결과 보고서

임베디드응용및실습.
기말 프로젝트

과 목 명	임베디드응용및실습
학 번	2019161068
이 름	백승진
제 출 일	2024년 12월 11일

1. 서론

1.1. 자율주행 개요

자율주행의 핵심은 차량 주변 환경을 정확히 인지하고 적절한 주행 명령을 내리는 것인데, 이를 위해 컴퓨터 비전과 인공지능 기술이 활발히 활용되고 있습니다.

컴퓨터 비전은 카메라 등 센서를 통해 수집한 영상 데이터를 분석하여 차선, 교통 표지판, 장애물 등을 인식하는 기술입니다. 본 강의에서는 OpenCV 라이브러리를 사용하여 카메라 영상에서 차선을 검출하고 주행 방향을 판단하는 과정을 다루었습니다.

인공지능, 특히 딥러닝 기술은 방대한 주행 데이터를 학습하여 최적의 주행 방식을 스스로 터득합니다. 수집한 주행 영상과 조향 각도 등의 데이터를 딥러닝 모델에 입력하여 학습시키면, 모델이 영상에 따른 적절한 주행 명령을 출력하도록 만들 수 있습니다.

이처럼 컴퓨터 비전과 딥러닝의 발전은 자율주행 구현에 크게 기여하고 있습니다. 다만 완전 자율주행을 위해서는 더욱 정교한 인식 및 판단 알고리즘 개발, 다양한 주행 상황에 대한 대응, 안전성 확보 등이 뒷받침되어야 할 것입니다. 강의 자료에서 소개된 기술들이 자율주행 발전의 토대가 될 것으로 기대됩니다.

1.2. 프로젝트 목표

본 프로젝트의 목표는 OpenCV와 딥러닝 기술을 활용하여 카메라 기반의 자율주행 자동차를 구현하는 것입니다. 세부적인 목표는 다음과 같습니다.

1) OpenCV를 사용하여 카메라 영상에서 차선을 인식하고 차선을 따라 주행하는 라인트레이싱 알고리즘을 개발합니다. 이를 통해 자율주행차가 정해진 트랙을 안정적으로 주행할 수 있도록 합니다.
2) 딥러닝 기술을 활용하여 주행 중 마주치는 장애물이나 표지판 등을 인식하고 그에 맞는 주행 명령을 내리는 모델을 학습시킵니다. 이를 통해 자율주행차가 주변 환경을 이해하고 상황에 적절히 대응할 수 있게 합니다.
3) 라인트레이싱과 딥러닝 모델을 통합하여 자율주행 시스템을 완성합니다. 카메라 영상을 입력받아 차선 인식과 장애물 감지를 수행하고, 그 결과에 따라 주행 명령을 차량 제어부에 전달하는 일련의 과정을 구현합니다.
4) 실제 자율주행차를 제작하고 위에서 개발한 자율주행 시스템을 탑재하여 테스트를 진행합니다. 다양한 주행 환경에서의 성능을 평가하고 필요한 부분을 개선해나갑니다.

궁극적으로 본 프로젝트를 통해 소형 자율주행차의 기본 원리를 학습하고, 카메라 기반 자율주행 시스템 구현 역량을 기르고자 합니다. 이는 실제 자율주행 기술 개발에 있어 중요한 토대가 될 것입니다.

