Yolo-Tiny를 활용한 농작물 상태 식별 스마트 카봇

김지현 배문규 이지아

01

contents

History

Version	Date	Update Info	Remark
V.0.1	2021.06.02	개발계획서 최초 작성	
V.0.2	2021.06.07	기본 페이지 화면구성, 로그인 처리, 차량 주행 진행	
V.0.3	2021.06.11	MariaDB을 이용한 차트 페이지 구성, yolo-tiny를 활용한 데이터 모델링 진행	
V.0.4	2021.06.20	데이터 베이스 값을 winForm에서 받아와 시각화	
V.0.5	2021.06.30	모의환경에서 테스트 실시 및 버그수정	

01

contents

프로젝트 개요

구분	내용
프로그램명	• Yolo-Tiny를 활용한 농작물 상태 식별 스마트 카봇
개발자	• 김지현, 배문규, 이지아
사용언어	• C, C#, Python
개발환경	• Arduino IDE, Raspbian, Visual Studio
프로그램 개요	• 주어진 트랙을 따라 카트가 주행하면서 카메라로 농작물의 성숙도를 실시간 구분하여 상태를 프로그램으로 확인함
프로그램 기능정의	• 주행 : 정해진 트랙을 따라 카트가 계속 주행할 수 있게 함 (초음파 센서 사용으로 충돌 방지 기능 추가)
	• 상태 판정 : 라즈베리파이와 연결한 카메라로 농작물의 성숙도를 식별함
	• 환경 감지 : 온습도 센서를 통해 각 섹터별 환경을 감지함
	• 확인 : WinForm을 통해 농작물의 상태와 환경정보를 실시간으로 확인

01

contents

서비스 개요

구분	내용
기획배경	• 카메라를 활용해여 스마트 팜 또는 팩토리등에 적용할 수 있는 프로젝트를 고안하다 아이디어를 정함
기획목적	• 딥러닝 기반의 농작물의 성숙도 정보를 사용자가 더 편하게 열람할 수 있는 플랫폼 제공
기대효과	• 사용자가 농작물의 성숙상태를 카봇이 확인하기 때문에 노동력 절감 기대, 농장의 환경 정보를 섹터별 정확하게 측정
기능요약	• 라인트레이싱 RC카, 카메라로 상태 식별, 센서값 감지, 웹 폼을 통한 정보 열람
기타사항	• winForm, mariaDB, yolo-Tiny를 연동하여 만든 프로그램

기획일정	1W					2W					3W					4W					5W				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
벤치마킹 및 아이디어 기획																									
개발 계획서 작성																									
설계서 작성																									
프로그래밍																									
QA																									

01

contents

사용설명서

작동방법

1

메인 페이지에서 로그인을 하여 정보 열람 페이지로 이동할 수 있습니다.
(+ 아이디, 비밀번호 변경 기능)

2

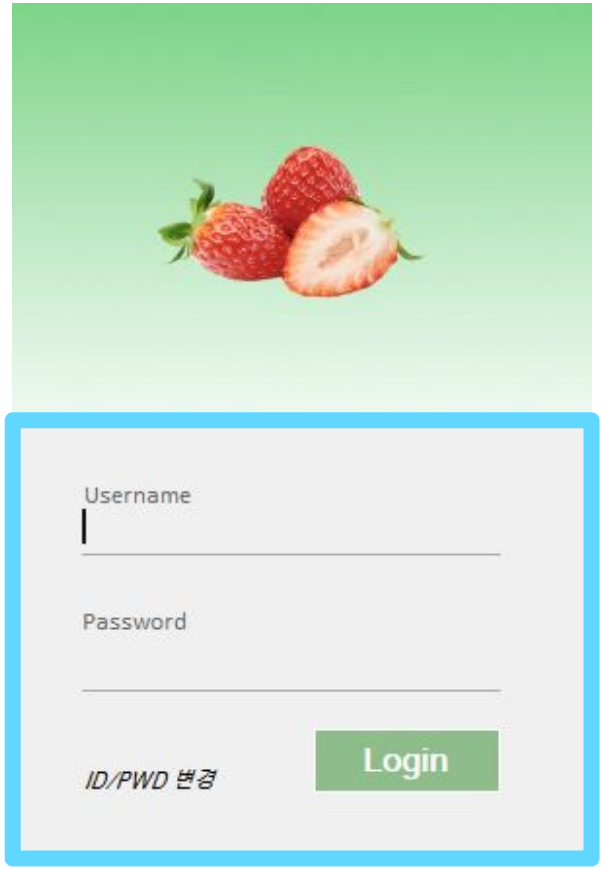
페이지에서 **딸기 정보 확인**과 **환경 정보 확인** 페이지로 이동할 수 있습니다.

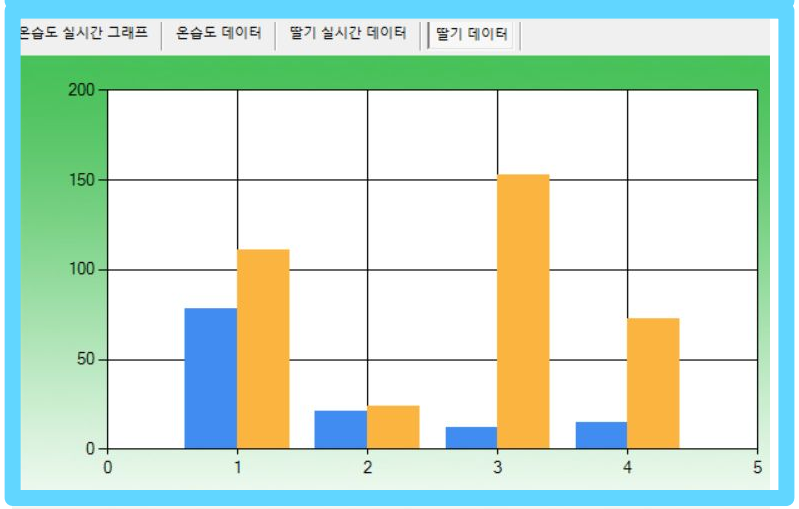
3

딸기 정보 확인 페이지에서는 덜 익은 딸기와 잘 익은 딸기와의 비율을 계산해 페이지에서 열람할 수 있게 해줍니다.

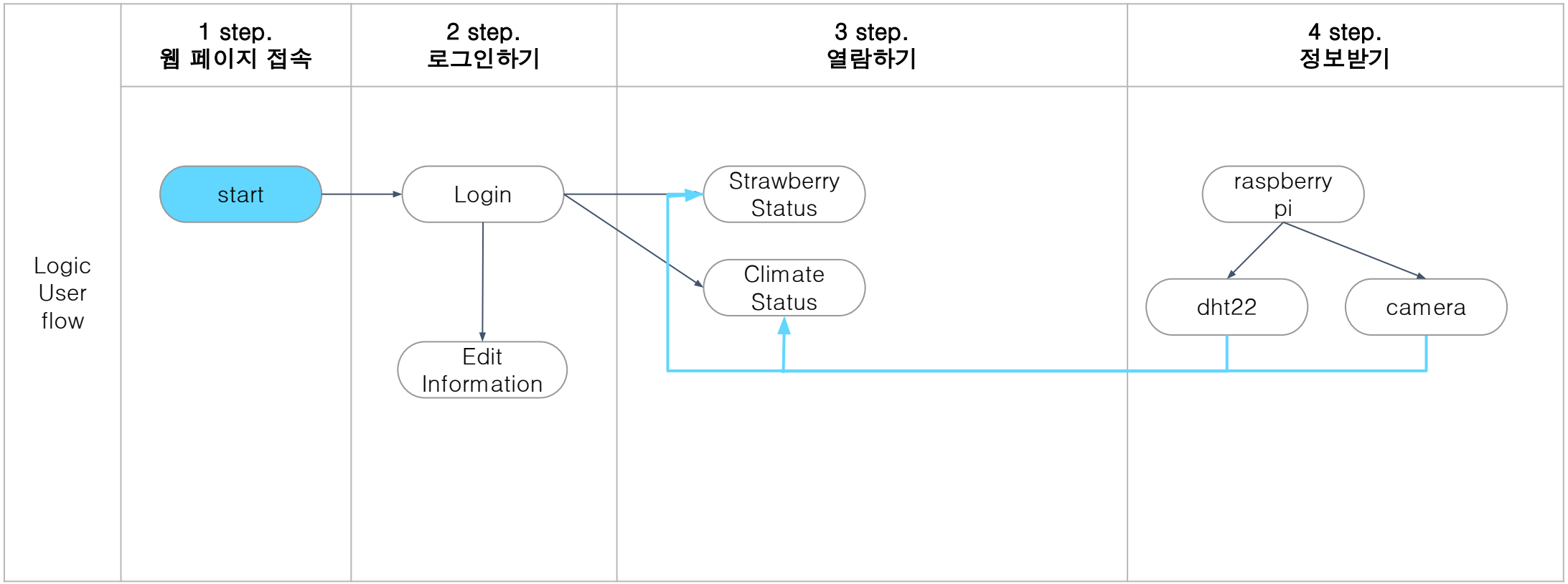
4

환경 정보 확인 페이지에서는 센서 값을 통한 온습도 값의 변화를 그래프와 그리드뷰로 확인할 수 있게 해줍니다.





