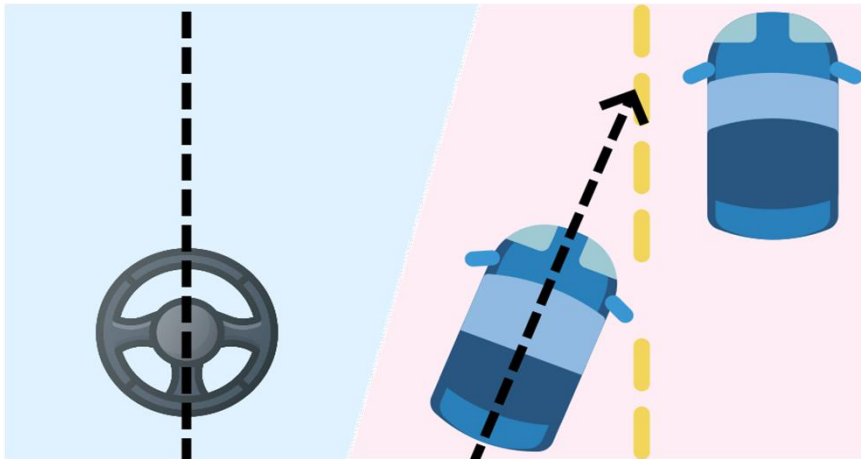


AIoT AutoCar Prime 으로 배우는 온디바이스 AI 프로그래밍

8 온디바이스 AI

휠 얼라인먼트

- 6축 센서 응용, 조향 모터의 오차 보정 인공지능 예제
 - ▣ 실제 차량의 휠 얼라인먼트를 보정하는 과정과 유사



월 얼라인먼트

- ▣ 학습을 위해 데이터를 수집

- 파이썬 조향 제어값과 6축 센서의 값을 데이터셋으로 구성
- 데이터셋은 다양한 각도로 조향해보고 이에 따른 6축 센서 값을 수집

- ▣ 차량 제어를 위한 Pop.Pilot 라이브러리와 Numpy, Time 라이브러리 import

- ▣ AutoCar 객체 생성

```
01:         from pop import Pilot
02:         import numpy as np
03:         import time
04:
05:         Car = Pilot.AutoCar()
```

월 얼라인먼트

- ▣ 여러 각도로 조향하고, 6축 센서의 짜이로 값 중 Z축 값을 읽어 저장
 - 조향 제어값은 좌회전을 음수(-), 우회전을 양수(+)로 저장
 - 센서 측정 시간을 고려해 조향 후 0.5초가 지난 시점에 6축 센서 값 읽음
- ▣ 다음 데이터 수집을 위해 전진한 만큼 후진하여 제자리로 복귀
- ▣ 이 과정을 여러 번 반복하여 데이터셋을 많이 수집할수록 정확한 오차 보정 가능

월 일라인먼트

06:	dataset={ 'gyro' : [], 'steer' : [] }	18:	time.sleep(0.5)
07:		19:	
08:	for n in np.arange(-1, 1.1, 0.2):	20:	Car.backward()
09:	n = round(n,1)	21:	
10:		22:	time.sleep(1)
11:	Car.steering = n	23:	
12:	Car.forward()	24:	Car.stop()
13:		25:	
14:	time.sleep(0.5)	26:	dataset['gyro'].append(m)
15:		27:	dataset['steer'].append(n)
16:	m = Car.getGyro('z')	28:	
17:	print({ 'gyro' : m , 'steer' : n })	29:	print({ 'gyro' : m , 'steer' : n })

월 일라인먼트

- ▣ 선형 회귀를 위해 Pop.AI 라이브러리 import
- ▣ Linear_Regression 객체 생성.
 - X_data와 Y_data에 각각 gyro와 steer를 지정
- ▣ train 메소드의 파라미터에 times를 5000으로 설정. 데이터 학습
 - 충분한 학습을 위해 times 설정

```
30:         from pop import AI
31:
32:         LR=AI.Linear_Regression(input_size=1, output_size=1)
33:         LR.X_data=dataset['gyro']
34:         LR.Y_data=dataset['steer']
35:
36:         LR.train(times=5000, print_every=100)
```

