

캡스톤 디자인 최종보고서

주제	국문	자폐 스펙트럼 증상 완화를 위한 저비용 AR 감정 인식 보조 기기 개발
	영문	Proposal of low-cost AR emotion recognition assistance device to alleviate ASD symptoms

팀명	012
팀원	214823 박종현, 214722 모아림

1. 연구과제의 개요

본 연구는 자폐 스펙트럼 장애(ASD)를 가진 사람들의 감정 인식을 보조하는 증강현실(AR) 기반 기기를 개발하고, 이를 통해 자폐 증상 완화의 가능성을 탐구하는 것을 목표로 한다. 현재 자폐 스펙트럼 장애는 전 세계적으로 많은 사람들이 겪고 있는 문제로, 감정 인식 및 사회적 상호작용의 어려움이 자폐 아동과 성인의 주요 증상 중 하나로 지적되고 있다. 기존의 치료법과 교육 프로그램들은 어느 정도 효과가 있지만, 비용이 많이 들고 지속적인 상호작용이 어렵다는 단점이 존재한다.

연구는 엔드포인트 하드웨어(글래스), 감정 인식 처리 서버(처리기), 중계 애플리케이션(중계기)로 구성된 시스템을 개발함으로써 저비용 고효율의 솔루션을 제공하는 것을 목표로 한다. 이를 통해 자폐 스펙트럼 장애를 가진 사람들이 감정 인식 능력을 높이고, 일상 생활 속에서의 사회적 상호작용을 개선할 수 있는 가능성을 제시하고자 한다.

이 연구의 가설은 “AR 감정 인식 보조 기기를 통해 자폐 스펙트럼 장애를 가진 사람들이 상대방의 감정을 더 잘 인식하게 되면, 사회적 상호작용이 향상되고 자폐 증상이 완화될 수 있다”는 것이다. 이는 자밀자키의 저서 『공감은 지능이다』에서 소개된 자폐장애 글래스 프로젝트에서 영감을 얻었다.[1] 해당 프로젝트에서는 구글 글래스를 사용해 자폐 스펙트럼 장애를 가진 사람들이 감정 인식 능력을 향상시키는데 도움을 주었으며, 이러한 기술의 사용이 자폐 증상 완화에 기여할 수 있다는 가능성을 보여주었다.

1.1) 연구과제의 필요성

첫째, 자폐 증상 완화에 대한 새로운 가능성을 제시한다. 자폐 스펙트럼 장애는 감정 인식과 사회적 상호작용에 어려움을 겪는다는 특성을 가지고 있다. 본 연구에서는 자폐 환자들이 감정을 더 명확하게 인식할 수 있도록 도와, 일상생활에서의 사회적 상호작용을 향상시킬 수 있을 것으로 기대된다.

둘째, 구글 글래스 단종 후의 대안 하드웨어 개발 필요성을 충족한다. 구글 글래스는 가격적으로 최적화되지 않았고, 더 이상 정규 경로로 구할 수 없다. 본 연구는 감정 인식에 초점을 맞춘 저비용 AR 하드웨어를 설계함으로써, 자폐 치료나 기타 감정 인식이 중요한 분야에서의 실질적인 대안을 제시할 수 있다.

셋째, AI와 최신 기술의 융합을 통해 감정 인식 기술의 성능을 개선할 수 있다. 본 연구는 최신 임베디드 기술과 AI를 도입하여 기존 감정 인식 알고리즘의 성능을 높이는 데 중점을 둔다. 이러한 시도는 자폐 치료 뿐 아니라 감정 인식이 중요한 다양한 분야에 응용될 가능성이 있다.

1.2) 연구목표 및 내용

1.2.1) 연구 목표

1. 대면 중인 사람의 감정 상태를 평가하고 결과를 화면에 출력하는 AR 기기를 구현하는 것:
이 프로젝트는 실물 기기를 포함하여 기기 구동에 요구되는 일련의 시스템 일체를 구현하는 것을 목표로 한다. 이 시스템은 Section 1 와 Section 1.2.2 에서 정의한 글래스, 중계기, 처리기를 포함한다.
2. 이 기기를 임상 실험하여 이 연구를 시작하게 된 선행 연구의 주장을 검증하는 것:
단순히 우리 팀이 심리학 전공자가 아니고 자폐 증상에 대한 정보를 필요로 하지 않기 때문에 잘 모르는 것인지, 혹은 실제로 전공자에게도 잘 알려지지 않은 사실인지 제대로 파악되지는 않았으나, 자밀자키[1] 의 주장은 널리 알려지지 않았다. 특히 자폐 환자가 상대방의 감정을 제대로 인식하는 것, 더 나아가 감정 인식 보조 장치의 도움을 받는 것이 자폐 증상을 완화시킨다는 주장은 [1] 에서 처음 접하였다. 따라서 이러한 장치가 실제로 효과를 내는지 검증하는 것은 좋은 기회라고 생각한다.

1.2.2) 연구 과제 내용

대면 중인 사람의 감정 상태를 평가하고 사용자에게 그 결과를 출력하는 AR HMD 기기와 부속 소프트웨어를 구현한다.

1. 엔드포인트 하드웨어(글래스)

사용자가 실제로 착용하는 임베디드 하드웨어로, 안경에 부착하거나 안경처럼 착용하는 방식으로

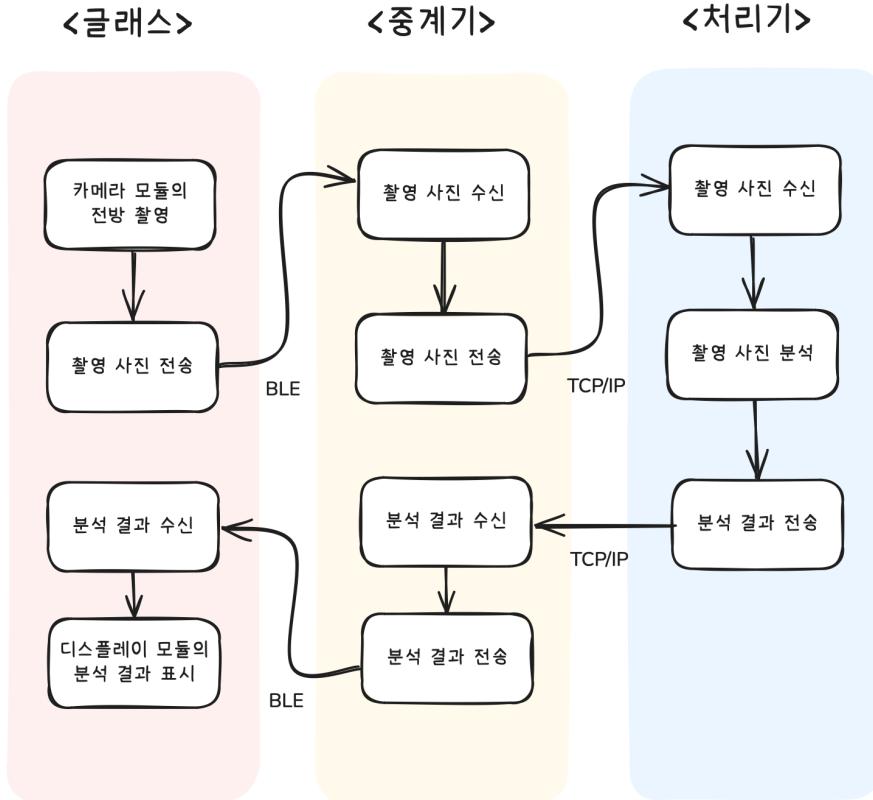


Figure 1: 시스템의 개요

구현할 계획이다. 소형 카메라 모듈을 사용해 상대방의 얼굴을 인식하고, 감정 평가 결과를 사용자가 쉽게 확인할 수 있도록 시야 한쪽 구석에 소형 디스플레이를 설치한다.

2. 중계 애플리케이션(중계기)

사용자의 스마트폰을 중계기로 사용하여 클래스와 처리기가 원활하게 통신할 수 있도록 한다. 스마트폰 애플리케이션은 클래스에서 수집한 데이터를 서버로 전송하고, 서버에서 처리된 결과를 클래스로 다시 전달하는 역할을 수행한다.

3. 감정 인식 처리 서버(처리기)

클래스에서 수집한 데이터를 외부의 고성능 서버로 보내어 감정 인식을 처리한다. 이 서버는 기계학습 감정 인식 알고리즘을 활용하여 얼굴 데이터를 분석하고 감정을 평가하며, 그 결과를 처리기로 다시 전달한다.

2. 연구과제의 수행 과정 및 수행 내용

시스템의 상세한 작동 시나리오는 다음과 같다: 우선 클래스의 카메라 모듈이 전방을 촬영한다. 촬영한 사진을 Base64 문자열로 인코딩하여 BLE 인터페이스를 통해 전송한다. 중계기는 BLE 인터페이스로 데이터를 수신한다. 이어서 처리기로 처리를 요청하기 위해 RESTFUL JSON을 생성한 후 수신한 Base64 문자열을 생성한 JSON 데이터에 담는다. 중계기는 생성한 JSON 데이터를 이용하여 처리기에 HTTP 요청을 발생시킨다.

처리기는 요청을 수신하여 파라미터를 역직렬화한다. 이 과정에서 요청에 포함된 Base64 문자열을 이미지로 디코딩하고 학습시킨 분류 모델에 입력한다. 분류 모델은 입력에서 얼굴을 추출하고 얼굴에 드러난 감정을 분류한다. 분류 결과는 응답 규격으로 생성된 RESTFUL JSON에 담아 HTTP 통신 결과로 반환한다.

중계기는 통신 결과를 역직렬화하여 결과 데이터를 추출한다. 추출한 결과 데이터는 BLE 인터페이스를 통해 클래스에 반환된다. 클래스는 반환된 결과 데이터를 디스플레이 모듈에 표현한다.

2.1) 클래스: 기반 보드 선정과 시행착오, 변화

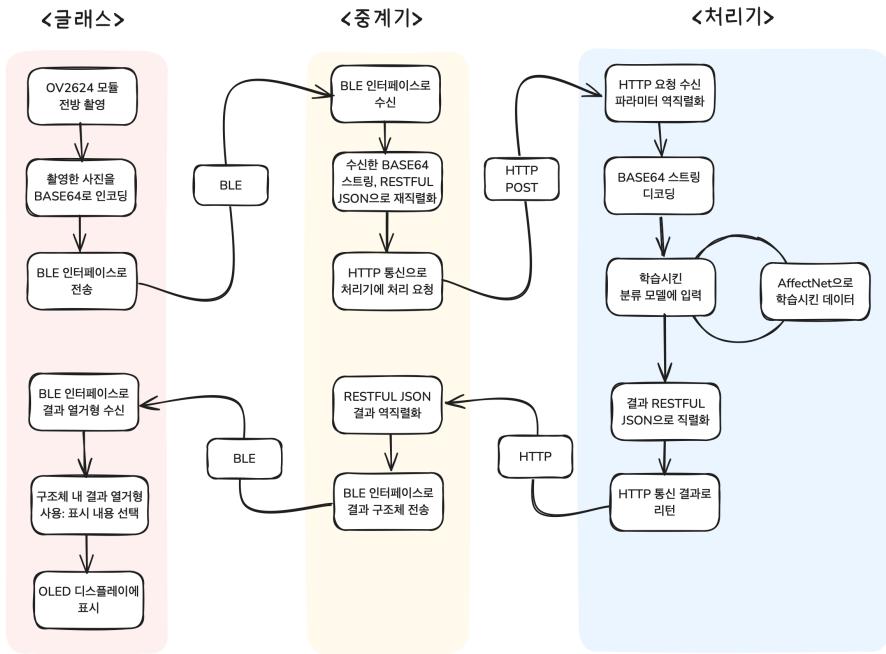


Figure 2: 시스템의 상세 개요

프로젝트 수행 기간, 팀의 재정 상황과 지원 사업의 진척, 거듭되는 시행착오에 따라서 프로젝트에 사용하는 글래스 기반 보드를 계속해서 변경하였다. 최초 단계에서는 Arduino Uno, 이어서는 Arduino Nano 33 IoT, Raspberry Pi Zero/Raspberry Pi Zero 2 W, Ai Thinker ESP32-CAM을 차례로 도입하고 실험하였다.