User Flow

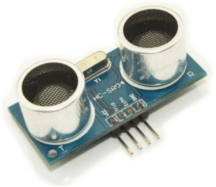

Logic
User
flow
1 step.
웹 페이지 접속
2 step.
로그인하기
3 step.
열람하기
4 step.
정보받기
start
Login
Edit
Information
Strawberry
Status
Climate
Status
raspberry
pi
dht22
camera

Yolo-Tiny를 활용한 농작물 성숙도 판정 스마트 카봇

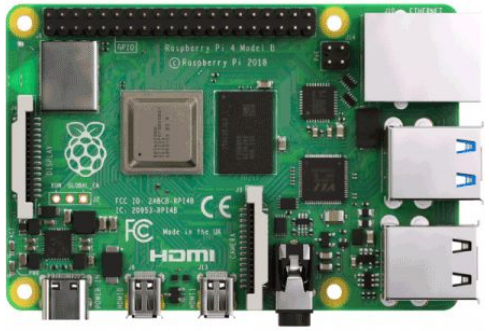
Structure

CarBot

아두이노 우노
초음파 센서, 라인 센서(적외선)



라즈베리 파이
온습도센서, 컬러센서,
라즈베리파이카메라, yolo-tiny
모델 학습



yolo-tiny

DataBase

온습도 데이터베이스
temp, hum, sector, time

딸기 정보 데이터베이스
ripe, unripe, sector, time

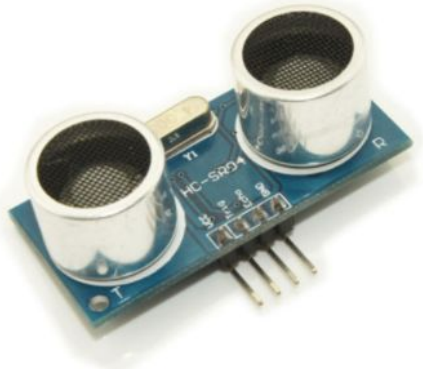
winForm

- 온습도 실시간 확인창
- 온습도 데이터 확인
- 딸기 실시간 비율 확인
- 구역별 전체 비율 확인창

CarBot-Arduino

전원

9v 배터리 1개, aa건전지 4개
아두이노에 전원을 공급했고,
모터드라이버에도 전원을 공급



초음파센서

일정 거리 이하로 거리가 짧아져
장애물이 감지가 되면
RC카가 정지하게 함

라인센서

좌측과 우측을 감지하는 라인센서 2개
적외선을 감지하는 원리로 동작
하얀 선을 따라가며 RC카가 움직임

동작

양쪽 센서 라인 감지되지 않을 때 정지
왼쪽 센서 라인 감지될 때 왼쪽으로 전진
오른쪽 센서 라인 감지될 때 오른쪽으로 전진
양쪽 센서 모두 라인 감지될 때 전진

라인센서를 통해 동작하면서
초음파 센서로 장애물이 감지될 때 정지

CarBot-Arduino

동작

양쪽 센서 라인 감지되지 않을 때 전진
왼쪽 센서 라인 감지될 때 왼쪽으로 전진
오른쪽 센서 라인 감지될 때 오른쪽으로 전진
양쪽 센서 모두 라인 감지될 때 정지

라인센서를 통해 동작하면서
초음파 센서로 장애물이 감지될 때 정지

모터 구동 함수

```
void motor_drive()  //모터를 구동하는 함수
{
  if(m_a_dir == 0)
  {
    digitalWrite(MOTOR_A_a, LOW);      //모터A+ LOW
    analogWrite(MOTOR_A_b, m_a_spd);   //모터A-의 속력을 PWM 출력
  }
  else
  {
    analogWrite(MOTOR_A_a, m_a_spd);   //모터A+의 속력을 PWM 출력
    digitalWrite(MOTOR_A_b, LOW);      //모터A- LOW
  }
  if(m_b_dir == 1)
  {
    digitalWrite(MOTOR_B_a, LOW);      //모터B+ LOW
    analogWrite(MOTOR_B_b, m_b_spd);   //모터B-의 속력을 PWM 출력
  }
  else
  {
    analogWrite(MOTOR_B_a, m_b_spd);   //모터B+의 속력을 PWM 출력
    digitalWrite(MOTOR_B_b, LOW);      //모터B- LOW
  }
}
```

CarBot-Arduino

초음파센서

일정 거리 이하로 거리가 짧아져
장애물이 감지가 되면
RC카가 정지하게 함

거리 측정 함수

```
int get_dist()
{
  long duration, distance;

  digitalWrite(TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);

  duration = pulseIn (ECHO, HIGH); //물체에 반사되어돌아온 초음파의 시간을 변수에 저장합

  //34000*초음파가 물체로 부터 반사되어 돌아오는시간 /1000000 / 2(왕복값이아니라 편도값이기때문에 나누기2를 합
  //초음파센서의 거리값이 위 계산값과 동일하게 Cm로 환산되는 계산공식 입니다. 수식이 간단해지도록 적용
  distance = duration * 17 / 1000;

  //PC모니터로 초음파 거리값을 확인 하는 코드
  return distance;
}
```

장애물 확인 후 주행

```
void drive()
{
    int dist = get_dist(); //장애물이 없으면
    if(dist >= 10)
    {
      linetrace_val();  //입력된 데이터에 따라 모터에 입력될 변수를 조정하는 함수
      motor_drive();
    }
    else
    {
      pause();
    }
}
```

CarBot-Arduino

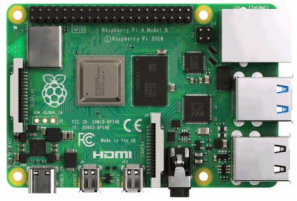
라인센서

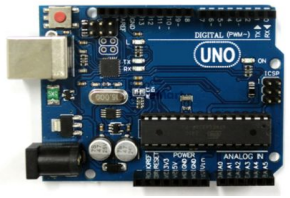
좌측과 우측을 감지하는 라인센서 2개
적외선을 감지하는 원리로 동작
하얀 선을 따라가며 RC카가 움직임

라인 트레이싱 함수

```
void linetrace_val() //입력된 데이터에 따라 모터에 입력될 변수를 조정하는 함수
{
  boolean line_l = digitalRead(LINESENS_L), line_r = digitalRead(LINESENS_R);
  //왼쪽과 오른쪽 라인센서의 감지값을 변수에 저장합니다.

  if(line_l == 0 && line_r == 0)      //라인이 감지되지 않을때 전진
  {
    m_a_dir = 0;  //모터A 정방향
    m_b_dir = 0;  //모터B 정방향
    m_a_spd = MOTOR_SPEED;  //모터A의 속력값 조정
    m_b_spd = MOTOR_SPEED;  //모터B의 속력값 조정
  }
  else if(line_l == 1 && line_r == 0) //왼쪽 센서 감지시 왼쪽으로 전진
  {
    m_a_dir = 1;  //모터A 역방향
    m_b_dir = 0;  //모터B 정방향
    m_a_spd = 0;  //모터A의 정지
    m_b_spd = constrain(MOTOR_SPEED*2
    , 0, 255);  //모터B의 속력값 조정
  }
  else if(line_l == 0 && line_r == 1)  //오른쪽 센서 감지시 오른쪽으로 전진
  {
    m_a_dir = 0;  //모터A 정방향
    m_b_dir = 1;  //모터B 역방향
    m_a_spd = constrain(MOTOR_SPEED*2, 0, 255);  //모터A의 속력값 조정
    m_b_spd = 0;  //모터B의 정지
  }
  else
  {
    m_a_dir = 0;  //모터A 정방향
    m_b_dir = 0;  //모터B 정방향
    m_a_spd = 0;  //모터A의 정지
    m_b_spd = 0;  //모터B의 정지
  }
}
```

CarBot-Linetrace





메인보드가 두개나 존재하고, 해당 카트 내에
보조배터리도 올려져 있어 카봇의 무게가 무거움
-> 높은 속도로 동작하도록 보완

높은 속도로 주행하면서 카메라로 포착하는
딸기의 상이 깨지는 현상 발생, 인식률 저하
-> 라인트레이싱에 millis() 함수를 사용해
drive, pause를 반복하며 촬영하도록 보완

주행 반복 함수

```
void loop(){
  unsigned long now = millis();

  if(now - past >= 0 && now - past <= 1500)
  {
    drive();
  }
  else if(now - past <= 3500)
  {
    pause();
  }
  else if(now - past >= 3500)
  {
    past = now;
  }
}
```