월 얼라인먼트

- 학습된 모델의 run 메소드에 0을 입력하여 실행

- Z축 쪽으로 값이 0에 가까울 때의 조향 값이 정중앙

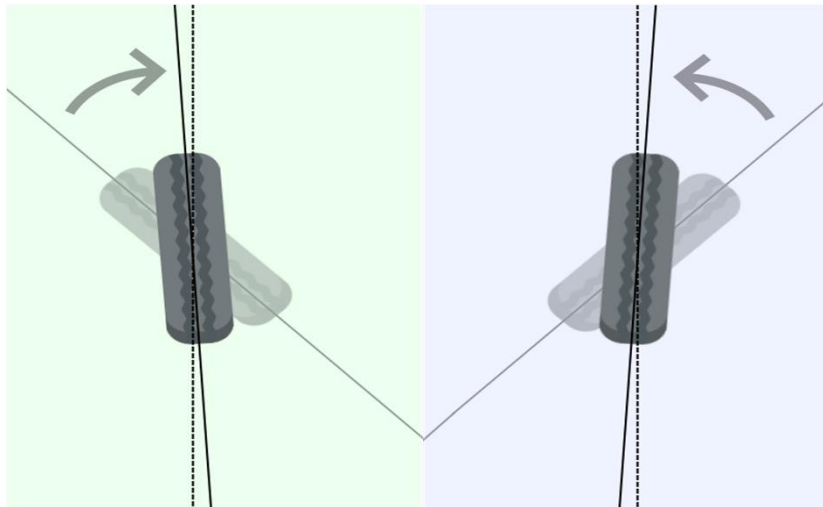
```
37:         value = LR.run([0])
38:         print(value)
```

- 정중앙으로 주행하기 위한 조향 제어값을 알 수 있고, 이 값을 입력하여 조향 보정

```
39:         Car.steering = value[0]
40:         Car.forward()
```

월 얼라인먼트

- ▣ 차량이 비교적 정확하게 직진하는 것을 확인 가능
- ▣ 단, 베어링으로 인한 오차가 발생하여 완벽한 보정은 불가능



월 얼라인먼트

□ 전체 코드

01: from pop import Pilot	16:	
02: import numpy as np	17:	m = Car.getGyro('z')
03: import time	18:	
04:	19:	time.sleep(0.5)
05: Car = Pilot.AutoCar()	20:	
06:	21:	Car.backward()
07: dataset={ 'gyro' : [], 'steer' : [] }	22:	
08:	23:	time.sleep(1)
09: for n in np.arange(-1, 1.1, 0.2):	24:	
10: n = round(n,1)	25:	Car.stop()
11:	26:	
12: Car.steering = n	27:	dataset['gyro'].append(m)
13: Car.forward()	28:	dataset['steer'].append(n)
14:	29:	
15: time.sleep(0.5)	30:	print({ 'gyro' : m , 'steer' : n })

월 얼라인먼트

□ 전체 코드

```
31:
32: from pop import AI
33:
34: LR=AI.Linear_Regression(input_size=1, output_size=1)
35: LR.X_data=dataset['gyro']
36: LR.Y_data=dataset['steer']
37:
38: LR.train(times=5000, print_every=100)
39:
40: Car.steering = value[0]
41: Car.forward()
```

차량 자세 제어

- 6축 센서 응용, 스스로 자세 제어하여 주행하는 인공지능 예제
 - ▣ 6축 센서에서 좌회전이 감지되면 차량을 우회전으로 제어하여 자세를 유지
 - ▣ 학습 데이터는 직전 예제와 비슷한 방법으로 수집
 - 단, 직전 예제를 통해 조향 모터의 오차를 보정한 상황에서 진행
 - ▣ 차량 제어를 위한 Pop.Pilot 라이브러리와 Numpy, Time 라이브러리 import
 - ▣ AutoCar 객체 생성
 - ▣ correctError 메소드를 이용해 직전 예제에서 얻은 오차 보정값 적용

차량 자세 제어



```
01:      from pop import Pilot
02:      import numpy as np
03:      import time
04:
05:      Car = Pilot.AutoCar()
06:      Car.correctError( -0.1206646 )
```

