

AIoT SerBot Prime X로 배우는

# 온디바이스 AI 프로그래밍

# 사진 분류













- 카메라를 응용하여 특정 사물을 구분하는 사물 인식 인공지능 예제
  - ▣ 휴대전화 카메라나 장비의 카메라를 이용해 사진 데이터 수집
  - ▣ 이 예제에서는 마우스와 마우스가 아닌 것을 구분
  - ▣ 다양한 배경, 다양한 조명에서 찍힌 마우스 사진과 다른 사물 사진 준비
  - ▣ 정확한 학습을 위해서는 각 클래스별 사진 500장 이상이 적절
  - ▣ 간단한 설명과 빠른 진행을 위해 30장 내외의 사진으로 학습
  - ▣ OpenCV와 Numpy를 활용
    - 사진을 일일이 배열로 하드코딩 하기에는 무리가 있으므로 대체

# 사진 분류



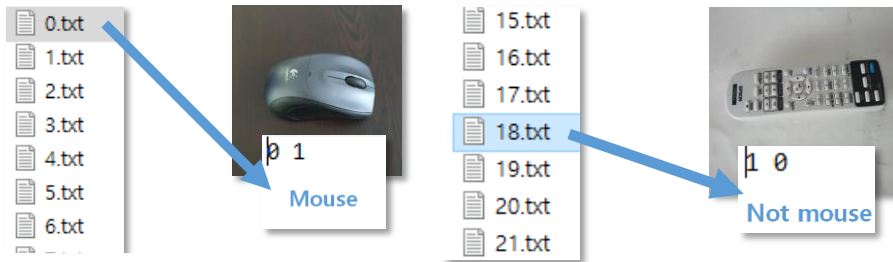
# 사진 분류

- 사진 수집이 끝났다면 Samba, USB 등을 이용해 이미지를 장비로 복사
- 파일명을 숫자로 수정
  - for 문을 이용해 쉽게 접근하기 위해

 0.jpg  
 1.jpg  
 2.jpg  
 3.jpg  
 4.jpg  
 5.jpg  
 6.jpg  
 7.jpg  
 8.jpg  
 9.jpg  
 10.jpg  
 11.jpg

# 사진 분류

- ▣ 각 사진에 관한 결과 데이터 생성
  - 텍스트 파일을 만들어 데이터 만들
  - 마우스가 아닌 사진이라면 첫 번째 숫자가 1
  - 마우스 사진이라면 두 번째 숫자가 1로 지정하고 두 숫자는 ' '(스페이스 바)로 구분
  - 텍스트 파일의 이름은 각 사진의 이름과 일치하도록 함
    - for 문 사용의 편의를 위해
  - 이 과정을 라벨링(Labeling)이라고 함



# 사진 분류

- ▣ 사용해야 할 패키지 import, 입력 데이터와 결과 데이터를 리스트로 선언
- ▣ for 문을 사진 개수만큼 반복하는 루프 생성
- ▣ cv2.imread() 메소드로 사진을 읽은 후 cv2.cvtColor() 메소드로 흑백사진으로 변환
- ▣ cv2.resize() 메소드로 이미지 사이즈를 50x50로 하고 X\_data 리스트에 추가
- ▣ 라벨링 했던 파일을 numpy의 loadtxt() 메소드로 읽어 Y\_data 리스트에 추가
- ▣ 이 과정은 사진 개수에 따라 수 분이 걸릴 수 있음

# 사진 분류

---

```
01:         from pop import Al, Camera
02:         import numpy as np
03:         import cv2
04:
05:         IMG_AMOUNT = 33
06:
07:         X_data=[]
08:         Y_data=[]
09:
10:         for i in range(IMG_AMOUNT):
11:             img=cv2.imread('./img/'+str(i)+'.jpg')
12:             img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
13:             img=cv2.resize(img, (50,50), interpolation=cv2.INTER_AREA)
14:             X_data.append(img.reshape(50,50,1).astype(float))
15:             y=np.loadtxt('./img/'+str(i)+'.txt')
16:             Y_data.append(y)
```

---

# 사진 분류

- ▣ CNN 객체의 파라미터에 input\_size를 [50, 50], output\_size를 2로 주고 생성
- ▣ CNN의 X\_data와 Y\_data 지정
- ▣ 충분한 학습을 위해 train() 메소드의 파라미터에 times를 500으로 설정
- ▣ 데이터 학습

---

```
17:         CNN=AI.CNN(input_size=[50,50], output_size=2)
18:         CNN.X_data=X_data
19:         CNN.Y_data=Y_data
20:
21:         CNN.train(times=500)
```

---



# 사진 분류

- ▣ 학습된 모델의 `run()` 메소드에 CNN의 `X_data` 중 임의 데이터를 입력하여 실행

---

```
22: CNN.run([CNN.X_data[20]])
```

```
23: CNN.run([CNN.X_data[7]])
```

---

# 사진 분류

- ▣ 카메라의 이미지를 읽어와 인식
- ▣ Pop 라이브러리의 Camera 객체 생성

---

```
24:         cam = Camera()
```

---

# 사진 분류

- ❑ Camera 객체에서 카메라의 장면을 읽어오고 cvtColor() 메소드로 흑백으로 변환
- ❑ resize() 메소드로 50x50 사이즈로 축소
- ❑ 카메라 이미지를 run() 메소드에 입력하고 결과 확인

---

```
25:         frame = cam.value
26:         frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
27:         frame = cv2.resize(frame, (50, 50), interpolation=cv2.INTER_AREA)
28:         CNN.run([frame.reshape(50,50,1)])
```

