1.1.1.1. 조이스틱으로 이동체 이동

트랙 데이터를 수집하기 전에 주피터에서 새로 “joystick_driving.ipynb”를 만듭니다.

```
01: from pop import Pilot
02: joy = Pilot.joystick()
```

Pilot 모듈을 로드한 후 joystick 객체가 만들어지면 show() 메소드를 호출합니다.

```
03: joy.show()
```

큰 원 안에 작은 원을 마우스로 특정 방향으로 끌면 속도와 방향이 실습장치에 적용되어 해당 방향으로 실습장치가 이동합니다.

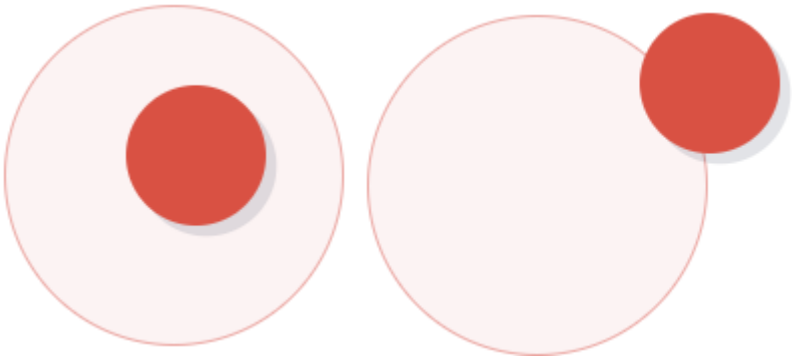

그림 1 조이스틱으로 실습장치 이동

1.1.1.2. 트랙 데이터 수집

트랙 데이터를 수집하기 위해 주피터에서 새로 “resnet_dataset.ipynb”를 만듭니다. 일부 Pop 라이브러리 버전은 pytorch.org 에서 resnet18 모델을 다운받기 위해 인터넷 연결을 요구하므로 실습장비를 인터넷에 연결한 후 실습을 진행합니다.

트랙 데이터 수집을 위해 필요한 라이브러리를 불러오고 카메라 객체를 생성합니다. 카메라 객체를 생성할 때 이미지의 크기를 지정해주는데, 저장되는 이미지의 용량, 해당 이미지를 사용하여 모델을 학습하고 실행시킬 때의 연산 속도 등을 고려하여 크기를 설정해야 합니다. 여기서는 300x300 사이즈를 사용합니다.

```
04: from pop import Pilot, Camera
05: cam = Camera(300,300)
```

Pilot 라이브러리는 이미지 형태의 주행용 데이터 수집을 지원하기 위해 Data_Collector 클래스를 제공하며 show() 메소드를 호출하면 GUI 환경이 표시됩니다.

표시되는 위젯의 상단에는 2 개의 이미지, 하단에는 조이스틱이 나타납니다. 위젯에 나타나는 2 개의 이미지 중 왼쪽 이미지는 현재 카메라에서 읽어오는 실시간 이미지(프리뷰)이며, 오른쪽은 SerBot 에 저장되는 카메라 이미지입니다.

위젯의 하단에 나타나는 조이스틱을 활용하여 이동체를 움직이면서 데이터를 수집할 수 있습니다. Auto Collect 버튼을 눌러 초록색으로 활성화되면 데이터를 자동으로 수집하기 시작하지만, 숙련된 조작이 필요해 수동 모드를 권장합니다.

그림 2 자동 수집 활성화와 비활성화

```
06: dc = Pilot.Data_Collector("Track_Follow", cam)
07: dc.show()
```

