

# GRU 모델 기반 낙상 감지 시스템

권준용, 이민하, 정현석, 박우길

영남대학교 디지털융합대학 컴퓨터학부 정보통신공학전공

(우) 38541 경북 경산시 대학로 280

june2908@naver.com, iminha0512@gmail.com, jhs20531194@gmail.com, wooguilpak@yu.ac.kr

## 요 약

본 논문은 증가하는 1인 가구 및 독거노인의 낙상 사고 즉각적으로 대응하여 환자의 골든타임을 지키기 위해 AI 기반 낙상 감지 시스템을 개발하고 이를 제안한다. 기존의 낙상 감지 시스템은 RADAR, Laser Imaging, Detection And Ranging(LIDAR) 센서를 장착해 높은 정확도를 보여주지만, 제품들 대부분이 Business to Business(B2B) 서비스를 지원하고, 고가의 제품으로 인해 접근하기 어렵다는 한계가 있다. 본 논문에서는 MediaPipe 모듈을 사용하여 주요 관절의 스켈레톤 시퀀스 데이터를 추출하고 이를 기반으로 낙상 여부를 추론하는 낙상 감지 시스템을 개발했다. Skeleton 데이터를 활용함으로써 센서 데이터를 대체할 수 있으며, 하드웨어의 모듈이 간소화되며 비용 또한 줄어들게 된다. 이러한 방식을 통해 1인 가구 및 독거노인의 위험성을 줄이고, 저비용으로 많은 사용자에게 도움이 되기를 기대한다.

## 1. 서 론

노인 낙상 사고의 발생은 점점 늘어나고 있으며, 심각한 손상을 동반하거나 낙상으로 인한 합병증으로 사망에까지 이르게 한다. 우리나라의 경우 65세 이상 노인의 신체 손상 중 반 이상의 원인이 낙상이다 [1].

특히 낙상 사고가 발생했을 때 독거노인 및 1인 가구는 대처하기가 매우 어렵다. 부상이 심할 경우 저혈압, 저산소증 등 2차 손상의 발생 위험이 높아 병원 이송이 신속하게 이뤄져야 한다 [2].

기존의 낙상 감지 시스템은 각종 센서를 연결하여 대상자의 행동, 건강을 파악하는 헬스케어 시스템과 낙상 감지 시스템이 결합 제품이 다수였다. 또한, 제품들 대부분이 B2B 서비스를 지원하고, 많은 기능이 탑재된 제품은 고가를 이루고 있어 일반인이 접근하기 어렵다.

본 논문에서는 기존 시스템이 활용하는 센서 데이터를 MediaPipe를 통해 추출한 Skeleton 데이터로 대체했다. Artificial Intelligence (AI)를 통해 데이터의 낙상 여부를 추론하여 최소화된 하드웨어로 저비용 낙상 감지 시스템을 개발했다.

이로써 보다 많은 1인 가구 및 독거노인이 서비스에 편하게 접근할 수 있게 되고, 낙상 사고 후 2차 손상 조치가 빠르게 이루어질 수 있어 피해를 막을 수 있다.

본 논문은 다음과 같이 구성되어 있다. 2절에서는 LIDAR 센서를 활용한 낙상 감지와 RADAR 센서를

활용한 낙상 감지를 포함한 관련 연구를 간략하게 소개하며, 이들 기존 방식의 문제점과 한계점에 대하여 설명한다. 3절에서는 본 논문에서 제안하는 Skeleton 데이터를 활용한 낙상 감지에 대하여 상세하게 설명한다. 4절에서는 본 논문에서 제안하는 기술/기능의 기능 구현에 대하여 설명하며, 5절에서는 제안된 기술/기능의 성능을 측정하고, 측정 결과를 분석한다. 6절에서 결론을 맺으며 향후 연구개발 계획을 간략히 소개한다.

## 2. 관련 연구

본 논문의 낙상 감지 시스템은 라즈베리파이에 AI 모델을 결합한 시스템이다. 이에 다른 연구에서의 낙상 감지 방식과 제한된 하드웨어 환경에서의 동작을 참고하고자 다음과 같은 논문을 참조했다.

