10주차 (5/6~5/12)

⏰ **2024-05-06 (월) 오프라인 미팅 14:00 ~ 23:00**
2024-05-07 (화) 오프라인 미팅 07:00 ~ 18:00
2024-05-08 (수) 오프라인 미팅 14:00 ~ 19:00
총 활동 시간 : 25시간

(1) 아두이노

- 장애물 정지 알고리즘
  - 아두이노에서 서보 모터, 모터 드라이버, 초음파 센서를 사용하여 로봇을 제어하는 것입니다. 로봇은 거리를 측정하고, 특정 거리 이내의 장애물을 감지하면 멈춥니다.

```
// 서보 모터 제어를 위한 Servo 라이브러리 포함
#include <Servo.h>

// 핀 설정: 모터 드라이버 연결 핀
int IN1=A1;
int IN2=A0;
int IN3=A3;
int IN4=A2;

// PWM 핀 설정: 모터의 속도 제어를 위한 핀
int MotorLPWM=5; // 왼쪽 모터 PWM
int MotorRPWM=6; // 오른쪽 모터 PWM

// 초음파 센서 핀 설정
int inputPin = 9;  // 초음파 센서 에코 핀
int outputPin = 8; // 초음파 센서 트리거 핀

Servo myservo;  // 서보 객체 생성

void setup() {
    Serial.begin(9600); // 시리얼 통신 시작, 9600bps
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
```

```
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);
    pinMode(MotorLPWM, OUTPUT);
    pinMode(MotorRPWM, OUTPUT);
    pinMode(inputPin, INPUT);
    pinMode(outputPin, OUTPUT);

    // 서보 모터를 디지털 10번 핀에 연결
    myservo.attach(10);
}

void stopp() {
    // 모든 모터 입력 핀을 LOW로 설정하여 모터 정지
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, LOW);
}

void detection() {
        // 서보 모터를 90도 위치로 회전
    myservo.write(90);

    // 초음파 센서 신호 발생
    digitalWrite(outputPin, LOW);
    delayMicroseconds(2);
    digitalWrite(outputPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(outputPin, LOW);

    // 에코 핀에서 HIGH 신호의 지속 시간 측정
    float distance = pulseIn(inputPin, HIGH);
    // 거리 계산 (센티미터 단위)
    distance = distance / 58.0;

    if (distance < 25) {  // 거리가 25cm 미만이면 정지
        stopp();
    }
```

```
}

void loop() {
    detection();  // 반복적으로 거리 감지
}
```