차량 자세 제어

- ▣ 여러 각도로 조향하고, 6축 센서의 짜이로 값 중 Z축 값을 읽어 저장
- ▣ 조향 제어값을 중앙 값(0)을 제외하고 저장
- ▣ 조향 후 0.5초가 지난 시점에 6축 센서의 값을 읽어옴
 - 센서 측정 시간 고려
- ▣ 다음 데이터 수집을 위해 전진한 만큼 후진하여 제자리로 복귀
- ▣ 조향 제어값을 저장할 때 좌회전을 양수(+), 우회전을 음수(-)로 저장
 - 6축 센서에서 우회전이 감지되면 차량을 좌회전시켜 자세를 유지하기 위해
- ▣ 이 과정을 여러 번 반복하여 데이터셋을 많이 수집할수록 정확한 오차 보정 가능

차량 자세 제어

```
07:         dataset={ 'gyro' : [], 'steer' : [] }
08:
09:         for n in np.arange(-1, 1.1, 0.2):
10:             n = round(n,1)
11:
12:             if n == 0.0 :
13:                 continue
14:
15:             Car.steering = n
16:             Car.forward()
17:
18:             time.sleep(0.5)
19:
20:             m = Car.getGyro('z')
```

```
21:
22:             time.sleep(0.5)
23:
24:             Car.backward()
25:
26:             time.sleep(1)
27:
28:             Car.stop()
29:
30:             n = -n #조향 제어가음 반전하여 저장
31:
32:             dataset['gyro'].append( m )
33:             dataset['steer'].append( n )
34:
35:             print({ 'gyro' : m , 'steer' : n })
```

차량 자세 제어

- ▣ Linear_Regression 학습을 위해 Pop.AI 라이브러리 import
- ▣ Linear_Regression 객체 생성
- ▣ Linear_Regression의 X_data와 Y_data에 각각 gyro와 steer를 지정
- ▣ 충분한 학습을 위해 train 메소드의 파라미터에 times를 5000으로 설정
- ▣ 데이터 학습

```
36:         from pop import AI
37:
38:         LR=AI.Linear_Regression(input_size=1, output_size=1)
39:         LR.X_data=dataset['gyro']
40:         LR.Y_data=dataset['steer']
41:
42:         LR.train(times=5000, print_every=100)
```

차량 자세 제어

- Z축 짜이로 값을 입력으로 선형 외귀 모델을 실행
- 반환 값을 steering에 입력하여 차량 제어
- 원활한 제어 값 송수신을 위해 0.1초 간격으로 반복 실행
 - 0.1초 미만으로 할 경우 반응성이 좋아지지만 모터 제어를 위한 통신이 불안정해질 수 있음
- 자세 복구를 위해 조향 값을 줄이지 않고 그대로 제어할 시
 - 학습 모델은 다시 그 반대 방향으로 복구해야 한다고 판단해 좌우로 크게 왔다갔다 함
 - 이러한 현상을 방지하기 위해 모델의 반환 값을 3으로 나눔

차량 자세 제어

```
43:         Car.forward()
44:
45:         while True:
46:             err = Car.getGyro('z')
47:             value = LR.run([err])
48:
49:             Car.steering = value / 3
50:             time.sleep(0.1)
```

차량 자세 제어

□ 전체 코드

```
01:         from pop import Pilot
02:         import numpy as np
03:         import time
04:
05:         Car = Pilot.AutoCar()
06:         Car.correctError( -0.1206646 ) #보정값 입력
07:
08:         dataset={ 'gyro' : [], 'steer' : [] }
09:
10:         for n in np.arange(-1, 1.1, 0.2):
11:             n = round(n,1)
12:
13:             if n == 0.0 :
14:                 continue
15:
16:             Car.steering = n
```

```
17:         Car.forward()
18:
19:         time.sleep(0.5)
20:
21:         m = Car.getGyro('z')
22:
23:         time.sleep(0.5)
24:
25:         Car.backward()
26:
27:         time.sleep(1)
28:
29:         Car.stop()
30:
31:         n = -n #조향 제어값을 반전하여 저장
32:
```

차량 자세 제어

```
33:         dataset['gyro'].append( m )
34:         dataset['steer'].append( n )
35:
36:         print({ 'gyro' : m , 'steer' : n })
```

```
37:
38:     from pop import AI
39:
40:     LR=AI.Linear_Regression(input_size=1, output_size=1)
41:     LR.X_data=dataset['gyro']
42:     LR.Y_data=dataset['steer']
43:
44:     LR.train(times=5000, print_every=100)
45:
46:     Car.forward()
47:
48:     while True:
49:         err = Car.getGyro('z')
```

```
50:         value = LR.run([err])
51:
52:         Car.steering = value / 3
53:         time.sleep(0.1)
```

