2022학년도 학부생 연구프로그램(UGRP)

최종보고서

연구책임자(학생)	하상범	소속/학번	전기전자공학과
과제명	**MI Kit를 이용한 운동 자세 교정 어플리케이션 개발**		
연구계획 대비 완성 정도: 100% / 100%			

1. 연구목적

- 스마트폰 카메라를 이용해 실시간으로 사용자 행동을 분석하고, 그에 따른 피드백을 제공하는 운동 자세 교정 어플리케이션(이하 POSFIT)을 개발하고자 한다. 미리 녹화된 강의를 제공하는 기존의 홈 트레이닝 어플리케이션들과는 달리 POSFIT은 실시간 피드백을 가능하게 해 사용자들에게 양질의 운동 환경 제공과 운동 효율 증가를 목표로 한다.

- 홈 트레이닝은 시공간적 비용을 아낄 수 있다는 장점을 지녀 여러 사람들의 관심을 받고 있다. 2019년 15조 원 규모였던 글로벌 홈 트레이닝 시장이 코로나 19사태로 2021년 21조 원까지 성장했으며, 오는 2026년 36조 원 규모까지 성장이 예상되는 만큼 앞으로가 기대되는 유망한 시장이다.[1)]

- 현재 홈 트레이닝 시장이 급격히 성장했지만, 참여자들의 운동에 대한 미숙한 이해로 인해 여러 문제가 발생한다. 홈 트레이닝은 운동 진행 과정을 스스로 판단하며 진행해야 하기 때문에 부정확한 자세를 유발할 가능성이 높다. 이로 인해 운동의 효과가 미미하거나, 척추나 관절에 무리를 줘 부상을 입기도 한다. 단순히 영상을 따라하는 방법으로는 자신의 동작이 정확한지 확실하게 알 수 없는 만큼, 실시간 분석 및 피드백을 통한 운동 자세 교정 프로그램을 개발해 앞서 언급한 문제점을 해결하고자 한다.

2. 연구 진행 과정

월	연구 진행 과정
4	연구 주제 선정 Dart 언어 학습
5	Pose detection model 조사(AI model 조사)
6	어플리케이션에서 구현하고자 할 운동 결정 런지, 스쿼트, 사이드 플랭크에 대한 공부 및 정확한 자세 각도 조사

7	어플리케이션 개발 시작
10	어플리케이션 내 자세 교정 피드백 제공 방법 논의 Pose Detection 기능을 이용해 사용자가 운동을 정확하게 하고 있는지에 대한 시각적 전달 방법 논의 TTS(Text To Speech) 음성 기능을 이용한 청각적 전달 방법 연구
11	최적화 관련 문제점 논의 개선방향 탐색
12	운동 촉진 방법, 어플리케이션 내 이미지와 문구 논의 로고, 캐릭터, 어플리케이션 내부 디자인
1	최종 보고서 작성, 영상 및 포스터 제작

3. 연구 내용

3.1 이론적 배경

Pose Detection

- ML Kit 내 Pose Detection과 Selfie Segmentation API를 이용해 사진과 라이브 영상의 골격 구조를 인식한다. Live Camera Detection 동안 카메라로부터 이미지를 전달받는 과정과 Detection 과정이 동시에 진행된다.

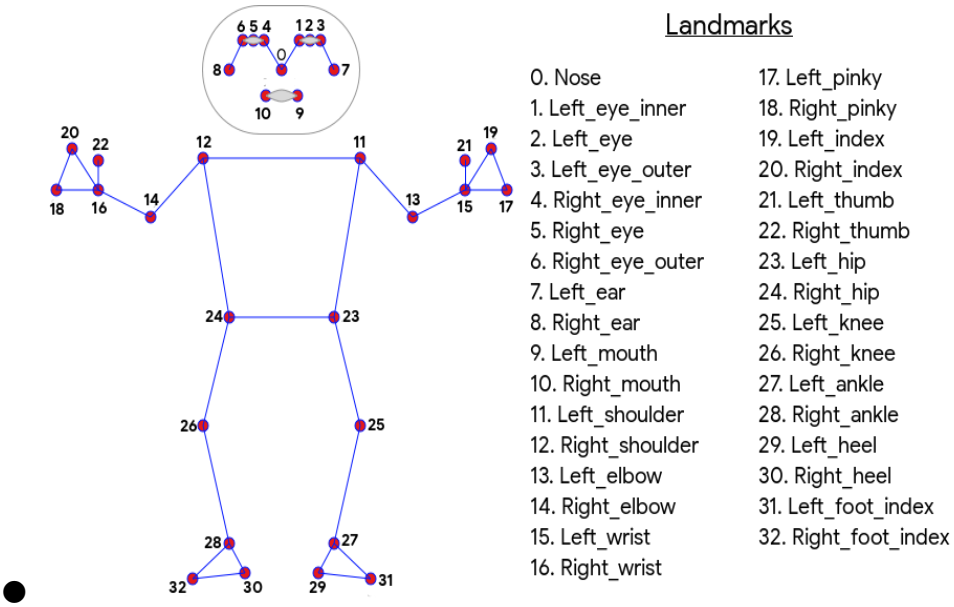


그림1 detectImagePose 함수

- 총 33개의 랜드마크를 설정하여 33개의 랜드마크의 카메라 화면 상 좌표를 얻는다.

Single image detection

- 단일 이미지를 활용해 사용자의 골격 구조를 인식한다. 플러그인에서 PNG Encoding 이미지를 받는데, 이는 Flutter 이미지 위젯을 PNG 형식으로 변환하는 기능을 가졌기에 가능하다. 구글 ML Kit의 API(Application Programming Interface)가 플러그인에서 받은 이미지에 대해 작업을 수행해 단일 이미지의 골격구조를 파악한 뒤 그 결과를 Flutter로 전달한다.

```
void detectImagePose(Image source) async {
  PngImage? pngImage = await source.toPngImage();
  if (pngImage == null) return;
  final pose = await BodyDetection.detectPose(image: pngImage);
  ...
}
```

그림2 detectImagePose 함수

- BodyDetection.detectPose에서 ML Kit에서 분석된 정보를 포함하는 Pose 객체를 반환하고, 이를 CustomPainter 클래스에서 그려낸다. 인식한 골격 구조는 폭과 길이의 곱으로 반환되며, 특정 픽셀이 골격 구조로 인식되는 한 부분을 표시한다.