2. 본론

2.1. 시스템 구성

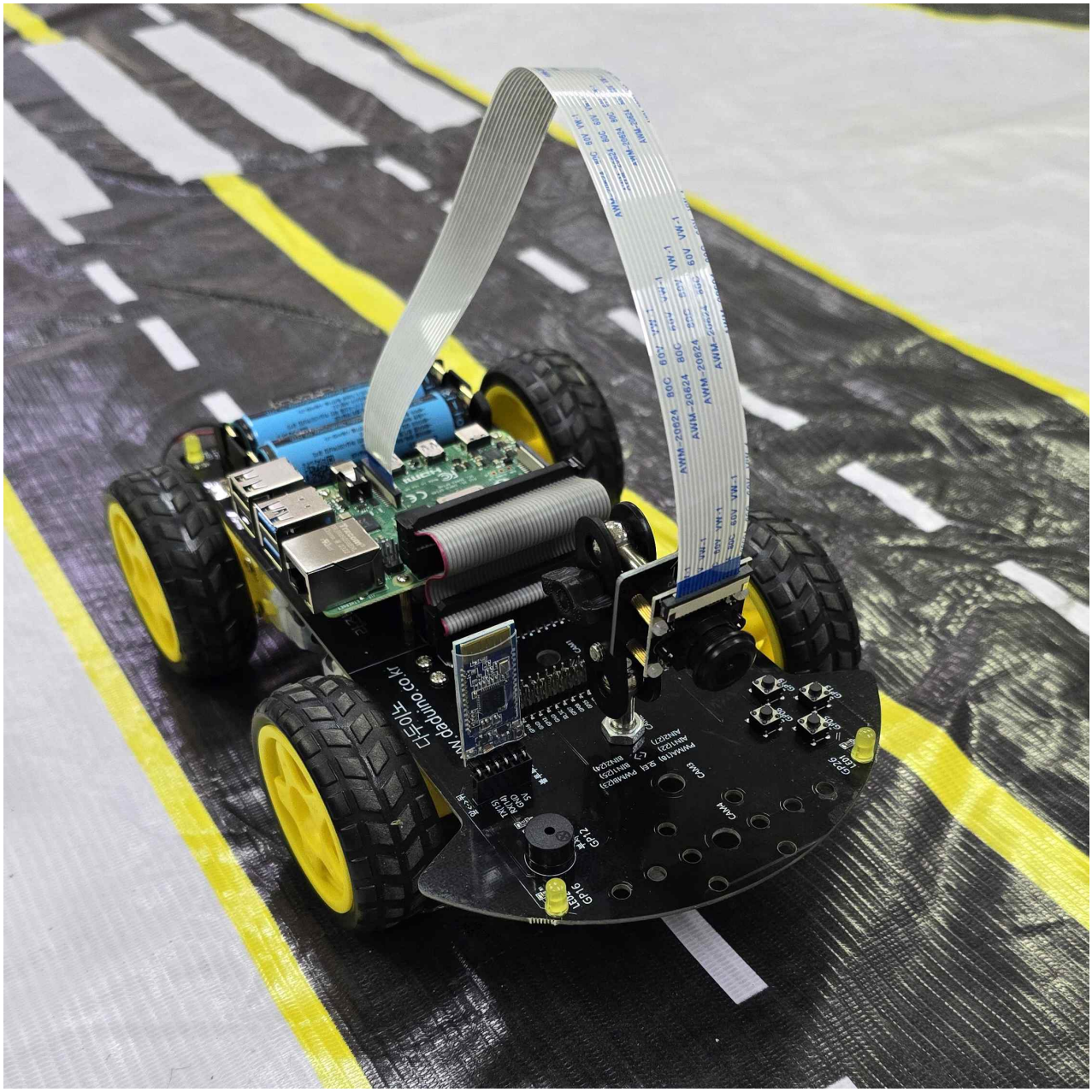
자율주행차는 센싱, 제어, 구동의 세 가지 시스템으로 구성되며, 센서를 통해 수집된 환경 정보를 바탕으로 주행 경로를 결정하고 모터를 제어하여 자율주행을 수행합니다. 다음 절에서는 이러한 자율주행 시스템을 구현하기 위한 하드웨어 및 소프트웨어 구성에 대해 구체적으로 기술합니다.

2.1.1. 하드웨어 구성

본 프로젝트의 자율주행차 하드웨어는 다음과 같이 구성됩니다.

1) 라즈베리파이
   i) 자율주행차의 두뇌 역할을 하는 소형 컴퓨터로, 센서 데이터 처리, 주행 알고리즘 수행, 모터 제어 등을 담당합니다.
   ii) 카메라, 블루투스 모듈 등 다양한 주변 장치와 연결되어 자율주행에 필요한 정보를 주고받습니다.
2) 모터 드라이버(TB6612)
   i) 라즈베리파이의 GPIO 신호를 모터가 동작할 수 있는 높은 전류로 변환해주는 모듈입니다.
   ii) 2개의 DC 모터를 제어할 수 있으며, 속도와 방향을 제어하기 위한 PWM 신호와 방향 신호를 받습니다.
3) DC 모터
   i) 자율주행차의 바퀴를 구동하는 모터로, 일반적으로 4개가 사용됩니다(왼쪽 2개, 오른쪽 2개).
   ii) 모터 드라이버를 통해 공급되는 전원으로 동작하며, 정/역회전이 가능합니다.
4) 스위치
   i) 사용자 입력을 받기 위한 버튼 스위치로, 자동차의 시동, 모드 변경 등을 제어하는데 사용됩니다.
   ii) 라즈베리파이의 GPIO에 연결되어 버튼 상태(눌림/뗌)를 감지할 수 있습니다.
5) 부저
   i) 자동차의 경적 역할을 하는 부품으로, 라즈베리파이의 PWM 신호에 따라 소리를 냅니다.
   ii) 장애물 감지 시 경고음 발생 등 다양한 용도로 활용 가능합니다.
6) 블루투스 모듈
   i) 스마트폰과 자동차 간 무선 통신을 가능하게 하는 모듈입니다.
   ii) UART 통신을 통해 라즈베리파이와 연결되며, 스마트폰 앱을 통해 자동차를 원격 제어할 수 있습니다.