---

# 사진 분류

## □ 전체 코드

```
01: from pop import AI, Camera
02: import numpy as np
03: import cv2
04:
05: IMG_AMOUNT = 33
06:
07: X_data=[]
08: Y_data=[]
09:
10: for i in range(IMG_AMOUNT):
11:     img=cv2.imread('./img/'+str(i)+'.jpg')
12:     img=cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
13:     img=cv2.resize(img, (50,50), interpolation=cv2.INTER_AREA)
14:     X_data.append(img.reshape(50,50,1).astype(float))
15:     y=np.loadtxt('./img/'+str(i)+'.txt')
16:     Y_data.append(y)
```

```
17:
18: CNN=AI.CNN(input_size=[50,50], output_size=2)
19: CNN.X_data=X_data
20: CNN.Y_data=Y_data
21:
22: CNN.train(times=500)
23:
24: cam = Camera()
25:
26: frame = cam.value
27: frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
28: frame = cv2.resize(frame, (50, 50), interpolation=cv2.INTER_AREA)
29: CNN.run([frame.reshape(50,50,1)])
```

# 음성 분류

- 마이크를 응용하여 0~9 사이 숫자 음성을 인식하는 인공지능
  - ▣ 0 부터 9 까지의 음성 데이터를 수집
  - ▣ 다양한 사람, 다양한 톤, 다양한 크기(dB)로 녹음
  - ▣ 적절한 학습을 위해서는 각 숫자별 음성 데이터셋 200개 이상이 필요

# 음성 분류

- ▣ 데이터를 수집하기에는 시간이 충분치 않은 경우
  - 오픈 데이터셋 사이트인 Kaggle에서 데이터셋을 구할 수 있음
  - Kaggle에 공개된 SDD(Spoken Digit Dataset) 프로젝트의 데이터셋만 추출한 파일
    - [https://github.com/hanback-docs/Spoken\\_Digit\\_Dataset](https://github.com/hanback-docs/Spoken_Digit_Dataset)
  - 영어로 녹음된 파일이며 4명의 목소리만 있음
    - 실제 사용했을 때 다른 사람의 목소리를 제대로 인식 못할 가능성이 있음
    - 이 데이터셋에 본인 또는 다른 사람의 목소리를 더 추가하여 사용하면 좋음
  - 녹음을 할 때는 다른 파일들과 비슷한 길이인 1초 내외로 해야함

# 음성 분류

- ▣ Pop.Dataset 라이브러리 import
  - 수월한 음성 데이터 수집을 위해 한백전자에서 제공하는 라이브러리
- ▣ Pop의 Dataset 라이브러리에서 Collector() 메소드에 "Audio" 인자를 입력
- ▣ 오디오 컬렉터 실행

---

01:            from pop import Dataset  
02:  
03:            Dataset.Collector("Audio")

---

The screenshot displays the Pop Dataset Collector interface, which is divided into two main sections. The left section contains recording controls: a 'Time' indicator with an orange bar, a 'Duration (sec)' input field set to '1', a red 'REC' button, and a playback slider showing '0:00 / 0:00'. The right section contains form fields for 'Path' (set to '/audio\_datasets'), 'Label' (set to '0'), and 'Name' (set to 'identifier'). A green 'Save' button is located at the bottom of the form section.

# 음성 분류

- ▣ REC 버튼을 클릭하면 1초 뒤 녹음 시작
- ▣ 녹음 결과 플레이어로 확인 가능
- ▣ Save 버튼으로 'Label\_Name\_녹음시간.wav' 이름의 파일로 저장
- ▣ Save 버튼으로 저장하지 않으면 삭제
  - Duration (숫자 텍스트) : 녹음 시간, 초 단위
  - REC (버튼) : 녹음 시작
  - Path (텍스트) : 데이터셋을 저장할 경로
  - Label (숫자 텍스트) : 학습 과정의 Y값에 해당하는 라벨
  - Name (텍스트) : 파일 식별을 위한 이름
  - Save (버튼) : 저장



# 음성 분류

- ▣ 데이터 수집이 끝났다면 사용해야 할 패키지 import
- ▣ 입력 데이터와 결과 데이터를 리스트로 선언
- ▣ os 모듈을 사용해 데이터셋 리스트를 수집
- ▣ 음성 데이터를 MFCC 데이터로 변환하는 과정을 데이터셋 개수만큼 for문 반복
  - MFCC : 부록 참고
- ▣ 파일명에서 Label을 추출해 One Hot 데이터로 변환

# 음성 분류

- ▣ Util의 toMFCC() 메소드를 사용할 때는 파일명과 재생 시간을 파라미터로 입력
- ▣ 해당 파일의 실제 재생 시간과 파라미터 입력 길이를 최대한 맞춰 녹음
  - 실재 재생 시간 < 파라미터 입력 값: 부족한 부분을 0으로 채움
  - 실재 재생 시간 > 파라미터 입력 값: 나머지 부분을 잘라냄
- ▣ 이 과정은 데이터셋 개수에 따라 수 분이 걸릴 수 있음

# 음성 분류

---

```
04:         from pop import AI
05:         from pop import Util
06:         import os
07:         import numpy as np
08:
09:         X_data=[]
10:         Y_data=[]
11:
12:         datalist = os.listdir("audio_datasets")
13:
```

---

---

```
14:         for data in datalist:
15:             feat = Util.toMFCC("audio_datasets/" + data,
                                duration=1)
16:             label = int(data.split("_")[0])
17:             label = Util.one_hot(label,10)
18:
19:             X_data.append(feat)
20:             Y_data.append(label)
21:
22:         X_data=np.array(X_data)
23:         Y_data=np.array(Y_data)
```

---

# 음성 분류

- CNN 객체의 파라미터 설정후 생성
  - input\_size : 데이터셋 하나의 크기, output\_size : 10
- CNN의 X\_data와 Y\_data 지정
- train() 메소드의 파라미터에 times를 100으로 설정하고 데이터 학습