Camera Feed Detection

- Single image detection과 함께 카메라에서 실시간으로 골격 구조를 인식을 지원한다. 카메라가 이미지를 인식하는 과정은 모두 native side에서 실행되는 만큼, 데이터 직렬화(Serialization)를 신경 쓸 필요가 없다. 카메라 시작을 위해 카메라 프레임의 탐지 결과가 준비됐을 때 콜백(Callback)을 전달하는 동안 비동기함수를 동기적으로 작동하게 만드는 await을 이용해 다음과 같이 사용한다.

```
Future<void> _startCameraStream() async {
  final request = await Permission.camera.request();
  if (request.isGranted) {
    await BodyDetection.startCameraStream(
      onFrameAvailable: _handleCameraImage,
      onPoseAvailable: (pose) {
        if (!_isDetectingPose) return;
        _handlePose(pose);},);}}
```

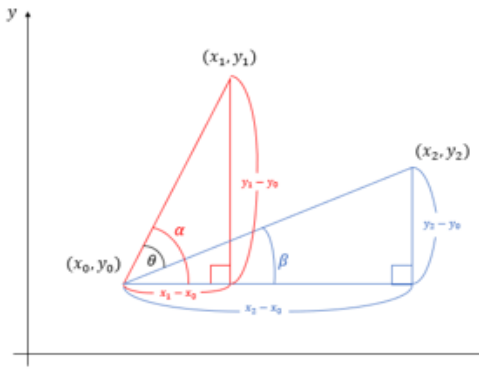
그림3 _startCameraStream 함수

- BodyDetection.startCameraStream 내의 onFrameAvailable 매개변수를 통해서는 실시간 카메라 이미지를 생성하고, onPoseAvailable 매개변수로 detectedPose를 생성한다. Flutter 내의 setState() 함수를 통해 프레임당 실시간으로 변하는 카메라 이미지에 Pose detection을 가능하게 해준다

Firebase

- Firebase는 유저 정보 저장 등 Backend에 필요한 서비스를 제공한다. 사용자가 직접 Backend에 코드를 작성하지 않아도 되며, 정보 저장을 위한 데이터 서버가 따로 필요없다는 장점이 있다. Backend의 데이터를 변경하면 Firebase가 실시간으로 데이터를 동기화한다. 클라우드와 충돌 방지를 위해 오프라인 상태의 데이터를 Local에 저장해뒀다가 온라인 연결 시 데이터를 자동으로 병합한다.
- Firebase Core는 Flutter App과 Firebase Project를 연결한다. Firebase Auth는 유저를 인증하기 위한 Backend Services, SDKs, UI library를 제공하고, △비밀번호 △전화번호 △federated identity 인증 방식이 지원된다. 어플리케이션에 사용자가 로그인을 시도하면 인증 자격증명을 받아 SDK의 Backend Service에서 확인 받은 후, 응답 결과를 돌려보낸다.
- Firebase Messaging은 유저들에게 메시지를 보내거나 받을 수 있는 기능을 제공하며, 유저로부터 정보들을 받아올 수 있다. Firebase Database는 클라우드 호스팅 데이터베이스로 HTTP 요청 대신 데이터 동기화를 사용해 데이터의 변경마다 연결된 기기들이 업데이트를 수신한다.

각도 계산 함수 원리



$$\tan\alpha = \frac{y_1 - y_0}{x_1 - x_0}$$

$$\tan\beta = \frac{y_2 - y_0}{x_2 - x_0}$$

$$\theta = \alpha - \beta$$
$$= \tan^{-1}\left(\frac{y_1 - y_0}{x_1 - x_0}\right) - \tan^{-1}\left(\frac{y_2 - y_0}{x_2 - x_0}\right)$$

그림4 각도 계산 함수 원리

3.2 앱 개발 내용

3.2.1 Wire Frame

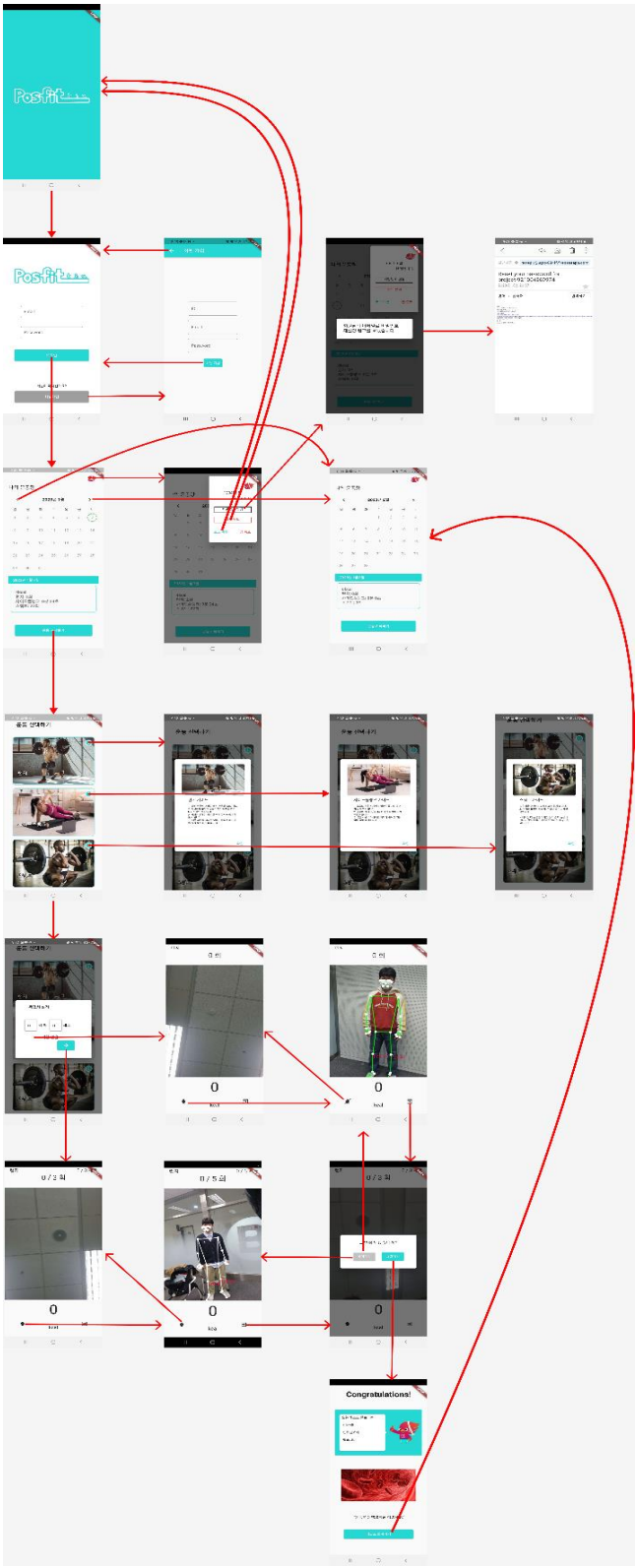


그림5 Wire Frame(페이지 연결 도식화)

3.2.2. LoginPage

1) 클래스 설명

class _LoginPage extends State<LoginPage>	설명
_databaseURL	데이터베이스 주소를 저장하는 변수
_tryValidation()	아이디, 비밀번호 입력이 유효한지 확인 및 저장하는 함수
_idTextController	사용자 아이디를 입력받는 Controller
_pwTextController	사용자 비밀번호를 입력받는 Controller
userEmail, userPW	TextformField 에서 이메일과 비밀번호를 할당하는 변수
_formKey	2개 이상의 TextFormField 상태를 관리하는 변수