### 2.1 RADAR 센서를 활용한 낙상 감지 시스템

참고문헌 [3]은 RADAR 센서를 활용하여 낙상 감지 시스템을 구현하였다. 환자가 활동하는 공간에 장착된 레이더를 통해 환자의 활동 중에 발생하는 데이터를 주기적으로 서버에 전송한다. 수집된 데이터와 환자의 동작을 학습하여 최적의 모델을 생성한다. 모델의 결과에 따라 낙상으로 판별될 경우 보호자, 가족, 의료진 등에게 환자의 낙상 위치 정보와 함께 낙상 여부를 메시지를 통해 전송한다. 낙상 감지 절차는 4단계로 이루어져 있다. 범위 처리 단계에서는 Raw Data를 Fast Fourier Transform (FFT)으로 처리하여 정적 클러스터를 제거해 움직이는 물체의 신호만을 남긴다. 객체 감지 단계는 Constant False Alarm Rate(CFAR) 알고리즘과 단일 피크 검색을 통하여 각 점의 고도각을 찾는다. 세 번째 단계에서는 카본 빔 가중치와 도플러 FFT를 활용하여 도플러를 추정 후 CFAR 및 고도 추정 중에 생성된 포인트 클라우드와 결합한다. 그룹 트랙커는 포인트 클라우드 데이터를 받아 대상의 위치 파악과 객체 목록을 출력하는 단계를 거친다. 마지막 낙상 감지는 Sequential Estimation Process(SEP) 알고리즘을 사용하여 추론하게 된다. 해당 논문에서는 수학적 알고리즘과 통계적 접근을 통해 정확도 95%를 달성했다. 결과를 도출하는데 걸리는 시간은 AI를 사용했을 때보다 빠를 것으로 예상된다. 하지만 결과를 도출하기까지 많은 전처리와 사전 지식을 필요로 한다. 이는 제품의 유지보수성이 떨어지는 결과를 보일 수 있다. 또한 AI를 활용한 시계열 예측보다 정확도가 조금 떨어지는 모습을 보인다. RADAR 데이터를 시계열 예측 AI 모델(Recurrent Neural Network (RNN), Autoreg

ressive Integrated Moving Average (ARIMA), Transformer 계열)로 추론하게 된다면 전처리 과정이 줄어들고 더 높은 정확도의 예측을 할 것으로 예상된다.

## 2.2 LIDAR 센서와 FSM 알고리즘을 활용한 낙상 감지 시스템

참고문헌 [4]은 LIDAR 센서를 활용하여 낙상 감지 시스템을 구현하였다. 해당 논문에서는 AI를 활용한 낙상 감지 알고리즘에서는 다음과 같은 복잡성이 포함된다고 한다. AI는 블랙박스 모델의 불투명성, 광범위한 데이터셋의 필요성, 그리고 대량의 매개변수와 계산 비용이 필요하다. 다양한 환경에서의 적용 가능성과 이러한 문제들을 해결하기 위해 3D LIDAR 센서와 Finite State Machine (FSM) 알고리즘 활용을 제안한다. 해당 실험에서 낙상을 감지하기 위하여 3D LIDAR 센서를 천장에 비치하였다. 센서 값을 입력받아 특정 조건을 만족하면 고정된 3개의 상태 NO\_HUMAN, HUMAN, FALL(or ALERT)로 전환된다. 사람이 있는지 여부를 판단하는 activity()는 2개의 파라미터를 통해 판단하고, human()에서 낙상과 비낙상을 각각 6개의 파라미터를 통해 판단한다. FSM 알고리즘은 속도가 빨라 임베디드 기기에 최적화 되어있고, LIDAR 센서 노이즈에도 강하다. 낙상 이벤트 기반 alarm logic 구축에도 용이하다. 또한, 이 실험에서 Artificial Neural Network (ANN), Support Vector Machine (SVM) 모델보다 높은 정확도인 89.7% 기록했다. FSM 알고리즘은 많은 장점을 가지고 있지만, 본 논문에서 다른 낙상 감지 시스템이나 앞서 언급된 관련 연구보다 상대적으로 낮은 정확도를 기록하고 있다. 낙상 감지의 속도를 생각한다면 FSM 알고리즘을 채택하는 것이 맞으나, 본 논문에서 제안한 Torchscript 변환 또는 양자화를 적용한 경량 AI 활용, 병렬 처리 설계를 통해 시스템을 구성하게 된다면 비용 대비 높은 정확도를 제공할 것으로 예상된다.

## 3. GRU를 이용한 낙상 감지 시스템

본 연구에서 제안하는 낙상 감지 시스템은 서버모터 기반 카메라 시야 제어 기능과 Skeleton 시계열 분석 기반 Gated Recurrent Unit (GRU) 모델을 통합한 실시간 동작 인식 구조로 설계되었으며, 주요 기능 블록은 상호 연동되는 단일 파이프라인으로 구성된다. 아래의 기능 블록도와 유사코드를 통해 이를 상세하게 설명하고자 한다.