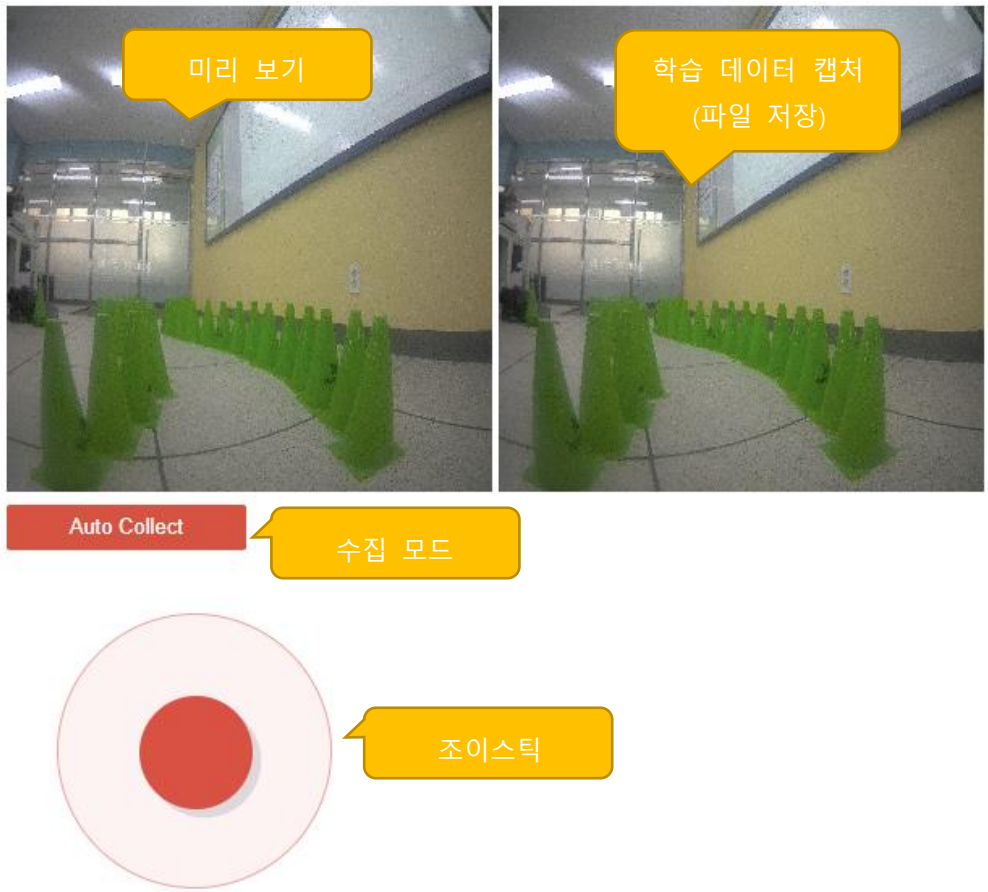

그림 3 주피터 환경에서 차선 인식 데이터 수집

마우스로 조이스틱 영역 가운데 원을 해당 방향으로 끌면 이동체가 움직입니다. 실습장비와 PC 는 공유기를 통해 Wi-Fi 로 연결되며, 공유기의 성능에 따라 영상의 지연이 발생할 수 있습니다. 원활한 실습을 위해서는 고성능 공유기 필요합니다.

4m x 3m 크기의 곡선과 직선이 혼합된 트랙 완주 기준으로 차선 이미지 데이터는 500 장 이상을 권장합니다. 수집한 데이터로 모델을 학습할 때 소요되는 시간은 500 장 기준 약 10 분 정도입니다. 정상적인 주행을 비롯해 좌우측 차선 치우침, 과도한 코너링 등 최대한 다양하고 많은 데이터 수집이 필요합니다.

수집한 차선 이미지는 실습장비에 학습 데이터로 저장되는데, 라벨(정답)에 해당하는 파일 이름은 이미지의 왼쪽 상단(0, 0) 기준으로 x_y_<날짜 시간>.jpg 형식으로, x_y_가 실습장치의 주행 방향을 나태는 목표점(x, y)입니다.