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

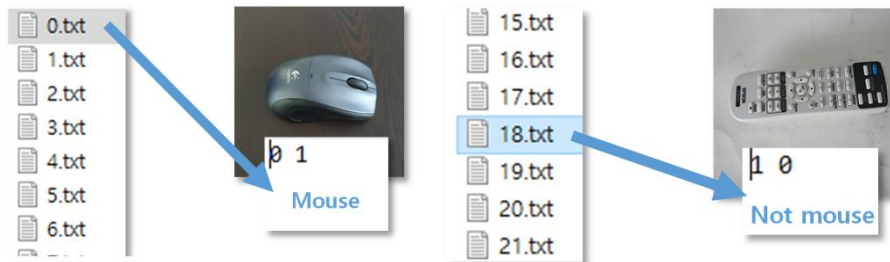


사진 분류

- ▣ 사용해야 할 패키지 import, 입력 데이터와 결과 데이터를 리스트로 선언
- ▣ for 문을 사진 개수만큼 반복하는 루프 생성
- ▣ cv2.imread메소드로 사진을 읽은 후 cv2.cvtColor메소드로 흑백사진으로 변환
- ▣ cv2.resize메소드로 이미지 사이즈를 50x50로 하고 X_data 리스트에 추가
- ▣ 라벨링 했던 파일을 numpy의 loadtxt메소드로 읽어 Y_data 리스트에 추가
- ▣ 이 과정은 사진 개수에 따라 수 분이 걸릴 수 있음

사진 분류

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

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

사진 분류

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

```
21:         CNN.run([CNN.X_data[20]])  
22:         CNN.run([CNN.X_data[7]])
```

사진 분류

- ▣ 카메라의 이미지를 읽어와 인식
- ▣ Pilot의 Camera 객체 생성
- ▣ Camera 객체에서 카메라의 장면을 읽어오고 cvtColor메소드로 흑백으로 변환
- ▣ resize메소드로 50x50사이즈로 축소
- ▣ 카메라 이미지를 run메소드에 입력하고 결과 확인

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

사진 분류

□ 전체 코드

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

```
15:
16: CNN=AI.CNN(input_size=[50,50], output_size=2)
17: CNN.X_data=X_data
18: CNN.Y_data=Y_data
19:
20: CNN.train(times=500)
21:
22: cam = Pilot.Camera()
23:
24: frame = cam.value
25: frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
26: frame = cv2.resize(frame, (50, 50), interpolation=cv2.INTER_AREA)
27: 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
 - 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' bar (orange) labeled 'Ready to record.', a 'Duration (sec)' input field set to '1', a red 'REC' button, and a playback slider showing '0:00 / 0:00' with a play button and a volume icon. The right section contains configuration fields: '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 right 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 과정 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 최대한 길이를 맞춰 녹음
- ▣ 이 과정은 데이터셋 개수에 따라 수 분이 걸릴 수 있음

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

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

음성 분류

- CNN 객체의 파라미터 설정 후 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:
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)
24:
25: dataset_size = X_data.shape[1:3]
26: CNN=AI.CNN(input_size=dataset_size, output_size=10)
27:
28: CNN.X_data=X_data
29: CNN.Y_data=Y_data
30:
```

음성 분류

```
31: CNN.train(times=100)
32:
33: Y=CNN.Y_data[20]
34: R=CNN.run([CNN.X_data[20]])
35:
36: print(Y)
37: print(R)
38:
39: Dataset.Collector("Audio")
40:
41: data=Util.toMFCC("audio_datasets/9_identifier_200413205
3280.wav",
duration=1)
42: CNN.run([data])
```