이러한 하드웨어 요소들이 유기적으로 연결되어 하나의 자율주행 시스템을 구성하게 됩니다. 각 요소의 제어와 데이터 처리는 라즈베리파이에서 파이썬 코드로 구현되며, 최종적으로 센서 정보에 기반한 주행 명령이 모터로 전달되어 자동차가 움직이게 됩니다.



[그림 1] 완성된 자율주행차 예시

2.1.2. 소프트웨어 구성

자율 주행 자동차 프로젝트의 소프트웨어 구성은 다음과 같은 주요 구성요소로 이루어져 있습니다.

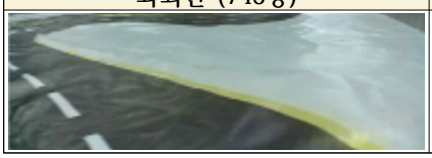
1) GPIO (General Purpose Input/Output) 설정
   i) 라즈베리파이의 GPIO 핀을 다양한 입출력 장치와 상호 작용하도록 설정하고 구성합니다.
   ii) 버튼 입력, 부저/스피커 출력, 모터 제어를 위한 GPIO 핀 설정을 포함합니다.
   iii) RPi.GPIO Python 라이브러리를 사용하여 GPIO 기능을 관리합니다.
2) 버튼 입력 처리
   i) 4개의 버튼(SW1, SW2, SW3, SW4)이 라즈베리파이 GPIO 핀에 연결됩니다.
   ii) 소프트웨어는 버튼 누름과 해제를 감지하고, 버튼 입력에 따라 적절한 동작을 수행합니다.
   iii) 버튼 입력은 자율주행 자동차의 전진, 후진, 좌회전, 우회전 등의 움직임을 제어하는 데 사용됩니다.

3) 부저/스피커 제어
   i) 부저 또는 스피커가 GPIO 핀에 연결되어 오디오 피드백이나 경고음을 제공합니다.
   ii) 소프트웨어는 펄스폭 변조(PWM)를 사용하여 다양한 음계와 사운드 패턴을 생성합니다.
   iii) 이를 통해 자율주행 자동차만의 고유한 경적 소리나 멜로디를 구현할 수 있습니다.
4) 모터 제어
   i) 모터 드라이버 보드(TB6612 모듈 등)를 사용하여 바퀴를 움직이는 모터를 제어합니다.
   ii) 소프트웨어는 적절한 GPIO 핀을 설정하여 모터 드라이버의 속도와 방향을 관리합니다.
   iii) 이를 통해 자동차를 전진, 후진, 좌회전, 우회전시킬 수 있습니다.
5) 블루투스 통신
   i) HM-10과 같은 블루투스 모듈이 라즈베리파이의 UART 핀에 연결됩니다.
   ii) 소프트웨어는 블루투스 연결을 설정하고, 페어링된 스마트폰이나 태블릿에서 보내는 명령어를 수신합니다.
   iii) 수신된 블루투스 명령어는 버튼 입력 처리와 유사하게 자율주행 자동차의 움직임을 제어하는 데 사용됩니다.
6) 멀티 스레딩
   i) 소프트웨어는 블루투스 통신과 자동차 제어 로직을 별도의 스레드에서 처리합니다.
   ii) 하나의 스레드는 블루투스 직렬 통신을 담당하고, 다른 하나의 스레드는 수신된 입력에 따라 자동차 운영을 관리합니다.
   iii) 이러한 접근 방식을 통해 블루투스 통신과 자동차 제어가 동시에 실행되어, 시스템의 응답성과 안정성이 향상됩니다.