- 트랙 이동 알고리즘
    - 센서를 사용하여 로봇의 움직임을 조정하는 데 사용됩니다. 센서 값에 따라 로봇이 전진, 후진, 좌회전 또는 우회전을 할 수 있습니다.

```
int IN1 = A1; // 모터 드라이버의 입력 핀 1
int IN2 = A0; // 모터 드라이버의 입력 핀 2

int IN3 = A3; // 모터 드라이버의 입력 핀 3
int IN4 = A2; // 모터 드라이버의 입력 핀 4

int L_IR = 1; // 왼쪽 적외선 센서 연결 핀
int R_IR = 4; // 오른쪽 적외선 센서 연결 핀

int MotorRPWM=6; // 오른쪽 모터 PWM 제어 핀
int MotorLPWM=5; // 왼쪽 모터 PWM 제어 핀

void setup()
{
    pinMode(IN1,OUTPUT); // 핀 모드 설정
    pinMode(IN2,OUTPUT);
    pinMode(IN3,OUTPUT);
    pinMode(IN4,OUTPUT);
    pinMode(L_IR,INPUT); // 센서 핀을 입력으로 설정
    pinMode(R_IR,INPUT);

    // 시리얼 통신 시작 (9600bps)
    Serial.begin(9600);
}

void loop()
{
```

```
  int L_VAL = digitalRead(L_IR); // 왼쪽 센서 값 읽기
  int R_VAL = digitalRead(R_IR); // 오른쪽 센서 값 읽기

  if((L_VAL == 0) && (R_VAL == 1)) // 왼쪽에 장애물이 있을 경우
  {
      motor_right(255);  // 오른쪽으로 회전
  }

  else if((L_VAL == 1) && (R_VAL == 0)) // 오른쪽에 장애물이 있을
  {
      motor_left(255);  // 왼쪽으로 회전
  }

  else if((L_VAL == 1) && (R_VAL == 1)) // 양쪽 모두 장애물이 없을
  {
      motor_forward(0); // 정지
  }

  else if((L_VAL == 0) && (R_VAL == 0)) // 양쪽 모두 장애물이 있을
  {
      motor_forward(255); // 전진
  }
}

void motor_forward(int SpeedVal)
{
    analogWrite(IN1,SpeedVal); // 모터 전진
    analogWrite(IN2,0);
    analogWrite(MotorRPWM,SpeedVal);

    analogWrite(IN3,0);
    analogWrite(IN4,SpeedVal);
    analogWrite(MotorLPWM,SpeedVal);
}

void motor_backward(int SpeedVal)
{
    analogWrite(IN1,0);
```

```
    analogWrite(IN2,SpeedVal); // 모터 후진

    analogWrite(IN3,SpeedVal);
    analogWrite(IN4,0);
    analogWrite(MotorRPWM,SpeedVal);
    analogWrite(MotorLPWM,SpeedVal);
}

void motor_right(int SpeedVal)
{
    analogWrite(IN1,0);
    analogWrite(IN2,SpeedVal); // 오른쪽 모터 후진으로 설정하여 오른
    analogWrite(MotorRPWM,SpeedVal);

    analogWrite(IN3,0);
    analogWrite(IN4,SpeedVal); // 왼쪽 모터 전진으로 설정
    analogWrite(MotorLPWM,SpeedVal);
}

void motor_left(int SpeedVal)
{
    analogWrite(IN1,SpeedVal); // 왼쪽 모터 전진으로 설정
    analogWrite(IN2,0);
    analogWrite(MotorRPWM,SpeedVal);

    analogWrite(IN3,SpeedVal); // 오른쪽 모터 후진으로 설정하여 왼쪽
    analogWrite(IN4,0);
    analogWrite(MotorLPWM,SpeedVal);
}

void motor_stop()
{
    analogWrite(IN1,0);
    analogWrite(IN2,0);

    analogWrite(IN3,0);
    analogWrite(IN4,0); // 모든 모터 정지
}
```

(2) 라즈베리파이와 아두이노 연동

- 사람의 얼굴이 인식되었을 때 로봇이 움직이도록 한다

```
import cv2
import numpy as np
import subprocess
import pyfirmata
import time

port = 'COM5'
board = pyfirmata.Arduino('/dev/ttyUSB0')
LED = board.get_pin('d:13:o')

process = subprocess.Popen(['libcamera-vid', '--codec', 'mjpe
                            stdout=subprocess.PIPE, bufsize=1

face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +

buffer = b''

try:
    while True:
        chunk = process.stdout.read(4096)
        buffer += chunk

        a = buffer.find(b'\xff\xd8')
        b = buffer.find(b'\xff\xd9')

        if a != -1 and b != -1:
            jpg = buffer[a:b+2]
            buffer = buffer[b+2:]

            image = cv2.imdecode(np.frombuffer(jpg, dtype=np.

            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

            faces = face_cascade.detectMultiScale(gray, scale
```

```
            for (x, y, w, h) in faces:
                cv2.rectangle(image, (x, y), (x+w, y+h), (0, 
                LED.write(1)
                time.sleep(1)
                LED.write(0)

            cv2.imshow('Face Detection', image)

            if cv2.waitKey(1) & 0xff == ord('q'):
                break

finally:
    process.kill()
    cv2.destroyAllWindows()
```

(3) 중간 실험

- 중간 실험 후 보고서 작성

실험 설계

1. **실험 환경**
    - 대학교 운동장 트랙
2. **참가자 선정**
    - 체육학과 학생 1명
3. **실험 날짜**
    - 2024년 5월 20일

실험 과정

1. **라즈베리파이 얼굴 인식**
    - 라즈베리파이가 얼굴을 인식하면 로봇이 움직이는지 확인합니다.
2. **트랙 따라가기**
    - 로봇이 트랙 위의 선을 정확히 따라가는지 테스트합니다.
3. **장애물 인식**

- 로봇이 장애물을 탐지하여 적절한 거리에서 멈추는지 확인합니다.

