딥러닝
프로젝트

20191769 김지수, 20191767 김민지, 20191730 민지민

아이디어 제시

“시각장애인을 위한 도로 안내 로봇”

→시각장애인들은 현재 훈련된 안내견을 통해 길을 안내받고 위험을 대비함

→ 하지만, 안내견이 대형견이라는 이유로 위험한 존재로 여겨지거나 출입을 거부당하는 등 안내견을 동반하는 시각장애인들을 향한 ‘차가운 시선’ 및 ‘출입 거부’와 같은 고충을 겪고 있는 여러 사례들로 찾아볼 수 있음

→ 따라서 이러한 문제를 해결하기 위해 로봇을 학습시켜 안내견의 역할을 대신하도록 하여 로봇을 통해 길을 안내 받으면 좋을 것 같다고 생각

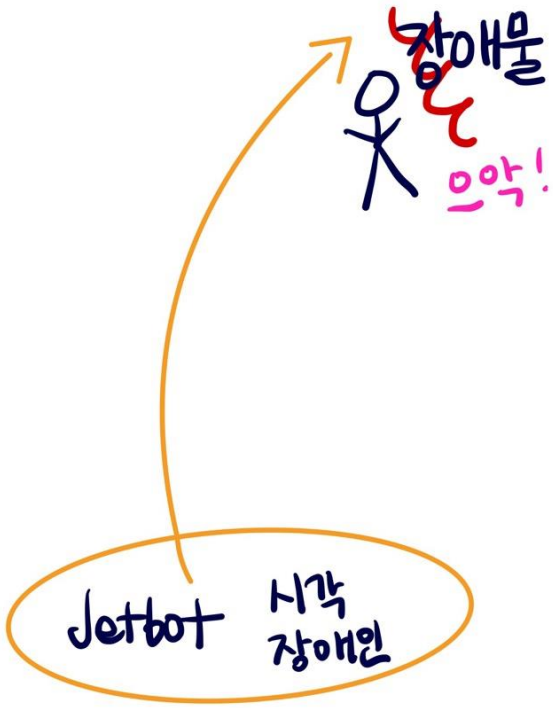
아이디어 제시

- 목표:

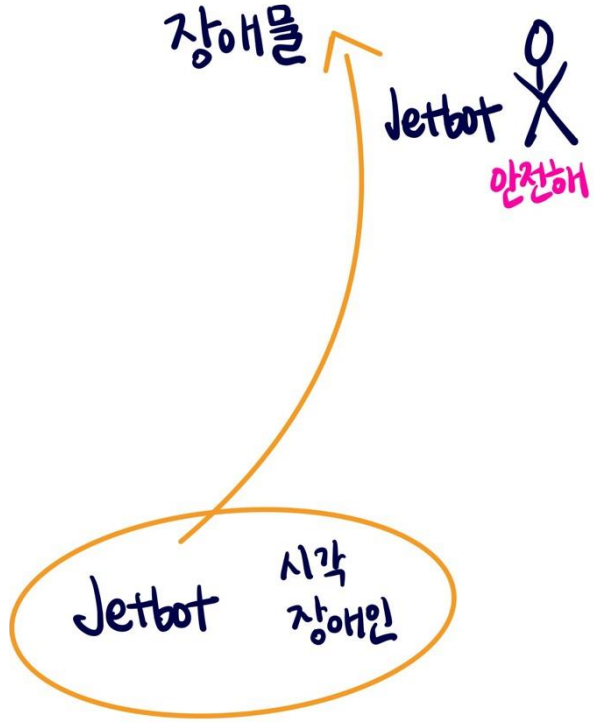
- Jetbot이 트랙의 특정 경로를 따라가는 동시에 장애물을 인지하여 피함으로써 장애물에 부딪히는 일이 없도록 하는 것

- 현재 안내견은 보통 시각장애인의 왼편에 서기 때문에 Jetbot의 경우에도 시각장애인의 왼편에 있다고 생각하고 장애물을 마주칠 경우 항상 오른쪽으로 피하도록 구현

아이디어 제시



• 장애물의 왼쪽으로 Jetbot이 이동할 경우
→ 시각 장애인이 장애물에 부딪힐 위험이 있음



• 장애물의 오른쪽으로 Jetbot이 이동할 경우
→ 시각장애인이 안전하게 장애물을 피할 수 있음

생성한 dataset

➔ Jetbot의 road following의 학습 데이터셋으로 직선구간, 라인을 이탈했을 경우 등 다양한 경우에서 Jetbot이 가야 할 방향을 지정한 데이터셋 생성



생성한 dataset

➔ Jetbot의 충돌 회피를 위한 학습 데이터셋의 경우 free, block 2개의 클래스로 구성


피하지 않을 dataset의 예시


피해야 할 dataset의 예시

➔ block 클래스에는 인형, 의자와 같은 장애물 이미지가 포함되어 있으며, free 클래스에는 Jetbot이 자유롭게 이동할 수 있는 빈 트랙의 배경이미지를 포함하고 있음