### 3.1 제안된 기능 구조 (기능 블록도)

라즈베리파이에 연결된 카메라로부터 입력된 영상은 MediaPipe Pose를 통해 사람의 관절을 의미하는 Skeleton Landmark로 변환되며, 본 연구는 낙상 판단에 기여도가 높은 13개 핵심 관절만을 사용하여 연산 효율성을 높였다. 추출된 Skeleton은 매 프레임 실시간으로 분석되어 좌·우 손목의 중점으로 정의한 사용자 중심점이 계산되고, 해당 좌표가 영상 중심과 일정 거리 이상 벗어날 경우 Pan/Tilt 서보모터가 자동으로 회전하여 사용자가 항상 카메라 중심에 위

치하도록 시야를 제어한다. 이는 Skeleton 데이터의 누락과 추적 실패를 최소화해 이후 단계의 시계열 분석 품질을 보장하는 핵심 안정화 기능으로 작동한다. Skeleton 데이터는 시간 순서에 따라 3초 길이의 슬라이딩 버퍼에 저장되며, 전처리 단계에서는 프레임 정렬, 랜드마크 결측 보정, 벡터화 과정을 거쳐 GRU 모델이 학습 가능한 시계열 텐서로 변환된다. GRU 모델은 Skeleton의 연속적 변화를 기반으로 정상 동작과 낙상 패턴을 구분하며, 최종 은닉 상태를 이용해 산출한 낙상 확률이 임계값을 초과하는 경우 시스템은 즉시 낙상을 감지한다. 감지 즉시 알림 모듈이 작동하여 보호자에게 Short Message Service (SMS)를 전송하고 부저를 활성화하며, 동시에 카메라 스트림과 서보모터 신호 및 분석 스레드를 안전하게 종료한다. 이러한 전체 기능 구조는 영상 획득, 추적 안정화, 시계열 분석, 의사결정, 경보까지 전 단계를 실시간으로 연동하여 고령자 환경에서의 낙상 상황을 신속하고 정확하게 탐지할 수 있는 하나의 통합 시스템을 구성한다.

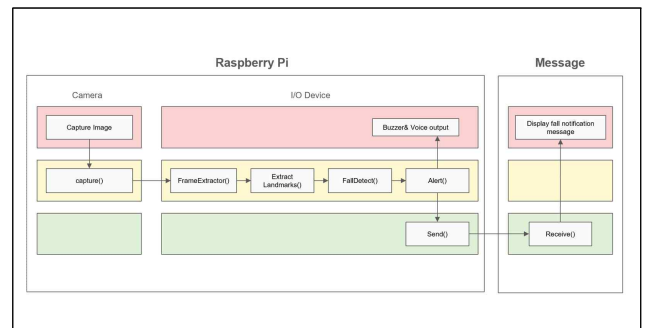


그림 1. 기능 블록도 (Functional Block Diagram)

낙상 감지 시스템은 Raspberry Pi Camera 로 받아 온 영상을 낙상 감지 모델을 통해 낙상을 분류하고 결과에 따라 알림 기능을 수행한다. 이 시스템의 기능은 Camera, Processing, Alarm의 3가지 기능으로 나뉘어있다. 먼저 Camera는 Pose\_process() 함수를 통해 영상에서 사람을 인식하고, Skeleton 좌표를 추출한다. 여기서 추출된 Skeleton 좌표를 통해 객체의 중심을 계산한다. 이 좌표와 영상의 중심좌표를 비교하여 객체가 영상에서 벗어났다고 판단되면, set\_servo\_angle() 함수로 Pan/Tilt 서보모터를 제어하여 객체의 추적을 원활하게 한다. 추적된 객체에서 추출된 Skeleton 좌표는 window.append() 함수를 통해 시계열 버퍼에 저장되며, 저장된 데이터는 Processing에서 Fall\_detecting() 함수에 의해 GRU 모델 기반 시계열 분석을 거쳐 낙상 여부가 분류된다. 낙상으로 판정되었을 경우, 결과값이 Alarm으로 전달되며 Alarm() 함수에 의해 낙상이 감지되었다는 음성메시지와 부저를 출력하고 보호자 연락처로 SMS가 전달된다.