millis() 함수를 사용하여 인식률을 증가시킴

- 0부터 1500까지 증가하는 동안 주행
- 1500부터 3500까지 정지시켜 촬영
- 3500초과 시 다시 0으로 돌아가게 함

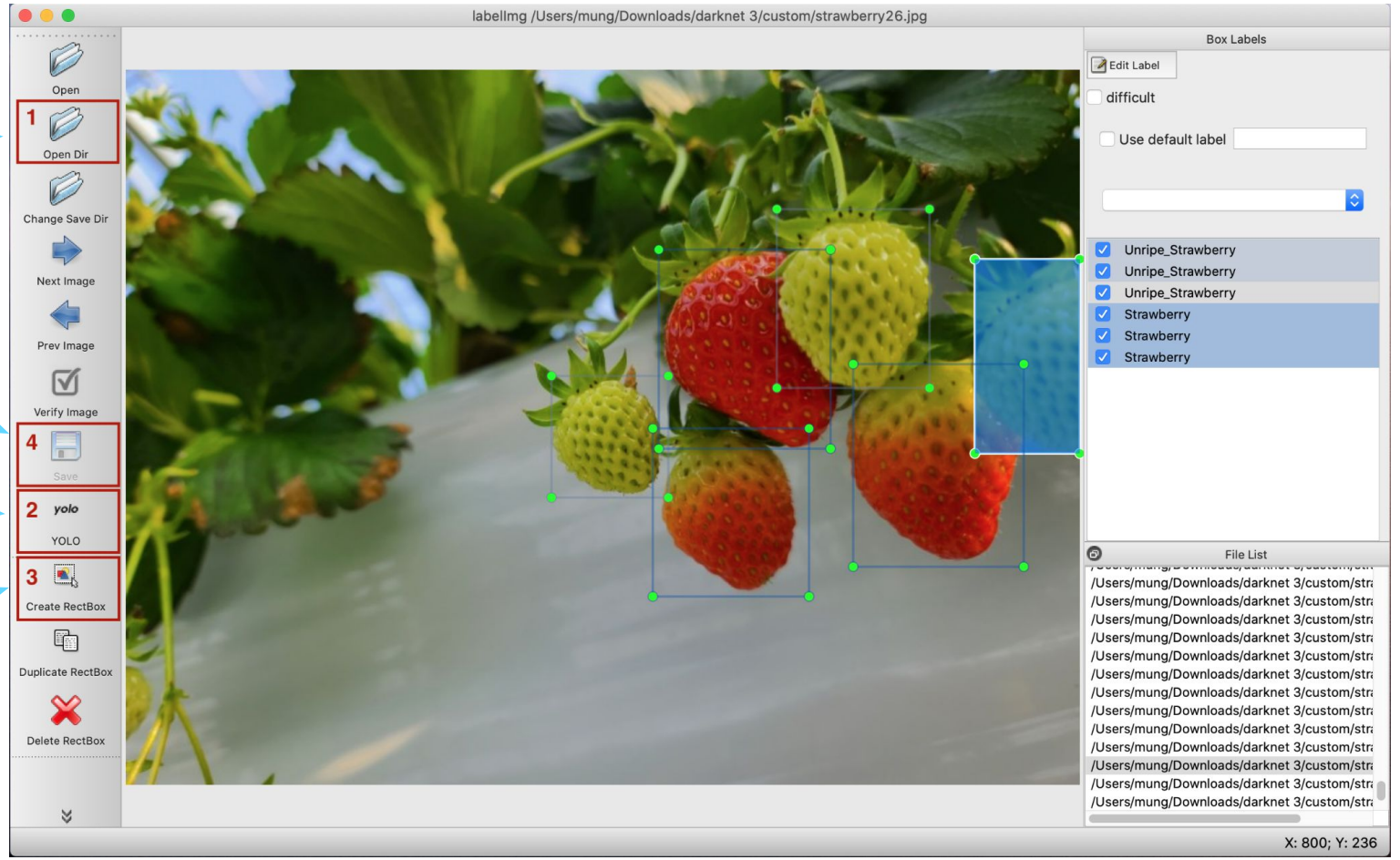
Yolo-Tiny > Yolo Model Image Labeling

이미지 데이터
디렉터리 열기

라벨링할 포맷 지정

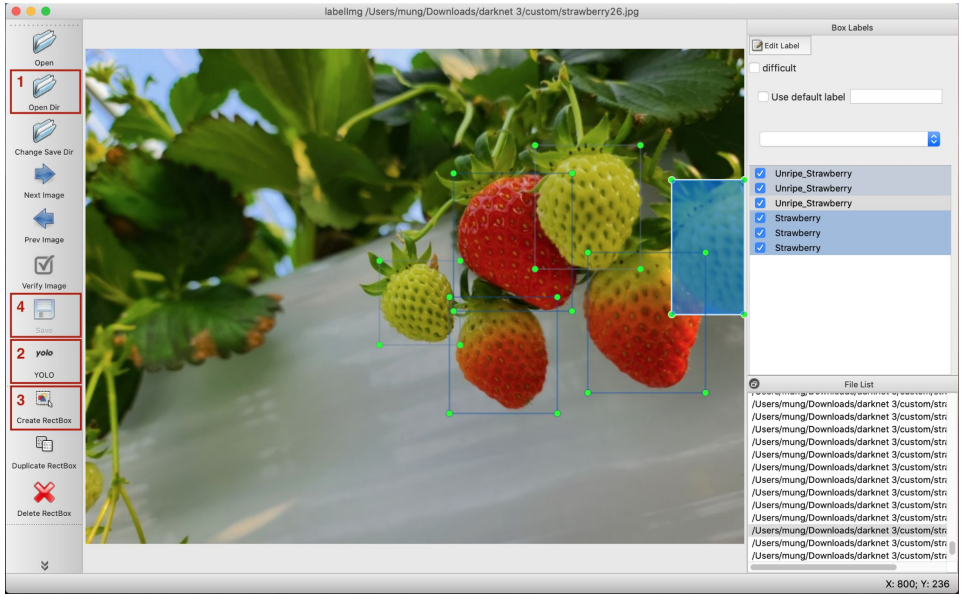
ROI 지정
Region of Interest

클래스 이름 정하고,
라벨링 후 저장



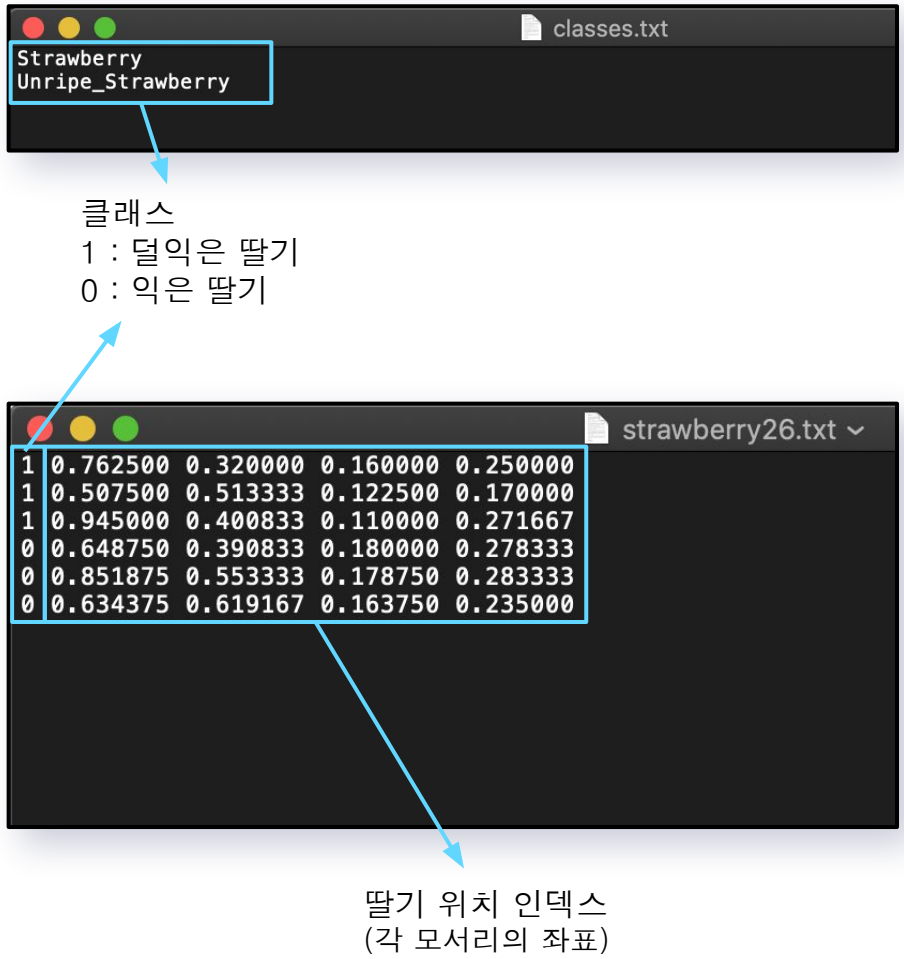
딸기 이미지를 크롤링한 뒤 yolo 모델 학습을 위해 딥러닝용 이미지 라벨링 툴인 labelImg를 사용하여 이미지 라벨링

Yolo-Tiny > Yolo Model Labeling



- 라즈베리파이에서도 yolo모델을 실시간으로 구동시킬 수 있을 정도로 경량화된 yolo-Tiny 모델을 활용하여 이미지를 모델에 학습하는 과정을 진행함
- 이미지를 모델에 학습시키기 위해서는 이미지의 클래스 라벨링 과정이 필요함
- yolo 모델을 학습시키기 위해 'labelImg'라는 이미지 라벨링 툴 사용

결과



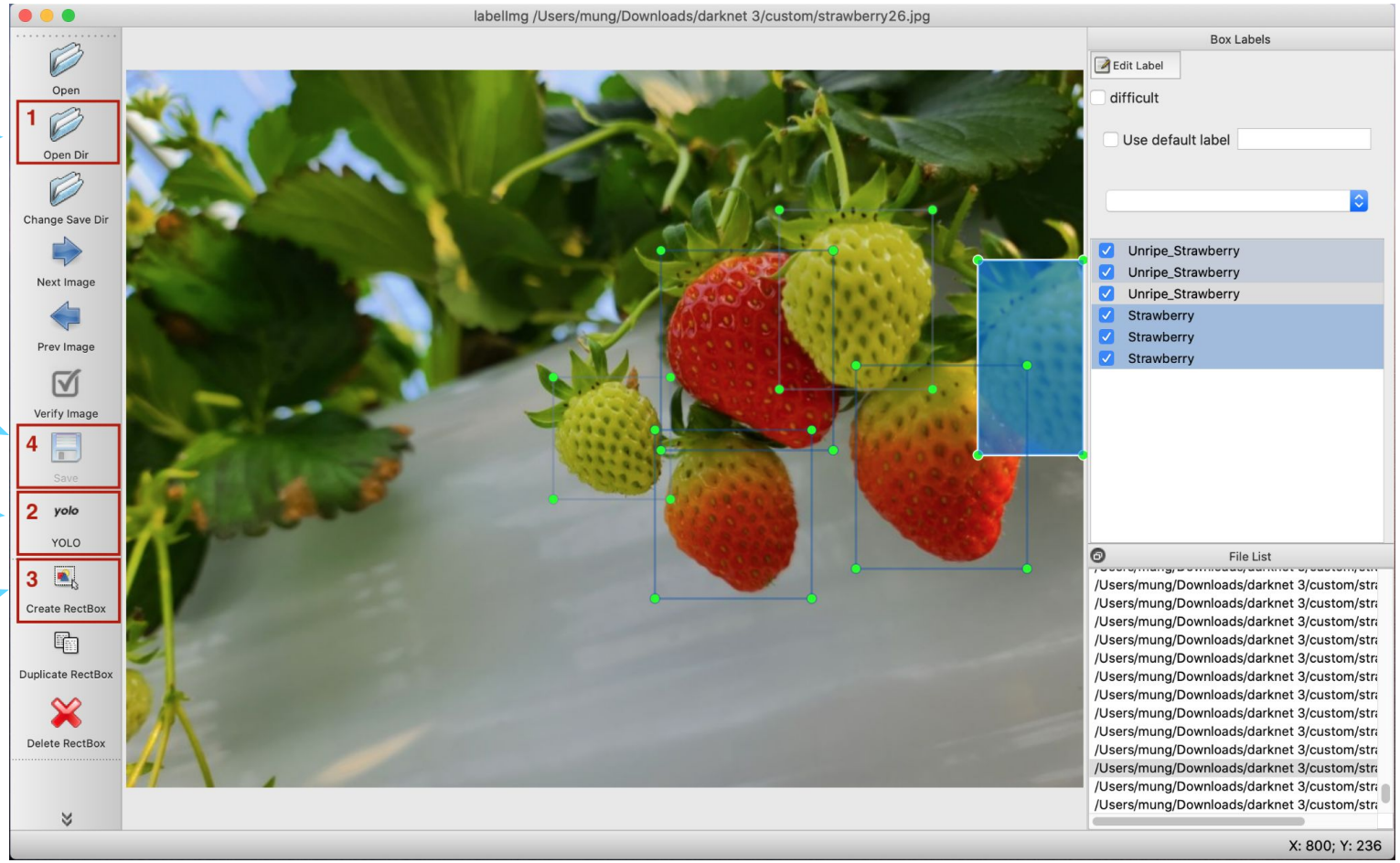
Yolo-Tiny > Yolo Model Labeling

이미지 데이터
디렉터리 열기

라벨링할 포맷 지정

ROI 지정
Region of Interest

클래스 이름 정하고,
라벨링 후 저장



딸기 이미지를 크롤링한 뒤 yolo 모델 학습을 위해 딥러닝용 이미지 라벨링 툴인 labelImg를 사용하여 이미지 라벨링

Yolo-Tiny > Configure Yolo Training Dataset

DarKnet은 Joseph Redmon이 독자적으로 개발한 신경망 프레임워크로서 DNN을 학습시키고 실행시킬 수 있다.

Darknet을 이용하면 학습된 신경망모델 중 하나인 yolo 뿐만 아니라 AlexNet, VGG-16, Resnet, Densenet 등 기존의 DNN모델들도 돌려 볼 수 있다.