학습진행

➔ 모아진 데이터셋을 이용하여 각각 학습을 진행하고 모델 생성

```
NUM_EPOCHS = 10
BEST_MODEL_PATH = 'best_model_1207.pth'
best_accuracy = 0.0

optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

for epoch in range(NUM_EPOCHS):

    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.cross_entropy(outputs, labels)
        loss.backward()
        optimizer.step()

    test_error_count = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        test_error_count += float(torch.sum(torch.abs(labels - outputs.argmax(1))))

    test_accuracy = 1.0 - float(test_error_count) / float(len(test_dataset))
    print('%d: %f' % (epoch, test_accuracy))
    if test_accuracy > best_accuracy:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_accuracy = test_accuracy
```

Collision Avoidance 학습

```
NUM_EPOCHS = 15
BEST_MODEL_PATH = 'best_rf_model_1207.pth'
best_loss = 1e9

optimizer = optim.Adam(model.parameters())

for epoch in range(NUM_EPOCHS):

    model.train()
    train_loss = 0.0
    for images, labels in iter(train_loader):
        images = images.to(device)
        labels = labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        train_loss += float(loss)
        loss.backward()
        optimizer.step()
    train_loss /= len(train_loader)

    model.eval()
    test_loss = 0.0
    for images, labels in iter(test_loader):
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        loss = F.mse_loss(outputs, labels)
        test_loss += float(loss)
    test_loss /= len(test_loader)

    print('%f, %f' % (train_loss, test_loss))
    if test_loss < best_loss:
        torch.save(model.state_dict(), BEST_MODEL_PATH)
        best_loss = test_loss
```

Road Following 학습

Road Following + Collision Avoidance

- Road following과 collision avoidance를 결합하기 위해 각각 학습된 모델을 로드

```
import torch
import torchvision

ca_model = torchvision.models.alexnet(pretrained=False)
ca_model.classifier[6] = torch.nn.Linear(ca_model.classifier[6].in_features, 2)
ca_model.load_state_dict(torch.load('best_model_1207.pth'))

rf_model = torchvision.models.resnet18(pretrained=False)
rf_model.fc = torch.nn.Linear(512, 2)
rf_model.load_state_dict(torch.load('best_rf_model_1207.pth'))
```

Road Following + Collision Avoidance

- Jetbot이 이동하면서 카메라 값이 변경될 때마다 호출되는 함수정의

```
def execute(change):
    global angle, angle_last, blocked_slider, robot, count_stops, stop_time, go_on, x, y, blocked_threshold
    global speed_value, steer_gain, steer_dgain, steer_bias

    steer_gain = steering_gain_slider.value
    steer_dgain = steering_dgain_slider.value
    steer_bias = steering_bias_slider.value

    image_preproc = preprocess(change['new']).to(device)

    #Collision Avoidance model:

    prob_blocked = float(F.softmax(ca_model(image_preproc), dim=1).flatten()[0])

    blocked_slider.value = prob_blocked
    stop_time=stopduration_slider.value

    if go_on == 1:
        if prob_blocked > blocked_threshold.value: # threshold should be above 0.5
            count_stops += 1
            go_on = 2
        else:
            #start of road following detection
            go_on = 1
            count_stops = 0
            xy = rf_model(image_preproc).detach().float().cpu().numpy().flatten()
            x = xy[0]
            y = (0.5 - xy[1]) / 2.0
            speed_value = speed_control_slider.value
    else:
        count_stops += 1
        if count_stops < stop_time:
            x = 0.0 #set x steering to zero
            y = 0.0 #set y steering to zero
            speed_value = 0 # set speed to zero (can set to turn as well)
        else:
            go_on = 1
            count_stops = 0


    angle = math.atan2(x, y)
    pid = angle * steer_gain + (angle - angle_last) * steer_dgain
    steer_val = pid + steer_bias
    angle_last = angle
    robot.left_motor.value = max(min(speed_value + steer_val, 1.0), 0.0)
    robot.right_motor.value = max(min(speed_value - steer_val, 1.0), 0.0)

execute({'new': camera.value})
```

이 함수는 다음 단계를 수행

- 카메라 이미지 전처리
- 도로 추적 및 충돌 방지를 위한 신경망 모델 실행
- Jetbot이 도로 추적을 수행하고 장애물이 감지될 때마다 오른쪽으로 회피하는 if 문
- 대략적인 스티어링 값 계산
- 비례/미분 제어(PD)를 사용하여 모터 제어

발생한 오류

- 함수를 실행시켜 보았으나 카메라 값이 변경 되지 않고 계속 한 프레임에 멈춰있는 카메라 오류가 발생하여 카메라로 보고 움직이는 행위 자체가 불가능
- 오류 해결 시도했으나 결국 실패하여 동작을 시연해보지 못함

감사합니다.