실험 결과

1. **라즈베리파이 얼굴 인식**
    - 라즈베리파이에서 얼굴 인식 시 로봇이 실시간으로 움직임을 확인할 수 있었습니다.
2. **트랙 따라가기**

- 로봇이 트랙 위의 선을 정확히 따라가지 못합니다.
- 라즈베리파이 카메라로 계속 얼굴 인식을 하고 있기 때문에 선을 인식하지 못해도 로봇은 움직입니다.
    - 문제 원인
        - 로봇의 구조상 무게가 한쪽으로 쏠려 직진을 하지 못하고 오른쪽으로 치우칩니다.
        - 라인트레이서 센서 두개 중 하나가 고장 나서 선을 제대로 인식하지 못합니다.

https://prod-files-secure.s3.us-west-2.amazonaws.com/de088e11-bc78-47d7-9a59-3df9ec2eadc2/b3d30c3f-e57d-4ff7-847c-374076dea350/%EB%8B%AC%EB%A6%AC%EA%B8%B0%EC%8B%9C%EC%97%B0.mp4

3. **장애물 인식**
    - 장애물을 잘 탐지하고 멈추나 너무 가까운 거리에서만 반응합니다.

https://prod-files-secure.s3.us-west-2.amazonaws.com/de088e11-bc78-47d7-9a59-3df9ec2eadc2/dafbc76b-3e2c-4645-8a23-f3d50c212af6/%EC%9E%A5%EC%95%A0%EB%AC%BC%EC%8B%9C%EC%97%B0.mp4

개선할 점

- **트랙 따라가기 정확도 개선**
    - 로봇의 구조를 바꿔 무게 중심이 맞도록 합니다.

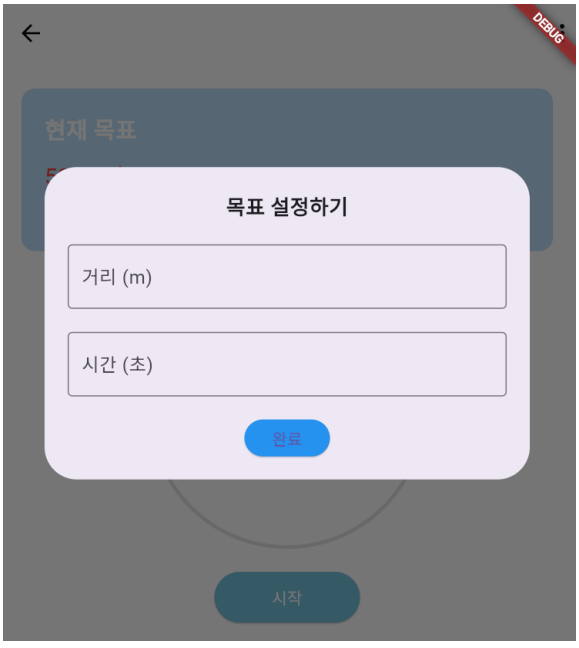
- 라인트레이서 센서를 교체하여 로봇이 트랙을 정확히 따라갈 수 있도록 합니다.

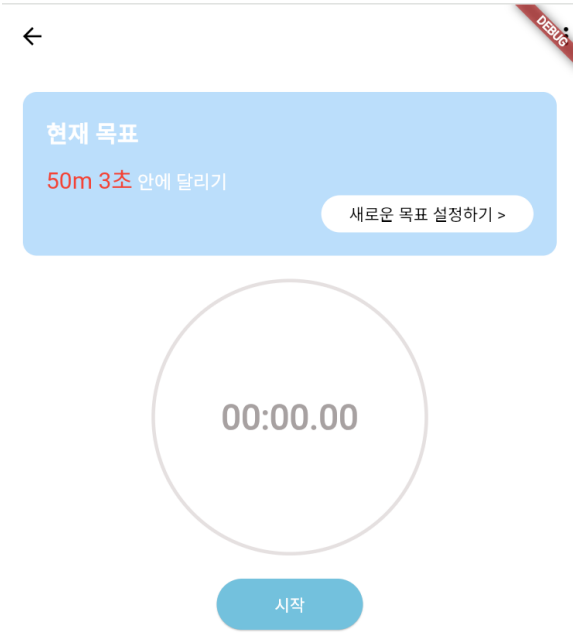
- **장애물 탐지 거리 조절**
  - 장애물 탐지 센서의 민감도를 조절하여 더 먼 거리에서도 반응할 수 있게 합니다.
- **실시간 얼굴 인식 성능 향상**
  - 얼굴 인식의 정확도와 반응 속도를 개선합니다.