2.1.1) 1차: Arduino Uno

Arduino Uno(아두이노 우노)는 가장 널리 보급된 IoT 컴퓨팅 보드여서, 프로젝트 시작 이전부터 글래스 담당 팀원이 사용할 수 있었다. 크기, 부품 등 사양을 고려했을 때 보드를 그대로 사용하는 것은 어려움을 인지하고, 프로젝트 초기 연구 방향을 설정하기 위해 사용하였다.

하지만 아두이노 우노는 블루투스 기능이 내장되어있지 않아 [2] 블루투스 모듈과의 GPIO 연결 설정을 비롯하여 이후 보드 변경 시 사용하기 어려운 코드를 많이 작성해야했다. 초기 연구 방향을 설정하는데에도 마찬가지로 큰 도움이 되지는 않았으나, 아두이노 개발 환경을 설정하고 팀 내부에서 사용할 워크플로우를 정리할 수 있었다.

```

# 의존성 확인, 누락된 의존성 다운로드
arduino-cli lib deps
arduino-cli lib download

# 컴파일하여 스케치 오류 검사
arduino-cli compile

# 스케치 업로드
arduino-cli upload

```

Listing 1: 개발 과정에서 되도록 Arduino IDE보다 CLI 환경을 사용할 수 있도록 하였다.

2.1.2) 2차: Arduino Nano 33 IoT

아두이노 우노에서 정리한 워크플로우를 바탕으로 캡스톤 프로젝트 시작 직전 Arduino Nano 33 IoT(아두이노 나노 33 IoT)로 기반 보드를 변경하였다. 아두이노 나노 33 IoT는 BLE 통신이 가능하면서 보드의 물리적인 크기가 만족할만한 수준으로 소형화되었으므로 [3] 작업에 가장 이상적인 보드라고 판단하였다.

이 과정에서 아두이노 나노 33 IoT와 카메라 모듈의 통합도 시도하였다. 카메라 모듈은 OmniVision 사의 OV7670 CAMERACHIP™ 이미지 센서(OV7670)를 채택하여 사용했다. OV7670은 640×480의 VGA 해상도를 최대 30fps로 촬영할 수 있다. [4]

```

// IMPORTANT: #include ALL of the arch-specific .h files here.
// They have #ifdef checks to only take effect on the active architecture.
#include "arch/samd51.h"

```

Listing 2: Adafruit_OV7670/src/ov7670.h [5]

Figure 4 SCCB Timing Diagram

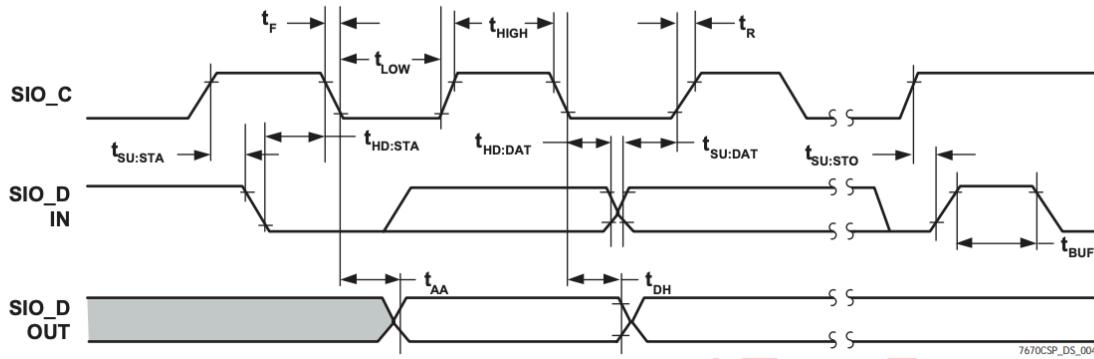


Figure 3: OV7670의 SCCB 타이밍도[4]

OV7670과의 호환 문제

하지만 카메라 모듈인 OV7670과 제대로 호환되지 않았다. 아두이노 라이브러리 저장소에 등록된 OV7670 드라이버 모듈인 Adafruit OV7670과 Arduino_OV767X 모두 아두이노 나노 33 IoT에 적절하게 대응되지 않았다. Adafruit OV7670은 SAMD51 아키텍처에 대응하여 작성되었기 때문에 OV7670의 공식 예제인 cameratest.ino와 selfie.ino 모두 아두이노 나노 33 IoT에서 동작하지 않았다. 드라이버의 소스코드 상 주석을 참조하면 라이브러리의 /src/arch/ 아래에 SAMD21에 대응하는 아키텍처 코드를 작성하고 추가해야 했다.

SAMD21 대응 OV7670 드라이버 코드 작성 시도

글래스 담당 팀원은 라이브러리 요구사항을 만족하기 위해 SAMD21 아키텍처 대응 드라이버의 작성을 시도하였다. 구체적으로 OV7670 데이터시트에 근거하여 SAMD51 아키텍처 대응 코드를 SAMD21 아키텍처 대응 코드로 마이그레이션하고자 했다. 꽤 많은 시간을 투입하여 데이터시트를 읽어냈으나, 데이터시트를 읽는데 요구되는 추가적인 관련 지식이 없었기 때문에 더 나아갈 수 없었다. 예를 들어 직렬 카메라 제어 버스(Serial Camera Control Bus; SCCB)의 타이밍도를 읽을 수는 있었으나, 이를 이용한 카메라 제어 코드를 제대로 구현해내지 못했다.

또한 SAMD51 아키텍처와 SAMD21 아키텍처의 차이로 SAMD51 대응 코드를 사용하지 못하는 부분도 많았다. 대부분의 경우 SAMD21에 없으나, SAMD51에 존재하는 요소가 많았고, 이 경우 구현 자체를 다른 방향에서부터 접근해야 했다. 결과적으로 SAMD21 드라이버는 수 주간 제대로 작동하지 않았고 성과가 나오지 않았다.

최종적으로 기한 내에 목표를 달성하기 어려울 것이라고 판단하여 OV7670의 SAMD21 드라이버 작성 을 포기하였다.

2.1.3) 3차: Raspberry Pi Zero / Raspberry Pi Zero 2 W

Raspberry Pi Zero와 Raspberry Pi Zero 2 W(라즈베리파이 제로)는 Raspberry Pi Foundation에서 개발한 저가형 싱글보드 컴퓨터로, 주로 Broadcom(브로드컴)의 ARM SoC를 사용한다. 아두이노 와는 다르게 완전히 독립적인 컴퓨터로서 기능할 수 있으므로, 더 다양한 처리가 가능하다. 일반적으로 리눅스 시스템을 설치해서 사용하여 아두이노의 사례와 같은 미호환 부품 대책도 충분히 마련할 수 있다. [6], [7]

Raspberry Pi의 도입

라즈베리파이 제로를 도입한 것도 위 사항을 고려하였기 때문이다. 아두이노 나노 33 IoT와 OV7670 간의 호환성 문제는 아두이노 나노 33 IoT와 OV7670 둘에게 한정된 문제로 판단하고 있었기 때문에, 팀은 보드에 맞는 카메라 모듈과 카메라 모듈에 맞는 보드를 각각 조사하였다. 그 결과 중 하나로서 라즈베리파이 제로의 도입을 시도하였다.

특히 라즈베리파이에서 주로 사용되는 arm 아키텍처는 아두이노 보드와 달리 범용적으로 사용되는 아키텍처이고, 리눅스를 설치해서 사용하므로 arm 아키텍처 IP별 파편화에 별도로 대응하지 않아도 된다는 점도 장점이다. 리눅스의 지원 아키텍처 풀은 굉장히 넓으므로 당장 아키텍처에 호환되지 않더라도 어딘가에는 호환성 레이어나 가상화 레이어가 있을 것이라는 기대도 있었다.

다만 아두이노와 완전히 다른 플랫폼이라는 점에서 기인하는 우려사항이 있었다.

우려사항 1. 큰 코드 재작성 소요

전용 C++ 방언만을 사용하여 프로그램을 구현해야하는 아두이노와 다르게, 라즈베리파이는 다양한 언어로 시스템 자원을 사용할 수 있다. 실제로 상당히 많은 코드가 파이썬으로 작성되고 있고, GPIO를 사용하는 많은 예제가 파이썬으로 작성되어 배포되고 있다. 따라서 이전까지 작성한 사실상 모든 코드를 처음부터 다시 작성해야 한다는 우려가 있었다. C++로 작성해서 극히 일부분의 코드를 재활용할 수 있겠지만, 상대적으로 양이 덜한 자료만을 고집하기에는 라즈베리파이로 전환한 후의 남은 시간이 얼마 없었다.

우려사항 2: 리눅스 시스템과 통합

또 리눅스 시스템에 코드를 통합하는 것도 큰 우려사항이었다. 라즈베리파이에 코드를 업로드하는 것 만으로 아두이노와 같은 작동을 기대할 수 없다. 리눅스 시스템에 서비스로 등록하고 실행 설정을 관리해야 한다. 이 점을 포함하여 전반적인 디버그 과정이 더욱 복잡해지기도 했다. 개발 환경에서 코드를 작성하고 검증한 뒤에 라즈베리파이에 적용하여 다시 한번 코드의 작동을 검증해야 했다. Git 등의 도구를 사용하면 코드의 전송이 크게 어렵지는 않으나 라즈베리파이에 물리적으로 접근하여 제어하거나 VNC, SSH 등의 수단을 사용하여 수동으로 제어해야하므로 아두이노와 비교하여 업로드 과정이 더욱 복잡한 것은 사실이다. 개발 환경과 라즈베리파이 간 디버그 파이프라인을 설정하는 것도 고려하였으나, 전환 시점에서의 남은 기간을 고려하였을 때 라즈베리에 직접 접근하는 것이 덜 복잡하며 현실적인 선택지였다.

위 우려사항은 감수할만 했으나, 다른 이유로 인해 라즈베리파이의 도입을 포기하였다.

팀원의 납땜 숙련도 및 실납의 품질 문제

라즈베리파이 제로에는 GPIO에 핀 헤더가 달려있지 않아, 라즈베리파이를 개발 과정에서 수월하게 사용하기 위해서는 GPIO 핀 헤더를 새로 납땜해야 한다. 하지만 작업 시점에서 모든 팀원이 납땜 경험이 없었으므로 숙련도와 요구 지식 수준을 달성하지 못한 채 투입되었다.