Darknet에서 yolo 학습을 위해 필요한 파일은 총 6개인데 이 중 4개는 파이썬 스크립트를 통해 생성했고, 두개는 darknet 홈페이지에서 다운로드를 받을 수 있었다.

본 프로젝트는 라즈베리파이에서 실행을 하기 때문에, 경량화 모델을 다운받았다.
적절하게 학습을 시켜주기 위해 학습시킬 모델의 클래스의 수에 알맞게 cfg 파일을 수정했다.

Yolo-Tiny > Making Yolo Learning Data Set

DNN 프레임워크 Darknet에서 Yolo 모델 학습을 위해 필요한 파일

라벨링한 데이터셋을 yolo 학습 데이터 셋으로 만들기 위해 6개의 파일이 필요하다.

1. classes.names
2. [custom_data].data
3. train.txt
4. test.txt
5. yolo.cfg
6. yolo.weights

파이썬 스크립트를 통해 생성

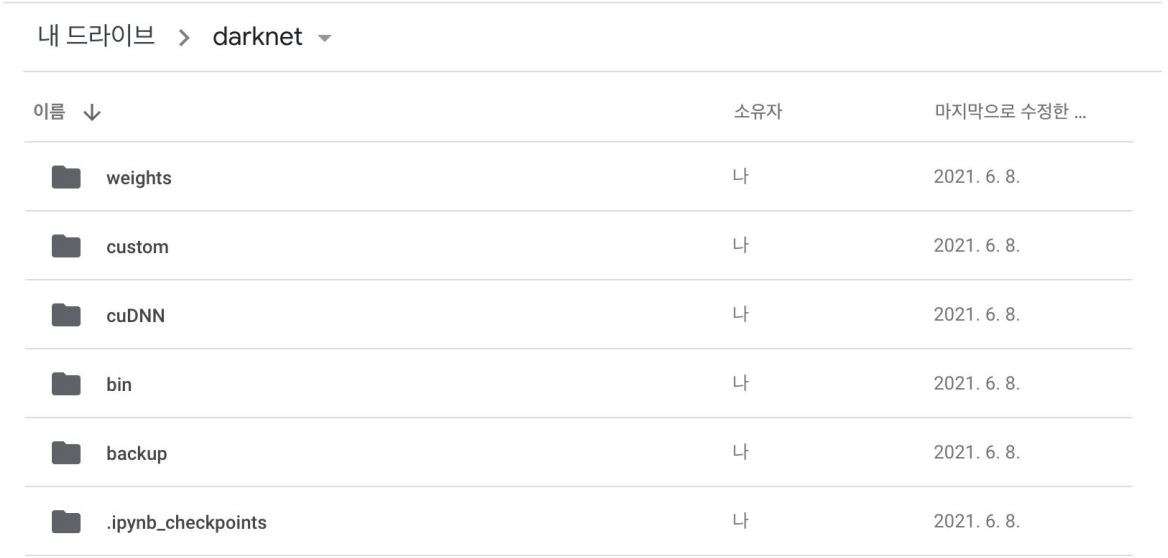
darknet 홈페이지에서 다운로드- 경량화 모델

Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

```
#PYTHON
import os

current_path = os.path.abspath(os.curdir)
COLAB_DARKNET_ESCAPE_PATH = '/content/drive/My\ Drive/darknet'
COLAB_DARKNET_PATH = '/content/drive/My Drive/darknet'

YOLO_IMAGE_PATH = current_path + '/custom'
YOLO_FORMAT_PATH = current_path + '/custom'

class_count = 0
test_percentage = 0.2
paths = []

with open(YOLO_FORMAT_PATH + '/' + 'classes.names', 'w') as names, \
     open(YOLO_FORMAT_PATH + '/' + 'classes.txt', 'r') as txt:
    for line in txt:
        names.write(line)
        class_count += 1
    print ("[classes.names] is created")

with open(YOLO_FORMAT_PATH + '/' + 'custom_data.data', 'w') as data:
    data.write('classes = ' + str(class_count) + '\n')
    data.write('train = ' + COLAB_DARKNET_ESCAPE_PATH + '/custom/' + 'train.txt' + '\n')
    data.write('valid = ' + COLAB_DARKNET_ESCAPE_PATH + '/custom/' + 'test.txt' + '\n')
    data.write('names = ' + COLAB_DARKNET_ESCAPE_PATH + '/custom/' + 'classes.names' + '\r
    data.write('backup = backup')
    print ("[custom_data.data] is created")

os.chdir(YOLO_IMAGE_PATH)
for current_dir, dirs, files in os.walk('.'):
    for f in files:
        if f.endswith('.jpg'):
            image_path = COLAB_DARKNET_PATH + '/custom/' + f
            paths.append(image_path + '\n')

paths_test = paths[:int(len(paths) * test_percentage)]
paths = paths[int(len(paths) * test_percentage):]

with open(YOLO_FORMAT_PATH + '/' + 'train.txt', 'w') as train_txt:
    for path in paths:
        train_txt.write(path)
    print ("[train.txt] is created")

with open(YOLO_FORMAT_PATH + '/' + 'test.txt', 'w') as test_txt:
    for path in paths_test:
        test_txt.write(path)
####     print ("[test.txt] is created")
```

Yolo-Tiny-> Learning Model



미리 컴파일을 완료한 darknet프레임워크를 google colab에서 작동시키기 위해 google drive에 업로드한다.

- **weights** : 학습이 완료하면 생성되는 모델을 최종적으로 담을 디렉토리
- **custom** : 앞에서 준비한 6개의 설정 파일과 학습시킬 이미지와 라벨링 데이터가 기록된 텍스트파일을 업로드할 디텍토리
- **cuDNN** : 컴파일된 darknet 프레임워크와 호환되는 cuDNN 압축파일이 담긴 디렉토리
- **bin** : darknet 실행파일이 담긴 디렉토리
- **backup** : 학습을 통해 생성된 모델들이 임시적으로 저장될 디렉토리

01

contents

Yolo-Tiny-> Learning Model

```
import os
if not os.path.exists('darknet'):
    os.makedirs('darknet')
%cd darknet
%ls
```

Darknet프레임워크를 로드하고 ls 명령어를 통해/content/darknet 이라고 정상적으로 뜬다면

학습시킬 때 권한문제가 발생하지 않기 위해 접근 권한을 미리 변경해준다.

```
!ls -la '/content/drive/My Drive/darknet/bin/darknet'
!cp /content/drive/My\ Drive/darknet/bin/darknet ./darknet
!chmod +x ./darknet
```

Yolo-Tiny Modeling

custom yolo3 모델을 만들 .conv.xx 파일 가져오기

1) yolov3일 때 : darknet53.conv.74 파일 다운받기. **(162.5MB)**

2) yolov3-tiny일 때 : yolov3-tiny.weights에서 yolov3-tiny.conv.15 추출하기.**(31MB)**

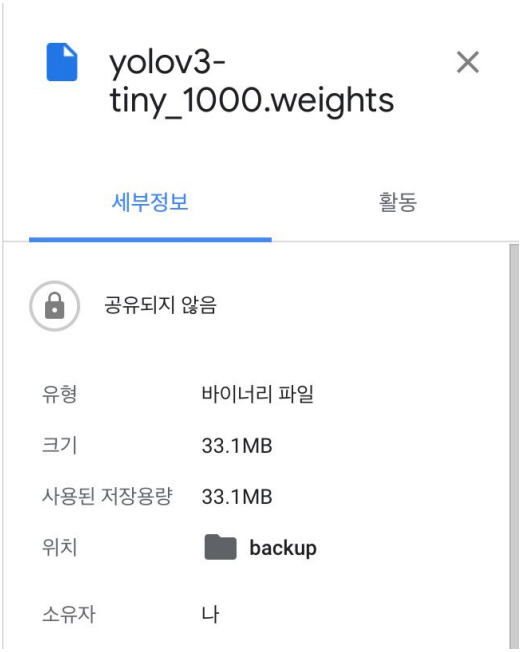
```
!./darknet partial custom/yolov3-tiny.cfg custom/yolov3-tiny.weights yolov3-tiny.conv.15 15
```

yolo.weights파일은 yolo모델을 사용할 수 있는 완제품이라 할 수 있고,
yolo.conv.74는 yolov3, yolov3-tiny.conv.15는 yolov3-tiny를 커스텀으로 최종학습시킬 중간정도 학습되어
웨이트값이 어느정도 잡혀있는 반제품이라 할 수 있음.라즈베리파이에서는 yolov3-tiny모델을 사용

Yolo-Tiny Modeling

yolo-tiny3 모델 학습시키고 산출모델들 다운받기

이름	소유자	마지막으로 수정한 ...
yolov3-tiny_1000.weights	나	오후 8:01
yolov3-tiny_2000.weights	나	오후 8:01
yolov3-tiny_3000.weights	나	오후 8:01
yolov3-tiny_4000.weights	나	오후 8:01




```
!./darknet detector train custom/custom_data.data custom/yolov3-tiny.cfg custom/yolov3-tiny.conv.15
-dont_show
from google.colab import files
files.download('/content/darknet/backup/yolov3-tiny_1000.weights')
files.download('/content/darknet/backup/yolov3-tiny_2000.weights')
files.download('/content/darknet/backup/yolov3-tiny_3000.weights')
files.download('/content/darknet/backup/yolov3-tiny_4000.weights')
```

01

contents

Yolo-Tiny Modeling

```
 3990: 0.675955, 0.659847 avg loss, 0.000010 rate, 0.673566 seconds, 127680 images
Resizing
480 x 480
 try to allocate additional workspace_size = 52.43 MB
 CUDA allocate done!
Loaded: 0.309272 seconds

 3991: 0.538666, 0.647729 avg loss, 0.000010 rate, 0.458960 seconds, 127712 images
Loaded: 0.061892 seconds

 3992: 0.449508, 0.627907 avg loss, 0.000010 rate, 0.437156 seconds, 127744 images
Loaded: 0.034381 seconds

 3993: 0.389076, 0.604023 avg loss, 0.000010 rate, 0.452050 seconds, 127776 images
Loaded: 0.057802 seconds

 3994: 0.648703, 0.608491 avg loss, 0.000010 rate, 0.463466 seconds, 127808 images
Loaded: 0.061149 seconds

 3995: 0.721022, 0.619744 avg loss, 0.000010 rate, 0.426656 seconds, 127840 images
Loaded: 0.018826 seconds

 3996: 0.580182, 0.615788 avg loss, 0.000010 rate, 0.449537 seconds, 127872 images
Loaded: 0.012295 seconds

 3997: 0.727844, 0.626994 avg loss, 0.000010 rate, 0.435157 seconds, 127904 images
Loaded: 0.213088 seconds

 3998: 0.650128, 0.629307 avg loss, 0.000010 rate, 0.422802 seconds, 127936 images
Loaded: 0.235546 seconds

 3999: 0.426436, 0.609020 avg loss, 0.000010 rate, 0.417323 seconds, 127968 images
Loaded: 0.000050 seconds

 4000: 0.500468, 0.598165 avg loss, 0.000010 rate, 0.458179 seconds, 128000 images
Saving weights to backup/strawberry_yolotiny_4000.weights
```

configuration 파일의 max_batches에 4000 (학습시킬 모델의 클래스 수 * 2000)로 설정하였기 때문에,
4000 Epoches 학습을 진행하고
1000 Epoches 마다 모델을 저장