2) 위젯트리 & 페이지 사진

그림6 LoginPage 어플리케이션 구현

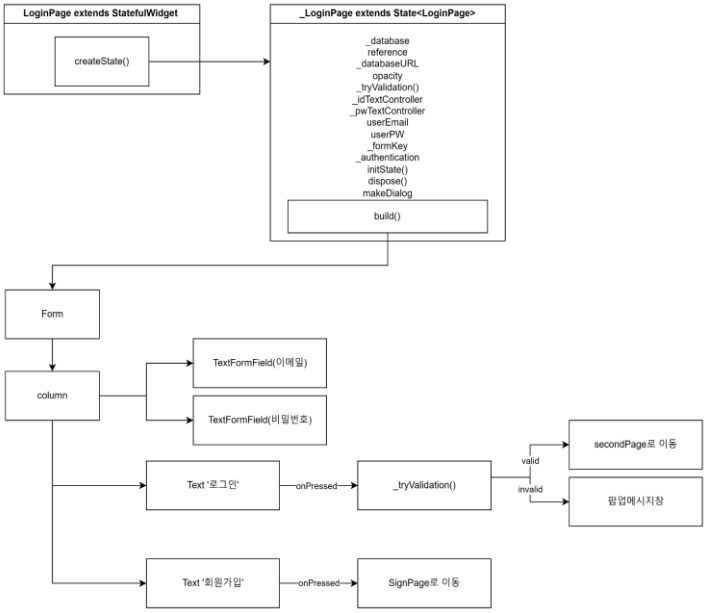


그림7 LoginPage Widget Tree

3) Backend 설명

TextEditingController를 사용해 TextFormField에 이메일과 비밀번호를 각각 입력받는다. 정보가 valid하면 캘린더가 있는 MainPage로 이동하고, invalid하면 팝업메시지 창을 출력하며, 이후 재입력을 해야한다. 회원가입 버튼을 누를 경우 SignUpPage로 이동한다.

3.2.3. SignUpPage

1) 클래스 설명

class _SignPage extends State<SignPage>	설명
_databaseURL	데이터베이스 주소를 저장하는 변수
userID, userEmail, userPW	TextformField 에서 user정보를 할당해주는 변수
_formKey	2개 이상의 TextFormField 의 상태 관리에 관여하는 변수
_tryValidation()	_formKey가 valid하면 true를 리턴하는 함수
makeDialog	AlertDialog를 사용해 하는 팝업메시지를 띄우는 함수

2) 해당 페이지 사진 & 위젯트리

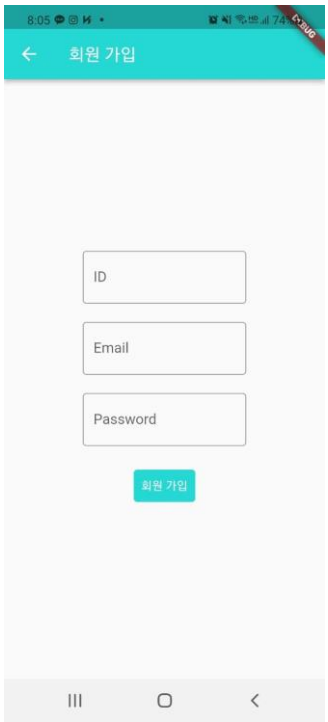


그림8 SignUpPage 어플리케이션 구현

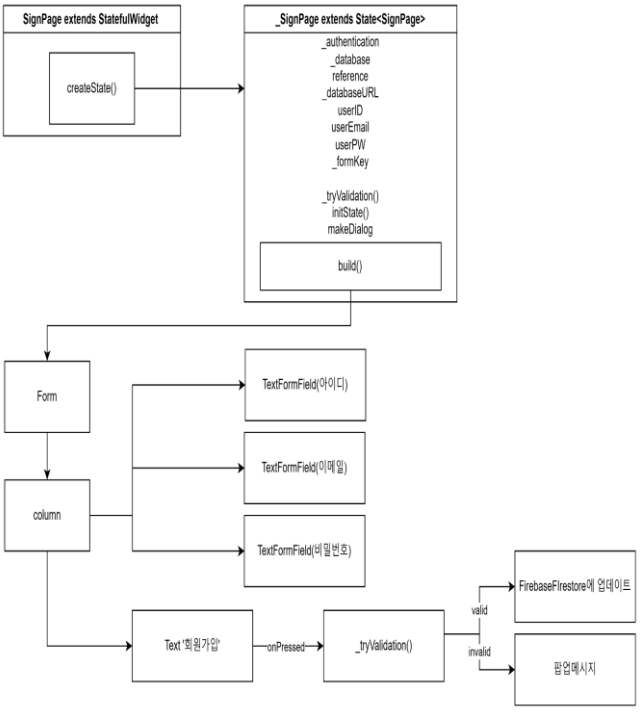


그림9 SignUpPage Widget Tree

3) Backend 기능

TextEditingController를 사용해 TextFormField에 아이디, 이메일, 비밀번호를 입력받는다. 정보 입력 후 회원가입 TextButton을 누를 때, valid하면 FirebaseFirestore에 정보가 업데이트 되고 invalid하면 팝업메시지 창이 출력되며, 이후 재입력을 해야한다.

3.2.4. MainPage

1) 클래스

class MainPage extends StatefulWidget	설명
kcal	소모한 칼로리 양을 인자로 받는 매개변수
number	총 운동 횟수(또는 운동 시간)를 인자로 받는 매개변수
index	운동 종류를 인자로 받는 매개변수
MainPage	this.kcal, this.number, this.index를 변수로 갖는 생성자

class _MainPage extends State<MainPage>	설명
loggedUser	currentUser 정보를 받는 변수
username	userID를 저장하는 변수
lastkcal lastlunge/plank/squat	{총 소모칼로리 / 운동당 횟수(또는 시간)} 값을 저장하는 매개변수
initState()	getCurrentUser, getandsetdata, getusername 실행 함수
getusername()	FirebaseFirestore로부터 username을 받는 함수
getandsetdata()	금일에 해당하는 각 운동과 칼로리 정보를 firebase로부터 받아 업데이트 한 후 서버에 업로드하는 함수
getCurrentUser()	currentUser를 loggedUser에 할당하는 함수
getInfor	FirebaseFirestore로부터 user의 해당날짜 운동정보를 가져와주는 함수이며, 존재하지 않으면 0을 반환한다.

2) 해당 페이지 사진 & 위젯트리

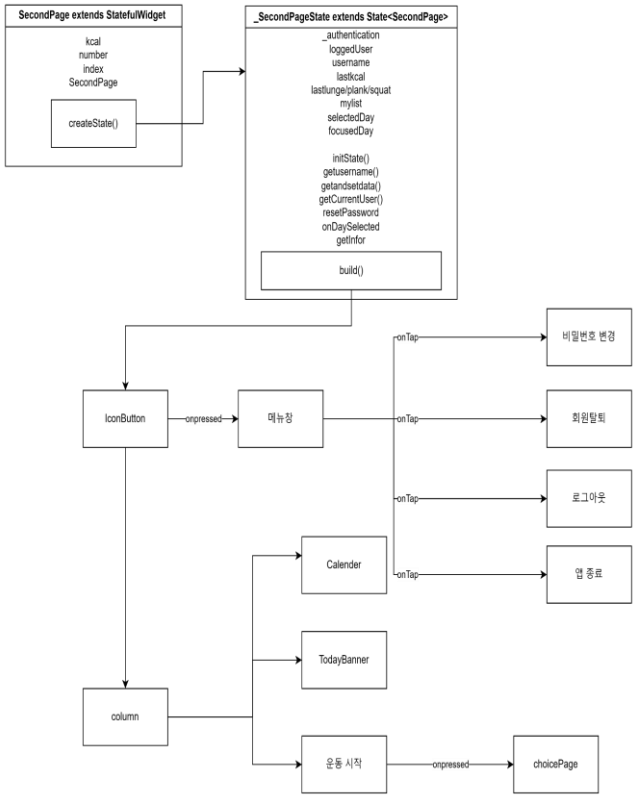


그림10 SignUpPage 어플리케이션 구현 그림11 SignUpPage Widget Tree

3) Backend 기능

캘린더와 이에 해당하는 날짜가 표시된다. 당일 운동에 대한 소모 칼로리, 운동 당 소요시간과 횟수가 나타난다. "운동 시작하기" 버튼을 누르면 ChoicePage를 호출한다. 오른쪽 상단의 캐릭터 버튼을 누르면 비밀번호 변경, 회원탈퇴, 로그아웃, 앱종료 버튼이 있는 메뉴창이 뜬다.

