주요 구성요소:

- **Python (라즈베리파이)**
  - 실시간 영상 처리 및 YOLO 탐지
  - ORB 기반 개체 식별
  - 결과 및 사료 정보 아두이노로 송신
- **Arduino (급식기 제어부)**
  - 시리얼 통신 수신
  - 체급별 모터 동작
  - HX711 로드셀을 통한 실시간 무게 측정

---

2. Python 측 주요 코드 구조

(1) detect_dog.py

- YOLOv8 을 사용하여 실시간으로 카메라 입력 프레임을 분석하고, class 16(dog)에 해당하는 객체만 탐지
- 탐지된 강아지의 ROI(영역)를 embedding_utils.py 로 전달하여 ORB 매칭 수행
- 가장 유사도가 높은 등록 이미지를 찾으면 이름, 체급 등의 정보를 pet_db.json 에서 불러와 시각화 및 송신

핵심 동작 요약:

1. YOLO 로 강아지 검출
2. ORB 로 등록 이미지와 매칭 → 신뢰도 계산
3. 매칭된 경우 이름, 크기, 신뢰도 표시
4. Arduino 로 {name, score, size, weight, activeLvl, calPerKg, feedingCount} 데이터 전송

화면 시각화 항목:

- 매칭 성공: 초록색 박스 + "Match: {이름}, Confidence: {신뢰도}"

- 매칭 실패: 빨간색 텍스트 "Match: None | Confidence: 0%"
- YOLO 탐지 결과가 없을 경우에도 화면 상단에 "None"이 항상 표시되도록 개선

---

(2) embedding_utils.py

- ORB(Oriented FAST and Rotated BRIEF) 알고리즘으로 특징점을 추출하고, 등록 이미지들과 비교하여 유사도를 계산
- 등록 이미지 폴더(pet_images)에서 각 강아지별 특징 디스크립터를 미리 로드
- 카메라에서 얻은 ROI 와 매칭하여 가장 유사한 강아지 이름과 점수를 반환

설명:
ORB 는 이미지의 특징점(코너, 경계 등)을 숫자로 추출해 다른 이미지와 비교하는 알고리즘이다.
SIFT 나 SURF 보다 가볍고, 실시간 임베디드 환경(라즈베리파이 등)에 적합하다.

---

(3) pet_db.json

각 강아지의 메타데이터를 저장한다. 예시:

```
{
  "coco": { "size": "small", "weight": 5.2, "activeLvl": 1.4, "calPerKg": 3500, "feedingCount": 2 },
  "bori": { "size": "medium", "weight": 12.8, "activeLvl": 1.6, "calPerKg": 3600, "feedingCount": 2 }
}
```

---

3. Arduino 측 코드 구조

(1) main.cpp

- Python 으로부터 전송된 문자열 데이터를 파싱
- 체급별로 사료량을 계산하고 해당 모터 구동
- 모터 작동 시작 시점을 타임스탬프(t_motor)로 기록하여 반응 시간 분석에 활용

```
unsigned long t_motor = 0;

if (size == "small") {
    t_motor = dispenseFoodWithTime(motorSmall, portion);
} else if (size == "medium") {
    t_motor = dispenseFoodWithTime(motorMedium, portion);
} else if (size == "large") {
    t_motor = dispenseFoodWithTime(motorLarge, portion);
}

Serial.print("[METRIC] Motor start time (t_motor) = ");
Serial.print(t_motor);
Serial.println(" ms");
```

---

(2) motor_control.cpp / motor_control.h

- 체급별 모터를 초기화하고 목표 무게에 도달할 때까지 사료를 추가 배출
- 목표량(targetGrams)에 도달하면 모터를 정지하고 완료 로그를 출력

```
void dispenseFood(int motorPin, float targetGrams) {
    while (true) {
        float currentWeight = getSuperStableWeight();
        float diff = targetGrams - currentWeight;

        if (diff <= 2.0) {
            digitalWrite(motorPin, LOW);
            break;
        } else {
```

```
                digitalWrite(motorPin, HIGH);
                delay(500);
                digitalWrite(motorPin, LOW);
                delay(500);
            }
        }
}
```

---

(3) 정량지표 측정 항목

- **t_start:** 강아지가 카메라 기준 20cm 이내로 접근한 시점 (수동 기록)
- **t_motor:** 모터가 실제로 동작하기 시작한 시점 (코드에서 자동 기록)
- **Δt = t_motor - t_start:** 반응시간
- **평균 반응시간:** (ΣΔt) / 9 — 9 마리 각각 1 회 측정 결과로 평가

---

4. 라즈베리파이 ↔ 아두이노 통신 구조

전송 방향:
Python (라즈베리파이) → Arduino (급식기 제어부)

프로토콜:
시리얼 통신 (9600bps)

{name},{score},{size},{weight},{activeLvl},{calPerKg},{feedingCount}₩n

데이터 예시:

coco,93,small,5.2,1.4,3500,2

역할:

- 라즈베리파이: 인식, 매칭, 정보 생성
- 아두이노: 모터 제어, 무게 확인, 반응 시간 기록

---

5. 카메라 인식 및 시각화 결과

- YOLOv8 으로 탐지된 객체 프레임마다 inference 시간(ms) 표시
  - 예시: 0: 384x640 1 dog, 37.7ms
  - 0: → 첫 번째 프레임
  - 384x640 → 입력 이미지 크기
  - 1 dog → 탐지된 객체 개수
  - 37.7ms → YOLO 추론 소요 시간
- 출력 예시:
- (프레임, 이미지 크기, 탐지된 객체 수, 추론 시간, 전처리·후처리 시간)
- = (0, 384x640, 1 dog, 37.7ms, Speed: 2.5/33.1/1.9ms)

---

6. 로드셀 캘리브레이션 보정 전략

모듈: hx711_calc.cpp

문제점:
로드셀은 시간이 지나면 온도, 진동, 장시간 하중 등으로 인해 **영점(Offset)**과 **스케일(Scale)**이 변함.
정확한 급식량 제어를 위해 자동 보정 로직이 필요하다.

방법 1 (추천): 자동 보정 알고리즘

1. 초기 설정 시 그릇(27g)을 올려둔 상태로 0 점 및 기준추 보정
2. 보정값을 EEPROM 에 저장
3. 재부팅 시 측정값이 27g±3g 범위에 있으면 기존 값 유지
4. 범위를 벗어나면 자동 재보정 수행
   → 환경 변화에도 항상 기준 그릇 무게 중심으로 보정 유지

방법 2: 다회 평균 보정

- 테스트 환경에서 캘리브레이션 값을 다수(예: 50 회) 측정

- 평균값을 산출해 EEPROM 에 저장
  → 통계적으로 안정적인 스케일 팩터 확보 가능