Yolo-Tiny Modeling

yolo-tiny3 산출모델들 성능평가

```
layer     filters    size              input                output
    0 conv     16  3 x 3 / 1   416 x 416 x   3   ->   416 x 416 x  16 0.150 BF
    1 max          2 x 2 / 2   416 x 416 x  16   ->   208 x 208 x  16 0.003 BF
    2 conv     32  3 x 3 / 1   208 x 208 x  16   ->   208 x 208 x  32 0.399 BF
    3 max          2 x 2 / 2   208 x 208 x  32   ->   104 x 104 x  32 0.001 BF
    4 conv     64  3 x 3 / 1   104 x 104 x  32   ->   104 x 104 x  64 0.399 BF
    5 max          2 x 2 / 2   104 x 104 x  64   ->    52 x  52 x  64 0.001 BF
    6 conv    128  3 x 3 / 1    52 x  52 x  64   ->    52 x  52 x 128 0.399 BF
    7 max          2 x 2 / 2    52 x  52 x 128   ->    26 x  26 x 128 0.000 BF
    8 conv    256  3 x 3 / 1    26 x  26 x 128   ->    26 x  26 x 256 0.399 BF
    9 max          2 x 2 / 2    26 x  26 x 256   ->    13 x  13 x 256 0.000 BF
   10 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512 0.399 BF
   11 max          2 x 2 / 1    13 x  13 x 512   ->    13 x  13 x 512 0.000 BF
   12 conv   1024  3 x 3 / 1    13 x  13 x 512   ->    13 x  13 x1024 1.595 BF
   13 conv    256  1 x 1 / 1    13 x  13 x1024   ->    13 x  13 x 256 0.089 BF
   14 conv    512  3 x 3 / 1    13 x  13 x 256   ->    13 x  13 x 512 0.399 BF
   15 conv     21  1 x 1 / 1    13 x  13 x 512   ->    13 x  13 x  21 0.004 BF
   16 yolo
   17 route  13
   18 conv    128  1 x 1 / 1    13 x  13 x 256   ->    13 x  13 x 128 0.011 BF
   19 upsample           2x    13 x  13 x 128   ->    26 x  26 x 128
   20 route  19 8
   21 conv    256  3 x 3 / 1    26 x  26 x 384   ->    26 x  26 x 256 1.196 BF
   22 conv     21  1 x 1 / 1    26 x  26 x 256   ->    26 x  26 x  21 0.007 BF
   23 yolo
Total BFLOPS 5.449
 Allocate additional workspace_size = 52.43 MB
Loading weights from custom/strawberry_yolotiny_2000.weights...
 seen 64
Done!
```

```
 calculation mAP (mean average precision)...
12
 detections_count = 106, unique_truth_count = 49
class_id = 0, name = Strawberry, ap = 87.89%      (TP = 17, FP = 1)
class_id = 1, name = Unripe_Strawberry, ap = 97.08%      (TP = 23, FP = 1)

 for thresh = 0.25, precision = 0.95, recall = 0.82, F1-score = 0.88
 for thresh = 0.25, TP = 40, FP = 2, FN = 9, average IoU = 73.49 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.924858, or 92.49 %
Total Detection Time: 1.000000 Seconds

Set -points flag:
 `-points 101` for MS COCO
 `-points 11` for PascalVOC 2007 (uncomment `difficult` in voc.data)
 `-points 0` (AUC) for ImageNet, PascalVOC 2010-2012, your custom dataset
```

```
class_id = 0, name = Strawberry, ap = 87.89%          (TP = 17, FP = 1)
class_id = 1, name = Unripe_Strawberry, ap = 97.08%          (TP = 23, FP = 1)
 for thresh = 0.25, precision = 0.95, recall = 0.82, F1-score = 0.88
 for thresh = 0.25, TP = 40, FP = 2, FN = 9, average IoU = 73.49 %
 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.924858, or 92.49 %
```

위 평가점수들을 종합적으로 평가하니, 2000Epoches에서 산출된 모델이 가장 성능이 좋아서 최종 프로젝트 모델로 선정함.

01

contents

Raspberry Pi4 > Source Code

strawberry Module

```
import numpy as np
import cv2
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

# Load Yolo
net = cv2.dnn.readNet("./model/strawberry_yolotiny_2000.weights", "./model/strawberry_yolotiny.cfg")
classes = []
with open("./model/classes.names", "r") as f:
    classes = [line.strip() for line in f.readlines()]

# initialize the camera and grab a reference to the raw camera capture
camera = PiCamera()
camera.resolution = frame_resolution
camera.framerate = frame_rate
rawCapture = PiRGBArray(camera, size=(frame_resolution))
```

1. numpy, opencv, time 패키지를 import하고 추가적으로 라즈베리파이 전용패키지인 PiRGBArray와 PiCamera를 import한다.
2. 글로벌로 앞서 학습된 yolo-tiny 모델과 cfg파일을 load하고 classes.names파일로 부터 클래스이름들을 읽어온다.
3. Picamera객체를 생성한다.

Raspberry Pi4 > Source Code

strawberry Module

```
class STRAWBERRY():
    def __init__(self):
        self.camera = camera
        self.rawCapture = rawCapture

    def detectAndDisplay(self, image):
        start_time = time.time()
        h, w = image.shape[:2]
        img = cv2.resize(image, (frame_width, frame_height))

        blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), swapRB=True, crop=False)

        net.setInput(blob)
        outs = net.forward(output_layers)

        confidences = []
        names = []
        boxes = []
        colors = []
```

1. STRAWBERRY라는 클래스를 생성하고,
   외부에서 호출될 camera, rawCapture 변수는 멤버로 지정한다.
2. 이미지가 입력되면 해당 이미지에서 객체들을 검출하기
   용이하게
   전처리를 한다.
3. 이미지를 모델에 입력하고 output된 Raw Data를 outs에
   저장한다.

Raspberry Pi4 > Source Code

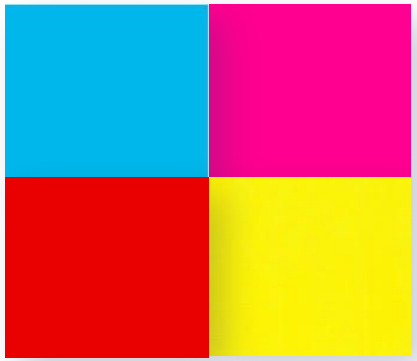
strawberry Module

```
for out in outs:
    for detection in out:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > min_confidence:
            center_x = int(detection[0] * frame_width)
            center_y = int(detection[1] * frame_height)
            w = int(detection[2] * frame_width)
            h = int(detection[3] * frame_height)

            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            names.append(classes[class_id])
            colors.append(color_lists[class_id])

indexes = cv2.dnn.NMSBoxes(boxes, confidences, min_confidence, 0.4)
font = cv2.FONT_HERSHEY_PLAIN
```

```
ripe_cnt = 0
unripe_cnt = 0
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = '{} {:,.2%}'.format(names[i], confidences[i])
        color = colors[i]
        if label.split()[0] == "Unripe_Strawberry":
            unripe_cnt += 1
        elif label.split()[0] == "Strawberry":
            ripe_cnt += 1

        print(i, label, x, y, w, h)
        cv2.rectangle(img, (x, y), (x + w, y + h), color, 2)
        cv2.putText(img, label, (x, y - 10), font, 1, color, 2)

cv2.imshow(title_name, img)
if ripe_cnt != 0 or unripe_cnt != 0:
    print("=== A frame took {:.3f} seconds".format(time.time() - start_time))
    return ripe_cnt, unripe_cnt
else :
    return 0, 0
```

1. 이미지에서 yolo모델이 학습된 객체들을 검출하면 outs에 해당 객체들에 대한 Raw Data가 모두 저장되고 각 객체들을 읽어온다.
2. 해당객체의 ripe, unripe 각각의 score를 가져와서 높은 score를 선택한다. 그리고 설정한 threshold보다 높으면 해당 객체의 ROI를 찾고 스코어, 클래스 이름, 해당 클래스의 박스와 폰트 컬러를 모두 배열로 각각 저장한다.
3. 객체들의 ROI 박스와 스코어, 클래스 이름을 영상에 보여주고, 저장된 객체들이 ripe인지, unripe인지 각각 카운트하여 리턴해준다.

Raspberry Pi4 > Color Sensor



TCS34725 컬러센서를 Carbot의 아래에 부착하여 RGB 색상을 인식

구역의 시작점을 색상으로 표시하여 라인의 구역을 색으로 구분함

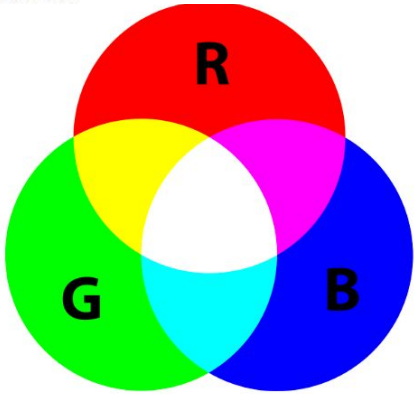
구역 별 데이터를 수집하여 확인할 수 있음

Raspberry Pi4 > Source Code

TCS34725 컬러센서를 통해 색깔별로 구역을 구분하는 과정

컬러센서는 조명에 따라 값이 달라지기에 비율값은 구동하면서 임의로 정해주었다.

```
self.red = data[3] * 256 + data[2]
self.green = data[5] * 256 + data[4]
self.blue = data[7] * 256 + data[6]
total = self.red + self.green + self.blue
sector = 0

if self.red/total > 0.45:
    sector = 1
elif self.blue/total > 0.4:
    sector = 2
elif (self.red+self.green)/total > 0.75:
    sector = 3
elif (self.red+self.blue)/total > 0.7:
    sector = 4

return sector
```



RED – 다른 색상에 비해 빨간색의 비율이 현저히 높다.
따라서, 전체 대비 빨간색이 차지하는 비율이 45퍼센트 이상이라면
빨간 구역(1 sector)에 있다고 지정했다.