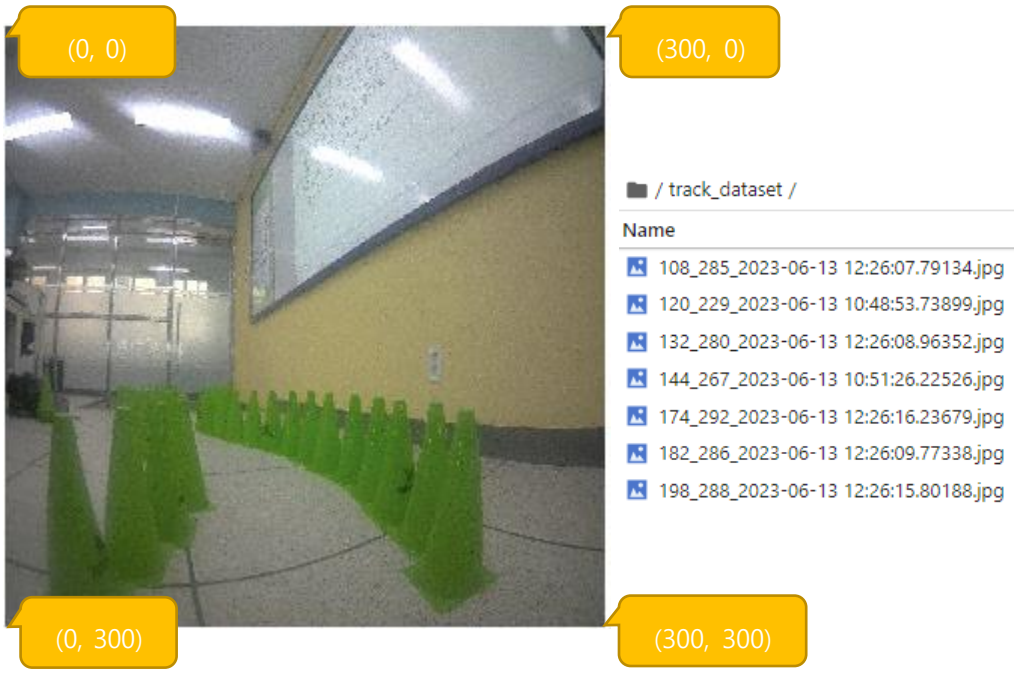

그림 4 데이터셋 샘플(좌)와 수집한 데이터 셋(우)

'Auto Collect' 버튼을 한 번 눌러 빨간색에서 녹색으로 바뀌면 5 초 단위로 미리보기를 캡쳐해 자동 저장합니다. 이때 파일 이름에 포함된 목표점은 조이스틱 상태를 이미지 상의 좌표로 변환한 값입니다. 'Auto Collect' 가능은 빠르게 대량의 데이터를 수집할 수 있지만 숙련도에 따라 정확한 데이터 수집이 힘들 수 있습니다.

그에 반해 수동 수집은 미리보기를 클릭할 때마다 해당 장면과 클릭한 상대 좌표

를 목표점으로 저장하는데, 실습장치의 이동은 조이스틱을 사용하거나 사용자가 직접 옮길 수 있습니다. 대부분은 조이스틱으로 원하는 위치까지 이동한 후 목표점을 찍지만, 예외 상황에 따라 직접 이동하기도 합니다.

수동으로 데이터를 수집할 때 목표점을 찍는 기준은 다음과 같습니다.

- x 값 = -이미지 중심 + (차선 중심 * 2)
- y 값 = 이미지의 아래 절반 중 1/2 위치

다음 이미지를 보면 수직으로 아래 절반에 트랙이 위치하므로 이 부분을 기준으로 y 값을 결정하는데, 실습장치가 이동한다는 점을 고려해 가장 아래 20 픽셀 정도를 제외하면 트랙 수직 범위도 130 ~ 280 픽셀로 약간 올라가게 됩니다. 따라서 이 구간을 1/2 로 나눈 205(130 + (280-130)/2)픽셀이 y 값이 됩니다. 하지만 실제 점을 찍을 때는 대략 205 픽셀에 가까우면 됩니다.

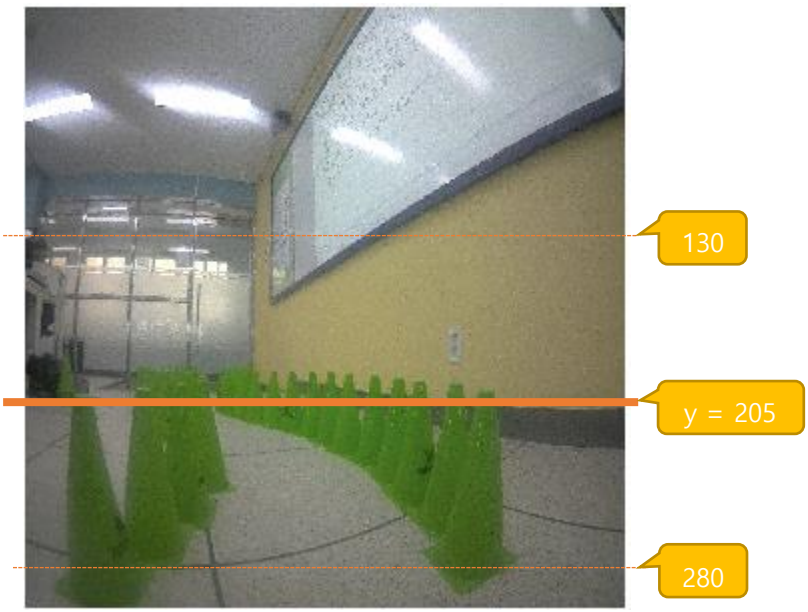

그림 5 목표점 중 y 값 기준

y 값은 하나의 기준으로 고정해도 되지만 x 값은 실습장비가 트랙을 벗어나지 않고 주행하는데 필요한 스티어링 값에 해당하므로 이미지의 중심에서 차선의 중심이 벗어난 정도와 방향에 따라 바뀝니다.