또한 지식이 충분하지 않았기 때문에 부적절한 재료와 장비를 구입하게 되었다. 납은 제대로 녹지 않았고, 인두기 거치 스탠드는 재료들을 제대로 고정하지 못하여 오히려 납땜 과정에서 주요한 방해물이 되었다. 잘못 녹인 납을 제거하기 위해 납 흡입기를 사용하였으나 납 흡입기는 납을 전혀 흡입하지 못했다. 무엇보다 인두기는 온도가 안정적으로 일관되지 않았다. 보드에 붙은 납도 제대로 녹이지 못했다.

일련의 납땜 경험으로 카메라 모듈과 보드의 연결 방안을 재평가하게 되었다. 하드웨어의 소형화를 위해서 카메라 모듈은 보드가 있어서는 안되었다. 보드가 있다면 보드에서 카메라 모듈을 떼어내야 했다. 이 때문에 카메라 모듈의 FFC 필름 케이블과 점프 케이블을 극세 납땜하여 아두이노 모듈과 연결하는 것을 고려하고 있었다.

글래스 담당 팀원은 GPIO 핀 헤더의 납땜을 실패한 것을 계기로 납땜 난이도를 저평가하고 있었다는 것을 확인하였다. 대책으로 FFC 커넥터를 이용해 FFC 필름 케이블을 연결하고 FFC 커넥터를 점프 케이블과 납땜하는 방안을 고려하였으나, 여전히 극세사 납땜이 필요하므로 작업의 난이도가 유의미하게 낮아지지 않았다. 결과적으로 FFC 필름 케이블을 극세 납땜하는 방안은 부적절했고, 카메라 모듈 연결 대책이 모두 무력화되었다.

송*재 ★★★★★ 2024.06.14
판매자: 유니밸리스

유니밸리스 납땜용 용접 와이어 납땜 실납 1.0mm 50g 1개

리뷰 잘 안다는 편인데 이건 진짜 아닙니다. 회로 납땜하다가 납이 떨어져서 급하게 주문했는데 이걸로 하다간 회로 다 망칠 것 같아서 급해도 작업중단했습니다. 인두기에 대고있는 도중에도 굳어버리는 납은 처음봅니다. 진짜 최악이네요

디자인	보통이에요
편리성	생각보다 불편해요
크기	적당해요
견고함	생각보다 부실해요

이 상품평가 도움 되었나요?

신고하기

박*우 ★★★★★ 2024.06.14
판매자: 유니밸리스

유니밸리스 납땜용 용접 와이어 납땜 실납 1.0mm 50g 1개

실 제품과 다른 제품이 온것같은 형태
제대로 녹지도 않고 고정도 안되어 품질도 매우 떨어짐 괜히 돈주고 샀음

디자인	별로에요
편리성	생각보다 불편해요
크기	예상보다 작아요
견고함	생각보다 부실해요

이 상품평가 도움 되었나요?

신고하기

Figure 4: 불량 실납의 구매자 리뷰: 실납 품질의 중요성을 몰랐기 때문에 이러한 구매자 리뷰를 주의깊게 확인하지 못했다.[8]

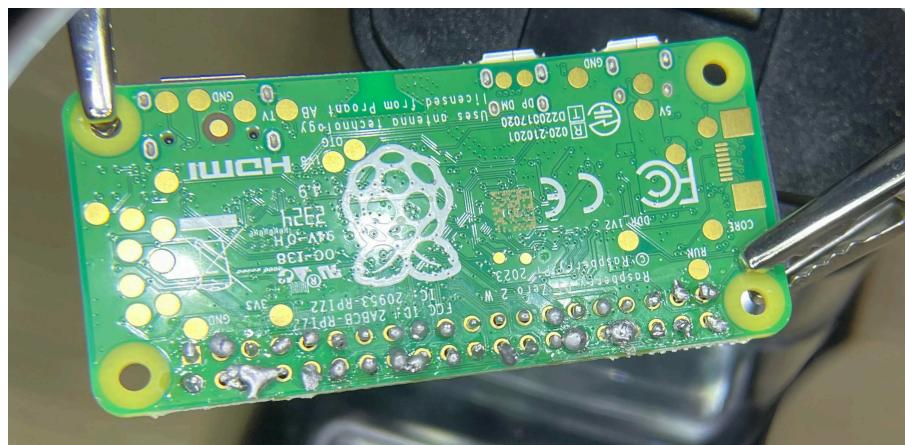


Figure 5: 불량 실납으로 라즈베리파이 보드에 납땜을 시도한 결과

2.1.4) 4차: AI Thinker ESP32-CAM

극세 납땜의 비현실성이 대두됨에 따라 카메라 모듈 연결 대책을 강구해야 했는데, 원래 고려하던 다른 보드로 돌아가더라도 카메라 모듈을 같은 방법으로 연결해야 한다는 점은 변하지 않았다. 그래서 카메라 모듈 대책으로서 FFC 커넥터가 통합된 보드, AI Thinker ESP32-CAM(esp32cam)을 새로 도입하였다.

우려사항: OV2640 FFC 케이블의 길이

이 보드에 있어 우려 사항은 보드의 위치에 따라 카메라의 위치가 결정된다는 점이었다. 국내에서 판매하는 사실상 모든 OV2640 카메라는 FFC 케이블의 길이가 짧으므로, 보드의 위치와 각도, 방향에 따라 카메라 모듈이 취할 수 있는 위치와 각도가 제한되었다. 국내에서는 카메라 전용 보드와 함께 제공하는 것이 주류이므로, 모듈과 기반 보드 사이 연결선의 길이는 전용 보드와 기반 보드 사이의 점프 케이블의 길이로 조정하는 것이 의도되었다. 이 문제를 해결하기 위해 해외에서 FFC 케이블의 길이가 긴 OV2640 모듈을 주문하였으나 프로젝트 기한 내에 배송이 완료되지 않았다.

Platform IO 마이그레이션

아두이노 IDE와 아두이노 CLI를 사용할 수는 있으나 제 3자 소프트웨어로서 동작하는 ESP32 계열 보드들이 아두이노 개발 환경에서 다소의 성능 문제와 이에 의해 유발되는 작업 과정 지연의 우려가 있었다. 또한 아두이노 소프트웨어로 불편한 CI/CD나 배치 작업이 더욱 부적절하게 되었다. 특히 배치 작업의 편의성은 다양한 환경과 기기를 사용하는 팀원에게는 더욱 중요한데, 아두이노 소프트웨어는 배치작업을 위해 추가로 작성해야하는 스크립트나 환경에 따라 수동으로 수행해야하는 작업이 존재하므로 다소 피로감이 있었다. 그 외에 아두이노 IDE에서 .ino파일의 이름과 부모 폴더의 이름이 같아야 하고, 같지 않으면 IDE가 임의로 이 규칙을 준수시키는 등의 작은 불편함도 있었다.

```
adafruit/Adafruit GFX Library@^1.11.11  
adafruit/Adafruit SSD1306@^2.5.13
```

Listing 3: arduino/dependencies.txt

```
#!/usr/bin/env sh  
for each in $(cat ./dependencies.txt)  
do  
    exec "arduino-cli lib install $each"  
done
```

Listing 4: arduino/install-library

```
Get-Content ./dependencies.txt |  
ForEach-Object {  
    arduino-cli lib install "$_"  
}
```

Listing 5: arduino/install-library.ps1

```
[env:esp32cam]  
monitor_speed = 115200  
platform = espressif32  
board = esp32cam  
framework = arduino  
board_build.mcu = esp32  
board_build.f_cpu = 240000000L  
lib_deps =  
    SPI  
    WiFi @ 2.0.0  
    adafruit/Adafruit GFX Library@^1.11.11  
    adafruit/Adafruit SSD1306@^2.5.13
```

Listing 6: platformio/platformio.ini

PlatformIO로 마이그레이션한 후의 개선점 중 하나로, 다음과 같이 여러 개의 파일에 걸쳐 시스템별로 정의한 프로젝트 의존성 메타데이터를 다른 설정과 함께 한 번에 관리할 수 있게 되었다.

Wokwi 시뮬레이터 사용

동시에 Wokwi 시뮬레이터를 사용할 수 있게 되었다. Wokwi는 컴퓨팅 보드를 소스코드와 함께 시뮬레이션할 수 있는 소프트웨어로, 하드웨어가 실재하지 않더라도 소스코드의 런타임 오류를 확인할 수 있다. 시뮬레이터나 하드웨어 없이는 컴파일 오류만 검증할 수 있으나, C++에서 종종 발생하는 부적절한 메모리 관리나 Undefined Behavior에서 유발되는 시스템 패닉을 사전에 확인할 필요가 있었다.

학부과정 특성 상 고정된 장소보다는 여러 장소를 이동하면서 작업을 수행하게 되는데, 하드웨어를 항상 지참하고 다닐 수 있는 것은 아니므로 하드웨어를 지참하지 않더라도 코드를 디버깅할 수 있게 되는 것은 주목할만한 개선이다.

ESP32-CAM의 커널 패닉

esp32cam으로 보드를 변경하고 커널 패닉이 자주 발생하게 되었다. 아두이노에서 작업하던 시기에는 런타임 오류를 쉽게 발생하지 못해 런타임 오류에서 촉발되는 시스템 실패, 복구 과정에서의 시스템 재시작 상황을 경험하지 못했다.

하지만 esp32cam으로 전환한 시점에서 기반 보드 제어 코드가 꽤 복잡해졌고, 시스템이 실패할 시나 리오가 더 많아졌으므로 시스템이 자주 실패하고 재시작되었다. 이것은 코드 자체의 규모가 커졌기 때문으로, esp32 계열 보드가 불안정한 것은 아니다.

시스템 실패/전원 부족

esp32cam은 5V 직류 전원을 GPIO로 입력받으며, BLE가 활성화되어있을 때 발신시 130mA, 수신시 95~100mA의 전류를 요구한다. ESP32-CAM-MB를 사용하지 않고 Arduino Uno와 WROVER 형식

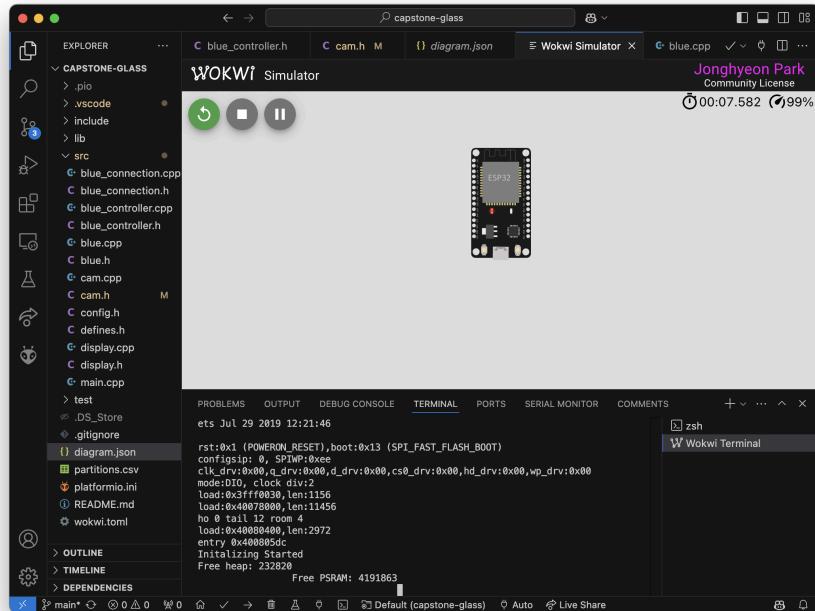


Figure 6: Wokwi 시뮬레이터의 동작

Work Mode	Min	Typ	Max	Unit
Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm	-	240	-	mA
Transmit 802.11g, OFDM 54 Mbps, POUT = +16 dBm	-	190	-	mA
Transmit 802.11n, OFDM MCS7, POUT = +14 dBm	-	180	-	mA
Receive 802.11b/g/n	-	95 ~ 100	-	mA
Transmit BT/BLE, POUT = 0 dBm	-	130	-	mA
Receive BT/BLE	-	95 ~ 100	-	mA

Table 1: ESP32 RF Current Consumption Depending on RF Modes[9]

으로 연결할 경우 전원 요구사항을 충분히 충족하지 못해 핀의 연결 상황에 따라 시스템 실패가 발생하였다.