추가사항

- **실험 데이터 로깅**
  - 실험 결과의 데이터를 수치화하여 로그로 남깁니다.
- **속도 및 거리 조절**
  - 사람과의 거리에 따른 로봇의 속도를 조절합니다.
- **목표 거리 달성**
  - 로봇이 목표 거리인 2km를 완주할 수 있도록 합니다.
- **어플리케이션 연동**
  - 로봇을 제어하기 위한 어플리케이션과의 연동을 구현합니다.

(4) 소프트웨어 - 타이머 기능 및 개인 목표 설정



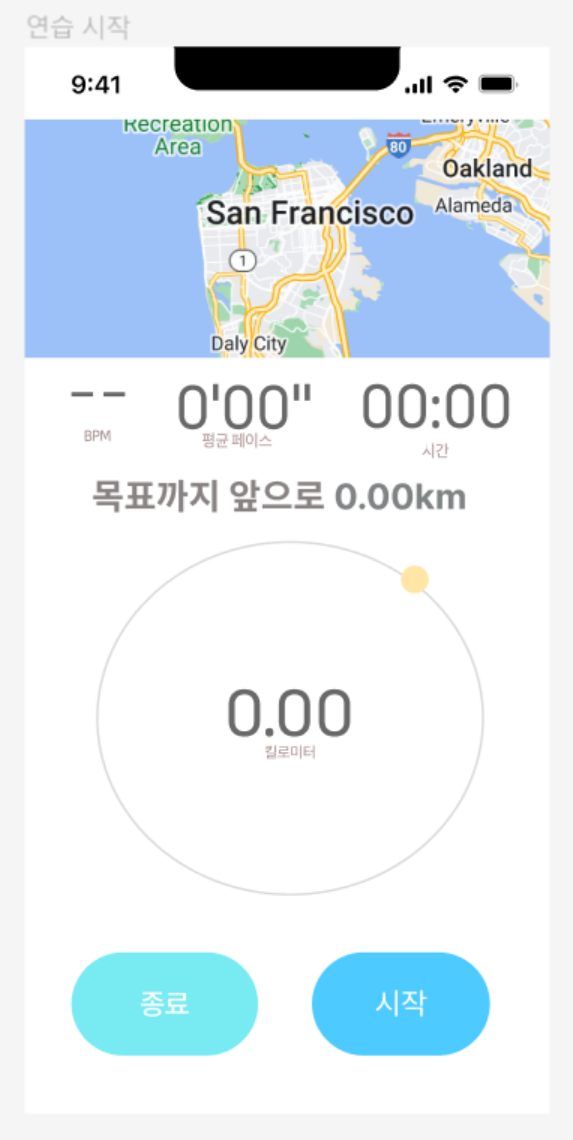


→ 원하는 목표 및 시간을 설정하면 어플에
뜨도록 하였습니다.

→ ui 수정을 위한 회의를 진행

- 시작 버튼을 누르면 타이머가 켜지며, 종료시 타이머 종료 후 기록됨.
- 취소 버튼을 누르면 타이머 중단.

(5) figma를 이용하여 추가 디자인



→ 타이머 시작 후 핸드폰/워치에 기록되는 평균 페이스, 타이머를 킨 시점부터 멈추는 시점까지의 킬로미터를 나타내는 페이지를 생성

→ 그 위에 google map을 이용하여 현재 사용자가 어디에 있는 지 위치를 나타내는 디자인

→ 사용자가 설정한 목표 km까지 얼마나 남았는지 알려주는 문구를 추가하였습니다.