이 소프트웨어 구성은 모듈식이며 확장 가능한 설계를 가지고 있어, 추후 추가 기능이나 센서의 통합이 용이합니다. 다양한 기능을 별도의 구성요소로 분리함으로써 코드 유지 관리와 테스트 가능성이 향상되어, 자율주행 자동차의 기능을 디버깅하고 발전시키기 쉽습니다.

2.1.3. 데이터 수집 및 전처리

자율주행 모델 학습을 위한 데이터 수집은 키보드 방향키를 이용한 수동 주행 방식으로 진행되었습니다. 주행 중 방향키 입력 시마다 해당 순간의 카메라 이미지와 조향 정보가 자동으로 저장되도록 구현하였습니다. 최종적으로 총 3,292장의 이미지를 수집했으며, 이는 여러 차례의 반복적인 데이터 수집 과정을 통해 최적화되었습니다.

좌회전 (743장)	직진 (2,018장)	우회전 (531장)
		

[표 1] 수집 데이터 예시

특히 주행 코스의 특성을 고려하여 좌회전(6회)과 우회전(2회)의 불균형을 해소하기 위해 우회전 구간에 대한 추가 데이터 수집을 진행했습니다. 이러한 데이터 균형화 작업을 통해 모든 주행 상황에서 안정적으로 동작하는 모델을 구현할 수 있었습니다.

2.2. 프로젝트 진행 일정

2.2.1. 1주차 (11.27~11.30)

- 11.27 (수) : 자율주행 기능 테스트

- 11.30 (토) : 1차, 2차 데이터셋 수집

2.2.2. 2주차 (12.02~12.07)

- 12.02 (월) : 3차, 4차 데이터셋 수집, 자율주행 기능 추가
- 12.04 (수) : 객체 인식 기능 테스트
- 12.06 (금) : 객체 인식 기능 추가, 부가 기능 추가(비상등, 경적, 시리얼 통신), 키보드 입력 기능 삭제
- 12.07 (토) : 부가 기능 추가(음악 재생), 코드 리팩터링

2.2.3. 3주차 (12.10)