메모리 부족

코드 규모가 증가함에 따라 기본 설정된 파티션 프리셋으로는 제어 코드가 제대로 동작하지 않는 문제가 발생하였다. The Last Outpost Workshop의 플래시 메모리 파티션 빌더를 이용하여 파티션을 커스터마이징해야했다.[10]

런타임 오류

부적절한 메모리 관리와 Undefined Behavior에서 런타임 오류가 유발되어 커널 패닉이 발생하였다. 하지만 시리얼 통신으로 출력되는 오류 메시지는 바이너리로 표현되는 코어 덤프뿐이었다.

Platform IO가 제공하는 디버깅 옵션이 존재할 것이라고 생각하지만, 프로젝트 마감까지 남은 기간이 새 도구를 찾아보고 학습하고 익히는데 걸리는 시간을 유의미하게 고려해야 할 정도로 거의 남지 않았기 때문에 새 도구를 더 도입하는 것은 고려하지 않았다. 패닉을 일으킬 것으로 의심되는 코드 부분 앞에 return을 작성하는 등의 방법으로 직접 중단점을 설정하여 버그 패턴을 분석하였다.

2.2) 클래스: 모델 구현과 시행착오, 변화

최초 구상 단계에서 클래스는 투명 OLED 디스플레이와 소형 리튬이온 배터리를 포함하여 독자적으로 작용할 수 있는 기기로 설계되었다.

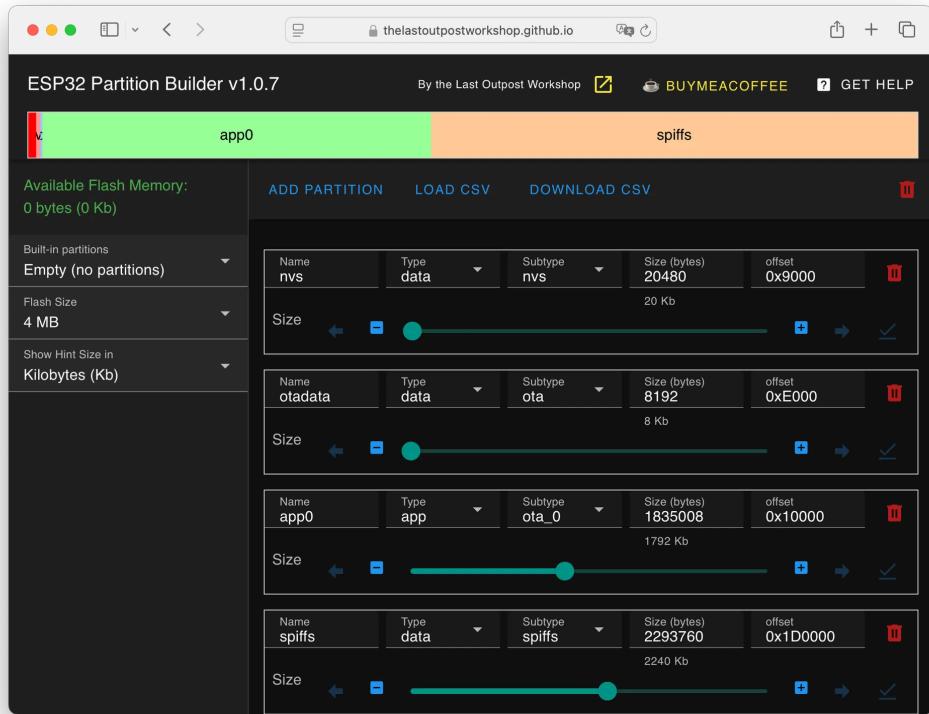


Figure 7: 플래시 메모리 파티션 빌더[10]

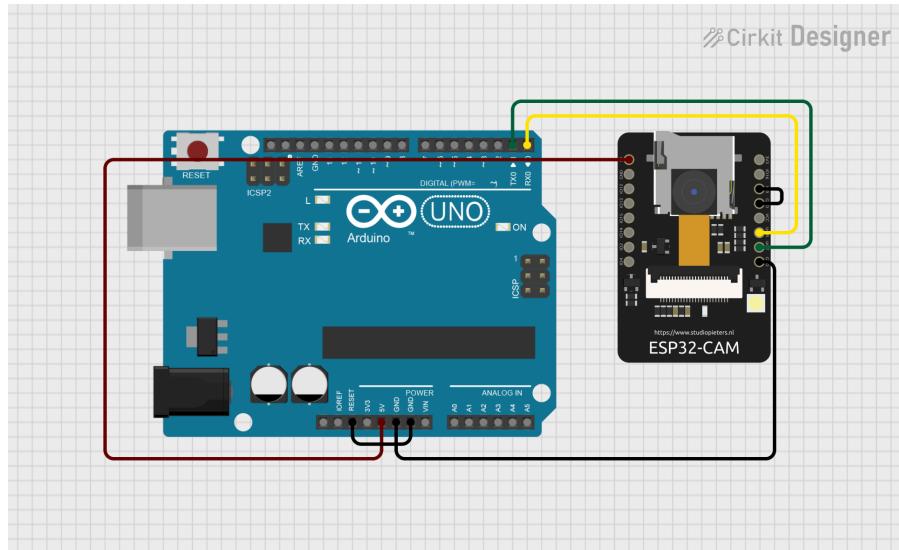


Figure 8: Arduino Uno와 ESP32-CAM의 WROVER 연결

하지만 아이디어를 구현하는 과정에서 투명 OLED의 배송이 매우 지연되었다. 배송 지연 대책으로 투명 디스플레이를 구현하는 다른 방법으로 DD ElectroTech의 아이디어를 참고하여 글래스 내부를 어둡게 만든 후, 일반 OLED 디스플레이의 내용물이 내부로부터 반사되어 바깥의 아크릴에 표시되도록 하는 것으로 구현 방식을 변경하였다.[11] 아이디어 변경의 계기가 된 자료의 신빙성이 매우 부족하였으므로 다소의 CG나 데이터 조작이 있을 가능성이 제기되었으나, 다소의 프로젝트 자료가 공개되어 있었으므로 신빙성 문제는 실험을 통해 검증하기로 하였다.

2.2.1) 1차: Thingiverse 오픈소스 모델 사용

초기 단계에서는 adafruit과 bshugs13의 험메이드 구글 글래스 모델을 활용하여 글래스의 기본 구조를 확정하였다. [12], [13] 1차 모델은 각 모델의 실 치수와 모듈 호환성을 확인하기 위해서 사용한 것이므로, 각 모듈의 호환성은 고려되지 않았다. 모델은 우려보다 잘 호환되었으나, 디스플레이가 사용자의 시야를 크게 방해하는 문제가 있었다.

```
ELF file SHA256: d3c95733ba82e342
```

```
E (285) esp_core_dump_flash: Core dump flash config is corrupted! CRC=0x7bd5c66f instead  
of 0x0 E (293) esp_core_dump_elf: Elf write init failed! E (297) esp_core_dump_common:  
Core dump write failed with error=-1 Rebooting... ets Jun 8 2016 00:22:57
```

```
rst:0xc (SW_CPU_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT) configsip: 0, SPIWP:0xee  
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00 mode:DIO, clock  
div:2 load:0x3fff0030,len:4832 load:0x40078000,len:16440 load:0x40080400,len:4 ho 8 tail 4  
room 4 load:0x40080404,len:3504 entry 0x400805cc Guru Meditation Error: Core 1 panic'ed  
(StoreProhibited). Exception was unhandled.
```

```
Core 1 register dump: PC : 0x400e53e3 PS : 0x00060030 A0 : 0x8008383d A1 : 0x3ffe7d50  
A2 : 0x3f42acac A3 : 0x00000002 A4 : 0x00000001 A5 : 0x3f42acc4  
A6 : 0x00000000 A7 : 0x00000000 A8 : 0x00000000 A9 : 0x3ffe7d50  
A10 : 0x00000000 A11 : 0x00000000 A12 : 0x00000000 A13 : 0x3ffe7d60  
A14 : 0x00000007 A15 : 0x00000006 SAR : 0x0000001f EXCCAUSE: 0x0000001d  
EXCVADDR: 0x00000080 LBEG : 0x40092a9c LEND : 0x40092ab2 LCOUNT : 0x00000000
```

```
Backtrace: 0x400e53e0:0x3ffe7d50 0x4008383a:0x3ffe7d80 0x40082bad:0x3ffe7da0  
0x40007c31:0x3ffe7dc0 0x4000073d:0x3ffe7e30
```

Listing 7: 런타임 오류에서 발생하는 커널 패닉 로그와 코어 덤프



Figure 9: 반사식 투명 디스플레이 아이디어[11]



Figure 10: adafruit의 모델을 수정하여 출력한 최초판 모델[12]

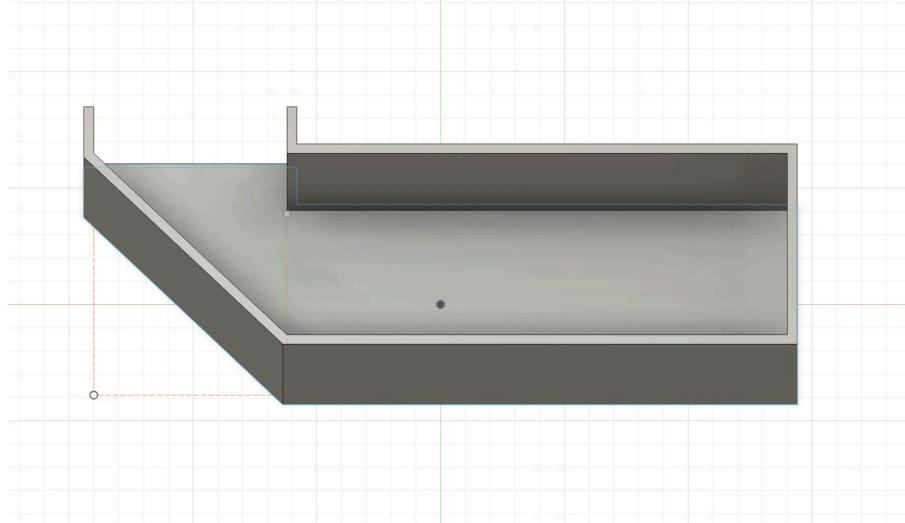


Figure 11: DD ElectroTech의 자료를 근거로 작성한 2차 모델

디스플레이 위치의 아크릴	굴절 유발 아크릴	플라스틱의 상태	실험 결과
2t 2cm×3cm	5t 2cm× 2cm	흰색	표시되지 않음
		검은색 도색	표시되지 않음
	8t 2cm× 2cm	흰색	표시되지 않음
		검은색 도색	표시되지 않음
3t 1cm×2cm	10t 2cm× 2cm	흰색	표시되지 않음
	5t 2cm× 2cm	흰색	표시되지 않음
	8t 2cm× 2cm	흰색	표시되지 않음

Table 2: 반사형 모델 아크릴 반사-굴절 실험 결과

2.2.2) 2차: 반사형 모델

2차 모델은 투명 OLED 대책으로서 대두된 반사식 투명 디스플레이 구현안[11], Figure 11 을 실제로 채택할 것인지 검토하기 위한 목적으로 사용하였다. 구체적으로는 디스플레이를 사용자에게 직접 노출시키는 대신 글래스 내부에서 디스플레이를 표시시키고, 거울로 반사시켜 아크릴에 표시되도록 하는 것이 골자이다.