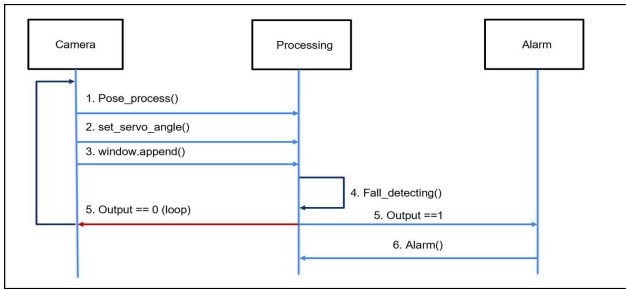


그림 2. 기능 순서도 (Sequence Diagram)

### 3.2 GRU를 통한 낙상 감지 시스템 기능 구조의 알고리즘 (pseudo code)

카메라 기반 자세 추적 과정에서 피사체의 상반신 위치를 안정적으로 추종하기 위해 서보모터를 제어하며 관절 좌표를 수집하는 전체 동작 흐름을 설명한다. 먼저 480×360 해상도의 Red, Green, Blue (RGB) 카메라를 초기화하고 미리보기 화면을 연 뒤, 루프 내에서 주기적으로 프레임을 획득하여 Mediapipe Pose로부터 주요 관절(어깨, 팔, 코, 다리 등)의 3차원 좌표를 추출한다. 관절 검출이 성공하면 시각적 신뢰도(visibility)가 일정 기준 이상인 어깨 및 상체 관절을 활용해 상반신이 화면 중심에 존재하는지를 판단하고, 상체가 좌우로 치우친 경우 pan 값을 동적으로 조절하며 최대·최소 범위 내에서 서보모터를 구동하여 시야 중심을 맞춘다. 또한 한쪽 팔만 인식될 때, 상체 없이 머리만 보일 때, 혹은 다리만 보일 때와 같이 특정 신체 부위만 검출되는 상황에서도 조건별 제어 규칙을 적용해 pan·tilt 값을 미세 조정함으로써 추적 안정성을 보완한다. 이후 검출된 중요 관절들의 (x, y, z) 정보를 리스트 형태로 저장하여 후속 낙상 감지 모듈에서 사용할 수 있도록 지속적으로 window 버퍼에 누적하며, 'q' 입력 시 시스템을 종료하고 카메라·서보모터·General Purpose Input/Output (GPIO)를 안정적으로 해제한다. 이 과정은 실시간 자세 추적을 위한 영상 처리, 조건 기반 로봇 관절(서보) 제어, 신체 부위 기반 가시성 판정 및 데이터 스트리밍을 결합한 전체 파이프라인을 구성한다.

```

1. function pose_tracking()
2.   init camera (480×360 RGB), start preview
3.   pan, tilt ← 90; set_servo(pan, tilt)
4.   start_time ← now
5.
6.   loop forever:
7.     frame ← capture_frame()
8.     if now - start_time < 0.1: continue
9.     start_time ← now
10.
11.    image_rgb ← BGR2RGB(frame)
12.    result ← pose_process(image_rgb)
13.
14.    if result has pose_landmarks:
15.      (l_sh, r_sh, l_wr, r_wr, nose, l_an, r_an) ← select key joints
16.      upper_body_detected ← (vis(l_sh) > 0.5 or vis(r_sh) > 0.5)
17.
18.      if upper_body_detected:
19.        (cx, cy) ← shoulder_center_in_pixels()
20.        (mx, my) ← image_center()
21.        if cx < mx - x_th: pan ← min(180, pan + 2)
22.        else if cx > mx + x_th: pan ← max(0, pan - 2)
23.        if cy < my - y_th: tilt ← max(0, tilt - 2)
24.        else if cy > my + y_th: tilt ← min(180, tilt + 2)
25.
26.        left_arm_only ← vis(l_wr) > 0.5 and vis(r_wr) ≤ 0.5
27.        right_arm_only ← vis(r_wr) > 0.5 and vis(l_wr) ≤ 0.5
28.        head_only ← vis(nose) > 0.5 and not upper_body_detected
29.        leg_only ← (vis(l_an) > 0.5 or vis(r_an) > 0.5) and not upper_body_det
30.      ect
31.      if left_arm_only and not upper_body_detected: pan ← min(180, pan
32.      + 3)
33.      else if right_arm_only and not upper_body_detected: pan ← max(0,
34.      pan - 3)
35.      if head_only: tilt ← min(180, tilt + 2)
36.      else if leg_only: tilt ← max(0, tilt - 2)
37.      update_servo_pwm(pan, tilt)
38.
39.      for each idx in important_landmarks:
40.        (x, y, z) ← result.landmark[idx]
41.        window.append((x, y, z))
42.      else:
43.        for each idx in important_landmarks:
44.          window.append((0, 0, 0))
45.
46.      show(frame)
47.      if key_pressed('q'): break
48.      if output == 1: set stop_event: break
49.      stop camera: stop servos
50.      close windows: GPIO_cleanup(): pose.close()
  