---

```
23: dataset_size = X_data.shape[1:3]
24: CNN=AI.CNN(input_size=dataset_size, output_size=10)
25:
26: CNN.X_data=X_data
27: CNN.Y_data=Y_data
28:
29: CNN.train(times=100)
```

---

# 음성 분류

- ▣ 학습된 모델의 run 메소드에 CNN의 X\_data 중 임의 데이터를 입력하여 실행
- ▣ 결과 비교를 위해 Y\_data를 함께 출력

---

```
30: Y=CNN.Y_data[20]
31: R=CNN.run([CNN.X_data[20]])
32:
33: print(Y)
34: print(R)
35: Y=CNN.Y_data[7]
36: R=CNN.run([CNN.X_data[7]])
37:
38: print(Y)
39: print(R)
```

---

# 음성 분류

- ▣ 마이크에서 음성을 읽어와 인식
  - Pop.Dataset의 Collector() 메소드 활용

---

```
40:         Dataset.Collector("Audio")
```

---

- ▣ 오디오 컬렉터로 숫자 음성을 녹음, 경로를 지정해 파일 저장
  - 파일명은 'Label\_Name\_녹음시간.wav' 형식으로 저장
- ▣ 저장된 녹음 파일을 MFCC 데이터로 변환하고 학습된 모델에 입력

---

```
41: data=Util.toMFCC("audio_datasets/9_identifier_2004132053280.wav", duration=1)
```

```
42: CNN.run([data])
```

---

# 음성 분류

## □ 전체 코드

```
01:from pop import Dataset
02:
03:Dataset.Collector("Audio")
04:
05:from pop import AI
06:from pop import Util
07:import os
08:import numpy as np
09:
10:X_data=[]
11:Y_data=[]
12:
13:datalist = os.listdir("audio_datasets")
14:
15:for data in datalist:
16:    feat = Util.toMFCC("audio_datasets/" + data, duration=1)
```

```
17:    label = int(data.split("_")[0])
18:    label = Util.one_hot(label,10)
19:
20:    X_data.append(feat)
21:    Y_data.append(label)
22:
23:X_data=np.array(X_data)
24:Y_data=np.array(Y_data)
25:
26:dataset_size = X_data.shape[1:3]
27:CNN=AI.CNN(input_size=dataset_size, output_size=10)
28:
29:CNN.X_data=X_data
30:CNN.Y_data=Y_data
31:
```

# 음성 분류

---

```
32:CNN.train(times=100)
33:
34:Y=CNN.Y_data[20]
35:R=CNN.run([CNN.X_data[20]])
36:
37:print(Y)
38:print(R)
39:
40:Dataset.Collector("Audio")
41:
42:data=Util.toMFCC("audio_datasets/9_identifier_2004132053280.wav", duration=1)
43:CNN.run([data])
```

---



# 장애물 회피

## □ 장애물 회피

- ▣ 카메라 전방에 근접한 사물이 있는 지 학습하여 장애물을 감지하는 모델
  - 합성곱 신경망을 이용
- ▣ 여러 장애물 데이터셋을 수집

# 장애물 회피

- 데이터 수집
  - ▣ Pop.Pilot라이브러리의 Data\_Collector 클래스
    - 이미지 데이터 수집
  - ▣ Pop.Pilot라이브러리의 Camera 클래스
    - 카메라 사용

# 장애물 회피

- ▣ Pop에서 Pilot라이브러리 import
- ▣ Camera 클래스를 width와 height 파라미터에 300을 입력
  - 300 x 300 사이즈로 생성
- ▣ Camera 클래스의 show() 메소드를 이용해 실시간 영상 확인

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:         cam.show()
```

---

# 장애물 회피

- stop() 메소드 : 영상 정지

- 실제 촬영은 백그라운드에서 지속되고 있고 보이는 영상만 정지

---

05:            `cam.stop()`

---

- Pilot라이브러리에서 Data\_Collector클래스 생성

- 생성 파라미터로 Pilot.Collision\_Avoid 입력

- Collision\_Avoid 데이터 수집이 목적임을 명시하기 위해

- camera 파라미터에 앞서 생성한 카메라 클래스 입력

- 카메라 영상 이용

---

06:            `dataCollector=Pilot.Data_Collector(Pilot.Collision_Avoid, camera=cam)`

---

# 장애물 회피

## ▣ Data\_Collector 클래스의 show() 메소드

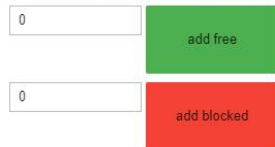
### ■ 데이터 수집을 위한 GUI환경 표시

---

07:            dataCollector.show()

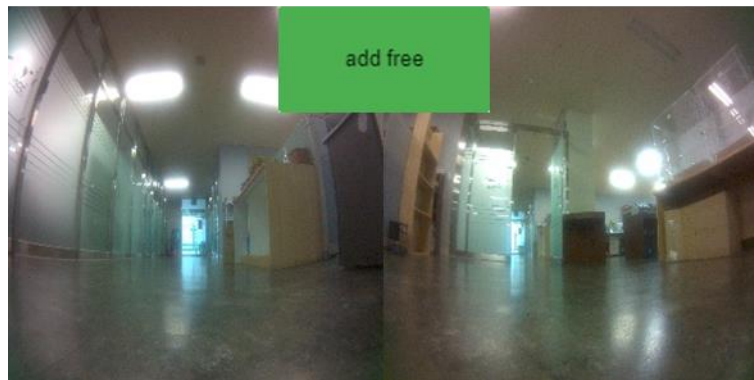
---

- 데이터를 위해 실시간 영상, 2개의 버튼, 조이스틱 표시
- 조이스틱 : 차량 제어
- add free 버튼 : 현재 표시되고 있는 장면을 장애물이 없는 데이터로 분류
- add blocked 버튼 : 현재 표시되고 있는 장면을 장애물 데이터로 분류



# 장애물 회피

- 조이스틱을 이용해 차량을 움직이며 데이터 수집
- 데이터는 각각 300장 이상이 적절
  - 실내, 실외, 타일, 장판 등 최대한 다양하고 많은 데이터를 수집해야 양질의 모델 생성 가능
- 사진 데이터는 현재 경로에서 collision\_dataset 디렉토리의 free, blocked 디렉토리에 저장



# 장애물 회피

## □ 데이터 딥러닝

### ▣ Collision\_Avoid 클래스 생성

- 장애물 인식을 위한 합성곱 신경망이 사전 구성된 클래스

### ▣ 생성 파라미터에 앞서 생성한 camera 클래스 입력하여 생성

- 카메라 영상 이용

### ▣ 앞서 수집한 데이터셋들 로드

- Collision\_Avoid 클래스의 load\_datasets() 메소드 이용

---

08: CA=Pilot.Collision\_Avoid(cam)

09: CA.load\_datasets()

---

# 장애물 회피

- ▣ 데이터셋 로드가 완료되면 train() 메소드를 이용해 학습 시작
  - train() 메소드의 times 파라미터 : 학습할 횟수 지정
  - train() 메소드를 통해 학습이 진행되면 매 스텝마다 자동으로 학습 모델 저장
  - 자동 저장을 비활성화 하고싶다면 autosave 파라미터를 False로 지정
  - 정확도가 1에 가까울수록 오차가 작은 모델

---