글래스 외부에서 유입되는 빛의 영향으로 디스플레이 표시 내용의 반사와 표시가 제대로 이루어지지 않을 것이라는 우려가 있었다. [11] 에서 등장하는 구현체의 화면이 지나치게 뚜렷하게 표시되는 점을 들어 아크릴에 디스플레이가 표시되는 장면의 합성의 의심되기도 하였다. 위와 같은 우려를 확인하기 위해 실제로 이 구현안이 제대로 작동하는 것인지 확인하고자 실제로 구현하였다.

검증을 위해 다양한 각도와 아크릴을 이용해 반사 수준을 암실에서 실험하고, 모델로 출력하여 유의미한 결과가 나오는지 확인하였다. (Table 2, Figure 12)

그 결과 [11] 와는 상이한 결과를 확인할 수 있었다. 재료 정보와 구체적인 설계 정보를 획득할 수 없었으므로 잘못된 실험일 가능성성이 있었으나, 가능성이 낮다는 점과 구현이 잘못되었다면 실제로 활용할 때 까지 더 많은 시간을 투입해야 한다는 점을 근거로 반사형 모델을 폐기하였다.

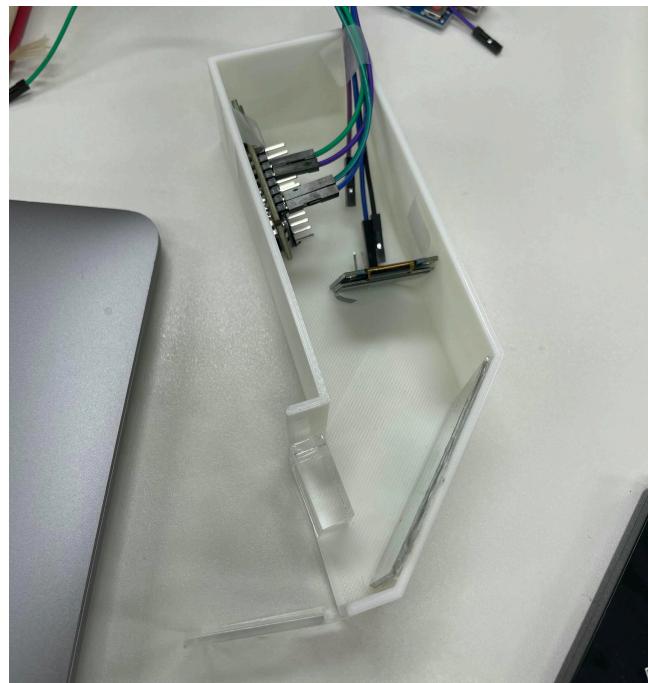


Figure 12: 반사형 모델의 아크릴 반사-굴절 실험 설정 과정(2t 2cm×3cm; 10t 2cm×2cm, 흰색)



Figure 13: 노출형 모델 리비전 1(좌), 리비전 2(우)

2.2.3) 3차: 노출형 모델

반사형 모델의 폐기 이후 다시 사용자에게 디스플레이를 직접 노출하는 방향으로 선회하였다. 최초 버전에는 들어갈 모듈 보드의 수와 구체적인 회로 위치, 배선이 확정되지 않았으므로 이전의 두 모델과 같이 시범적인 성격이 강했다.

이후 발전 과정에서 여러 모듈이 통합되는 보드를 사용하여 투입되는 보드를 줄이거나, 보드의 헤더 핀을 제거하고 전선을 납땜하여 모델의 물리적인 크기를 축소하였다. 카메라 노출을 위한 구성 요소와 글래스 모델을 안경에 부착할 수 있는 안경 걸이를 추가했다. (Figure 13, Figure 14, Figure 15)

2.3) 중계기: 사용하는 주 기술 스택의 의도되지 않은 사용 문제

중계기로서 사용할 기기의 시스템이 특정되지 않았으므로, 중계기 코드는 특정한 시스템에 종속되지 않도록 Dart(dart)와 Flutter(플러터) 프레임워크를 사용하여 작성하였다. 플러터는 하나의 코드베이스에 여러 시스템에 공통적으로 대응하도록 지원한다.[14]

전역 상태 관리 및 역방향 제어 대책



Figure 14: 노출형 모델 리비전 3의 실제 출력물



Figure 15: 노출형 모델 리비전 4의 실제 출력물, 조립

플러터는 애플리케이션 작성 프레임워크이면서 UI 프레임워크이므로, UI 설계에 있어 의도된 사용 방법이 있다. 하지만 이것 때문에 전역 상태 관리와 역방향 제어(IoC; Inversion of Control)의 구현 대책을 마련하기 어려웠다.

중계기 담당 팀원은 부모 UI 요소와 자식 UI 요소 사이에 데이터를 주고받는 양방향 UI 제어 설계에 익숙하지만, 플러터는 부모 UI 요소로부터 자식 UI 요소 방향으로 데이터가 이동하는 단방향 제어가 의도되었다. 예를 들어 텍스트 필드의 경우, 웹 기술에서는 부모 UI 요소에서 `textfield.value`와 같이 부모 요소에서 자식 요소의 상태 데이터를 가져온다. 혹은 `textfield.addEventHandler('onChange', () => {})`와 같이 부모 요소에서 자식 요소에게 콜백 루틴을 전달하여 대응할 수도 있다. 하지만 플러터는 단방향, 부모 요소에서 자식 요소로 데이터를 전달하므로, 역방향 제어를 위해서는 콜백 함수를 주입해야만 한다. 이 때문에 값을 교환해야하는 두 UI 요소의 깊이 차이가 커질 수록 콜백의 구성은 더욱 복잡해졌다.

중계 작업에서 시간이 많이 투입된 부분 중 하나는 이러한 역방향 제어의 대응 대책을 마련하는 것이었다. 다른 기술 스택에서 전역 상태관리 대책으로 자주 활용하는 방법은 싱글톤 객체를 두고 필요한 데이터를 참조하도록 하는 것이었으나, 플러터에서는 단순히 전역 상태 관리용 싱글톤 객체를 사용하는 것으로는 충분하지 않았다. 특히 플러터의 UI 요소는 요소의 상태 데이터를 싱글톤 객체의 참조형과 연결하는 것이 아니라 값을 그대로 복사하므로, 플러터에서는 싱글톤 객체의 값을 변경하더라도 값의 변경

```

class _UIWidgetState extends
State<UIWidget> {
void updateFromConfig() {
Config config = Config();
setState(() {
config.currentDevice.cachedName;
});
}
}

class UIWidget : public Widget {
UIWidget() {
Config config = Config::get();
this->title = config->current_device-
>cached_name;
}
};

@Override
void initState() {
super.initState();
updateFromConfig();
}

@Override
void navigate(BuildContext context,
MaterialPageRoute route) {
Navigator.of(context).push(route);
if (!mounted) return;
updateFromConfig();
}
}
}

```

Listing 8: C++(좌) 및 Flutte(우)에서의 전역 상태 설정 예시

```
[ERROR:flutter/runtime/dart_vm_initializer.cc(41)] Unhandled Exception:
PlatformException(setNotifyValue, neither NOTIFY nor INDICATE properties are supported by
this BLE characteristic, null, null)
```

Listing 9: iOS에서 setNotifyValue 호출 시 오류 메시지

을 전역적으로 고지하거나 각 UI 컴포넌트에서 직접 지속적으로 전역 싱글톤 객체로부터 값을 받아와야 하는 문제가 있었다.

2.4) 중계기: 플랫폼 별 파편화된 API의 통합

안드로이드와 iOS의 많은 부분이 공통적으로 구현되어 있다고 하더라도 두 시스템은 발전 과정에서 기능들이 일정 수준 수렴된 것이다. 두 시스템의 코드베이스는 상이하기 때문에 여전히 서로 다른 접근 방법으로 API를 사용하거나 특정한 기능을 구현해야 했다.

대표적으로 BLE Service의 Characteristic 값을 가져오는 데 있어서, 안드로이드 시스템에서는 Characteristic 값의 갱신을 고지하도록 설정할 수 있으나, iOS에서는 사용할 수 없었다.[15] 동일한 기능을 iOS에서도 제공하고 있음에도[16] 파편화에 제대로 대응하지 못한 것으로 보인다.

시스템 별 보안 대책과 권한 획득 방식도 상이하므로, 시스템 별 시스템 권한 획득 대책도 대응해야 한다. 백그라운드 작업과 블루투스 사용이 그러하였다. 이러한 권한 선언은 안드로이드 시스템의 경우 레거시 지원 코드도 포함하는 것이 일반적이어서 다양한 상황을 고려해야 했다.

iOS의 경우 테스트 앱도 애플 개발자 시스템으로부터 유효기간 7일의 인증서를 발급받아서 코드 서명을 수행해야 한다. 이 때 인증서가 포함할 앱 번들 식별자가 발급된 다른 인증서들과 중복되지 않아야 하므로, 작업 과정에서 테스트 코드나 스니펫을 실행시키기 위해 앱 번들 식별자를 수정하고 인증서를 재발급받는 등의 반복 작업이 요구되었다.

이와 같이 시스템 별로 고려해야 할 대응 상황이 존재하고, 경우에 따라 동일한 기능을 이중으로 구현해야 할 소요가 있었다. 한쪽 시스템을 기준으로 작성한 코드가 다른 시스템에서는 작동하지 않아 추가적인 디버깅 소요가 존재했다.