3.2.5 ChoicePage (운동선택하는 부분)

1) 클래스

class _ChoicePageState extends State<ChoicePage>	설명
_targetNumber	세트당 (횟수/시간) 값을 입력받는 변수
_targetSet	목표 세트 수를 입력받는 변수
_targetLimit	제한없음 체크 유무를 표현하는 변수

2) 해당 페이지 사진 & 위젯트리

그림12 ChoicePage 어플리케이션 구현

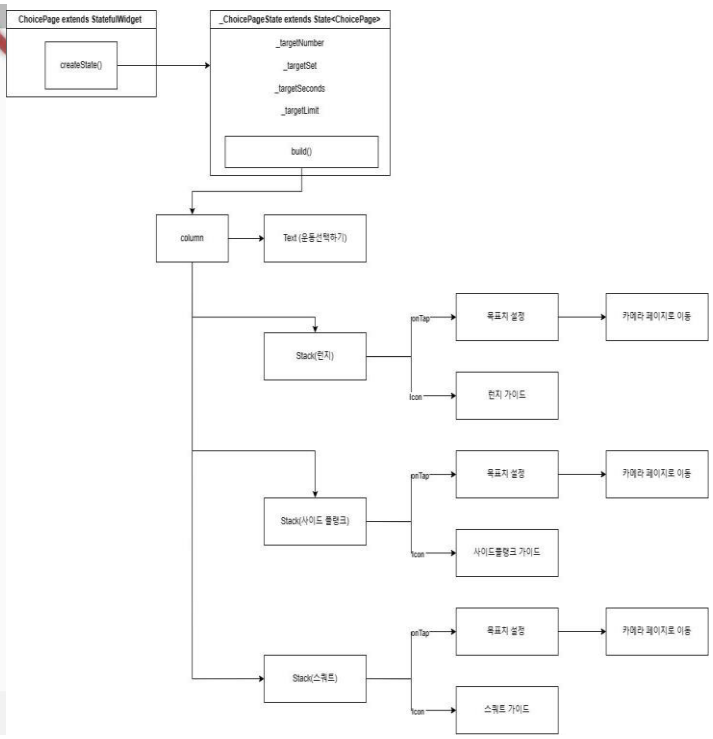


그림13 ChoicePage Widget Tree

3) Backend 기능

각각의 운동 중 하나를 선택해 목표치를 설정하거나 제한없음을 선택 시 CameraPage로 이동한다. 목표치 설정에서 입력받은 _targetNumber , _targetSet을 CameraPage로 넘겨주기 위해, Camera 클래스의 인자로 입력받도록 구현했다. 또한, 어떤 운동이 선택됐는지 알려주기 위해 index 인자에 숫자를 입력한다.

Index	**0**	**1**	**2**
운동	**런지**	**사이드플랭크**	**스쿼트**

세트와 횟수를 입력하는 경우, _targetLimit == true을 만들어 인자를 넘겨준다. 제한없음 선택 하는 경우 기존의 방식 대신, _targetNumber = -1 , _targetSet = 1이라는 특수한 값을 지정 후 인자를 넘겨줌으로써 제한없음을 인식해 필요한 동작을 실행하도록 구현했다.

3.2.6 CameraPage(메인)

1) 클래스

class CameraPage extends StatefulWidget	설명
targetnumber	ChoicePage에서 목표 세트당 운동 횟수(or 시간)를 받아오는 매개변수
targetset	ChoicePage에서 목표 세트 수를 받아오는 매개변수
index	ChoicePage에서 운동 종류를 받아오는 매개변수

class _CameraPageState extends State<CameraPage>	설명
_isDetectingPose	Pose detection의 turn on/off 여부를 확인하는 변수
stopwatch	운동시간을 측정하는 변수
_detectedPose	detected된 PoseLandmarkType 리스트와 connection 리스트를 멤버 변수로 갖는 객체
_cameraImage	카메라 이미지를 만들어내는 객체
_imageSize	카메라 이미지 사이즈 객체
mykcal	소모한 칼로리 양을 나타내는 int 변수
curnum	현재 운동 횟수를 나타내는 int 변수
curset	현재 운동한 세트 수를 나타내는 int 변수
isFinished	목표치 달성 후 이어하기 실행을 위한 변수
stopwatchStart()	운동 시간 start 또는 stop 상태 해제 함수
stopwatchStop()	운동시간을 stop 하는 함수
_incrementCounter()	운동횟수인 curnum을 1씩 증가시키는 함수
_startCameraStream()	_handleCameraImage() 와 _handlePose() 를 통해 카메라 이미지와 PoseDetection 을 만들어내는 함수

_handleCameraImage()	**setState()를 통해 카메라 이미지를 실시간으로 업데이트 하는 함수**
_handlePose()	**setState()를 통해 detectedPose를 실시간으로 업데이트 하는 함수**
_toggleDetectPose()	**Pose Detection의 on/off 를 조절하는 함수**
setNumber()	**현재 운동 횟수(또는 시간)를 문자열로 반환하는 함수**

2) 해당 페이지 사진 & 위젯트리

그림14 CameraPage 어플리케이션 구현

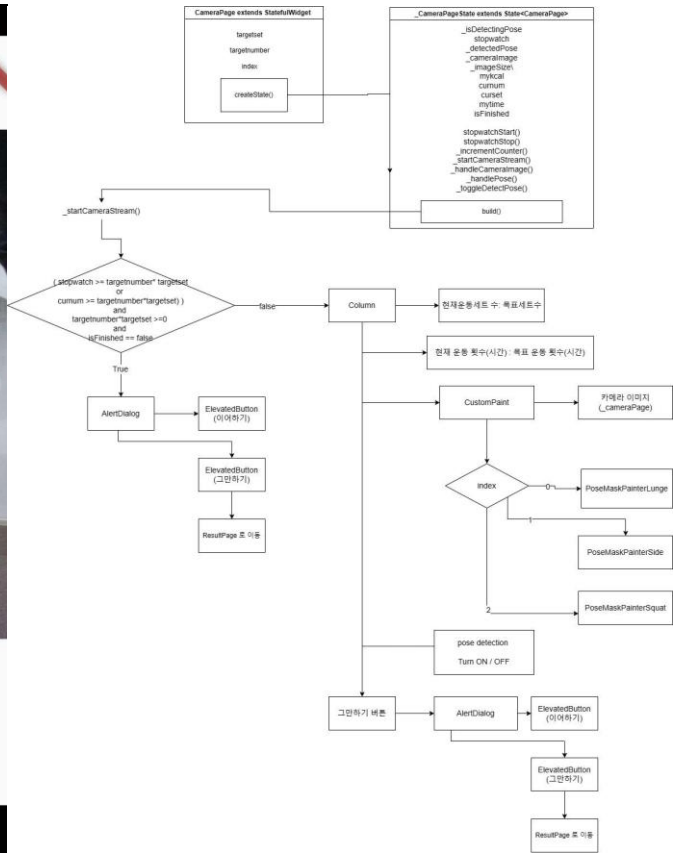


그림15 CameraPage Widget Tree

3) BackEnd 기능

- 카메라 이미지가 나오는 위젯에서 detected 된 Pose 의 line들이 그 위에 그려져야 하므로 CustomPaint 위젯을 이용했다. CustomPaint에서, 시각화되는 _cameraImage에는 _startCameraStream()의 _handleCameraImage를 통해 변경되는 값이 실시간으로 할당된다.
- 카메라 이미지 위에 그려지는 CustomPainter 객체는 삼항연산자를 통해, 각각의 운동에 따라 PoseMaskPainterLunge, PoseMaskPainterSide, PoseMaskPainterSquat 객체를 호출한다.