- 12.10 (화) : 객체별 인식 검증 및 수정, 최종 테스트 진행

2.3. 소스 코드

2.3.1. Train Code (총 242 Lines)

```
# GPU 설정
import os
os.environ['CUDA_VISIBLE_DEVICES'] = '1'
os.environ["HDF5_USE_FILE_LOCKING"] = "FALSE"

# 필요한 라이브러리 임포트
import random
import fnmatch
import datetime
import pickle
import cv2
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow.keras
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential, load_model
from tensorflow.keras.layers import Conv2D, MaxPool2D, Dropout, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import matplotlib.image as mping
from PIL import Image

# 이미지 데이터 로드
data_dir = "crop_img"
file_list = os.listdir(data_dir)
image_paths = []
steering_angles = []
pattern ="*.png"

# 이미지 경로와 조향각 데이터 수집
for filename in file_list:
    if fnmatch.fnmatch(filename, pattern):
        image_paths.append(os.path.join(data_dir, filename))
```

```
        angle = int(filename[-7:-4])  # 파일명에서 각도 추출
        steering_angles.append(angle)

# 수집된 데이터 확인
print('image_paths', image_paths)
print('steering_angles', steering_angles)

# 샘플 이미지 시각화
image_index = 1
plt.imshow(Image.open(image_paths[image_index]))
print("image_path: ", image_paths[image_index])
print("steering_Angle: ", steering_angles[image_index])

# 데이터프레임 생성
df = pd.DataFrame()
df['ImagePath'] = image_paths
df['Angle'] = steering_angles

# 조향각 히스토그램 시각화
num_of_bins = 10
hist, bins = np.histogram(df['Angle'], num_of_bins)
print(hist)
print(bins)
fig, axes = plt.subplots(1,1,figsize=(10,4))
axes.hist(df['Angle'], bins=num_of_bins, width=1, color='blue')

# 조향각 레이블 변환 (0도 : 0 / 45도 : 1 / 135도 : 2)
s = np.array(steering_angles)
new_angles = np.copy(steering_angles)
new_angles[np.where(s==0)] = 0
new_angles[np.where(s==45)] = 1
new_angles[np.where(s==135)] = 2
print(new_angles)
print(steering_angles)
print(len(np.where(new_angles==0)[0]))
print(len(np.where(new_angles==1)[0]))
print(len(np.where(new_angles==2)[0]))
df['Angle'] = new_angles

# 학습 데이터와 검증 데이터 분할 (8:2)
X_train, X_valid, y_train, y_valid = train_test_split(image_paths, new_angles, test_size=0.2)
print("Training data: {}, Validation data : {}".format(len(X_train), len(X_valid)))

# 학습/검증 데이터 분포 시각화
fig, axes = plt.subplots(1,2, figsize=(12,4))
axes[0].hist(y_train, bins=num_of_bins, width=1, color='blue')
axes[0].set_title("Training data")
axes[1].hist(y_valid, bins=num_of_bins, width=1, color='red')
axes[1].set_title("Validation data")

# 입력 데이터와 레이블 매핑 확인
for a, b in zip(X_train, y_train):
    print(a,b)
```

```
# 이미지 로드 함수 정의
def my_imread(image_path):
    image = cv2.imread(image_path)
    return image

# 이미지 전처리 함수
def img_preprocess(image):
    # 이미지 정규화
    image = np.float32(image) / 255.
    return image

# 전처리 전/후 이미지 비교 시각화
fig, axes = plt.subplots(1,2,figsize=(15,10))
image_ori = my_imread(image_paths[image_index])
image_processed = img_preprocess(image_ori)
axes[0].imshow(image_ori)
axes[0].set_title("ori")
axes[1].imshow(image_processed)
axes[1].set_title("processed")
print(image_ori.shape)

# CNN 모델 정의
def neural_net():
    model = Sequential()
    model.add(Conv2D(32, (5,5), input_shape=(66,200,3), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (5,5), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(64, (5,5), activation='relu'))
    model.add(MaxPool2D((2,2)))
    model.add(Dropout(0.2))
    model.add(Conv2D(128, (3,3), activation='relu'))
    model.add(Dropout(0.2))

    # 완전연결 층 (분류)