2.5) 클래스-중계기: 블루투스 통신 중 데이터 손실 문제

```

<!-- Tell Google Play Store that your app uses Bluetooth LE
     Set android:required="true" if bluetooth is necessary -->
<uses-feature android:name="android.hardware.bluetooth_le" android:required="false" />

<!-- New Bluetooth permissions in Android 12
https://developer.android.com/about/versions/12/features/bluetooth-permissions -->
<uses-permission android:name="android.permission.BLUETOOTH_SCAN"
    android:usesPermissionFlags="neverForLocation" />
<uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />

<!-- legacy for Android 11 or lower -->
<uses-permission android:name="android.permission.BLUETOOTH" android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"
    android:maxSdkVersion="30" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
    android:maxSdkVersion="30" />

<!-- legacy for Android 9 or lower -->
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
    android:maxSdkVersion="28" />

```

Listing 10: 안드로이드에서의 블루투스 사용 권한 요청 선언

```

<key>NSBluetoothAlwaysUsageDescription</key>
<string>Need BLE permission</string>
<key>NSBluetoothPeripheralUsageDescription</key>
<string>Need BLE permission</string>

```

Listing 11: iOS에서의 블루투스 사용 권한 요청 선언

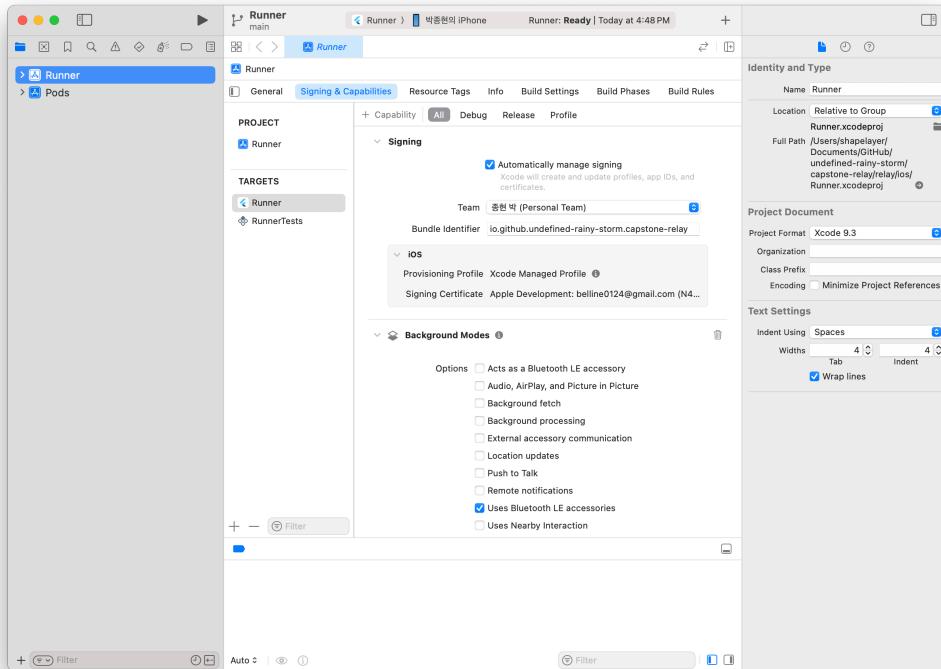


Figure 16: XCode, 테스트 앱 서명 유필리티

BLE 통신을 통해 esp32cam의 촬영 데이터를 중계기에 전송하는 과정에서 촬영 데이터가 소실되는 문제가 발생한다. 이 문제는 블루투스 규격 자체의 하자나 물리적 요인에 의해 발생하는 것은 아니다.

BLE 규격상 각 Characteristic은 한 번에 512바이트까지만 Notify할 수 있지만[17] 클래스의 촬영 해상도는 qVGA(quarter VGA; 320 × 240)으로 픽셀 당 1바이트 값을 가지더라도 프레임 당 76,800바이트를 가진다. 따라서 이를 512바이트로 분할해서 전송해야 한다.

```

...
try {
  if (Platform.isAndroid) {
    await FlutterBluePlus.turnOn();
  }
} catch (e) {
  print(e);
}
...
if (Platform.isIOS) {
  await FlutterBluePlus.adapterState
    .where((state) => state == BluetoothAdapterState.on)
    .first
    .timeout(
      const Duration(seconds: 10),
      onTimeout: () => throw BluetoothAdapterStateIsNotOnException(),
    );
}
if (Platform.isAndroid) {
  await FlutterBluePlus.turnOn();
}

```

Listing 12: 시스템 별 파편화된 기능의 대응 코드

```

E/flutter (16221): [ERROR:flutter/runtime/dart_vm_initializer.cc(41)] Unhandled Exception:
PlatformException(readCharacteristic, The READ property is not supported by this BLE
characteristic, null, null)
E/flutter (16221): #0      StandardMethodCodec.decodeEnvelope (package:flutter/src/
services/message_codecs.dart:648:7)
E/flutter (16221): #1      MethodChannel._invokeMethod (package:flutter/src/services/
platform_channel.dart:334:18)
E/flutter (16221): <asynchronous suspension>
E/flutter (16221): #2      FlutterBluePlus._invokeMethod (package:flutter_blue_plus/src/
flutter_blue_plus.dart:630:13)
E/flutter (16221): <asynchronous suspension>
E/flutter (16221): #3      BluetoothCharacteristic.read (package:flutter_blue_plus/src/
bluetooth_characteristic.dart:133:7)
E/flutter (16221): <asynchronous suspension>
E/flutter (16221):
D/[FBP-Android](16221): [FBP] onMethodCall: readCharacteristic

```

Listing 13: 안드로이드 빌드에서 단시간에 BLE Characteristic READ를 호출했을 때 오류

```

/// read a characteristic
Future<List<int>> read({int timeout = 15}) async {
  ...
  // Only allow a single ble operation to be underway at a time
  _Mutex mtx = _MutexFactory.getMutexForKey("global");
  await mtx.take();
  ...
  try {
    ...
  } finally {
    mtx.give();
  }
}

```

Listing 14: flutter_blue_plus의 데이터 수신 메서드 중 뮤텍스 사용 부분

하지만 현재의 중계기 구현체에서 사용하는 flutter_blue_plus 라이브러리는 각 시스템 별 API를 여러 쪽에 걸쳐 래핑하였고, 수신 루틴에서 뮤텍스를 사용하고 있었기 때문에 수신 루틴이 느리게 동작했다. 클래스에서 수십 밀리초 내에 쪼개진 데이터를 연속해서 전송하도록 하고, 이를 중계기에서 읽어들이도록 하면 수신하는 함수의 호출 시점이 뒤로 밀려 데이터 손실이 발생하였다.

이어진 발전 과정에서 데이터 자체의 크기를 줄이고 중계기에서 처리기로의 HTTP 요청 과정에서 추가적인 처리 소요를 줄이고자, 클래스에서 중계기로 이미지 데이터를 base64로 인코딩하였는데, 누락 데이터 블록에 대한 fallback 대책을 마련하고 수정한 것이 아니어서 오히려 대부분의 상황에서 클래스로부터 수신한 데이터를 이미지 프레임으로 디코딩하지 못하는 문제가 발생하였다.

이어서 fallback 루틴을 추가하고자 하여 추가 설계에서 제어용 신호를 주고받는 Characteristic의 추가, fallback 루틴 코드의 작성과 같은 소요를 도출하였다. 이러한 소요로 미루어보아 fallback을 위해 추가하고 수정해야하는 코드베이스의 양이 남은 과제 수행 기간 내에 완성되기 어렵다고 판단하였다. 따라서 이후의 후속 연구 목표로 설정하기로 하였다.

2.6) 처리기: 감정 분류 AI 학습

처리기가 투입된 이미지에서 사람의 얼굴을 추출하고 얼굴에 나타나는 감정을 분류할 수 있어야 한다. 얼굴 추출에는 Paul Viola 등의 Haar Cascade 검출기[18]를 사용하였고, 감정 분류기는 직접 모델을 구현하여 AffectNet[19]으로 학습시켰다.

처리기 AI 모듈 담당 팀원의 컴퓨팅 환경이 대규모 데이터 셋을 모두 처리하기 어려움에 따라 Google Colab(코랩) Pro Plus 플랜을 구입하여 사용하였다.

Haar Cascade 검출기 튜닝

Haar Cascade 분류기의 주요 매개변수는 다음과 같이 설정하였다.

- `scaleFactor = 1.1`: 이미지 크기를 10%씩 축소하며 탐지
- `minNeighbors = 6`: 얼굴 후보를 검증하기 위한 최소 이웃 개수
- `minSize = (48, 48)`: 탐지할 얼굴의 최소 크기

탐지된 얼굴 영역에 추가적인 전처리를 더해 모델에 적합한 입력 형식으로 변환했다. 우선 눈썹이나 턱선과 같은 얼굴의 주요한 특징들이 잘 포함되도록 탐지된 얼굴 좌표에 다음의 오프셋을 추가하여 얼굴 영역을 조정, 확장하였다.

- 가로 오프셋: 얼굴 너비 w 의 7.5% (`offset_coefficients[0] = 0.075`)
- 세로 오프셋: 얼굴 높이 h 의 5% (`offset_coefficients[1] = 0.05`)

데이터셋 전처리 및 모델 입력 조정

코랩 플랜을 구입하였더라도 런타임 연결 세션 유지가 제대로 안되거나 연결한 구글 드라이브의 인증이 만료되는 등의 문제가 확인되어 모델의 원활한 학습이 어려웠다. 때문에 모델 학습 시간을 줄일 필요가 있었고, 이미지의 채널 수를 축소하여 학습 시간을 줄이고자 했다.

3채널 컬러 입력 이미지를 1채널의 그레이 스케일로 변환하고, 크기를 (48, 48)로 조정하였다. 이어서 변환된 이미지 벡터를 [0, 255] 범위의 정수형 데이터에서 [0, 1] 범위의 float32 데이터로 정규화하였다.

감정 분류 모델 학습

모델 학습 성능을 향상시키기 위해 데이터를 증강하였다. 증강은 ImageDataGenerator를 사용하여 실시간으로 적용되도록 하였으며 학습 과정에서 다양한 변형 이미지를 생성하여 모델의 일반화 성능을 높였다.

최초의 감정 분류기 모델은 4개 유형의 레이어들로 구성되는 간단한 구조로 구성하였다. 하지만 이 모델의 정확도를 0.6 이상 끌어올릴 수 없었다. 때문에 Arriaga 등이 제안한 모델[20]을 참고하여 새 모델을 구성하였고, 정확도의 개선을 확인할 수 있었다.

2.7) 처리기: 감정 분류기 인터페이스 웹서버 작성

중계기는 클래스로부터 넘겨받은 이미지를 다시금 처리기에게 중계한다. 따라서 감정 분류기 모델의 입출력을 처리할 인터페이스를 구현하였다.