<PoseMaskPainterLunge>

class PoseMaskPainterLunge extends CustomPainter	설명
flutterTTS	사운드를 출력하는 TTS 객체
notifyParent	PoseMaskPainterLunge 객체가 호출될 때, 함수 형태의 인자를 전달 받는 매개변수다. 객체를 호출하는 CameraPage 내의 특정 함수(_incrementCounter)를 Function() 형 의 매개변수(notifyParent)로 받아서 객체 내부에서 사용할 수 있도록 한다.
pose	PoseMaskPainterLunge객체가 호출될 때, Pose 형의 인자(_detectedPose)를 전달 받는 매개변수
imageSize	PoseMaskPainterLunge객체가 호출될 때, Size 형의 인자(_imageSize)를 전달 받는 매개변수
pointPaint	detected된 PoseLandmark의 point 들을 색칠할 때 사용되는 Paint함수
linePaint	detected된 PoseLandmark의 line의 색표현을 위해 사용되는 Paint함수
paintRed	detected된 자세가 런지 가이드에서 잘못된 경우 line을 빨간색으로 표시하는 paint 함수
paint()	_paintPose 함수를 호출하는 함수, 상속받은 CustomPainter의 paint()를 overriding한다.
shouldRepaint()	pose, imageSize가 기존에서 변경되어 paint를 다시 해야 할 때를 알려주는 함수로 bool값을 반환한다.
inUpPosition()	Up position일 때 운동 횟수를 증가시키고, Flutter TTS로 운동 상황을 설명하는 자세 인식 함수
inDownPosition()	Down position일 때, TTS를 통해 운동 상황을 설명하는 자세 인식 함수
calculateAngle()	세 point의 사이각을 구하는 함수
angles	런지 자세를 올바르게 여부를 결정을 위해 필요한

	PoseLandmark 리스트를 구성하는 변수
connections	**모든 PoseLandmark가 연결되는 point끼리 묶어 나열한 리스트**
_paintPose()	**detected된 PoseLandmark의 point와 line을 색칠하고, inUpPosition() 과 inDownPosition() 을 호출하는 함수**

- 런지나 스쿼트는 운동을 할 때 마다, 그 횟수(curnum)가 1씩 증가하고, 이를 camerapage에서 시각화해야 한다. 이때 CameraPage에서 정의된 curnum 변수를 PoseMaskPainter 객체에서 바로 사용할 수 없는 문제가 발생한다.
- 이를 해결하기 위해 curnum을 1씩 증가시키는 _incrementCounter 함수를 정의해 PoseMaskPainterLunge, PoseMaskPainterSquat 의 매개변수로 받아서 사용했다.
- PoseMaskPainterSquat은 PoseMaskPainterLunge와 거의 동일하며, 스쿼트 자세 교정을 위해 inUpPosition(), inDownPosition() 함수를 사용했다.

<PoseMaskPainterSide>

- 사이드 플랭크는 다른 운동과는 달리, 자세 유지 시간을 측정해야 한다. checkAngle() 함수를 이용해 자세가 correct한 경우 시간을 흐르게 하고, uncorrect한 경우 시간을 멈추도록 구현했다.
- CameraPage에서 선언한 stopwatch 변수는 그 내부에서 호출된 PoseMaskPainterSide 객체 내부에서 바로 사용할 수 없다. 이를 해결하기 위해 시간을 흐르게 하는 함수인 stopwatchStart와 시간을 멈추게 하는 함수인 stopwatchStop를 PoseMaskPainterSide의 매개변수로 받아 객체 내부에서 사용할 수 있게 함으로써 문제를 해결했다.
- 목표치를 달성했거나 나가기 버튼을 누른 경우, AlertDialog 위젯에 의해 운동을 이어서 할지 여부를 선택할 수 있다. 그만하기를 선택한다면 ResultPage를 호출하고, 소모한 칼로리양(kcal) , 운동 종류(index), 운동 횟수(또는 운동 시간)(number)를 인자로 전달해준다.
- CameraPage 내부에는 직접적으로 kcal를 나타내는 변수를 선언하지 않고, 인자를 전달해주는 과정에서 삼항연산자를 이용해 운동 종류에 따른 kcal을 계산했다.

```
MaterialPageRoute(
    builder: (context) => ResultPage(
        kcal: widget.index == 0
            ? curnum ~/ 4
            : widget.index == 1
                ? (stopwatch.elapsedMilliseconds / 1000) ~/ 10
                : curnum ~/ 4,
        index: widget.index,
        number: widget.index == 1 ? (stopwatch.elapsedMilliseconds / 1000).floor() : curnum))
```

그림16 ResultPage 호출 및 인자 전달

<운동 종류에 따른 횟수(시간)당 소모 칼로리>

운동 종류	횟수(시간)당 소모 칼로리
런지	4회당 1kcal
사이드플랭크	10초당 1 kcal
스쿼트	4회당 1kcal

- 세트당 운동 횟수(시간)와 목표 세트 수를 설정한 경우, 목표치를 달성했는지 여부는 (세트당 운동 횟수(시간) * 목표 세트 수)를 기준으로 설정하고, 현재 운동 횟수(시간)이 이 기준치를 넘겼는지를 통해 확인한다.
- 사이드플랭크의 경우
현재 운동 시간 = (전체 운동 시간 % 세트당 운동 시간)
현재 세트 수 = (전체 운동 시간 / 세트당 운동시간)으로 계산한다.
- 런지와 스쿼트의 경우
현재 운동 횟수 = (전체 운동 횟수 % 세트당운동횟수)
현재 세트 수 = (전체 운동 횟수 / 세트당 운동 횟수)를 계산한다.
※ 제한없음을 선택한 경우, 목표세트 수가 없으므로 운동 횟수(시간)를 stopwatch, 또는 curnum 그대로 표현한다.

```
String setNumber(int i){
  if(widget.targetnumber == -1){
    return i==1? '${(stopwatch.elapsedMilliseconds / 1000).floor()} 초' : '${curnum} 회';
  } else if(i == 1){
    return '${(stopwatch.elapsedMilliseconds / 1000).floor() % widget.targetnumber} / ${widget.targetnumber} 초';
  } else{
    return '${curnum % widget.targetnumber} / ${widget.targetnumber} 회';
  }
}
```

그림17 현재 운동 시간, 현재 운동 세트수 구현 코드

```
SizedBox(
    width: 100.w,
    child: Text(widget.targetnumber == -1
        ? ''
        : widget.index == 1
            ? '${(stopwatch.elapsedMilliseconds / 1000).floor() ~/ widget.targetnumber} / ${widget.targetset}세트'
            : '${curnum ~/ widget.targetnumber} / ${widget.targetset} 세트', style: TextStyle(fontSize: 20.sp)))
```

그림18 제한 없음 선택시 측정 구현 코드

- **TTS를 통한 피드백 제공 프로그램 설계**

- 기존 프로그램 설계는 자세가 incorrect 한 경우, 정자세로 교정하기 위한 교정 차원의 피드백을 제공한다. 사이드 플랭크는 자세의 correctness에 따라 교정 피드백을 제공하도록 구현했다. 런지나 스쿼트의 경우, 자세의 correctness 를 정확하게 구별하기 어렵다.