```
10:         CA.train(times=10)
```

---

- ▣ run() 메소드를 실행하고 모델의 예측값 출력

---

```
11:         value=CA.run()  
12:         print(value)
```

---



# 장애물 회피

- ▣ 현재 카메라의 장면을 이용해 앞에 장애물이 있을 확률을 0~1의 범위로 출력
- ▣ show() 메소드 실행
  - 어떤 장면을 장애물이라 판단했는지 알기 위해 사용
  - 현재 카메라 영상과 run()메소드의 결과를 직관적으로 표시
  - show() 메소드는 run() 메소드와 별개로 한 번만 실행하면 계속해서 표시 됨

---

13:           CA.show()

---

# 장애물 회피

- ▣ Callback 메소드를 선언하고 run() 메소드의 callback 파라미터로 입력
  - 실행 결과값을 해당 메소드로 넘김

---

```
14:         def is_blocked(value):
15:             print(value>0.5)
16:
17:         value=CA.run(callback=is_blocked)
18:         print(value)
```

---

# 장애물 회피

- ▣ 차량 제어를 위해 Pop.Pilot 라이브러리에서 SerBot 클래스 생성
  - Callback 기능을 이용해 차량 제어
- ▣ Callback 메소드 선언
  - 모델 예측값이 0.5를 초과하면 우측으로 후진
  - 0.5 이하면 전진

```
19:         bot=Pop.Pilot.SerBot()
20:         bot.setSpeed(50)
21:
22:         def drive(value):
23:             if value<=0.5:
24:                 bot.steering=0
25:                 bot.forward()
```

```
26:         else:
27:             bot.steering=1
28:             bot.backward()
29:
30:         while True:
31:             CA.run(callback=drive)
```

# 장애물 회피

## ▣ 부가적 기능

- load\_model(path) 메소드 : 저장된 학습 모델을 로드하여 이어서 학습

---

```
CA.load_model(path= "collision_avoid_model.pth")
```

---

- save\_model(path) 메소드 : 원하는 경로에 수동으로 모델 저장

---

```
CA.save_model(path= "collision_avoid_model.pth")
```

---

# 장애물 회피

## □ 전체 코드 : 데이터 수집

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         dataCollector=Pilot.Data_Collector(Pilot.Collision_Avoid, camera=cam)
06:         dataCollector.show()
```

---

# 장애물 회피

## □ 전체 코드 : 딥러닝

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         CA=Pilot.Collision_Avoid(cam)
06:         CA.load_datasets()
07:
08:         CA.train(times=10)
09:
10:         CA.show()
11:
12:         bot=Pilot.SerBot()
```

---

---

```
13:         bot.setSpeed(50)
14:
15:         def drive(value):
16:             if value<=0.5:
17:                 bot.steering=0
18:                 bot.forward()
19:             else:
20:                 bot.steering=1
21:                 bot.backward()
22:
23:         while True:
24:             CA.run(callback=drive)
```

---

# 목표물 추적

## □ 목표물 추적

- ▣ 특정 사물을 지정하여 차량이 따라가는 예제
- ▣ 사전 학습된 사물 인식 모델을 이용하여 여러 사물들을 인식
- ▣ Pop.Pilot 라이브러리 사용

# 목표물 추적

## □ 사전 학습 모델 다운로드

### ▣ 공개된 사물 인식 모델 사용

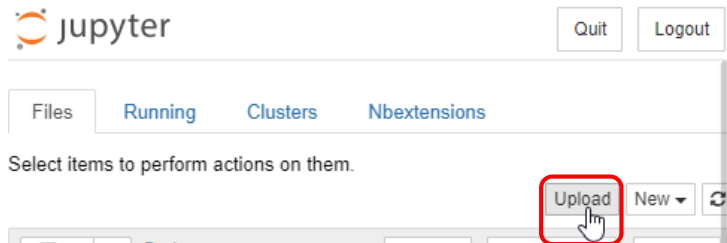
### ▣ `ssd_mobilenet_v2_coco.engine` 파일 다운로드

#### ■ 인식 가능한 사물 목록 확인 가능

#### ■ [https://github.com/hanback-docs/ssd\\_mobilenet\\_v2\\_coco\\_engine](https://github.com/hanback-docs/ssd_mobilenet_v2_coco_engine)



### ▣ Jupyter 경로에 업로드





# 목표물 추적

## □ 학습 모델 활용

### ▣ 카메라 클래스를 생성하고 Object\_Follow 클래스 생성

- Object\_Follow 클래스 : 최적의 사물 인식을 위한 합성곱 신경망 구현되어 있음

### ▣ 다운로드 받은 모델 파일 로드

- Object\_Follow 클래스의 load\_model(path) 메소드 사용
- 모델 파일을 로드할 때 path가 입력되지 않은 경우 현재 경로에서 가져 옴

---