아래 그림에서 y 값을 기준으로 x 값을 찾을 때, 차선 중심은 이미지 중심에서 약간 왼쪽에 있습니다. 이미지 중심이 150 이고, 차선 중심이 135 라면 x 값은 -150 + (135*2) = 120 입니다. 하지만 x 값도 정확할 필요는 없으며, 차선 중심이 이미지 중심 왼쪽에 있으면 왼쪽 차선에서 좀더 왼쪽, 오른쪽에서 있으면 오른쪽 차선에서 좀더 오른쪽에 찍으면 됩니다.

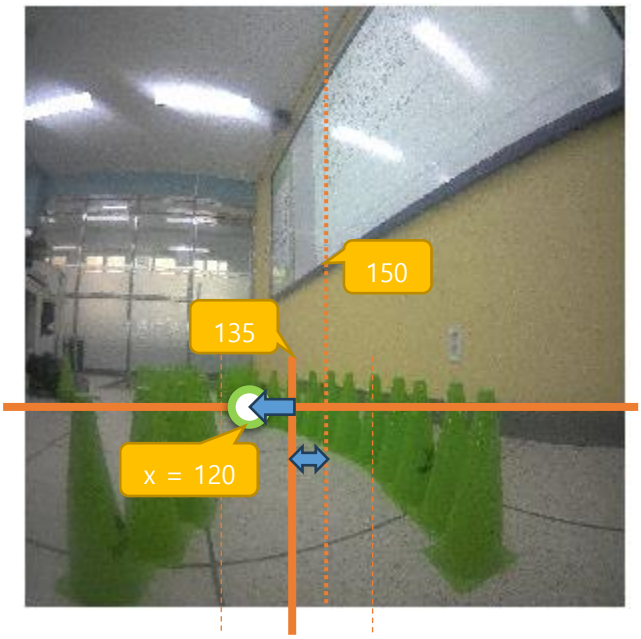

그림 6 목표점 중 x 값 기준

데이터를 수집할 때 앞서 소개한 기준으로 목표점을 정하면서 최대한 다양한 상황의 데이터를 많이 수집하는 것이 중요한데, 이렇게 수집한 데이터셋은 ~/Project/python/notebook/track_dataset 에 위치합니다.

데이터 수집이 끝나면 백그라우드에서 실행 중인 카메라를 중지시키기 위해 **"ShutDown Kernel"을 클릭해 파이썬 커널을 종료**합니다.

1.1.1.3. 트랙 주행 모델 생성

수집한 데이터셋으로 모델을 학습하고 저장합니다. 이번 단계부터는 jupyter 환경이 아닌, vscode 에서 진행합니다.

vscode 의 작업폴더가 ~/Project/python 이라면 이곳에 주피터에서 수집한 데이터셋을 복사합니다. 두 번째 cp 명령 끝의 점(.)은 현재 폴더를 나타냅니다.

```
cd ~/Project/python
cp -R notebook/track_dataset .
```

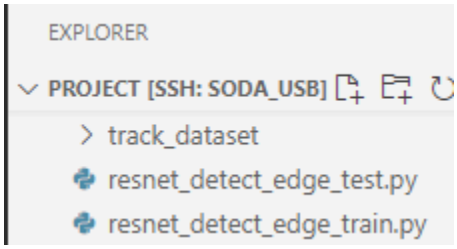

그림 7 dataset 복사

수집한 데이터셋으로 모델 학습 및 저장할 때 사용되는 메소드들은 다음과 같습니다.

- Track_Follow(camera): 차선 데이터셋 학습 및 트랙 주행 예측 객체 생성
- Track_Follow.load_dataset(path='./track_dataset'): 수집한 데이터셋 로드
  - path: 데이터 셋이 저장된 경로. 기본값은 현재 경로의 track_dataset 폴더
- Track_Follow.train(times=5, autosave=True): 수집한 데이터셋으로 학습
  - times: 학습 횟수로 기본값은 5
  - autosave: True 이면 학습된 매개변수 자동 저장. False 는 저장 안함

- Track_Follow.save_model(path='Track_Follow.h5'): 학습한 모델 수동 저장
  - path: 학습된 모델을 저장할 경로. 기본값은 현재 경로의 'Track_Follow.h5'

작업폴더에 "resnet_detect_edge_train.py"라는 새 파일을 만들고, 아래 코드를 작성하여 실행합니다. 만약 track_dataset 폴더를 vscode 의 작업폴더로 복사하지 않았다면, load_datasets()에 "../notebook/track_dataset"를 전달해야 합니다.

```
01: from pop import Pilot, Camera
02:
03: cam = Camera(width=300, height=300)
04: track_follow = Pilot.Track_Follow(camera=cam)
05:
06: track_follow.load_datasets()
07:
08: track_follow.train(5, False)
09: track_follow.save_model(path="Track_Follow.h5")
```

학습시간은 데이터셋 양에 따라 수 분이 걸릴 수도 있습니다. Loss 가 0 에 가까울수록 오차가 작은 모델입니다. 학습을 완료한 모델이 저장되면 다음과 같은 메시지를 확인할 수 있습니다.