- 사이드 플랭크의 경우, 특정 자세를 유지함으로써 운동을 하는 것과 달리, 런지와 스쿼트의 경우 연속 동작을 통해 운동을 하기 때문에, 잘못된 자세로 운동을 하더라도 정자세로 운동하는 과정의 순간과 구별할 수 없는 문제가 존재한다.

ex) 스쿼트를 할 때 엉덩이를 더 내리지 않는 경우

런지를 할 때 무릎을 제대로 굽히지 않는 경우 등

- 이러한 문제점을 해결하기 위해 런지와 스쿼트 운동에 한해 기존의 TTS 사용 목적을 실시간 교정 피드백을 제공하는 것 대신, 사용자의 움직임을 감지하여 단순한 notification을 제공하는 것으로 변경해 오류없이 음성을 구현하는 코드를 작성했다.

```
bool checkAngle(list) {
  if (list[0] > 160 && list[1] > 160) {
    stopwatchStart();
    isCorrectPose = true;
    return true;
  } else {
    stopwatchStop();
    if(isCorrectPose == true)
      {
        flutterTts.speak('correct your pose');
        isCorrectPose = false;
      }
    return false;
  }
 }
}
```

```
void inUpPosition(list) {
  if ((list[0] > 170 ) && (list[1] > 170) ) {
    if(downPosition==true){
      notifyParent();
      flutterTts.speak('Up');
      downPosition=false;
    }
    upPosition = true;

  }
}

void inDownPosition(list) {
  if ((list[0] > 70 && list[0] < 110) && (list[1] > 70 && list[1] < 110)) {
    downPosition = true;
    if(upPosition==true) {
      flutterTts.speak('down');
      upPosition=false;
    }
  }
}
```

그림19 올바른 사이드플랭크 자세 함수 구현 코드 **그림20** 올바른 런지 자세 함수 구현 코드

```
void inDownPosition(list) {
  if (isCorrectAngle1(list[0]) == true && list[1] < 100) {
    downPosition = true;
    if (upPosition == true) {
      flutterTts.speak('Down');
      upPosition = false;
    }
  }
}

void inUpPosition(list) {
  if (isCorrectAngle1(list[0]) == true && list[1] > 160) {
    if (downPosition == true) {
      notifyParent();
      flutterTts.speak('Up');
      downPosition = false;
    }
    upPosition = true;
  }
}
bool isCorrectAngle1(_angle) {
  // 무릎이 발 앞으로 안나가게 만드는 각도
  return _angle > 70 ? true : false;
}
```

그림21 올바른 스쿼트 자세 함수 구현 코드

● **운동별 올바른 자세 각도 설정**

운동	중요 관절	각도 조건
사이드 플랭크	**왼쪽 허리, 오른쪽 허리(BodyDetection LandMark 23,24번)**	왼쪽 허리 각도 > 160° 오른쪽 허리 각도 > 160°
런지	**왼쪽 무릎, 오른쪽 무릎(BodyDetection LandMark 25,26번)**	**inDownPosition** 70 < 왼쪽,오른쪽 무릎 각도 < 110° **inUpPosition** 왼쪽,오른쪽 무릎 각도 > 170°
스쿼트	**왼쪽 무릎, 왼쪽 발목(BodyDetection LandMark 25,27번)**	**inDownPosition** 왼쪽 발목 각도 > 70°, 왼쪽 무릎 각도 < 100° **inUpPosition** 왼쪽 발목 각도 > 70°, 왼쪽 무릎 각도 > 160°

※ 위와 같은 중요 각도의 설정은, 전문가의 자문을 구해 설정하였다.

3.2.7. ResultPage(결과창)

1) 클래스 설명

Class ResultPage extends StatefulWidget	설명
index	**운동 종류를 인자로 받아오는 매개변수**
kcal	**소모한 칼로리 양을 인자로 받아오는 매개변수**
number	**총 운동 횟수(또는 운동 시간)를 인자로 받아오는 매개변수**

class _ResultPage extends State<ResultPage>	설명
_authentication	**FirebaseAuth.instance 를 간략화하기 위한 변수**
loggedUser	**currentUser 정보를 받는 변수**
getCurrentUser()	**currentUser를 loggedUser 에 할당하는 함수**

2) **해당 페이지 사진 & 위젯트리**

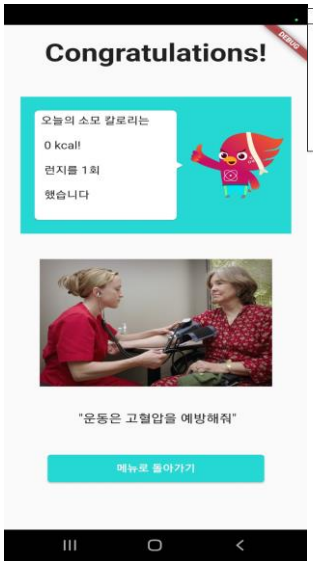


그림22 ResultPage 어플리케이션 구현

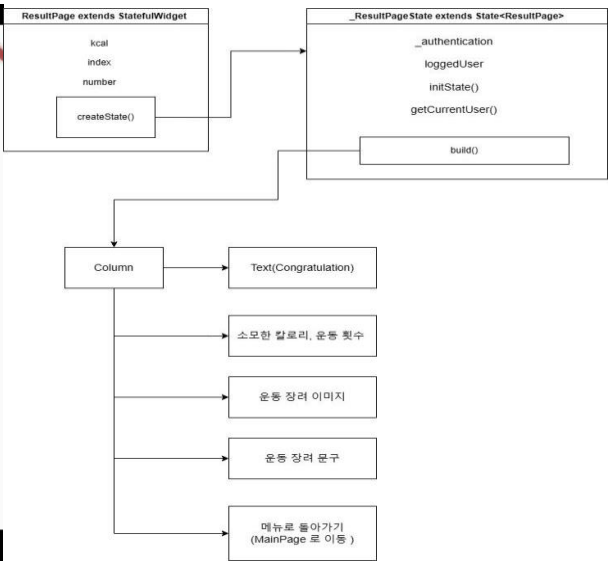


그림23 ResultPage Widget Tree

3) **Backend 설명**

메뉴로 돌아가기 버튼을 누르면 MainPage를 호출하고, 매개변수의 인자로 kcal, number, index를 전달한다.