```

그림 4. 좌표 추출 및 객체 인식의 유사코드 (pseudo code)

낙상 감지 과정에서 데이터 전처리부터 낙상 감지, 보호자 알림, 로컬 알림을 제어하는 전체 동작 흐름을 설명한다. 먼저 5초 주기로 이전 유사코드에서 수집된 관절 좌표 데이터를 확인한다. Window에 5초 동안의 관절 좌표를 누적 시키고, 50 프레임이 확보되지 않은 경우 분석을 지연하여 안정적인 시퀀스 입력을 확보한다. 라즈베리파이의 처리 속도를 감안하여 프레임을 전부 사용하지 않고 1프레임 처리 후 2프레임을 건너뛰는 다운샘플링을 수행한다. 샘플링 된 인덱스를 기반으로 데이터프레임을 생성하고, 결측치를 보간하게 된다. 전처리 된 데이터프레임은 index는 프레임, columns는 관절 id, 값은 관절의 (x, y, z) 좌표로 피벗 변환된다. 이후 모델 입력 형식에 맞추어 x\_id, y\_id, z\_id 형태로 컬럼 명이 재정렬된다. 피벗 결과는 Pytorch Tensor로 변환되어 배치 차원을 추가한 후, 미리 학습된 낙상 탐지 모델에 입력된다. 모델 출력은 sigmoid 함수를 거쳐 확률값으로 변환되고, 0.5 초과 시 낙상 발생으로 판단한다. 낙상이 감지되면 보호자에게 문자 메시지 전송, 부저 음성 출력된다. 낙상이 감지되지 않은 경우, 현재 분석 결과를 로그 형태로 출력하고 루프를 지속한다.

```

1. function fall_detecting():
2.     output ← 0
3.
4.     loop forever:
5.         sleep 5 seconds
6.         lock.acquire()
7.         if window length < 13 × 50:
8.             lock.release()
9.             continue
10.        window_list ← copy of window
11.        lock.release()
12.
13.        index_list ← empty list
14.        j ← 1
15.        for i in range(length of window_list):
16.            append (j - 1) to index_list
17.            j ← j + 1
18.            if j mod 13 == 1:
19.                j ← j + 26
20.
21.        fill test_df rows at index_list with (x, y, z) from window_list
22.
23.        test_df2 ← group test_df by landmark_id
24.        for each group:
25.            sort by (frame, landmark_id)
26.            interpolate missing values
27.            forward-fill, backward-fill
28.            merge groups and sort again
29.
30.        df_pivot ← pivot test_df2
31.        index = frame
32.        columns = landmark_id
33.        values = (x, y, z)
34.
35.        rename columns to "x_id", "y_id", "z_id"
36.        sort by frame and reset index
37.
38.        X_tensor ← convert df_pivot to float tensor
39.        X_tensor ← add batch dimension
40.        model_output ← sigmoid( fall_detect_model(X_tensor) )
41.        output ← (model_output > 0.5)
42.
43.        if output == 1:
44.            message ← "fall detected!!!"
45.            send SMS via send_many(message)
46.            print "낙상 감지됨!"
47.            set stop_event
48.            break loop
49.        else:
50.            print "낙상 없음", output value
51.
52. def alarm():
53.     while not stop_event.is_set():
54.         if output == 1:
55.             subprocess.run(["aplay", "-D", "plughw:3,0", "fall_detected.wav"])
56.             for _ in range(5):
57.                 buzzer_pwm.start(50)
58.                 time.sleep(0.2)
59.                 buzzer_pwm.stop()
60.                 time.sleep(0.2)
61.             stop_event.set()
62.             break
63.         time.sleep(0.1)

```

그림 5. 낙상 감지 및 알림 기능의 유사코드 (pseudo code)

## 4. 기능 구현

본 연구에서 구현한 낙상 감지 시스템의 하드웨어 구성은 라즈베리파이(Raspberry Pi 4 Model B, 4GB RAM)를 중심으로 설계되었다. 라즈베리파이는 저전력·고효율의 임베디드 보드로서, 다양한 센서와 모듈을 통합 제어할 수 있는 GPIO 포트를 제공하며, Python 기반의 딥러닝 모델 실행이 가능하다는 점에서 본 연구의 요구사항에 적합하다. 저비용으로 낙상 감지 시스템을 개발한다는 연구의 취지에 맞게 사용한 플랫폼은 다음과 같다.

