14주차 (6/3~6/9)

⏰ **2024-06-03 (월) 오프라인 미팅 19:00 ~ 22:30**
2024-06-04 (화) 오프라인 미팅 11:30 ~ 20:00
2024-06-06 (목) 오프라인 미팅 13:30 ~ 21:30
2024-06-07 (금) 오프라인 미팅 15:00 ~ 20:00
총 활동 시간 : 25시간

(1) 로봇 보조배터리 연결

- 기존의 로봇은 AA 사용하여 구동되었으며 이는 비교적 짧은 시간 내에 전력이 소진되는 단점이 있었음

  이러한 문제를 해결하고자 보조배터리를 사용하여 보다 장시간 안정적으로 전원을 공급하는 방법을 적용하였음

- 개조 내용

  1. 건전지 홀더 제거
  2. DC 배럴잭 모터 드라이버와 연결
  3. 보조배터리 연결

⇒ 보조배터리를 사용하여 로봇의 구동 시간이 크게 늘어났으며 장시간 안정적으로 전원을 공급받을 수 있게 되었음

(2) 로봇 구동 테스트

로봇의 주요 기능들을 통합하고, 이를 실제로 구동하는 시험을 진행함

- 방향 틀기 : 모든 방향 전환 명령이 정확하게 수행됨
- 얼굴 인식 : 라즈베리파이 카메라로 얼굴이 인식되면 로봇의 DC모터가 작동함
- 장애물 인식 후 멈춤 : 로봇은 장애물을 감지한 즉시 멈춤
- 사용자와의 거리 조정 : 로봇이 사용자와의 거리를 측정하여 DC모터의 속도를 조절해 사용자와의 거리를 일정하게 유지함

```
void loop() {
  // 초음파 센서로부터 거리 측정
  int rightDistance = getDistance(rightTrigPin, rightEchoPin);
  int centerDistance = getDistance(centerTrigPin, centerEchoPin);
  int leftDistance = getDistance(leftTrigPin, leftEchoPin);

  // 만약 어느 하나의 센서에서 10cm 이내에 장애물이 감지된다면
  if (rightDistance < 20 || centerDistance < 20 || leftDistance < 20) {
    // DC 모터 멈추기
    digitalWrite(port3Pin, LOW);
    digitalWrite(port4Pin, LOW);
    analogWrite(enbPin, 0); // 모터 멈춤
  } else {
    // DC 모터 회전
    digitalWrite(port3Pin, HIGH);
    digitalWrite(port4Pin, LOW);
    analogWrite(enbPin, 80); // 모터 속도 설정
  }
```

```
if (currentMillis - lastServoChange >= 5000) { // 5초마다 서보모터 각도 변경
  lastServoChange = currentMillis;
  static bool servoAtZero = false;

  if (servoAtZero) {
    servo.write(90);
  } else {
    servo.write(0);
  }

  servoAtZero = !servoAtZero;
}

// 특정 조건에서 시스템 종료: 서보모터, DC 모터, 초음파 센서 멈춤
// 여기서는 예시로 30초 후에 멈추도록 설정
if (currentMillis >= 30000) {
  stopAll();
}
```

```
// 초음파 센서로부터 거리를 측정하는 함수
int getDistance(int trigPin, int echoPin) {
  // 트리거 핀을 HIGH로 유지하여 초음파 송신
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  // 에코 핀으로부터 초음파 수신 시간 측정
  long duration = pulseIn(echoPin, HIGH);

  // 거리 계산 (음속 340m/s 기준)
  int distance = duration * 0.034 / 2;

  return distance;
}
```

- **거리 측정 :**
  - `getDistance` 함수를 사용하여 거리를 측정

    - `rightDistance`, `centerDistance`, `leftDistance` 변수에 각각 거리를 저장
- **장애물 감지 및 DC 모터 제어**:
    - 세 개의 초음파 센서 중 하나라도 장애물이 감지되면 DC 모터를 멈춤 (`analogWrite(enbPin, 0)`)
    - 그렇지 않으면 DC 모터를 전진시키며 속도를 설정 (`analogWrite(enbPin, 80)`)
- **서보 모터 제어** :
    - `currentMillis`와 `lastServoChange`를 비교하여 서보 모터의 각도를 변경
    - `servoAtZero` 변수에 따라 서보 모터 각도 설정
    - 서보 모터의 각도를 변경한 후 `servoAtZero` 변수를 반전
- **시스템 종료**:
    - `currentMillis`가 `stopAll()` 함수를 호출하여 시스템을 종료
        - 이 함수는 서보 모터, DC 모터, 초음파 센서 모두를 멈춤
- **초음파 센서를 통한 거리 계산**:
    - 초음파 센서에서 트리거 핀을 통해 초음파를 방출하고 에코 핀을 통해 수신된 시간을 측정
    - `pulseIn` 함수를 사용하여 에코 핀에서 신호를 수신하는 시간을 측정하고, 이를 기반으로 거리를 계산

(3) 블루투스 연결

플러터 앱에 BLE 통신 코드를 추가함

- BLE 모듈이 배송되기 전에 진행하여서 HM-10모듈 대신 ESP32 모듈과 공기계를 사용해 블루투스 검색 및 연결이 되는지 까지 확인했음

1. **사용자 인터페이스 (UI)**
    - 사용자가 달릴 거리와 목표 시간을 입력할 수 있는 텍스트 필드와 버튼을 제공하는 간단한 UI를 만듭니다.
    - 입력한 데이터를 ESP32 장치로 전송할 수 있는 버튼을 제공합니다.
2. **블루투스 통신**

- FlutterBlue 라이브러리를 사용하여 블루투스 장치와 연결하고 데이터를 주고받는 기능을 포함합니다.
- ESP32 장치와 연결하고 데이터를 전송할 수 있습니다.

▼ UI 구성

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: Text('ESP32 통신'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          TextField(
            controller: distanceController,
            decoration: InputDecoration(labelText: '달릴
          ),
          TextField(
            controller: timeController,
            decoration: InputDecoration(labelText: '목표
          ),
          TextButton(
            onPressed: _sendData,
            child: Text('전송'),
          ),
        ],
      ),
    ),
  );
}
```

▼ 데이터 전송

```
void _sendData() {
  String distance = distanceController.text;
```

```
  String time = timeController.text;

  String data = '$distance:$time';

  if (characteristic != null) {
    characteristic.write(data.codeUnits);
  }
}
```

▼ **블루투스 장치 연결**

```
void _connectToDevice() async {
  List<BluetoothDevice> devices = await flutterBlue.con
  for (BluetoothDevice dev in devices) {
    if (dev.name == 'ESP32') {
      device = dev;
      break;
    }
  }

  if (device == null) {
    List<ScanResult> results =
        await flutterBlue.scan(timeout: Duration(second
    for (ScanResult result in results) {
      if (result.device.name == 'ESP32') {
        device = result.device;
        break;
      }
    }
  }

  if (device != null) {
    await device.connect();
    List<BluetoothService> services = await device.disc
    for (BluetoothService service in services) {
      for (BluetoothCharacteristic characteristic
          in service.characteristics) {
        if (characteristic.uuid.toString() == 'YOUR_CHA
```

```
          this.characteristic = characteristic;
          break;
        }
      }
    }
  }
}
```

▼ 상태 초기화 및 해제

```
@override
void initState() {
  super.initState();
  _connectToDevice();
}

@override
void dispose() {
  super.dispose();
  device?.disconnect();
}
```