    model.add(Flatten())
    model.add(Dropout(0.5))
    model.add(Dense(128, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(64, activation='relu', kernel_regularizer=regularizers.l2(0.01)))
    model.add(Dropout(0.5))
    model.add(Dense(3, activation='softmax'))

    # 모델 컴파일 설정
    opt = Adam(lr=1e-3)
    model.compile(
        loss='sparse_categorical_crossentropy',
        optimizer=opt
    )

    return model
```

```
# 모델 생성 및 구조 출력
model = neural_net()
print(model.summary())

# 학습 데이터 생성기 정의
def image_data_generator(image_paths, steering_angles, batch_size):
    while True:
        batch_images = []
        batch_steering_angles = []
        len_imgs = len(image_paths)

        # 배치 크기만큼 랜덤하게 데이터 선택
        for i in range(batch_size):
            random_index = random.randint(0, len_imgs-1)
            image_path = image_paths[random_index]
            image = my_imread(image_path)
            steering_angle = steering_angles[random_index]

            # 이미지 전처리 적용
            image = img_preprocess(image)
            batch_images.append(image)
            batch_steering_angles.append(steering_angle)
        yield(np.asarray(batch_images), np.asarray(batch_steering_angles))

# 생성된 배치 데이터 확인
ncol = 2
nrow = 2

print(len(X_train))

X_train_batch, y_train_batch = next(image_data_generator(X_train, y_train, nrow))
X_valid_batch, y_valid_batch = next(image_data_generator(X_valid, y_valid, nrow))

# 배치 데이터 시각화
print(len(X_train_batch))
fig, axes = plt.subplots(nrow, ncol, figsize=(15,6))
fig.tight_layout()

for i in range(nrow):
    axes[i][0].imshow(X_train_batch[i])
    axes[i][0].set_title("training, angle={}".format(y_train_batch[i]))
    axes[i][1].imshow(X_valid_batch[i])
    axes[i][1].set_title("validation, angle={}".format(y_valid_batch[i]))

# 모델 저장 디렉토리 설정
save_dir = datetime.datetime.now().strftime('%Y%m%d_%H%M')
model_output_dir = os.path.join(save_dir)
os.mkdir(model_output_dir)

print('model_output_dir ', model_output_dir)

# 학습 파라미터 설정
train_bSize = 16
valid_bSize = 128
```

```
# 체크포인트 콜백 설정 (최적 모델 저장)
checkpoint = 'lane_navigation_'+ save_dir + '.h5'
checkpoint_callback = ModelCheckpoint(
    filepath=os.path.join(model_output_dir, checkpoint),
    verbose=1,
    save_best_only=True
)

# 모델 학습
history = model.fit_generator(
    image_data_generator(X_train, y_train, batch_size=train_bSize),
    steps_per_epoch=len(X_train) // train_bSize,
    epochs=150,
    validation_data=image_data_generator(X_valid, y_valid, batch_size=valid_bSize),
    validation_steps=len(X_valid) // valid_bSize,
    verbose=1,
    shuffle=True,
    callbacks=[checkpoint_callback]
)

# 학습 히스토리 저장
history_path = os.path.join(model_output_dir, 'history.pickle')
with open(history_path, 'wb') as f:
    pickle.dump(history.history, f, pickle.HIGHEST_PROTOCOL)

# 학습된 모델로 예측 테스트
img_path = X_valid[0]
image = my_imread(img_path)
steering_angle = steering_angles[0]
image = img_preprocess(image)
print(image.shape)
print(model.predict(np.expand_dims(image, 0))[0])
print(np.argmax(model.predict(np.expand_dims(image, 0))[0]))
plt.imshow(image)

print(model_output_dir)

# 학습 히스토리 로드 및 손실 그래프 시각화
history_path = os.path.join(model_output_dir, 'history.pickle')
with open(history_path, 'rb') as f:
    history = pickle.load(f)

plt.plot(history['loss'], color='blue')
plt.plot(history['val_loss'], color='red')
plt.legend(['training loss', 'validation loss'])
```