BLUE – 다른 색상에 비해 파란색의 비율이 현저히 높다.
따라서, 전체 대비 파란색 값의 비율이 40퍼센트 이상이라면
파란 구역(2 sector)에 있다고 지정했다.

YELLOW – 초록색과 빨간색이 비슷한 비율로 섞인 것이 특징이다.
따라서, 초록색과 빨간색을 합친 비율이 75퍼센트 이상이라면
노란 구역(3 sector)에 있다고 지정했다.

MAGENTA – 빨간색과 파란색이 비슷한 비율로 섞인 것이 특징이다.
따라서, 빨간색과 파란색을 합친 비율이 70퍼센트 이상이라면
자홍 구역(4 sector)에 있다고 지정했다.

01

contents

Raspberry Pi4 > Source Code

Main Script

```
import time
import cv2
import pymysql
import Adafruit_DHT as dht

from color import TCS34725
from strawberry import STRAWBERRY

#MariaDB
Conn = pymysql.connect(port=3306, host='192.168.0.XX', user='root', password='XXXXXXXX', db='kosta', charset='utf8')
Cur = Conn.cursor()

#Strawberry
strawberry = STRAWBERRY()
QueryBerry = "insert into berrystatus values (%s, %s, %s, now())"

#DHT22
Outpin = 4
QueryDHT = "insert into sensorvalue values (%s, %s, %s, now())"

#Sector
color = TCS34725()
lastSector = 0
```

1. apt-get으로 pymysql과 Adafruit_DHT 패키지를 다운받아서 import하고, color, strawberry 모듈을 import해준다.
2. pymysql 객체를 생성하여 MariaDB와 연결을 한다.
3. strawberry 모듈의 STRAWBERRY 객체를 생성하고 해당 객체 정보를 데이터베이스에 보낼 쿼리를 미리 선언한다.
4. 온습도 데이터 센서가 연결된 핀번호를 지정해주고 센서 정보를 데이터베이스에 보낼 쿼리를 미리 선언한다.
5. color 모듈의 TCS34725 객체를 생성하고 섹터 변화를 인식하기 위한 lastSector를 변수를 선언한다.

01

contents

Raspberry Pi4 > Source Code

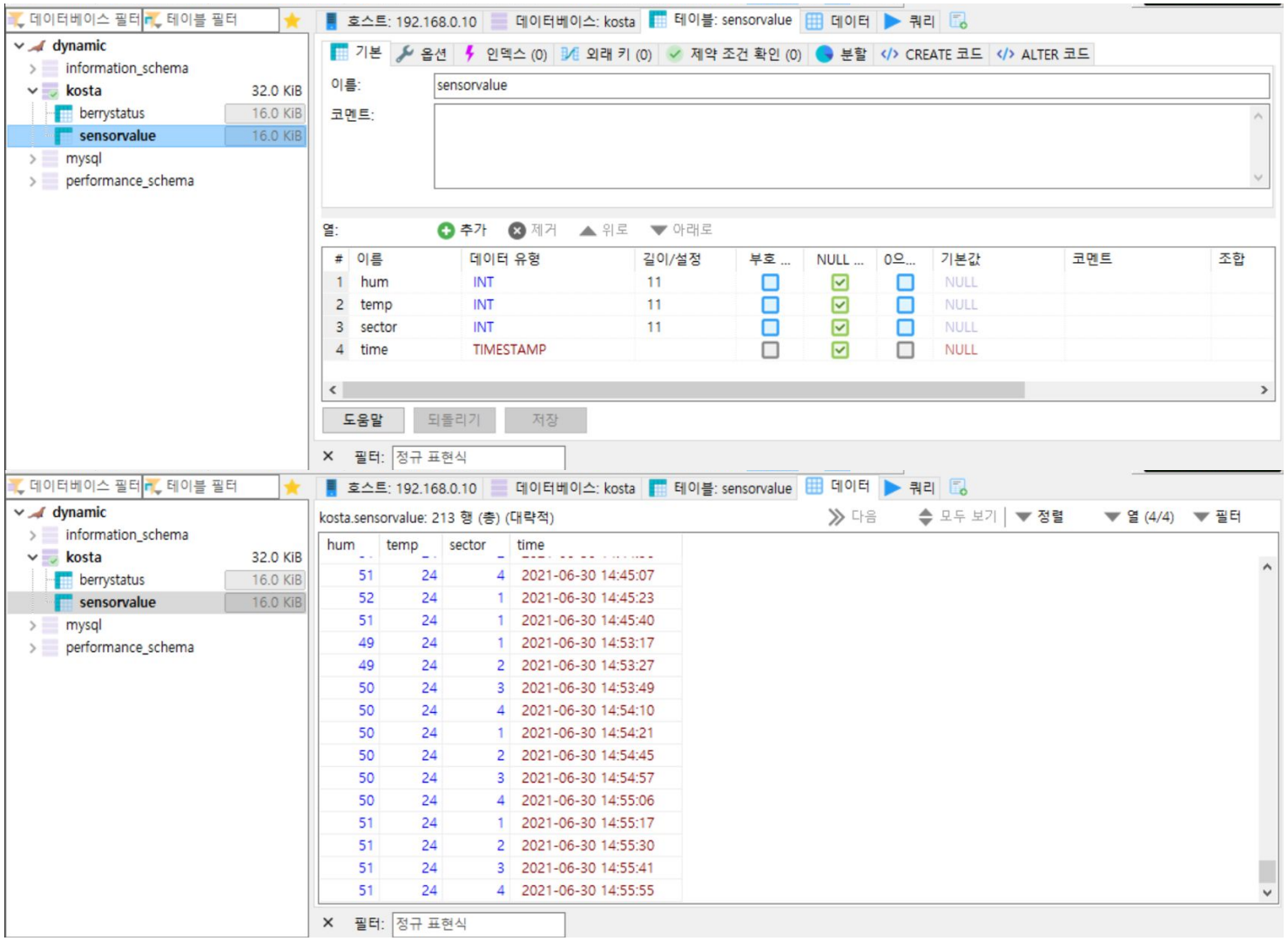
Main Script

```
for frame in strawberry.camera.capture_continuous(strawberry.rawCapture, format="bgr", use_video_port=True):
    Ripe, unRipe = strawberry.detectAndDisplay(frame.array)
    strawberry.rawCapture.truncate(0)

    Sector = color.get_sector()
    print(Sector)
    if Ripe + unRipe != 0 and Sector != 0:
        Values = Ripe, unRipe, Sector
        Cur.execute(QueryBerry, Values)
        Conn.commit()

    if Sector != lastSector and Sector != 0:
        Hum, Temp = dht.read_retry(dht.DHT22, Outpin)
        Values = int(Hum), int(Temp), Sector
        print(Sector)
        Cur.execute(QueryDHT, Values)
        Conn.commit()

    lastSector = Sector
```

1. pycamera가 실시간으로 이미지를 읽어오면 strawberry 객체가 해당 이미지에서 학습된 객체 데이터를 인식 후 객체의 개수를 카운트한다.
2. color 객체가 해당 이미지가 어디 섹터인지 구한다.
3. 딸기의 ripe와 unripe가 모두 0이 아니면서, 섹터가 인식이 되면 해당 데이터를 데이터베이스에 전송한다.
4. 섹터가 변할 때 마다, 해당 섹터의 온습도를 구하여 데이터베이스에 전송한다.

Raspberry Pi4 Strawberry Status Detection

Raspberry Pi 4 > Database(sensor value)



hum(습도), temp(온도), sector(구역), time(시간) 데이터를 sensorvalue 데이터베이스로 전송한다. mariaDB를 활용해 mysql 구문으로 작성하였다.

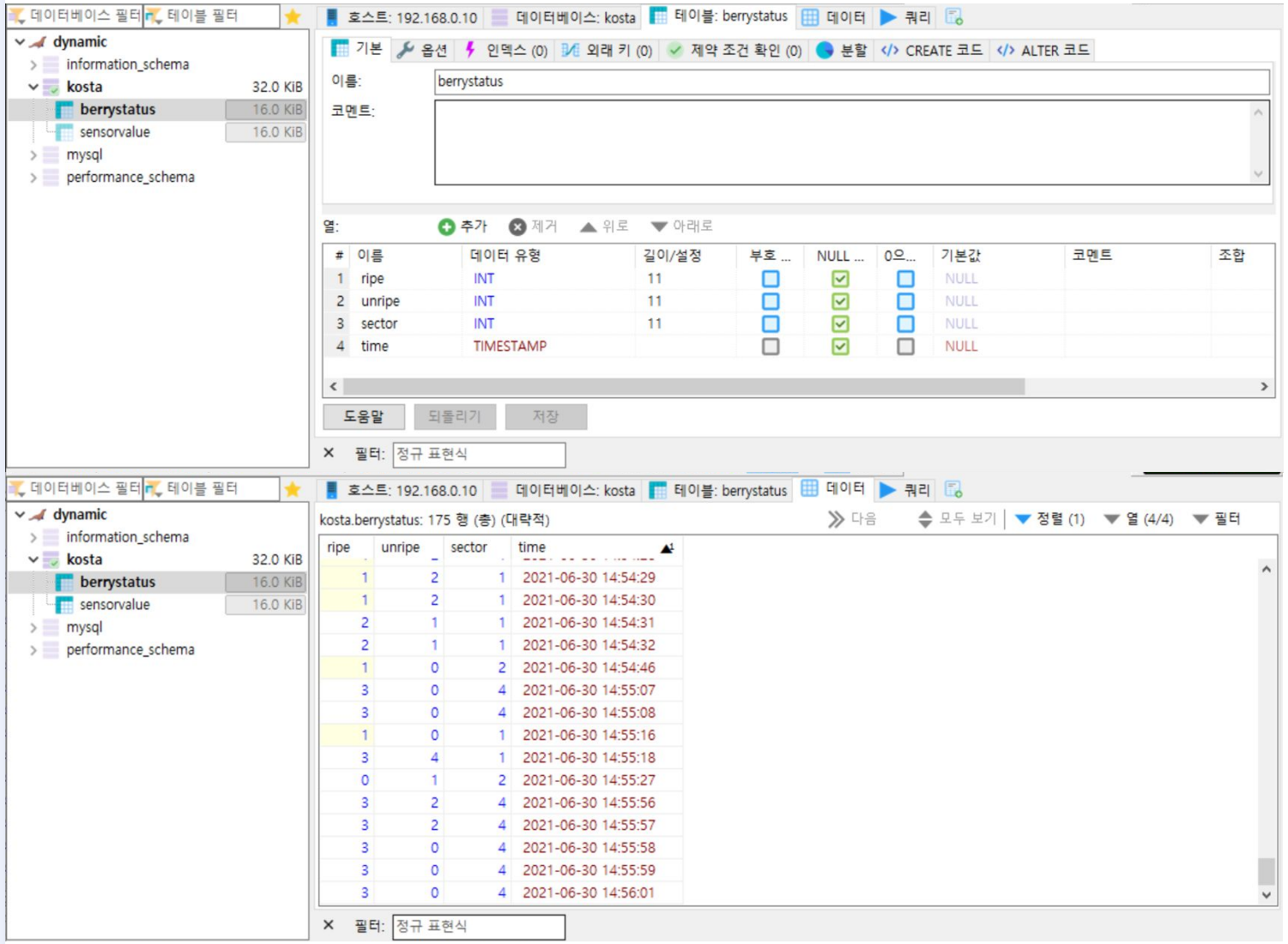
구역이 변경될 때마다 온습도 값을 데이터베이스에 전송하였다.

insert 구문을 활용하여 **insert into sensorvalue values %s, %s, %s, now()** 로 데이터를 전송했다.