### 4.1 하드웨어 및 소프트웨어 플랫폼

시스템의 주요 하드웨어 구성 요소는 카메라 모듈, 서보모터, 앰프, 스피커, 부저 등으로 구성되어 있다. Pi Camera V3 모듈은 사람의 움직임을 실시간

으로 촬영하여 영상 데이터를 제공하며, SG90 서보모터는 카메라의 상하 및 좌우 회전을 제어하여 객체가 카메라 시야에서 벗어나지 않도록 추적한다. 또한, MAX98357A I2S 앰프와 8Ω 1W 스피커는 낙상 상황 발생 시 음성 알림을 출력하는 역할을 수행한다.

각 구성 요소는 라즈베리파이의 GPIO 핀을 통해 제어된다. SG90 서보모터는 GPIO13(좌우) 및 GPIO23(상하) 핀에 연결되어 Pulse Width Modulation (PWM) 신호를 받아 카메라 각도를 제어하며, 앰프 모듈은 GPIO18, GPIO19, GPIO21 핀을 이용하여 DIN, BCLK, LRCLK 신호를 전송받아 스피커로 음성을 출력한다. 낙상 발생 시 GPIO에 연결된 부저가 작동하여 즉각적인 경고음을 발생시키도록 설계되었다.

하드웨어 구성은 브레드보드 기반의 통합 회로를 통해 구성되었으며, 전류 안정화를 위해 저항 및 커패시터를 삽입하였다. 전체 하드웨어는 약 14만 원 내외의 비용으로 구축 가능하며, 모듈화를 통해 유지보수가 용이하고, 저비용으로도 충분한 실시간 성능을 확보할 수 있다. 또한, 각 모듈은 독립적으로 동작 가능하도록 구성되어 있으며, Thread 기반 병렬 처리 구조를 적용하여 실시간성과 응답성을 향상시켰다.

플랫폼	주요 내용
하드웨어 플랫폼	rasberry pi4 model B 4gb, rasberry pi camera module v3 8mp, max 98357a amp, Intel(R) Core(TM) i5-14400F(2.50 GHz), NVIDIA GeForce RTX 4060
소프트웨어 플랫폼, 웹웨어	Microsoft Windows 11 Debian GNU/Linux 12
응용 프로그램 플랫폼	Python 3.9.21, pandas 2.2.3, numpy 1.26.4, matplotlib 3.9.4, opencv-python 4.11.0.86, scikit-learn 1.3.2, mediapipe 0.10.21, torch 2.4.1+cu124, torchmetrics 1.5.2, Microsoft Visual Studio Code 1.106.2

### 4.2 낙상 감지 시스템 구현

본 논문에서 제안하는 낙상 감지 시스템은 카메라를 통해 입력된 영상을 전처리하여 낙상 여부를 판단하는 낙상 감지 모델, 낙상 발생 시 부저에서 경보음을 발생시키는 로컬 알림 시스템, 카메라에 입력된 객체의 위치에 따라 카메라 방향을 조정하는 객체 추적 시스템으로 크게 3가지의 기능을 가지고 있다. 이에 대한 상세한 설명은 다음과 같다.

#### 4.2.1 낙상 감지 모델

낙상 감지 모델은 Mediapipe Pose를 이용하여 영상으로부터 사람의 관절 좌표를 추출하고, 이를 GRU 기반의 딥러닝 모델에 입력하여 낙상 여부를 분류한다. 그리고 모델 학습 과정에서는 관절 좌표의 길이를 보정하기 위해 보간(Interpolation) 과정을 수행하였으며, 시간적 시퀀스 데이터를 반영하기 위해 RNN 구조를 활용하였다.

테스트 데이터 약 600개를 대상으로 검증한 결과, 본 모델은 F1 Score 0.976의 정확도를 달성하였으며, 낙상 발생 후 5초 이내의 감지 반응 시간을 확보하였다.

#### 4.2.2 로컬 알림 시스템

낙상 감지 이벤트 발생 시, 시스템은 Text-to-Speech(TTS) 모듈을 이용하여 “낙상 감지되었습니다.”라는 음성 파일을 생성한다. 생성된 .wav 파일은 Inter-IC Sound(I2S) 프로토콜을 통해 MAX98357A 앰프로 전송되어 스피커로 출력되며, 동시에 부저가 작동하여 경고음을 발생시킨다.

이 과정은 Python의 threading 모듈을 활용한 병렬 구조로 구현되어, 감지 이후 평균 1초 이내에 알림이 출력된다. 이러한 병렬 처리 구조는 실시간성을 보장하며, 하드웨어 간 신호 지연을 최소화한다.