2.3.2. Inference Code (총 363 Lines)

```
# 필요한 라이브러리 임포트
import cv2 as cv
import numpy as np
import threading, time
import SDcar
```

```
import sys
import tensorflow as tf
import RPi.GPIO as GPIO
import serial
from tensorflow.keras.models import load_model

# GPIO 핀 번호 설정
BUZZER = 12  # 부저 핀
L_Light = 26  # 왼쪽 비상등 핀
R_Light = 16  # 오른쪽 비상등 핀

# GPIO 초기화
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
GPIO.setup(L_Light, GPIO.OUT)
GPIO.setup(R_Light, GPIO.OUT)

# 시리얼 통신 설정
bleSerial = serial.Serial("/dev/ttyS0", baudrate=9600, timeout=1.0)
gData = ""  # 수신 데이터 저장 변수

# 전역 변수 초기화
frame = None  # 카메라 프레임
class_name = None  # 감지된 물체 이름
is_running = True  # 프로그램 실행 상태
enable_AIdrive = False  # 자율주행 활성화 여부
enable_detection = False  # 객체 인식 활성화 여부
enable_music = False  # 음악 재생 상태
current_note_index = 0  # 현재 재생 중인 음표 인덱스
last_note_time = 0  # 마지막으로 음표가 변경된 시간

# 음계 사전 정의
notes_dict = {
    'Do': 261.63,
    'Di' : 277.18,
    'Re': 293.66,
    'Ri' : 311.13,
    'Mi': 329.63,
    'Fa': 349.23,
    'Fi' : 369.99,
    'Sol': 392.00,
    'Si' : 415.30,
    'La': 440.00,
    'Li' : 466.16,
    'Ti': 493.88,
    '2Do': 523.25,
    '2Di' : 554.37,
    '2Re' : 587.33,
    '2Ri' : 622.25,
    '2Mi' : 659.25,
    '2Fa' : 698.46,
    '2Fi' : 739.99,
    '2Sol' : 783.99,
```

```
    '2Si' : 830.61,
    '2La' : 880.00,
    '2Li' : 932.33,
    '2Ti' : 987.77
}

# 노래 데이터 정의
song_scale = [
    ('2Di', 0.22), ('Rest', 0.22), ('2Di', 0.22), ('Rest', 0.22), ('2Di', 0.22), ('Rest', 0.22), ('2Di', 0.22), ('2Ri', 0.22),
    ('Rest', 0.22), ('2Ri', 0.22), ('Rest', 0.22), ('2Ri', 0.22), ('2Ri', 0.22), ('Rest', 0.22), ('2Ri', 0.22), ('Rest', 0.22),
    ('Si', 0.22), ('Rest', 0.22), ('Si', 0.22), ('Rest', 0.22), ('Si', 0.22), ('Rest', 0.22), ('Si', 0.22), ('2Fa', 0.22),
    ('Rest', 0.22), ('2Fa', 0.22), ('Rest', 0.22), ('2Fa', 0.22), ('2Fa', 0.22), ('Rest', 0.22), ('2Fa', 0.22), ('Rest', 0.22)
]

# 자율주행 모델 로드
model_path = 'my_checkpoint/lane_navigation_20241201_1705.h5'
model = load_model(model_path)

with open('object_detection_classes_coco.txt', 'r') as f:
    class_names = f.read().split('₩n')
print(class_names)
COLORS = np.random.uniform(0, 255, size=(len(class_names), 3))

# 객체 인식 클래스 및 모델 로드
od_model = cv.dnn.readNetFromTensorflow(
    model='frozen_inference_graph.pb',
    config='ssd_mobilenet_v2_coco_2018_03_29.pbtxt'
)

# 시리얼 통신 스레드 함수
def serial_thread():
    global gData, class_name, is_running
    global enable_AIdrive, enable_detection, enable_music

    while is_running:
        try:
            # 시리얼 데이터 읽기
            data = bleSerial.readline().decode().strip()
            if data == "start_A":  # 자율주행 활성화 수신
                enable_AIdrive = True
            elif data == "stop_A":  # 자율주행 비활성화 수신
                enable_AIdrive = False
                time.sleep(0.1)
                car.motor_stop()
            elif data == "start_D":  # 물체감지 활성화 수신
                enable_detection = True
            elif data == "stop_D":  # 물체감지 비활성화 수신
                enable_detection = False
                class_name = None
                time.sleep(0.1)
                cv.destroyWindow('detection')
            elif data == "start_M":  # 음악 재생 활성화 수신
                enable_music = True
                p.ChangeDutyCycle(1)
```

```
            elif data == "stop_M":  # 음악 재생 비활성화 수신
                enable_music = False
                p.ChangeDutyCycle(0)
            elif data == "finish":  # 프로그램 종료 수신
                car.motor_stop()
                enable_AIdrive = False
                enable_detection = False
                p.stop()
                car.clean_GPIO()
                GPIO.cleanup()
                bleSerial.close()
                is_running = False
        except Exception as e:
            print(f"Serial reading error: {e}")

# 객체 인식 스레드 함수
def detection_thread():
    global frame, is_running, enable_detection, class_name

    frame_count = 0
    process_interval = 30  # 객체 인식 주기

    while is_running:
        current_frame = None
        if frame is not None:
            lock.acquire()
            current_frame = frame.copy()
            lock.release()

            if enable_detection and frame_count % process_interval == 0:
                try:
                    object_detection(current_frame)
                except Exception as e:
                    print(f"Error in object detection: {e}")
            elif frame_count % process_interval == 0:  # detection이 꺼져있을 때만 초기화
                class_name = None

        frame_count = (frame_count + 1) % 31
        time.sleep(0.01)

# 음악 재생 스레드 함수
def music_thread():
    global current_note_index, last_note_time, enable_music, is_running

    while is_running:
        if enable_music:
            current_time = time.time()

            if current_note_index == 0:
                # 첫 음 재생 시 특별 처리
                note, duration = song_scale[0]
                if note != 'Rest':
                    p.ChangeFrequency(notes_dict[note])
                    p.ChangeDutyCycle(1)
```

```
                    else:
                        p.ChangeDutyCycle(0)