장애물 외피

- 장애물 외피

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

장애물 외피

- 데이터 수집

- ▣ Pop.Pilot라이브러리의 Data_Collector 클래스

- 이미지 데이터 수집

- ▣ Pop.Pilot라이브러리의 Camera 클래스

- 카메라 사용

장애물 외피

- ▣ 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 외피

- stop() 메소드 : 영상 정지

- 실제 촬영은 백그라운드에서 지속되고 있고 보여지는 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)
```

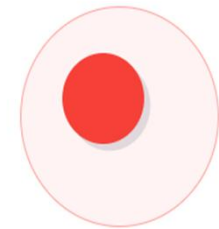
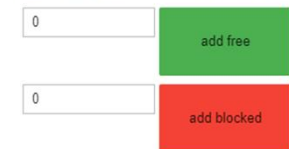
장애물 외피

▣ Data_Collector 클래스의 show() 메소드

■ 데이터 수집을 위한 GUI환경 표시

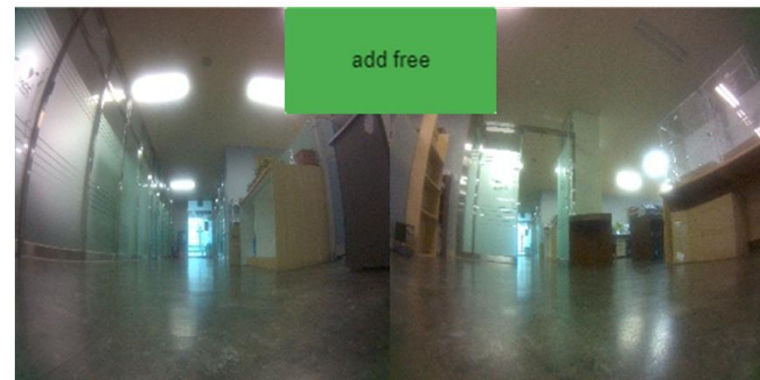
07: dataCollector.show()

- 데이터 수집을 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 외피

- 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 외피

□ 데이터 담러닝

▣ 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 외피

- ▣ 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 `False`로 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 외피

- ▣ 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 외피

- ▣ 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 AutoCar 클래스 생성
 - Callback 기능을 이용해 차량 제어
- ▣ Callback 메소드 선언
 - 모델 예측값이 0.5를 초과하면 우측으로 우진
 - 0.5 이하면 전진

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

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

장애물 외피

▣ 추가적 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 외피

□ 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 : 덩러닝

01:	from pop import Pilot	13:	Car.setSpeed(50)
02:		14:	
03:	cam=Pilot.Camera(width=300, height=300)	15:	def drive(value):
04:		16:	if value<=0.5:
05:	CA=Pilot.Collision_Avoid(cam)	17:	Car.steering=0
06:	CA.load_datasets()	18:	Car.forward()
07:		19:	else:
08:	CA.train(times=10)	20:	Car.steering=1
09:		21:	Car.backward()
10:	CA.show()	22:	
11:		23:	while True:
12:	Car=Pilot.AutoCar()	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

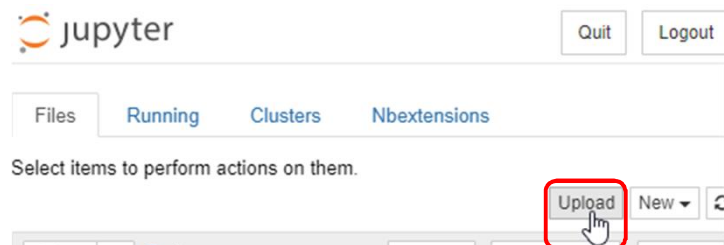
▣ `ssd_mobilenet_v2_coco.engine` 파일 다운로드

■ 인식 가능한 사물 목록 확인 가능

■ https://github.com/hanback-docs/ssd_mobilenet_v2_coco_engine



▣ 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 가져옴

```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300,height=300)
04:
05:         OF=Pilot.Object_Follow(cam)
06:         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



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

```
07:      v=OF.detect(index='person')
08:      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 비율
- 인식된 개체의 크기가 클수록 거리가 가깝다는 의미로 해석 가능

목표물 추적

▣ index 입력없이 모든 사물에 대한 데이터 확인