#### 4.2.3 객체 추적 시스템

객체 추적 시스템은 OpenCV와 Mediapipe Pose Tracking을 이용하여 사용자의 이동을 실시간으로 추적한다. 카메라 영상 내에서 관절 좌표의 중심값을 계산하고, 객체가 프레임 중심에서 벗어날 경우 SG90 서보모터를 제어하여 카메라의 각도를 조정한다. 이를 통해 카메라는 사용자의 움직임을 지속적으로 따라가며, 낙상 인식의 사각지대를 최소화한다.

### 5. 성능 측정 및 분석

낙상 감지 모델별 성능 비교를 위해 GRU 및 2 layers Multi-layer Perceptron (MLP) 모델의 학습과 검증을 거쳐 성능 측정을 진행하였다. 분석 방법과 결과는 다음과 같다.

#### 5.1 낙상 감지 모델의 성능 측정 및 분석

각 모델들은 100 Epoch의 학습을 진행하였고, 매 Epoch마다 F1 Score를 계산하여 낙상 감지 정확도를 측정하였다. 두 모델 다 70 Epoch 부근에서 수렴하여 성능이 최고치를 보였다. 검증은 학습 데이터의 20%, 600개의 낙상/비낙상 영상 데이터를 사용했다. GRU 모델은 Sequence 데이터에 강점을 보이는 RNN 계열의 모델이니만큼 첫 Epoch부터 높은 성능을 보여주었다. 이후 꾸준히 성능이 상승하여 70 Epoch에서 F1 Score 0.976을 달성하였다. 반면, MLP 모델은 학습 초반 단계에서는 낮은 성능을 보였다.

나, 빠르게 성능이 향상되는 모습을 보였다. 최종 성능은 65 Epoch에서 F1 Score 0.864를 달성하여 GRU 모델보다는 좋지 않은 성능을 보였다. 2 layers MLP라는 얇은 모델의 한계 때문일 수 있지만, 모델의 특성상 구조를 바꿔더라도 RNN 계열 모델의 성능보다는 떨어질 것으로 예상된다.

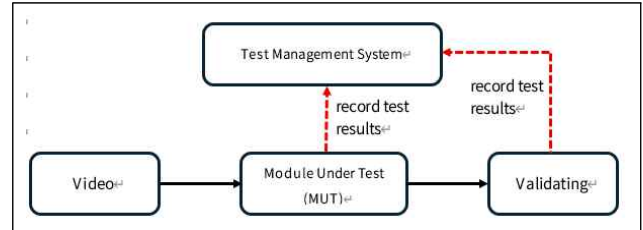


그림 6. 성능 측정 기능 구성도

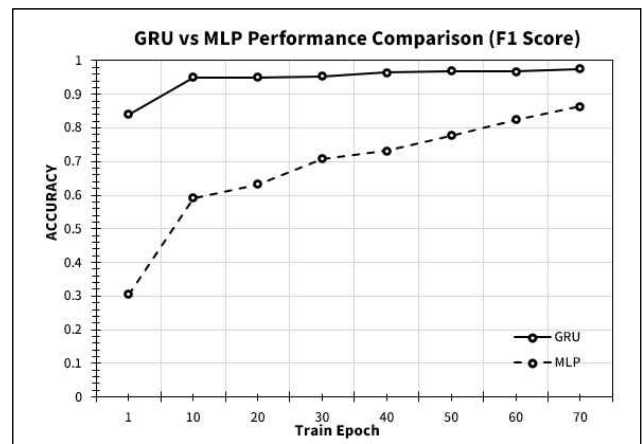


그림 7. 성능 측정 결과 분석 (꺾은선 그래프)

#### 5.2 낙상 감지 모델의 추론 속도 측정 및 분석

라즈베리파이에서 낙상 감지 모델 추론을 진행하기에는 사양이 낮아 추론 속도가 느리다. 따라서, 엣지 디바이스에 AI 모델을 사용하기 위해서는 Quantization(양자화) 또는 TorchScript 변환이 필수적이다. 본 논문에서는 낙상 감지 기능의 실시간성을 확보하기 위해 TorchScript 변환을 사용했다. 일반 모델과 TorchScript 모델의 추론 속도를 비교하기 위하여 일반 데스크탑의 CPU(Intel(R) Core(TM) i5-14400F(2.50 GHz))로 동일한 환경에서 성능을 비교했다. 그래프에서 나타나듯 Normal 모델과 TorchScript 모델의 추론 속도는 약 2배 정도의 차이를 보이고 있다. 이는 CPU의 성능이 낮은 라즈베리파이에서는 더욱 큰 차이로 벌어질 것으로 예상된다.