[출력]

```
...
0 step loss : 0.025837895073226893

1 step loss : 0.014605101206779094
2 step loss : 0.007547952315493215
3 step loss : 0.006795135795176357
4 step loss : 0.004984153598415068
Save completed.
```

그림 8 학습 실행 결과

1.1.1.4. 트랙 주행 모델 로드 및 사용

학습된 모델을 로드하고 트랙을 주행할 때에는 Track_Follow 객체를 사용합니다.

- Track_Follow.load_model(path='Track_Follow.h5'): 학습된 모델 로드
  - path: 로드할 모델 경로. 기본값은 현재 경로의 'Track_Follow.h5'
- Track_Follow.run(value=None, callback=None): 이미지 또는 카메라 프레임을 입력값으로 이동 방향 예측
  - value: 입력 데이터. 기본값인 None 은 카메라에서 얻음
  - callback: 반환 외에 예측값을 추가로 받을 사용자 함수
  - 반환된 예측값은 이동체가 이동할 지점의 딕셔너리 타입 상태 좌표로 키는 'x'. 'y', 값은 -1 ~ 1 사이. 화면 정 중앙이 {'x':0, 'y':0}

작업폴더에 "resnet_detect_edge_test.py"라는 새 파일을 만들고, 아래 코드를 작성하여 실행합니다. 모델을 로드하는데 수분의 시간이 걸릴 수도 있습니다.

```
01: from pop import Pilot, Camera
02:
03: cam = Camera(width=300, height=300)
04: track_follow = Pilot.Track_Follow(camera=cam)
05:
06: track_follow.load_model(path="Track_Follow.h5")
07:
08: while True:
09:    value = track_follow.run()
10:    print(x = value['x'], y = value['y'])
```

코드를 실행한 후 실습장비를 움직이면 예측값인 목표지점의 x, y 좌표를 출력합니

다. 실습장비의 이동은 앞서 주피터에서 테스트한 “조이스틱으로 이동체 이동”을 사용하면 됩니다.

예측 결과에서 중요한 것은 주행 방향을 예측하는 x 값으로 음수는 왼쪽, 양수는 오른쪽, 0은 정면으로 이동해야 함을 의미합니다. 하지만 이값은 일반적으로 실습장비에서 요구하는 스트어링 값보다 작으므로 실습장비에 적용할 때는 n 만큼 곱해 사용합니다.

```
x = -0.11514842510223389, y = -0.9856566190719604
x = -0.1344992220401764, y = -0.9894192814826965
x = -0.17146293818950653, y = -0.9891960024833679
x = -0.1840624213218689, y = -0.9892452359199524
x = -0.14653095602989197, y = -0.9891765713691711
x = -0.13884778320789337, y = -0.9899221658706665
```

그림 9 코드 실행 결과

Ctrl+c 를 눌러 코드를 종료합니다.

1.1.1.5. 실습 장비로 트랙 자율 주행

끝으로 SerBot 객체를 이용해 실습장비의 트랙 자율주행을 수행해보겠습니다.

작업폴더에 “resnet_detect_edge_drive.py”파일을 생성하여 다음과 같이 코드를 작성하고 실행합니다.

```
01: from pop import Pilot, Camera
02:
03: cam = Camera(width=224, height=224)
04: track_follow = Pilot.Track_Follow(camera=cam)
05: bot=Pilot.SerBot()
06:
07: track_follow.load_model(path="Track_Follow.h5")
```

```
08:
09: bot.setSpeed(50)
10: bot.forward()
11:
12: def driving(value):
13:    steer = value['x']
14:
15:    steer = steer * 1.5
16:    steer = 1 if steer > 1 else -1 if steer < -1 else steer
17:    bot.steering = steer
18:
19: while True:
20:    value = track_follow.run()
21:    driving(value)
```

위 코드는 속도 50 으로 직진주행을 하되, 예측값 x 에 1.5 배한 값을 스티어링 값으로 사용해 트랙 상황에 따라 좌회전 또는 우회전을 하며 트랙을 주행합니다. 예제에는 정지코드가 없으므로 코드를 실행할 때에는 주의하여 주시기 바랍니다.

Ctrl+c 를 눌러 코드를 종료합니다.