3.2.8 최적화 관련 연구 및 개선방안

- pose detection을 사용하자 버퍼링이 발생했는데 이에 대한 원인으로 Jank를 꼽을 수 있다. Flutter는 Skia engine을 통해 위젯을 생성하고 제거하는데, 일반적으로 Skia engine은 60Hz로 동작하므로, 16.7ms 안에 Rendering해야 어플이 버벅거리지 않는다. Jank는 우리가 Skia engine의 주기에 맞춰 Rendering을 끝내지 못해 새로운 UI가 그려지지 않고, 화면이 업데이트 되지 않아 버벅이는 것을 사용자가 보는 현상을 말한다.
- Jank는 UI와 Raster 스레드 모두에서 오랜 시간이 걸리는 경우와, Raster 스레드에서만 문제가 나타나는 경우에서 주로 발생한다. 전자의 경우 UI 스레드 또는 Widget Tree의 변경 횟수가 잦거나, UI 스레드에서 무거운 작업이 수행되고 있는 경우다. 후자의 경우 saveLayer, Opacity, Shadow, Clip이 원인일 가능성이 높다.
- 정적이지 않은 이미지 캐싱도 많은 Cost를 소모하는데 setState가 동작하는 statefulWidget에서 const를 사용하면 Jank 현상을 줄일 수 있다. 아래는 StatefulWidget인 Camera class으로, Jank를 줄이기 위해 const를 사용한 여러 부분 중 하나다.

```
OutlinedButton(
  onPressed: _toggleDetectPose,
  child: _isDetectingPose
      ? const Text('Turn off pose detection')
      : const Text('Turn on pose detection'),
),  // OutlinedButton
IconButton(
  icon: const Icon(Icons.exit_to_app_sharp),
  onPressed: (() {
    showDialog(
        context: context,
        barrierDismissible: true,
        builder: (BuildContext ctx) {
```

그림24 Jank 감소를 위해 코드 내 const 사용 예시

-pose detection을 사용하자 버퍼링이 발생해 원인을 알아보기 위해 Flutter의 DevTools를 사용했다.

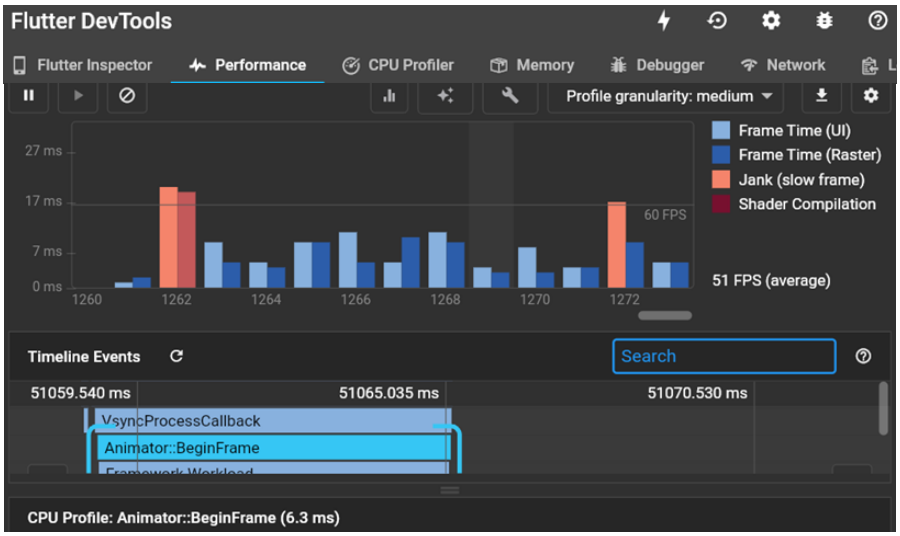


그림25 Flutter DevTools 사용 예시

- DevTool을 통해 나온 결과를 json파일로 다운로드 받은 후, 이를 python의 dataframe으로 바꿔, seaborn, matplot, pandas, numpy를 사용해 데이터를 확인했다.

```
================== turn off 상태 : build, raster 평균 ================
카메라실행 후 build 평균 :  24952.86
카메라실행 후 raster 평균 :  13955.0
느린모션 시작 후 build 평균 :  19632.2
느린모션 시작 후 raster 평균 :  13987.9
빠른모션 시작 후 build 평균 :  22912.73
빠른모션 시작 후 raster 평균 :  13874.67


================== turn on 상태 : build, raster 평균 ==================
카메라실행 후 build 평균 :  23712.57
카메라실행 후 raster 평균 :  19667.0
느린모션 시작 후 build 평균 :  20616.86
느린모션 시작 후 raster 평균 :  13181.0
느린모션 + 발동작 추가 후 build 평균 :  22365.66
느린모션 + 발동작 추가 후 raster 평균 :  18874.64
빠른모션 시작 후 build 평균 :  23556.37
빠른모션 시작 후 raster 평균 :  21003.17
```

그림26 AI model 사용 전후의 17,000ms 넘는 build와 raster 평균

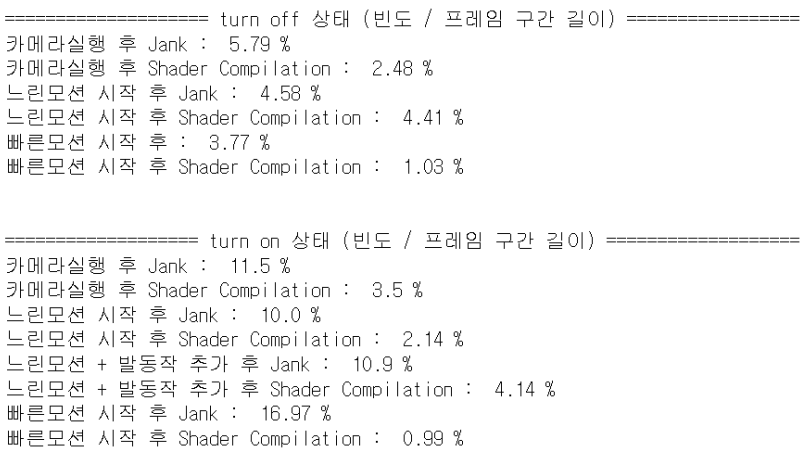


그림27 AI model 구간별 프레임 길이 당 빈도수

- 각 구간에서 delay를 발생시키는 17ms가 넘는 시간들의 평균을 조사한 결과, build의 경우 turn on/off 여부에 상관없이 값이 비슷하다. raster의 경우, turn on을 했을 때가 off한 경우보다 평균 시간이 높았다.
- 각 구간의 측정시간이 다른 만큼, 빈도 / 측정시간 값을 계산한 결과 pose detection 실행한 후(turn on)가 실행하기 전(turn off)보다 빈도가 적게는 50%에서 많게는 500%가량 증가함을 확인했다.