```
01: from pop import Pilot
02:
03: cam=Pilot.Camera(width=300, height=300)
04: bot=Pilot.SerBot()
```

---

---

```
05:
06: OF=Pilot.Object_Follow(cam)
07: OF.load_model()
```

---

# 목표물 추적

- ▣ 모델 로드가 완료되면 detect() 메소드를 이용해 사물을 인식하고 결과 반환
  - index파라미터에 1 또는 'person'이라고 입력하고 실행하면 사람 인식
  - index가 입력되지 않으면 인식할 수 있는 모든 사물에 대한 인식 결과 반환
  - index에 관한 입력과 인식가능한 목록
    - [https://github.com/hanback-docs/ssd\\_mobilenet\\_v2\\_coco\\_engine](https://github.com/hanback-docs/ssd_mobilenet_v2_coco_engine)



# 목표물 추적

## ■ detect 메소드의 파라미터

### ■ image

- 직접 이미지 데이터를 입력하여 인식
- 현재 클래스에 지정된 카메라 영상이 아닌 다른 데이터를 인식해보고자 할 때 사용
- 기본값: Camera.value

### ■ index

- 어떤 사물을 인식할 지 지정 가능
- index를 입력하지 않을 경우 인식가능한 모든 사물에 대한 결과 반환
- 기본값: All

### ■ show

- 실행 결과를 그래픽 요소로 표현할 지에 대한 여부
- 기본값: True

# 목표물 추적

## ▣ index에 'person'을 입력

---

```
08:         v=OF.detect(index='person')
09:         print(v)
```

---

[출력]

```
{'x': 0.02924691140651703, 'y': 0.26995646953582764, 'size_rate': 0.18161612565622676}
```

- 사람을 구분하여 반환되는 값 확인
- 인식된 'person'개체들 중 가장 사이즈가 큰 개체에 대한 결과를 딕셔너리 형태로 반환
- x와 y는 인식된 개체의 상대적 좌표이며 화면 정중앙을 0으로 하고 -1 ~ 1의 범위로 표현
- size\_rate는 개체의 크기
- 인식된 개체의 크기가 1에 가까울수록 거리가 가깝다는 의미로 해석 가능

# 목표물 추적

- ▣ index 입력없이 모든 사물에 대한 데이터 확인

---

```
10:         v=OF.detect()  
11:         print(v)
```

---

[출력]

```
[[{'label': 76, 'confidence': 0.9231778383255005, 'bbox': [0.143454909324646,  
0.062126800417900085, 0.8638043403625488, 0.3971424102783203]}],  
{ 'label': 74, 'confidence': 0.6338636875152588, 'bbox': [0.6076321601867676,  
0.5661253333091736, 0.7583435773849487, 0.8835135102272034]}],  
{ 'label': 77, 'confidence': 0.6270381212234497, 'bbox': [0.259377121925354,  
0.4745787978172302, 0.4690379500389099, 0.9588668942451477]}]]
```

- index가 76인 'Keyboard'인식
- index가 74인 'mouse'인식
- index가 77인 'cell phone' 인식

# 목표물 추적

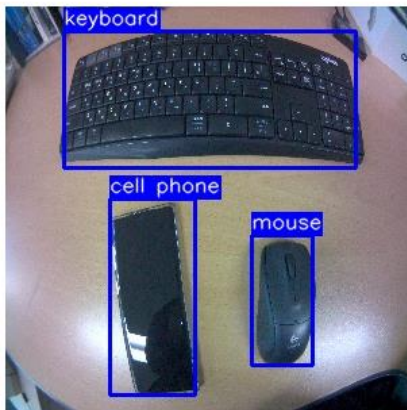
- show() 메소드를 이용하여 인식 결과를 시각적으로 확인

---

12:            OF.show()

---

[출력]



# 목표물 추적

- ▣ detect() 메소드를 사용하여 사람을 인식
- ▣ 사람이 좌측에 있으면 좌회전, 우측에 있으면 우회전 하도록 함
- ▣ size\_rate 값을 이용하여 SerBot제어
  - 화면 대비 크기가 20% 이상이 되면 정지, 그 미만일 경우에만 전진하여 충돌 방지
- ▣ steer값에 4를 곱함
  - 조향 각도를 더 민감하게 하기 위해

# 목표물 추적

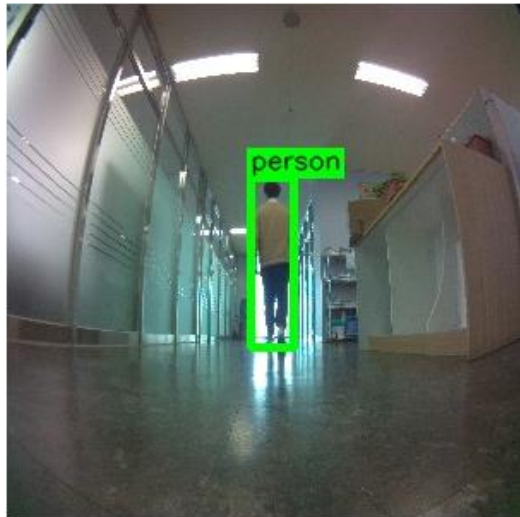
```
13: while True:
14:     v=OF.detect(index='person')
15:
16:     if v is not None:
17:         steer=v['x']*4
18:
19:         if steer > 1:
20:             steer=1
21:         elif steer < -1:
22:             steer=-1
23:
24:         bot.steering=steer
25:
26:         if v['size_rate']<0.20:
27:             bot.forward(50)
28:         else:
29:             bot.stop()
30:     else:
31:         bot.stop()
```



# 목표물 추적

- ▣ 표시된 화면에서 초록색 박스 표시가 된 사람을 따라감
- ▣ 근접하거나 아무것도 인식되지 않으면 정지

[출력]



# 목표물 추적

## □ 전체 코드

---