문자열로 보낸 데이터를 데이터베이스에서는 정수형으로 변환해 저장했다.

시간은 timestamp 단위로 받으며 이를 활용해 winForm에서 시간순으로 받은 데이터를 토대로 실시간으로 그래프를 그리게 했다.

Raspberry Pi 4 > Database(berry status)



ripe(익은 딸기의 수), unripe(덜익은 딸기의 수), sector(구역), time(시간) 데이터를 berrystatus 데이터베이스로 전송한다.

mariaDB를 활용해 mysql 구문으로 작성하였다.

딸기가 인식되었고 특정한 구역에 위치했을 경우 딸기 값을 데이터베이스에 전송하였다.

insert 구문을 활용하여

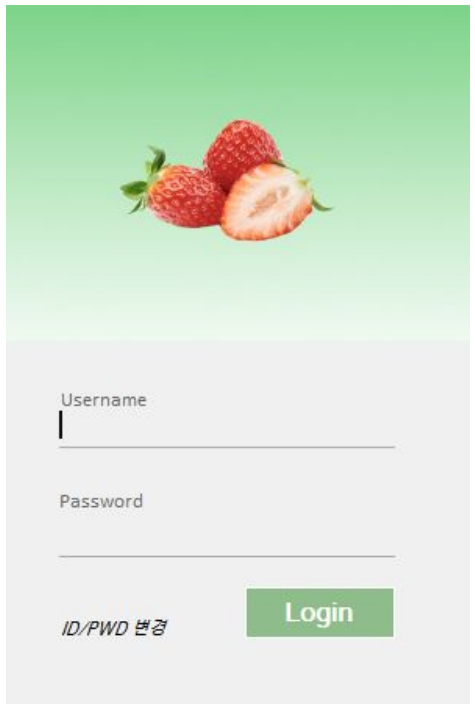
insert into berrystatus values %s, %s, %s, now() 로 데이터를 전송했다.

문자열로 보낸 데이터를 데이터베이스에서는 정수형으로 변환해 저장했다.

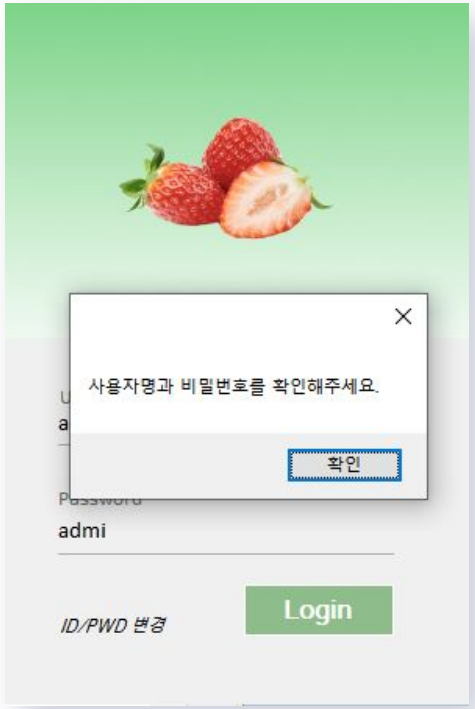
시간은 timestamp 단위로 받으며 이를 활용해 winForm에서 시간순으로 받은 데이터를 토대로 실시간으로 그래프를 그리게 했다.

WinForm For Data Visualization

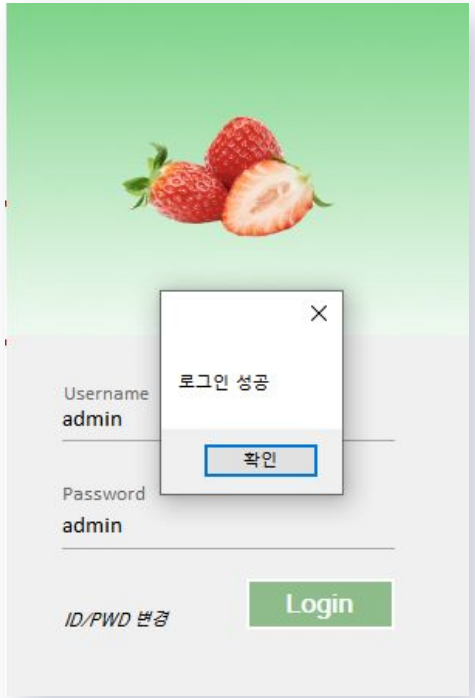
1



2



3

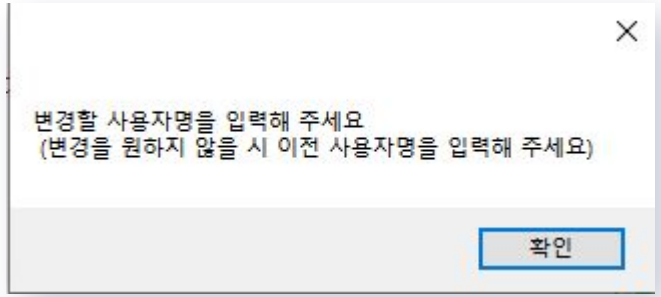


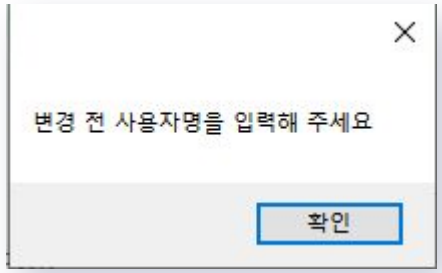
1. **로그인** 화면을 통해 관리자가 데이터 확인 페이지로 접속할 수 있습니다.
2. 로컬 데이터베이스의 사용자 정보를 확인하여 정확한 정보인지 확인합니다.
3. 로컬 데이터베이스의 사용자 정보와 동일함이 확인되면 로그인 성공

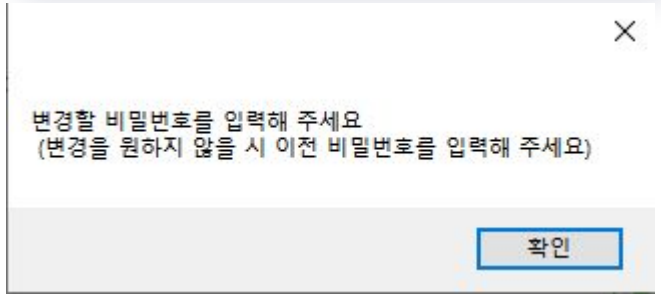
WinForm For Data Visualization

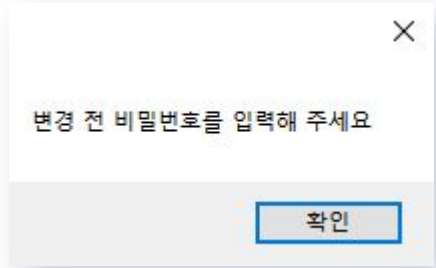
1

2









1. **정보수정** 화면을 통해 관리자의 ID와 PW를 수정할 수 있습니다.
2. 정확한 정보를 입력하도록 오류 메시지를 세분화하여 보여줍니다.

WinForm For Data Visualization

1 **온습도 실시간 그래프** 탭에서 실시간 온습도 데이터를 그래프로 확인할 수 있습니다.

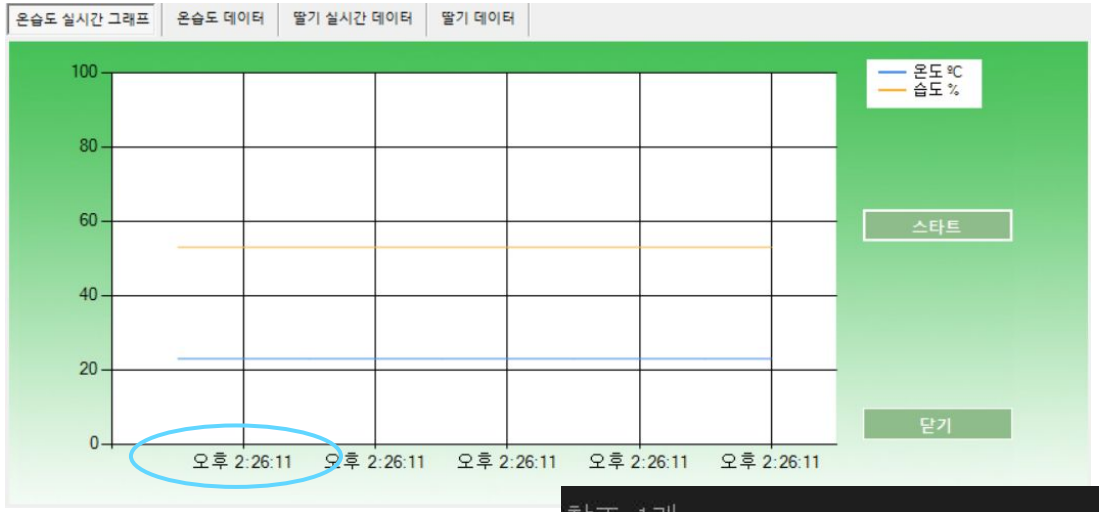
2 **온습도 데이터** 탭에서 DB에 기록된 온습도 데이터 전부를 그리드뷰로 확인할 수 있습니다.

3 **딸기 실시간 데이터** 탭에서 실시간 딸기 숙성도 데이터를 그래프로 확인할 수 있습니다.

4 **딸기 데이터** 탭에서 섹터 별 딸기 숙성도 데이터를 그래프로 확인할 수 있습니다.

WinForm For Data Visualization

온습도 실시간 그래프 탭에서 실시간 온습도 데이터를 그래프로 확인할 수 있습니다.