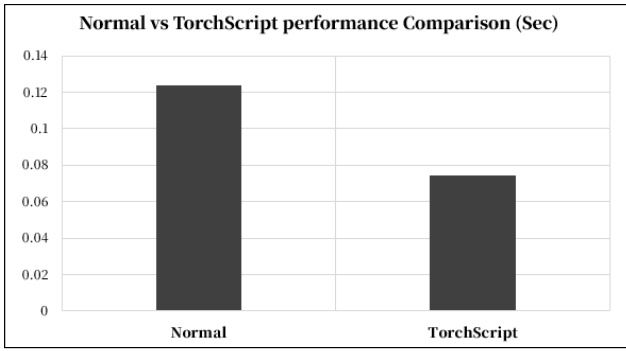


그림 8. 성능 측정 결과 분석 (막대 그래프)

## 6. 결 론

기존의 낙상 감지 시스템은 RADAR, LIDAR 센서를 장착해 높은 정확도를 보여주지만, 대부분 제품들이 B2B 서비스 제공과 높은 가격으로 인해 1인가구 및 독거노인 등 개인이 접근하기 어렵다는 한계가 있다. 이에 본 논문에서는, MediaPipe 모듈을 사용하여 주요 관절의 스켈레톤 시퀀스 데이터를 추출하고 이를 기반으로 GRU 모델을 통해 낙상 여부를 판단하는 GRU 기반 낙상 감지 시스템을 개발하였다. Skeleton 시퀀스 데이터를 활용함으로써 센서 데이터를 대체할 수 있으며, 서보모터, 부저, 라즈베리파이 등 최소화된 장치로 필수적인 기능만 구현하여 하드웨어의 모듈이 간소화되어 비용 또한 줄어들게 된다. 낙상 감지 모델별 성능 비교를 위해 GRU 및 2 layers MLP 모델의 학습과 검증을 거쳐 성능 측정을 진행하였다. 각 모델들은 100 Epoch의 학습을 진행하였고, 매 Epoch 마다 F1 Score를 계산하여 낙상 감지 정확도를 측정하였다. 두 모델 다 70 Epoch 부근에서 수렴하여 성능이 최고치를 나타냈으며, 검증은 학습 데이터의 20%, 600개의 낙상/비낙상 영상 데이터를 사용했다. GRU 모델은 첫 Epoch부터 높은 성능을 보여주면서 이후 꾸준히 성능이 상승하여 70 Epoch에서 F1 Score 0.976을 달성하였다. 반면, MLP 모델의 최종 성능은 65 Epoch에서 F1 Score 0.864를 달성하여 GRU 모델보다는 좋지 않은 성능을 보였다. 또한, 낙상 감지 기능의 실시간성을 확보하기 위해 TorchScript 변환을 사용했다. 일반 모델과 TorchScript 모델의 추론 속도를 비교하기 위하여 일반 데스크탑의 CPU(Intel(R) Core(TM) i5-14400F(2.50 GHz))로 동일한 환경에서 성능을 비교했다. Normal 모델과 TorchScript 모델의 추론 속도는 각각 0.12, 0.07 정도의 값으로 약 2배 정도의 차이를 보였다. 이는 라즈베리파이와 같은 CPU의 성능이 낮은 임베디드 시스템에서는 큰 차이로 벌어질 것으로 예상된다. 이를 통해, GRU 기반 낙상 감지 시스템이 고가 센서 기반 시스템을 대체할 수 있는 비용 효율적이고 경량화된 낙상 감지 기술의 대표적인 예시로 보여질 수 있다고 생각한다. 향후 연구에서는 라즈베리파이의 CPU 성능 한계로 인해 발생한 낮은 프레임율과 반응 속도 문제를 개선하기 위해 모델 경량화, 데이터 처리 최적화를 수행하여, 더 안정적이고

실사용에 적합한 시스템으로 확장할 계획이다.

## 참고문헌

- [1] 질병관리청, 건강정보 - 추락/낙상, 2020.03.23
- [2] 안지호, &고령 1인 가구, 안전사고 목숨까지 위협&, 백세인생, 2023.04.26
- [3] Jin-Il Kim, Development of a Fall Detection System Based on Hospital Inpatients Data Using Radar, Journal of Knowledge Information Technology and System, Vol. 17, No. 2, pp. 375-384, April 2022
- [4] Miguel Piñero, David Araya, David Ruete, Carla Taramasco, Low-Cost LIDAR-Based Monitoring System for Fall Detection, IEEE Access, Vol. 12, pp. 72051-72061, May 2024