```
01: from pop import Pilot
02:
03: cam=Pilot.Camera(width=300, height=300)
04: bot=Pilot.SerBot()
05:
06: OF=Pilot.Object_Follow(cam)
07: OF.load_model()
08:
09: while True:
10:     v=OF.detect(index='person')
11:
12:     if v is not None:
13:         steer=v['x']*4
14:
```

---

---

```
15:     if steer > 1:
16:         steer=1
17:     elif steer < -1:
18:         steer=-1
19:
20:     bot.steering=steer
21:
22:     if v['size_rate']<0.20:
23:         bot.forward(50)
24:     else:
25:         bot.stop()
26: else:
27:     bot.stop()
```

---

# 트랙 주행

- 트랙 주행

- ▣ 벽, 차선, 콘 등을 이용해 만든 트랙 주행
- ▣ 차량이 주행할 때 발생하는 여러 상황을 수집하여 학습

# 트랙 주행

## □ 데이터 수집

- ▣ 트랙과 차선 등 주행에 필요한 정보를 학습하기 위해 이미지 데이터를 수집
- ▣ Pop.Pilot라이브러리의 Data\_Collector 클래스
  - 이미지 데이터 수집
- ▣ Pop.Pilot라이브러리의 Camera 클래스
  - 카메라 사용

# 트랙 주행

- ▣ Pop에서 Pilot라이브러리 import
- ▣ Camera 클래스를 width와 height 파라미터에 300 입력
  - 300 x 300 사이즈로 생성
- ▣ Camera 클래스의 show() 메소드를 이용해 실시간 영상 확인 가능

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:         cam.show()
```

---

# 트랙 주행

- ▣ Pilot라이브러리에서 Data\_Collector클래스 생성
- ▣ 생성 파라미터로 Pilot.Track\_Follow 입력
  - Track\_Follow 데이터 수집이 목적임을 명시
- ▣ camera 파라미터에 앞서 생성한 카메라 클래스 입력

---

05:           dataCollector=Pilot.Data\_Collector(Pilot.Track\_Follow, camera=cam)

---

- ▣ Data\_Collector 클래스의 show() 메소드 사용
  - 데이터 수집을 위한 GUI환경 표시

---

06:           dataCollector.show()

---

# 트랙 주행

- ▣ 데이터 수집을 위해 실시간 영상, Auto Collect 버튼, 조이스틱 표시
- ▣ Track Follow 데이터는 조이스틱과 클릭 두 가지 방법으로 데이터 수집 가능
- ▣ 데이터가 수집되면 오른쪽 장면에 결과 표시

# 트랙 주행

## ▣ 첫 번째 방법

- 조이스틱을 이용해 차량을 제어
- 차량 제어 동안 자동으로 영상 데이터와 조이스틱 제어 데이터를 초당 5회 수집
- Auto Collect 버튼을 클릭해 초록색이 되어야 해당 기능 활성화
- 조이스틱으로 제어하는 Auto Collect 기능은 빠르게 대량의 데이터를 수집 가능
- 제어 숙련 정도에 따라 정확한 데이터 수집이 어려움
- 차선에 근접하거나 벗어나는 등 예외적인 상황에 대한 데이터를 수집하기 어려움



# 트랙 주행

## ▣ 두 번째 방법

- 실시간 영상을 클릭하면 해당 장면과 클릭 좌표를 데이터로 수집
- 데이터를 수집할 때는 트랙의 중앙 지점 위주로 앞으로 진행 되어야할 지점을 클릭
- 원하는 장면에 원하는 좌표를 지정 할 수 있으므로 정확한 데이터 수집이 가능
- 데이터 수집이 반자동으로 이루어지기에 수집 속도가 느림
- 이 기능은 Auto Collect 기능이 활성화된 상태에서도 사용 가능
- 이 방법 사용하여 예제 진행

# 트랙 주행

- ▣ 두번째 방법으로 데이터를 수집
- ▣ 데이터는 최소한 500장 이상이 필요
  - 최대한 다양하고 많은 데이터를 수집
- ▣ 사진 데이터는 현재 경로에서 track\_dataset 디렉토리에 저장

# 트랙 주행

## □ 데이터 딥러닝

- 트랙 인식을 위한 합성곱 신경망이 사전 구성된 Track\_Follow 클래스 생성
- 생성 파라미터에 앞서 생성한 camera 클래스를 입력하여 생성
- Track\_Follow 클래스의 load\_datasets() 메소드로 수집한 데이터셋 로드

---

```
07:         TF=Pilot.Track_Follow(camera=cam)
```

```
08:         TF.load_datasets()
```