인터페이스는 Flask를 기반으로 한 RESTful API 웹 서버로 구현하였다. 중계기가 Base64 문자열로 인코딩된 이미지를 서버에 요청 파라미터로 전송하면, 처리기는 Base64 문자열을 이미지로 디코딩하

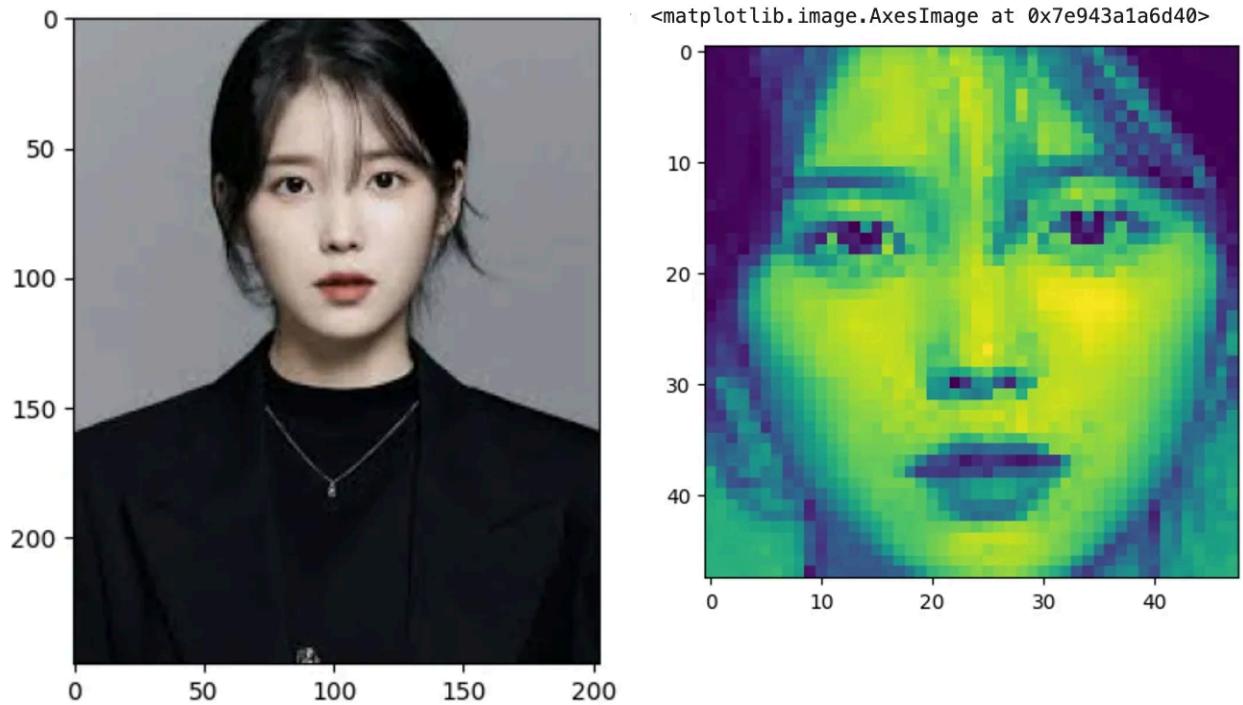


Figure 17: 전처리 이전의 원본 이미지(좌)와 전처리 규칙에 맞춰 처리된 이미지(우)

```
datagen = ImageDataGenerator(
    zoom_range=0.2,
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False
)
```

Listing 15: 이미지 데이터 증강기 설정값

Model: "functional_122"

Layer (type)	Output Shape	Param #
input_layer_15 (InputLayer)	(None, 48, 48, 1)	0
conv2d_78 (Conv2D)	(None, 48, 48, 32)	320
conv2d_79 (Conv2D)	(None, 46, 46, 32)	9,248
batch_normalization_2 (BatchNormalization)	(None, 46, 46, 32)	128
max_pooling2d_39 (MaxPooling2D)	(None, 23, 23, 32)	0
dropout_39 (Dropout)	(None, 23, 23, 32)	0
conv2d_80 (Conv2D)	(None, 23, 23, 64)	18,496
conv2d_81 (Conv2D)	(None, 21, 21, 64)	36,928
max_pooling2d_40 (MaxPooling2D)	(None, 10, 10, 64)	0
dropout_40 (Dropout)	(None, 10, 10, 64)	0
conv2d_82 (Conv2D)	(None, 10, 10, 64)	36,928
conv2d_83 (Conv2D)	(None, 8, 8, 64)	36,928
batch_normalization_3 (BatchNormalization)	(None, 8, 8, 64)	256
max_pooling2d_41 (MaxPooling2D)	(None, 4, 4, 64)	0
dropout_41 (Dropout)	(None, 4, 4, 64)	0
flatten_13 (Flatten)	(None, 1024)	0
dense_26 (Dense)	(None, 512)	524,800
dense_27 (Dense)	(None, 8)	4,104

Total params: 668,136 (2.55 MB)
Trainable params: 667,944 (2.55 MB)
Non-trainable params: 192 (768.00 B)

Figure 18: 1차 모델

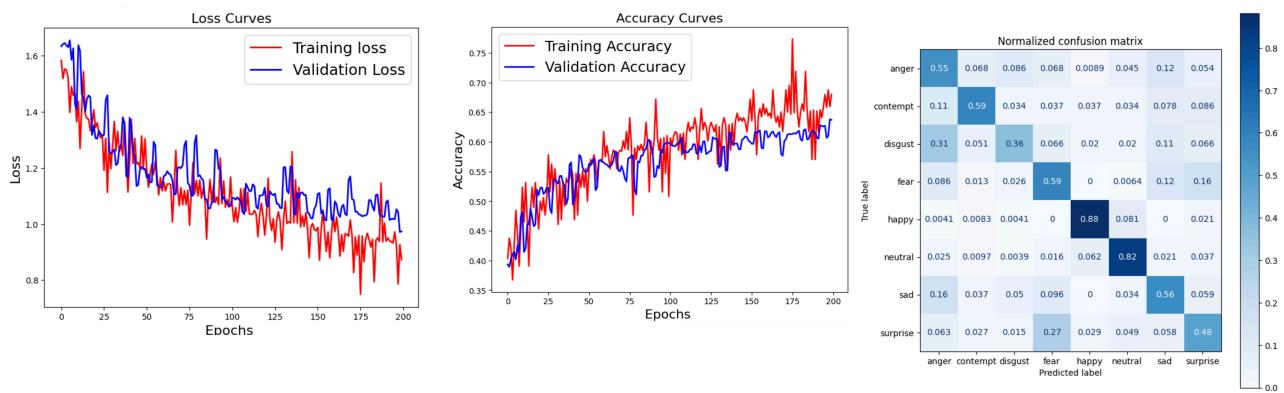


Figure 19: 1차 모델의 Loss, 정확도, 혼동행렬

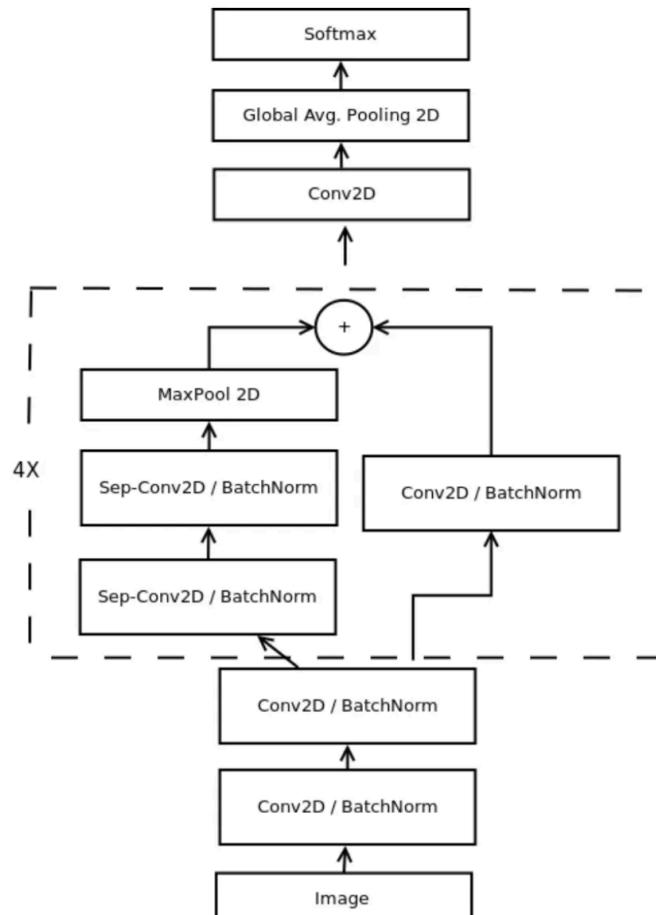


Figure 20: 2차 모델

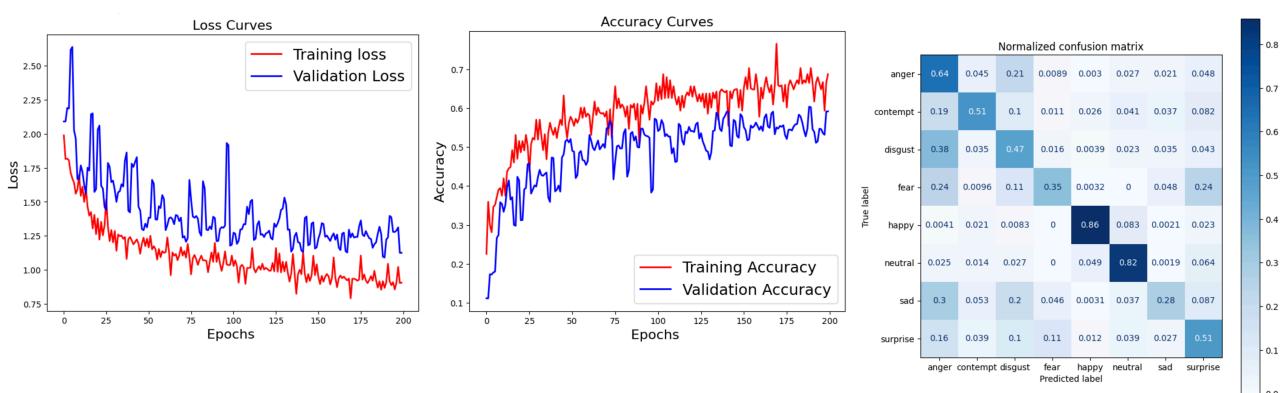


Figure 21: 2차 모델의 loss와 정확도, 혼동행렬

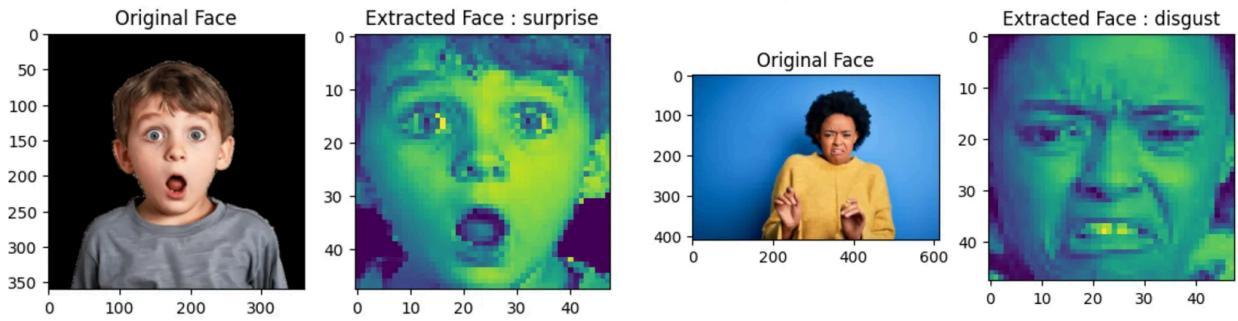


Figure 22: 2차 모델의 학습 결과 검증 갈무리

ANGRY (화남)	CONTEMPT (경멸)	DISGUST (혐오)	FEAR (공포)
HAPPY (행복)	NEUTRAL (중립)	SAD (슬픔)	SURPRISE (놀람)

Table 3: AffectNet의 감정 분류 [19]

```
{
  "image": "base64, /9j/4AAQSkZJRgABAQEAAAAAAAD... "
}
```

Listing 16: /query 엔드포인트에 POST 요청

```
{
  "status": "ok",
  "timestamp": "2024-12-10T12:00:00.000000",
  "result": "happy",
  "confidence": 0.95
}
```

Listing 17: /query 엔드포인트에 POST 요청 성공 시

여서 서버 로컬에 임시로 저장한다. Base64 이미지 데이터 문자열은 알파 채널을 갖는 PNG 이미지로부터 생성하므로 이 과정에서 RGBA 채널의 이미지를 RGB 채널의 이미지로 변환한다.