<build : pose detection off>	<raster : pose detection off>
df_off_start_build : 카메라 실행 후 아무 동작 하지 않음 df_off_slow_build : 느린 모션 실행 df_off_fast_build : 빠른 모션 실행	df_off_start_raster : 카메라 실행 후 아무 동작 X df_off_slow_raster : 느린 모션 실행 df_off_fast_raster : 빠른 모션 실행
<build : pose detection on>	<raster : pose detection on>
df_on_start_build : pose detection 후 아무 동작 하지 않음 df_on_slow_build : 느린 모션 실행 df_on_slow_plus_build : 느린 모션 + 발동작 실행 후 df_on_fast_build : 빠른 모션 실행	df_on_start_raster : pose detection 후 아무 동작 하지 않음 df_on_slow_raster : 느린 모션 실행 df_on_slow_plus_raster : 느린 모션 + 발동작 실행 후 df_on_fast_raster : 빠른 모션 실행

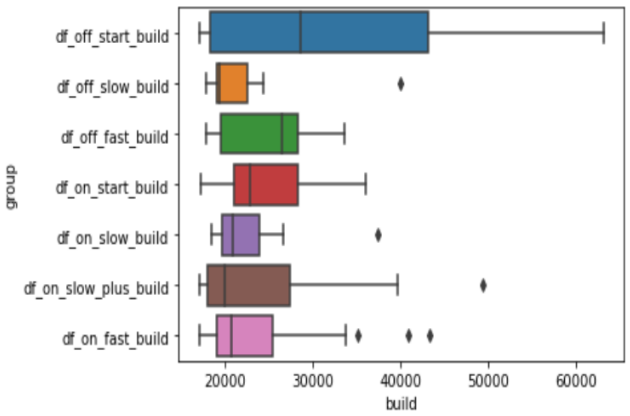


그림28 구간별 Build DataFrame Boxplot

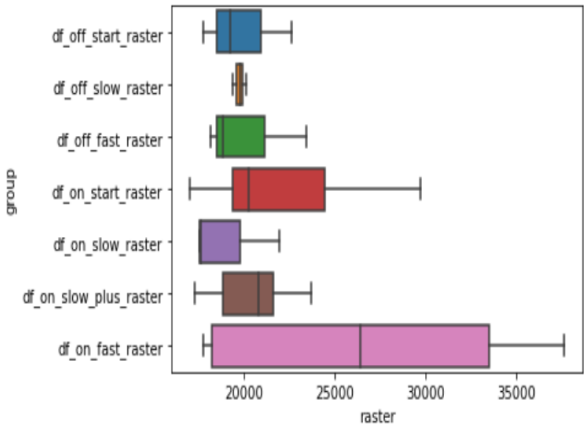


그림29 구간별 Raster DataFrame Boxplot

- build가 붙은 DataFrame 자료들의 경우 UI로 인해 걸리는 시간을 측정한 만큼, Pose Detection on/off 여부에 따라 자료들의 분포, 최대, 최소, 중간값 등이 크게 차이나지 않는다. 초기 시작하는 경우에는 setting을 위해 UI에서 많은 시간이 걸리므로 이를 나타낸 df_off_start_build 의 중간값, 최댓값등이 다른 자료들에 비해 크다.
- raster의 경우 Raster Thread에서 걸리는 시간을 측정한 자료들로 Shader Compilation Jank와 관련있다. Pose Detection on을 하게 되면 AI를 통해 화면에 골격을 그리면서 Raster Thread에서 많은 시간을 쓰게 된다. 이런 사실은 off상태보다 on상태의 boxplot의 최대, 최소 중간값이 모두 증가했음을 확인할 수 있다. 또한, 각각의 Pose Detection on/off상태에서 start, slow, fast로 동작 속도가 증가함에 따라 시간당 좌표위치 변화값이 커지고, 자료들의 중간값, 최댓값 등이 커짐을 확인할 수 있다.
- Raster Thread에서 걸리는 시간이 17ms를 넘으면 발생하는 Shader Compilation Jank는 앱 실행 초반 이후 최대한 발생하지 않아야 좋다. 어플리케이션이 최적화가 되기 위해서는 카메라와 AI model 모두 최적화가 필요하지만, 그 중 Shader Compilation을 해결을 우선해야 한다. 이를 위해 Pose Detection model을 최적화할 필요가 있다.

4. 한계점

1) 2차원 각도 계산

- 앱을 실제로 사용하는 상황을 가정했을 때, 정방향에 카메라를 놓고 찍어야 한다는 조건이 상당히 제한적이다. △카메라를 정면에 놓지 않을 때 △사용자의 측면에 놓고 찍을 때 △바닥에 놓고 찍을 때 등에는 각도를 정확하게 계산하지 못한다는 단점이 존재한다.
- 이는 카메라 이미지를 통해 2차원 좌표상에 여러 PoseLandMark를 구현했기 때문이며, 실제 3차원인 현실에서의 각도와 오차가 존재한다.

2) 최적화

최적화하기 위해서는 Pose Detection 모델 경량화가 필요하다. 그러나 모델의 경량화는 정확성을 낮추기 때문에 최적화와 정확성 모두를 높일 수 없다는 한계가 있다. 그렇기에 사람이 불쾌감을 느끼지 않을 정도의 Jank가 발생하는 최적화 정도와 사람이 인지할 수 있는 범위의 정확성의 적정선에서

최적화가 이뤄져야 한다.

○ 연구결과 및 활용방안

- 구글에서 개발한 ML Kit의 Vision API 중 하나인 Pose Detection을 이용하여 사용자가 운동을 할 때 그 운동에 맞는 특정한 조건을 만족할 시 측정되며, 골격을 통해 자세를 쉽게 볼 수 있다. 운동 기록은 달력을 통해 한 눈에 쉽게 볼 수 있으며, 그 외 회원가입, 회원탈퇴, 운동 결과 안내 등의 기능을 제공한다.
- 본 연구는 ML Kit AI 기술을 활용한 PT 어플리케이션을 개발했다. 이를 통해 사용자들이 더 정확한 자세로 운동하는 것을 도와 부상을 방지하고, 운동 효율을 향상시키는 점을 목표로 한다.

1) 헬스 관련 자격증 취득

- 우리가 개발한 어플리케이션은 정확한 자세를 취하지 않을 시 개수가 측정되지 않는다. 시간의 경우 증가하지 않는다. 이는 헬스 트레이너 실습 평가 시 객관적인 평가 기준을 제시할 수 있다. 헬스 트레이너 관련 자격증을 따는데 도움을 줄 것으로 기대된다.

2) 헬스의 보편화

헬스를 처음 시작하는 사람들은 홈 트레이닝을 할 때 정확한 자세를 유지하는 것이 어렵고 PT는 비용의 부담이 있어 쉽게 주저하는 경우가 많다. 본 어플리케이션은 혼자서도 정확한 자세로 운동하는 것을 도와줌으로 헬스에 대한 진입장벽을 낮춰 운동의 보편화에 일조할 수 있다.

3) 자세 교정

현대인들은 잘못된 자세로 인해 허리통증, 거북목 등을 흔히 경험한다. 본 연구의 목적은 헬스의 정확한 자세를 지도하는 것이다. 앉은 자세, 걷는 자세등을 분석하여 올바른 자세로 교정시켜, 사용자의 전반적인 건강을 관리할 수 있다.

4) 높은 확장성

개발된 앱은 추후 여러가지 운동 및 자세를 추가할 수 있는 만큼, 큰 확장성을 가진다. 더 나아가 홈 트레이닝을 넘어서 전반적인 운동에 대해서도 동작을 추가해줄 수 있다. 구기 종목의 여러 동작과 개인기, 여러 무술의 동작 등 특정 운동에 대해 특화하는 방향으로 확장이 가능하다.

○ 참고문헌

1) 홈 트레이닝 시장규모

https://www.statista.com/topics/5325/fitness-equipment-in-the-us/#topicHeader__wrapper

2) 운동별 칼로리

https://www.dietshin.com/calorie/sports-view.asp?cal_idx=465&cal_type=E

3) Flutter dev tool performance view

https://docs.flutter.dev/development/tools/devtools/performance