---

# 트랙 주행

- ▣ 데이터셋 로드가 완료되면 `train()` 메소드를 이용해 학습 시작
- ▣ `train()` 메소드의 파라미터인 `times`에 숫자를 입력해 학습할 횟수 조절
- ▣ `train()` 메소드를 통해 학습이 진행되면 매 스텝마다 자동으로 학습 모델 저장
  - 자동 저장을 비활성화 하고싶다면 `autosave` 파라미터를 `False`로 지정
  - Loss가 0에 가까울수록 오차가 작은 모델

---

```
09:         TF.train(times=5)
```

---

- ▣ `run()` 메소드를 실행하여 반환값(모델의 예측값 ) 출력

---

```
10:         value=TF.run()  
11:         print(value)
```

---

# 트랙 주행

- ▣ show() 메소드를 실행

- 현재 카메라 영상과 run()메소드의 결과를 직관적으로 표시

---

12:	TF.show()
-----	-----------

---

# 트랙 주행

- ▣ run() 메소드 Callback 기능
  - Callback 메소드를 선언
  - run() 메소드의 callback 파라미터로 입력
  - 실행 결과값을 해당 메소드로 넘김

---

```
13:         def is_Left(value):
14:             print(value<0)
15:
16:         value=TF.run(callback=is_Left)
17:         print(value)
```

---

# 트랙 주행

## ▣ Callback 기능을 이용해 차량 제어

- 차량 제어를 위해 Pop.Pilot 라이브러리에서 SerBot 클래스 생성
- Callback 메소드는 차량을 전진시키고 x값을 이용해 좌우 조향하도록 구성

---

```
18:         bot=Pop.SerBot()
19:         bot.setSpeed(50)
20:
21:         def drive(value):
22:             bot.forward()
23:             steer=value['x']
24:
```

---

---

```
25:         if steer > 1:
26:             steer=1
27:         elif steer < -1:
28:             steer=-1
29:
30:         bot.steering=steer*1.5
31:
32:         while True:
33:             TF.run(callback=drive)
```

---

# 트랙 주행

## ▣ 부가적 기능

- load\_model(path) 메소드 : 저장된 학습 모델을 로드하여 이어서 학습

---

```
TF.load_model(path= "track_follow_model.pth")
```

---

- save\_model(path) 메소드 : 원하는 경로에 수동으로 모델 저장

---

```
TF.save_model(path= "track_follow_model.pth")
```

---



# 트랙 주행

## □ 전체 코드 : 데이터 수집

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         dataCollector=Pilot.Data_Collector(Pilot.Track_Follow, camera=cam)
06:         dataCollector.show()
```

---

# 트랙 주행

## □ 전체 코드 : 딥러닝

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         TF=Pilot.Track_Follow(camera=cam)
06:         TF.load_datasets()
07:
08:         TF.train(times=5)
09:
10:         TF.show()
11:
12:         bot=Pilot.SerBot()
13:         bot.setSpeed(50)
14:
```

---

---

```
15:         def drive(value):
16:             bot.forward()
17:             steer=value['x']
18:
19:             if steer > 1:
20:                 steer=1
21:             elif steer < -1:
22:                 steer=-1
23:
24:             bot.steering=steer*1.5
25:
26:         while True:
27:             TF.run(callback=drive)
```

---

# 트래픽 콘 주행

## □ 데이터 수집

- ▣ 주행에 필요한 트래픽 콘 정보를 학습하기 위해 이미지 데이터를 수집
- ▣ 카메라를 생성
- ▣ Pilot라이브러리에서 Data\_Collector클래스를 생성
  - 생성 파라미터로 Pilot.Track\_Follow와 camera 파라미터에 앞서 생성한 카메라 클래스 입력

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:         cam.show()
05:         dataCollector=Pilot.Data_Collector(Pilot.Track_Follow, camera=cam)
```

---

# 트래픽 콘 주행

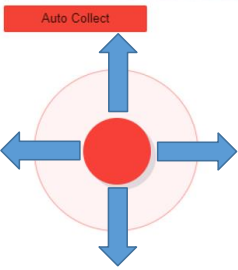
- ▣ Data\_Collector 클래스의 show() 메소드를 사용해 GUI환경 표시

---

06:            dataCollector.show()

---

[출력]



# 트래픽 콘 주행

- ▣ 데이터 수집을 위해 실시간 영상과 Auto Collect 버튼, 조이스틱이 표시
  - 이미지를 클릭하여 데이터셋을 수집하는 방법 사용
- ▣ Auto Collect 기능이 비활성화된 상태에서 데이터를 수집
- ▣ 데이터는 최소한 500장 이상이 필요하며 다양한 상황 필요.
- ▣ 사진 데이터는 현재 경로에서 track\_dataset 디렉토리에 저장

# 트래픽 콘 주행

## □ 데이터 딥러닝

- ▣ 트랙 인식을 위한 합성곱 신경망이 사전 구성된 Track\_Follow 클래스 생성
- ▣ 생성 파라미터에 앞서 생성한 camera 클래스를 입력하여 생성
- ▣ 앞서 수집한 데이터셋들을 로드
  - Track\_Follow 클래스의 load\_datasets() 메소드를 이용
  - 데이터셋을 로드할 때 현재 경로에 있는 track\_dataset 디렉토리에서 가져옴

---

07: CF=Pilot.Track\_Follow(camera=cam)

08: CF.load\_datasets()

---

# 트래픽 콘 주행

- ▣ 데이터셋 로드가 완료되면 train() 메소드를 이용해 학습 시작

---

```
09:         CF.train(times=5)
```

---

- ▣ run() 메소드를 실행하고 반환값(모델의 예측값)을 출력

---

```
10:         value=CF.run()  
11:         print(value)
```

---

- ▣ run() 메소드와 동기화 된 show() 메소드를 호출

---

```
12:         CF.show()
```

---

# 트래픽 콘 주행

- ▣ SerBot 클래스를 생성하고 Callback 메소드를 작성
  - Callback 메소드는 차량을 전진시키고 x값을 이용해 좌우 조향하도록 구성
- ▣ 모델의 run() 메소드에 callback 파라미터로 입력하여 실행

---

