게임그래픽, 콘텐츠 기획, 실시간 사용자 경험, 증강현실

허관(Gwan Heo)

2019년~현 재: 동서대학교 소프트웨어학과 (학사과정)
※관심분야 : 게임 클라이언트, 서버 개발, 콘텐츠 기획

이준석(David Junesok Lee)

2015년 : 동서대학교
영상애니메이션학과 공학사
2021년 : 중앙대학교 첨단영상대학원
영상제작학 석사

2015년~2019년: 스튜디오 스틸웜 대표
2019년~2021년: 스카에나 기술감독
2021년~현 재: 동서대학교 영상애니메이션학과 교수
※관심분야 : 생성형 AI, 애니메이션, VR