데이터를 받은 시간(timestamp)

SELECT MAX(TIME)구문을 통해 현재 시각에서 가장 최근의 데이터를 받도록 했습니다.

```
참조 1개
private void btnClimateStart_Click(object sender, EventArgs e)
{
    Tname = "sensorvalue";
    QueryTable = "SELECT * FROM " + Tname + " WHERE 1=1 AND TIME=(SELECT MAX(TIME) FROM " + Tname + ")";

    if (btnClimateStart.Text == "그래프 정지")
    {
        btnClimateStart.Text = "스타트";
        RefreshToggle();
    }
    else
    {
        btnClimateStart.Text = "그래프 정지";
        RefreshToggle();
        RefreshOn();
```

센서 정보를 담는 데이터베이스 이름

WinForm For Data Visualization

온습도 데이터 탭에서 DB에 기록된 온습도 데이터 전부를 그리드뷰로 확인할 수 있습니다.

53	23	4	2021-06-30 오후 2:24
53	23	2	2021-06-30 오후 2:24
53	23	3	2021-06-30 오후 2:24
53	23	4	2021-06-30 오후 2:25
53	23	4	2021-06-30 오후 2:25
53	23	3	2021-06-30 오후 2:25
53	23	4	2021-06-30 오후 2:25
53	23	3	2021-06-30 오후 2:26

습도 / 온도 / 구역번호 / 데이터를 받은 시간(timestamp)

```
참조 1개
private void TabPage2_Enter(object sender, EventArgs e)
{
    Tname = "sensorvalue";
    string tempsql = "SELECT * FROM " + Tname;
    using (MySqlConnection amySqlConnection = new MySqlConnection(QueryLogin))
    {
        MySqlDataAdapter mySqlDataAdapter = new MySqlDataAdapter(tempsql, amySqlConnection);
        DataSet ds = new DataSet();
        mySqlDataAdapter.Fill(ds);
        dataGridView1.DataSource = ds.Tables[0];
    }
}
```

센서 정보를 담는 데이터베이스 이름

WinForm For Data Visualization

딸기 실시간 데이터 탭에서 실시간 딸기 숙성도 데이터를 그래프로 확인할 수 있습니다.

DB에 기록된 딸기 개수를 바탕으로
STACKEDCOLUMN 형식으로
막대그래프로 확인할 수 있게 했습니다.

SELECT MAX(TIME) 구문을
통해
가장 최근의 시간의 데이터를
받을 수 있게 했습니다.

딸기 정보를 담는 데이터베이스 이름

```
참조 1개
private void btnBerryStart_Click_1(object sender, EventArgs e)
{
    Tname = "berrystatus";//여기에 그 음...딸기의 데이터베이스 이름
    QueryTable = "SELECT * FROM " + Tname + " WHERE 1=1 AND TIME=(SELECT MAX(TIME) FROM " + Tname + ")";


    //딸기의 그래프 시작 페이지
    if (btnBerryStart.Text == "그래프 정지")
    {
        btnBerryStart.Text = "스타트";
        RefreshToggle();
    }
```

WinForm For Data Visualization

딸기 데이터 탭에서 섹터 별 딸기 숙성도 데이터를 그래프로 확인할 수 있습니다.

DB의 실시간 데이터셋을 sector별 누적 개수를 파악하기 위해 덜 익은 딸기와 잘 익은 딸기의 개수를 배열에 누적시켜 막대그래프로 나타내었습니다.

```
int[] unripeArray = new int[4];
int[] ripeArray = new int[4];
int sector, lessripe, wellripe;

foreach (DataRow item in aDataSet.Tables[Tname].Rows)
{
    sector = int.Parse(item["SECTOR"].ToString());
    lessripe = int.Parse(item["UNRIPE"].ToString());
    wellripe = int.Parse(item["RIPE"].ToString());

    unripeArray[sector - 1] += lessripe;
    ripeArray[sector - 1] += wellripe;
}

chart1.Series[0].Name = "UNRIPE";
chart1.Series[1].Name = "RIPE";
chart1.Series[0].Points.DataBindY(unripeArray);
chart1.Series[1].Points.DataBindY(ripeArray);
```

01

contents

Summary

기획일정	1W					2W					3W					4W					5W				
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
벤치마킹 및 아이디어 기획																									
개발 계획서 작성																									
설계서 작성																									
프로그래밍																									
발표자료 및 최종 보고서																									

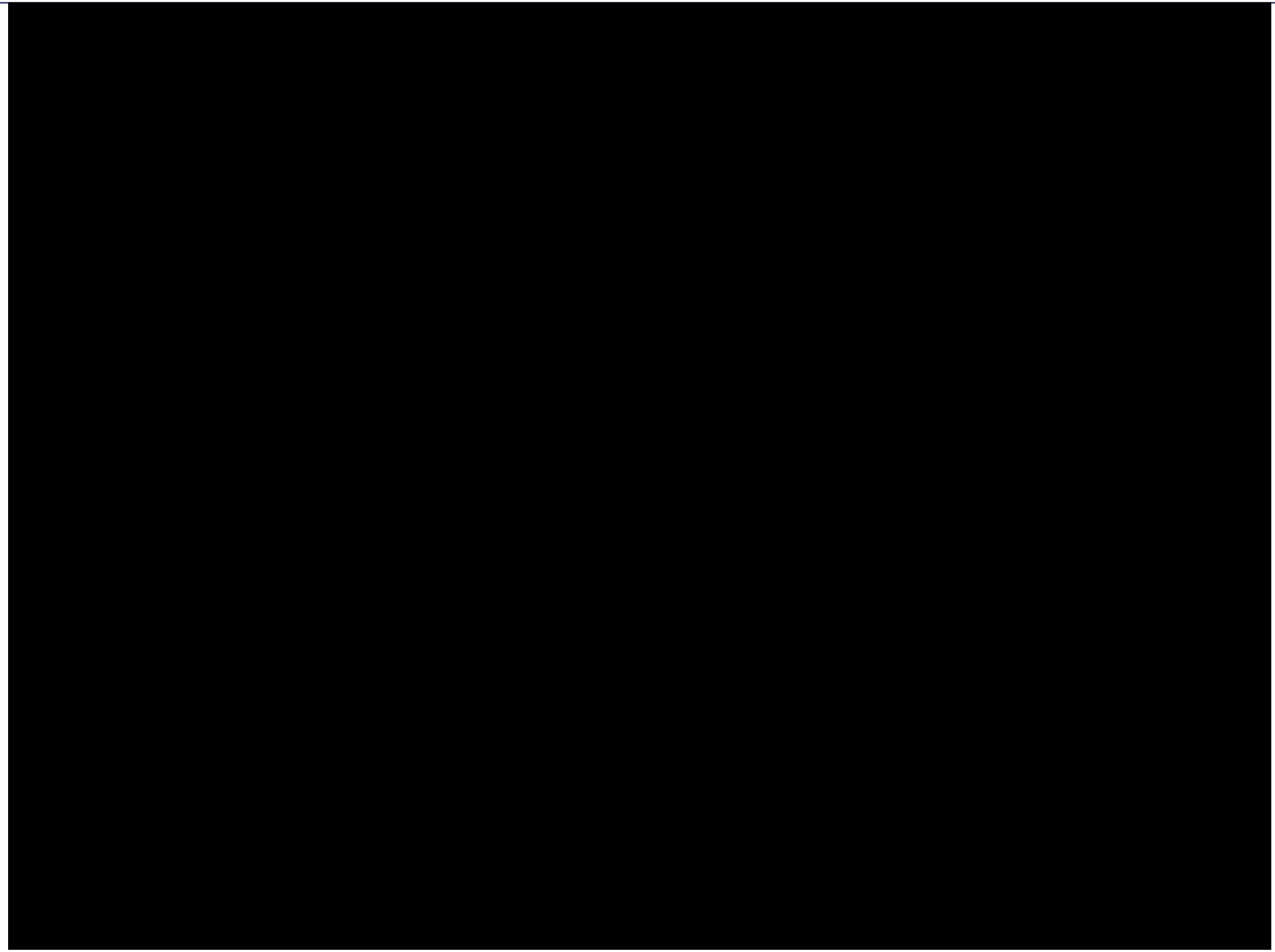
페이지 기능 간단하게 구현
로그인, 정보수정
카봇에서 전송받은 데이터를
데이터베이스화

구체적인 기능들을 위주로 프로그래밍
ex) 딥러닝 카메라 인식, 라인트레이싱,
데이터 가공 및 열람 페이지

01

contents

구현 동영상



계획 대비 결과 및 기대효과

- sector를 더 여러개 구현하고 싶었으나, 조명 값에 따른 컬러센서의 값 변화로 인해 4개까지 구현하는 것으로 마무리지었다.
-
- 주행하면서 데이터를 실시간으로 받아오고 싶었는데 카메라의 frame이 낮아 딸기 인식률이 낮았다.
- 따라서 CarBot의 주행과 정지를 반복하며 딸기의 인식률을 높였다.
-
- 온습도는 트랙의 사이즈가 작아지면서 유의미한 데이터를 받아올 수는 없었지만, 프로토타입인만큼 실제 농장에 적용된다면 값 변화가 있을 것으로 예상된다.
- 정확하게 개수를 세지 못한 점이 아쉽지만, 구역의 덜익은 딸기와 익은 딸기의 비율을 확인하는 페이지를 통해 대략적인 해당 구역의 비율을 확인할 수 있었다.

본 프로젝트는 식별하고자 하는 이미지를 직접 클래스를 나눠 라벨링하고 학습시킴으로써 실시간으로 객체 상태를 식별하고 다양한 주변데이터를 수집하는 프로젝트이다.
해당 데이터가 꾸준히 축적이되어 빅데이터화 된다면 다양한 인사이트를 도출할 수 있을것으로 기대되며 딸기 성장 상태 식별은 여러 주제들의 일부분일 뿐, 이러한 메커니즘은 스마트팜, 스마트팩토리 등 다양한 도메인에 구애받지 않고 적용가능하다는 점에서 그 의의가 있다.

역할 분담

김지현
아두이노 라인트레이싱 및 모터 제어
비주얼스튜디오 윈폼 제작
컬러 센서, 온습도 센서 데이터 활용 알고리즘 적용
mariaDB data set을 추출하여 winForm 그래프로 시각화

이지아
아두이노 라인트레이싱 및 모터 제어
비주얼 스튜디오 윈폼 제작
컬러 센서, 온습도 센서 데이터 활용 알고리즘 적용
mariaDB data set을 추출하여 winForm 그래프로 시각화

배문규
Training Image Dataset Crawling and Labeling
yolo-tiny Modeling
라즈베리파이 카메라로 객체 인식 및 검출
아두이노 초음파 센서, 컬러 센서 데이터 활용 알고리즘 적용
라즈베리파이에 mariaDB로 데이터 송신 기능 적용