```
13:         bot=Pilot.SerBot()
14:         bot.setSpeed(50)
15:
16:         def drive(value):
17:             bot.forward()
18:             steer=value['x']
19:
20:             if steer > 1:
```

---

---

```
21:                 steer=1
22:                 elif steer < -1:
23:                     steer=-1
24:
25:                 bot.steering=steer*1.5
26:
27:                 while True:
28:                     CF.run(callback=drive)
```

---



# 트래픽 콘 주행

## □ 전체 코드 : 데이터 수집

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         dataCollector=Pilot.Data_Collector(Pilot.Track_Follow, camera=cam)
06:         dataCollector.show()
```

---

# 트래픽 콘 주행

## □ 전체 코드 : 딥러닝

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         CF=Pilot.Track_Follow(camera=cam)
06:         CF.load_datasets()
07:
08:         CF.train(times=5)
09:
10:         CF.show()
11:
12:         bot=Pilot.SerBot()
13:         bot.setSpeed(50)
14:
```

---

---

```
15:         def drive(value):
16:             bot.forward()
17:             steer=value['x']
18:
19:             if steer > 1:
20:                 steer=1
21:             elif steer < -1:
22:                 steer=-1
23:
24:             bot.steering=steer*1.5
25:
26:         while True:
27:             CF.run(callback=drive)
```

---

# 라인 트레이서

## □ 데이터 수집

- ▣ 주행에 필요한 라인 정보를 학습하기 위해 이미지 데이터 수집

- ▣ 카메라를 생성

- ▣ Pilot라이브러리에서 Data\_Collector클래스를 생성

- 생성 파라미터로 Pilot.Track\_Follow와 camera 파라미터에 앞서 생성한 카메라 클래스 입력

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:         cam.show()
05:         dataCollector=Pilot.Data_Collector(Pilot.Track_Follow, camera=cam)
```

---

# 라인 트레이서

- ▣ Data\_Collector 클래스의 show() 메소드를 사용해 GUI환경 표시

---

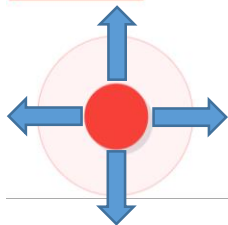
06:            dataCollector.show()

---

[출력]



Auto Collect



# 라인 트레이서

- ▣ 데이터 수집을 위해 실시간 영상과 Auto Collect 버튼, 조이스틱이 표시
  - 이미지를 클릭하여 데이터셋을 수집하는 방법 사용
- ▣ Auto Collect 기능이 비활성화된 데이터를 수집
- ▣ 데이터는 최소한 500장 이상이 필요. 다양한 상황의 데이터 필요
- ▣ 사진 데이터는 현재 경로에서 track\_dataset 디렉토리에 저장

# 라인 트레이서

## □ 데이터 딥러닝

- 라인 인식을 위한 합성곱 신경망이 사전 구성된 Track\_Follow 클래스 생성
- 생성 파라미터에 앞서 생성한 camera 클래스를 입력하여 생성
- 앞서 수집한 데이터셋들을 로드
  - Track\_Follow 클래스의 load\_datasets() 메소드를 이용
  - 데이터셋을 로드할 때 현재 경로에 있는 track\_dataset 디렉토리에서 가져옴

---

```
07:         LF=Pilot.Track_Follow(camera=cam)
```

```
08:         LF.load_datasets()
```

---

# 라인 트레이서

- ▣ 데이터셋 로드가 완료되면 train() 메소드를 이용해 학습 시작
  - train() 메소드를 통해 학습이 진행되면 매 스텝마다 자동으로 학습 모델 저장

---

```
09:         LF.train(times=5)
```

---

- ▣ run() 메소드를 실행하고 반환값(모델의 예측값) 출력

---

```
10:         value=LF.run()  
11:         print(value)
```

---

- ▣ run() 메소드와 동기화 된 show() 메소드를 호출

---

```
12:         LF.show()
```

---

# 라인 트레이서

- ▣ SerBot 클래스를 생성하고 Callback 메소드를 작성
  - Callback 메소드는 차량을 전진시키고 x값을 이용해 좌우 조향하도록 구성
- ▣ 모델의 run() 메소드에 callback 파라미터로 입력하여 실행

---

```
13:         bot=Pilot.SerBot()
14:         bot.setSpeed(50)
15:
16:         def drive(value):
17:             bot.forward()
18:             steer=value['x']
19:
```

---

---

```
20:         if steer > 1:
21:             steer=1
22:         elif steer < -1:
23:             steer=-1
24:
25:         bot.steering=steer*1.5
26:
27:         while True:
28:             LF.run(callback=drive)
```

---



# 라인 트레이서

## □ 전체 코드 : 데이터 수집

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         dataCollector=Pilot.Data_Collector(Pilot.Track_Follow, camera=cam)
06:         dataCollector.show()
```

---

# 라인 트레이서

## □ 전체 코드 : 딥러닝

---

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300, height=300)
04:
05:         LF=Pilot.Track_Follow(camera=cam)
06:         LF.load_datasets()
07:
08:         LF.train(times=5)
09:
10:         LF.show()
11:
12:         bot=Pilot.SerBot()
13:         bot.setSpeed(50)
14:
```

---

---

```
15:         def drive(value):
16:             bot.forward()
17:             steer=value['x']
18:
19:             if steer > 1:
20:                 steer=1
21:             elif steer < -1:
22:                 steer=-1
23:
24:             bot.steering=steer*1.5
25:
26:         while True:
27:             LF.run(callback=drive)
```

---