```
09:      v=OF.detect()  
10:      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]}]]
```

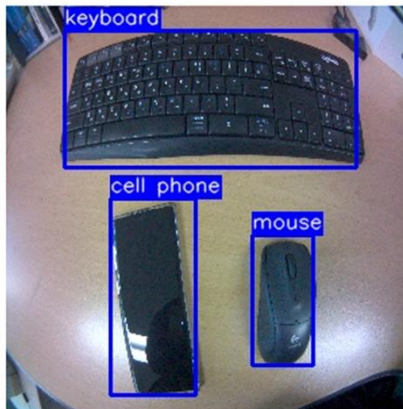
- index가 76인 ‘Keyboard’ 인식
- index가 74인 ‘mouse’ 인식
- index가 77인 ‘cell phone’ 인식

목표물 추적

- show() 메소드를 이용하여 인식 결과를 시각적으로 확인

11: OF.show()

[출력]



목표물 추적

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

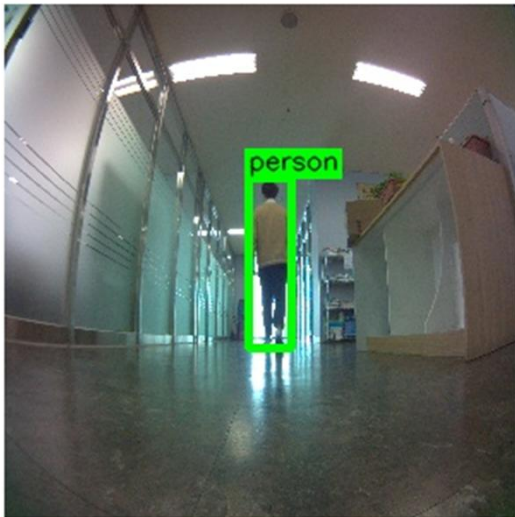
목표물 추적

```
12:         while True:
13:             v=OF.detect(index='person')
14:
15:             if v is not None:
16:                 steer=v['x']*4
17:
18:                 if steer > 1:
19:                     steer=1
20:                 elif steer < -1:
21:                     steer=-1
22:
23:                 ac.steering=steer
24:
25:                 if v['size_rate']<0.20:
26:                     ac.forward(50)
27:                 else:
28:                     ac.stop()
29:             else:
30:                 ac.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:
05:         OF=Pilot.Object_Follow(cam)
06:         OF.load_model()
07:
08:         while True:
09:             v=OF.detect(index='person')
10:
11:             if v is not None:
12:                 steer=v['x']*4
13:
```

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

- 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 times에 숫자를 입력해 학습할 횟수 조절
- ▣ train 메소드를 통해 학습이 진행되면 매 스텝마다 자동으로 학습 모델 저장
 - 자동 저장을 비활성화 하고싶다면 autosave 파라미터를 False로 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 AutoCar 클래스 생성
- Callback 메소드는 차량을 전진시키고 x값을 이용해 좌우 조향하도록 구성

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

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

트랙 주행

▣ 추가적 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 : 덤러닝

```
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:         Car=Pilot.AutoCar()
13:         Car.setSpeed(50)
14:
```

```
15:         def drive(value):
16:             Car.forward()
17:             steer=value['x']
18:
19:             if steer > 1:
20:                 steer=1
21:             elif steer < -1:
22:                 steer=-1
23:
24:             Car.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

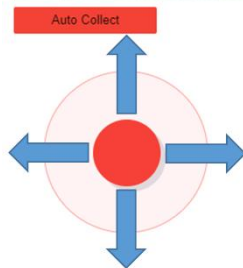
```
01:         from pop import Pilot
02:
03:         cam=Pilot.Camera(width=300,height=300)
04:         cam.show()
05:         dataCollector=Pilot.Data_Collector(Pilot.Track_Follow,camera=cam)
```

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 덤러닝

- 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 이용해
 - 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

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

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

```
21:             steer=1
22:             elif steer < -1:
23:                 steer=-1
24:
25:             Car.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 : 덤러닝

```
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:         Car=Pilot.AutoCar()
13:         Car.setSpeed(50)
14:
```

```
15:         def drive(value):
16:             Car.forward()
17:             steer=value['x']
18:
19:             if steer > 1:
20:                 steer=1
21:             elif steer < -1:
22:                 steer=-1
23:
24:             Car.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()
```

- 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 덤퍼

- 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 이용해
 - 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

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

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

```
20:             if steer > 1:
21:                 steer=1
22:             elif steer < -1:
23:                 steer=-1
24:
25:             Car.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 : 덩러닝

```
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:         Car=Pilot.AutoCar()
13:         Car.setSpeed(50)
14:
```

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