(Section 2.6) 이어서 OpenCV 패키지에 포함된 Caar Cascade 검출기를 사용하여 얼굴을 추출한 후, (48, 48)의 단일 채널 이미지 벡터로 변환한 후 정규화한다. 정규화한 벡터를 모델에 입력하여 감정을 분류하도록 한다.

모델의 출력물에서 컨피던스가 높은 감정 클래스와 분류된 클래스의 신뢰도를 Listing 17의 형식과 같은 RESTFUL JSON로 래핑하여 HTTP 요청에 반환한다.

반환하는 감정의 유형

모델은 [19] 만을 학습시키고, 클래스를 별도로 누락하지 않았으므로 [19] 의 분류와 동일하다. (Table 3)

API 엔드포인트

처리기 인터페이스는 모델의 입출력만을 담당하므로 단일한 엔드포인트만을 구현하였다.

- | |
|---|
| /query |
| • GET: 서버 상태를 타임스탬프와 함께 반환 |
| • POST |
| • 입력: Base64 문자열로 인코딩된 이미지 |
| • 출력: 감정 분류 결과 |
| • 자세한 규격은 Listing 16, Listing 17, Listing 18 참조 |

3. 연구과제의 수행 결과 및 목표 달성 정도

```

    }
    "status": "error",
    "timestamp": "2024-12-10T12:00:00.000000",
    "message": "image data not found"
}

```

Listing 18: /query 엔드포인트에 POST 요청 실패 시

연구과제의 수행은 다소의 성과를 거두었으나, 목표한 수준의 성과를 달성하지는 못했다. 특히 처리기와 중계기 사이의 상호작용, 중계기의 구현 부분에서 미흡함이 있었다. 그 중 처리기와 중계기 간 상호작용 과정의 하자(Section 2.5)는 전체 시스템의 품질을 현저히 떨어뜨리는 치명적인 문제였다.

#	항목	평가
1	클래스를 실제로 착용할 수 있는 수준으로 구현하였는가?	4/5
2	클래스가 디스플레이에 표시하는 내용을 쉽게 확인할 수 있는가?	2/5
3	클래스의 디스플레이는 전방 주시를 해치지 않는가?	2/5
4	클래스가 전방을 제대로 촬영할 수 있는가?	5/5
5	클래스가 촬영하는 이미지의 화질은 모델에 입력하기에 충분히 선명한가?	5/5
6	클래스의 처리 속도는 자연이 용인되는 수준으로 충분히 빠른가?	4/5
7	클래스는 중계기와 충분히 원활하게 상호작용하고 있는가?	2/5
8	클래스는 중계기에게 정확하게 데이터를 전송하고 있는가?	4/5
9	클래스는 중계기로부터 수신받은 데이터를 정확하게 표시하는가?	5/5

Table 4: 클래스 목표 달성 정도 자가평가

1. 클래스는 독자적으로는 착용 불가능하나, 안경에 설치하여 착용할 수 있다.
2. 디스플레이에 표시되는 내용은 초점이 맞지 않아 제대로 확인하기 어렵다. 제대로 확인하려면 디스플레이로 초점을 이동해야 하고 착용자는 전방을 확인하기 어려워진다.
3. 최초에는 투명한 디스플레이를 설치하여 전방 주시 요건을 만족하고자 하였으나, 발전 과정에서 투명한 디스플레이 대신 디스플레이 반사 방식을 도입하는 것으로 변경되었다. 이어서 디스플레이 반사 방식이 제대로 구현되지 않으면서 단순 디스플레이로 변경하였고, 최초 목표를 달성하지 못했다. 하지만 2.의 초점 문제로 인해 전방 주시를 현저히 해치지 않는다.
6. 클래스 자체의 처리 속도는 목표 요구사항을 충분히 달성할 수 있는 수준이다. (BLE Notify를 1ms에 1회 수행 가능; 카메라 촬영을 20ms에 1회 이상 수행 가능)
7. 중계기 구현 문제로 클래스는 중계기와 원활하게 상호작용하지 못하고 있다. Section 2.5

#	항목	평가
1	중계기는 안드로이드와 iOS 두 시스템에서 모두 동작하는가?	5/5
2	중계기는 클래스로부터 데이터를 정확하게 수신할 수 있는가?	0/5
3	중계기는 처리기로 HTTP 요청을 발신할 수 있는가?	4/5
4	중계기는 중계 과정의 모든 작업을 백그라운드에서 수행하는가?	4/5
5	중계기는 처리기로부터 반환되는 응답을 정확하게 수신할 수 있는가?	5/5
6	중계기는 처리기로부터 반환되는 응답을 클래스로 정확하게 발신할 수 있는가?	4/5

Table 5: 중계기 목표 달성 정도 자가평가

2. 중계기는 구현 문제로 클래스가 발신하는 BLE Characteristic 데이터를 모두 수신하지 못하고 있다. Section 2.5

#	항목	평가
1	처리기는 인공지능 처리가 가능한 다양한 시스템에서 일관되게 작동할 수 있는가?	5/5
2	처리기는 입력된 이미지의 사람 얼굴을 정확히 인식하는가?	4/5
3	처리기의 감정 인식 정확도가 높은가?	2/5
4	처리기의 중계기로부터 요청되는 값을 정확하게 수신할 수 있는가?	5/5
5	처리기의 중계기로 반환되는 응답을 정확하게 발신할 수 있는가?	5/5
6	처리기는 요청과 응답 사이 발생하는 다양한 예외 상황을 커버하고 있는가?	4/5

Table 6: 처리기 목표 달성 정도 자가평가

2. 처리기에서 얼굴을 인식하기 위해 사용된 Haar Cascade 분류기는 아주 낮은 확률로 그림자, 옷 문양 등을 사람 얼굴로 인식한다.
6. 처리기가 커버하고 있는 주요 예외 사항은 다음과 같다.
 - 요청 데이터에 이미지가 없는 경우.
 - 잘못된 Base64 이미지 데이터.
 - 임시 파일 저장 실패.
 - 이미지 처리 및 예측 중 발생하는 예외.
 - 기타 모든 예상치 못한 예외.

4. 연구 성과의 활용 방안 및 기대효과

당초 본 연구 성과는 아래와 같은 활용 방안과 기대효과를 기대하였다. 하지만 아직 구현물이 활용 방안을 실현하기에는 미흡하므로, 후속 연구 과정에서 아래 활용 방안을 담보할 수 있도록 구현을 발전시켜야 한다:

본 연구에서 개발되는 AR 감정 인식 보조 기기는 자폐 스펙트럼 장애를 가진 사람들의 삶의 질을 개선하고, 감정 인식 관련 기술 발전에 기여할 것으로 기대된다.

1. 자폐 증상 완화

실시간으로 타인의 감정을 인식할 수 있는 기술을 통해 자폐 환자들의 사회적 상호작용 능력이 향상되고, 이에 따라 자폐 증상이 완화될 가능성이 있다. 감정 인식 능력이 향상됨에 따라 자폐 환자들이 사회에서 더 원활하게 소통하고 관계를 형성할 수 있을 것이다.

2. 저비용, 고효율 솔루션

기존의 감정 인식 장치들은 비용이 높고 접근성이 떨어졌지만, 본 연구에서 개발되는 기기는 저비용으로 설계되어 많은 사람들이 쉽게 접근할 수 있다. 이로 인해 자폐 치료 분야에서 더 많은 사람들이 혜택을 받을 수 있을 것이다.

3. AI 기반 감정 인식 기술의 발전

AI와 최신 임베디드 기술을 활용한 감정 인식 알고리즘을 통해 기존 감정 인식 기술의 성능을 개선할 수 있다. 이는 자폐 치료뿐만 아니라 다양한 분야에서 감정 인식 기술이 응용될 수 있는 새로운 가능성을 열어줄 것이다.

4. 다양한 응용 가능성

연구 결과로 나온 기술은 자폐 치료 외에도 감정 인식이 중요한 여러 직업군 및 사회적 상황에서 응용될 수 있다. 예를 들어, 감정 노동자나 고객 서비스 분야에서 감정 인식을 지원하는 기술로 활용될 수 있으며, 이로 인해 직무 효율성과 스트레스 감소 효과가 기대된다.

4. 참고문헌

- [1] 자밀 자키(정지인 역), *공감은 지능이다.* 심심, 2021, pp. 1–2.
- [2] Arduino, “UNO R3.” [Online]. Available: <https://docs.arduino.cc/hardware/uno-rev3/>
- [3] Arduino, “Nano 33 IoT.” [Online]. Available: <https://docs.arduino.cc/hardware/nano-33-iot/>
- [4] OmniVision, “OV7670 Datasheet.”
- [5] adafruit, “ov7670.h.” [Online]. Available: https://github.com/adafruit/Adafruit_OV7670/blob/master/src/ov7670.h
- [6] R. P. Ltd, “Buy Raspberry Pi Zero.” [Online]. Available: <https://www.raspberrypi.com/products/raspberry-pi-zero/>
- [7] R. P. Ltd, “Raspberry Pi Zero 2 W Product Breif.”
- [8] 쿠팡 사용자, “유니팰리스 납땜용 용접 와이어 납땜 실납 1.0mm 50g 사용자 리뷰.” [Online]. Available: <https://www.coupang.com/vp/products/7803801064?itemId=21142173651&vendorItemId=88203691803>
- [9] ESPRESSIF, “ESP32 Series Datasheet Version 4.7.”
- [10] T. L. O. Workshop, “ESP32 Partition Builder.” [Online]. Available: https://thelastoutpostworkshop.github.io/microcontroller_devkit/esp32partitionbuilder/
- [11] D. ElectroTech, “How I Made my own Smart Glass Under \$10.” [Online]. Available: <https://www.youtube.com/watch?v=pkB1Nahi-X0>
- [12] adafruit, “DIY Video Glasses for Raspberry Pi.” [Online]. Available: <https://www.thingiverse.com/thing:302243>
- [13] bshugs13, “DIY Homemade Google Glass Parts.” [Online]. Available: <https://www.thingiverse.com/thing:505790>
- [14] “Flutter Official Website.” [Online]. Available: <https://flutter.dev/>
- [15] C. Weinberger, “setNotifyValue method.” [Online]. Available: https://pub.dev/documentation/flutter_blue_plus/latest/flutter_blue_plus/BluetoothCharacteristic/setNotifyValue.html
- [16] Apple, “setNotifyValue(_:for:).” [Online]. Available: [https://developer.apple.com/documentation/corebluetooth/cbperipheral/setnotifyvalue\(_:for:\)](https://developer.apple.com/documentation/corebluetooth/cbperipheral/setnotifyvalue(_:for:))
- [17] B. SIG, “Bluetooth Core Specification v5.3.” 2021.
- [18] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001, 2001, p. I–I.
- [19] A. Mollahosseini, B. Hasani, and M. H. Mahoor, “Affectnet: A database for facial expression, valence, and arousal computing in the wild,” IEEE Transactions on Affective Computing, vol. 10, no. 1, pp. 18–31, 2017.
- [20] O. Arriaga, M. Valdenegro-Toro, and P. Plöger, “Real-time convolutional neural networks for emotion and gender classification,” arXiv preprint arXiv:1710.07557, 2017.