                    time.sleep(0.2)
                    last_note_time = current_time
                    current_note_index = 1

                # 현재 음표의 지속 시간이 지났는지 확인
                elif current_time - last_note_time >= song_scale[current_note_index-1][1]:
                    if current_note_index < len(song_scale):
                        # 다음 음표 재생
                        note, duration = song_scale[current_note_index]

                        # 쉼표가 아닐 때만 주파수 변경
                        if note != 'Rest':
                            p.ChangeFrequency(notes_dict[note])
                            p.ChangeDutyCycle(1)
                        # 쉼표일 때는 소리를 끔
                        else:
                            p.ChangeDutyCycle(0)
                        last_note_time = current_time
                        current_note_index += 1
                    else:
                        # 노래가 끝나면 처음부터 다시 시작
                        current_note_index = 0
                        last_note_time = current_time
                        p.ChangeDutyCycle(0)

            time.sleep(0.01)

# 자율주행 함수
def drive_AI(img):
    global class_name

    img = np.expand_dims(img, 0)
    res = model.predict(img)[0]
    steering_angle = np.argmax(np.array(res))

    # class_name 읽기 전에 lock 획득
    lock.acquire()
    current_class = class_name  # 로컬 변수에 복사
    lock.release()

    if enable_detection and current_class == 'laptop':
        car.motor_stop()
        horn()
    else:
        if steering_angle == 0:
            speedSet = 40
            car.motor_go(speedSet)
        elif steering_angle == 1:
            speedSet = 20
            car.motor_left(speedSet)
        elif steering_angle == 2:
            speedSet = 20
```

```
            car.motor_right(speedSet)
        else:
            print("This cannot be entered")

# 객체 인식 함수
def object_detection(image):
    global class_name
    detected = False  # 객체 인식 여부를 추적하는 플래그

    # 관심 영역(ROI) 설정
    height = int(image.shape[0] * 0.6)
    roi = image[:height, :]

    image_height, image_width, _ = roi.shape
    blob = cv.dnn.blobFromImage(image=roi, size=(160, 160), swapRB=True)

    # 객체 인식 수행
    od_model.setInput(blob)
    output = od_model.forward()

    for detection in output[0, 0, :, :]:
        confidence = detection[2]
        if confidence > 0.5:
            class_id = detection[1]
            detected_class = class_names[int(class_id) - 1]
            color = COLORS[int(class_id)]

            print(f"name : {detected_class}, confidence : {confidence}")

            # 물체가 감지되었음을 표시
            detected = True
            class_name = detected_class

            box_x = detection[3] * image_width
            box_y = detection[4] * image_height
            box_width = detection[5] * image_width
            box_height = detection[6] * image_height

            cv.rectangle(image, (int(box_x), int(box_y)), (int(box_width), int(box_height)), color, thickness=2)
            cv.putText(image, detected_class, (int(box_x), int(box_y - 5)), cv.FONT_HERSHEY_SIMPLEX, 1, color, 2)

    # 물체가 감지되지 않았다면 class_name을 None으로 초기화
    if not detected:
        class_name = None

    # 감지 결과를 표시
    if enable_detection:
        cv.imshow('detection', roi)
        cv.waitKey(1)

# 경적 출력 함수
def horn():
    p.ChangeDutyCycle(1)
    p.ChangeFrequency(659.25)
```

```
        time.sleep(0.1)
        p.ChangeFrequency(659.25)
        time.sleep(0.4)
        p.ChangeDutyCycle(0)

# 비상등 출력 함수
def emergency_lights():
    global enable_AIdrive, is_running
    while is_running:
        if enable_AIdrive:
            GPIO.output(L_Light, GPIO.HIGH)
            GPIO.output(R_Light, GPIO.HIGH)
            time.sleep(0.5)
            GPIO.output(L_Light, GPIO.LOW)
            GPIO.output(R_Light, GPIO.LOW)
            time.sleep(0.5)
        else:
            GPIO.output(L_Light, GPIO.LOW)
            GPIO.output(R_Light, GPIO.LOW)
            time.sleep(0.1)

# 메인 함수
def main():
    global frame, is_running

    # 카메라 초기화
    camera = cv.VideoCapture(0)
    camera.set(cv.CAP_PROP_FRAME_WIDTH, v_x)
    camera.set(cv.CAP_PROP_FRAME_HEIGHT, v_y)

    try:
        while camera.isOpened() and is_running:
            lock.acquire()
            ret, frame = camera.read()
            frame = cv.flip(frame, -1)
            lock.release()

            cv.imshow('camera', frame)

            crop_img = frame[int(v_y/2):, :]
            crop_img = cv.resize(crop_img, (200, 66))

            if enable_AIdrive:
                drive_AI(crop_img)

            cv.waitKey(1)

    except Exception as e:
        exception_type, exception_object, exception_traceback = sys.exc_info()
        filename = exception_traceback.tb_frame.f_code.co_filename
        line_number = exception_traceback.tb_lineno

        print("Exception type: ", exception_type)
        print("File name: ", filename)
```

```
            print("Line number: ", line_number)

            is_running = False

if __name__ == '__main__':
    # PWM 및 기타 초기화
    p = GPIO.PWM(BUZZER, 1)
    p.start(0)

    # 카메라 해상도 설정
    v_x = 320
    v_y = 240

    # 스레드 초기화 및 시작
    lock = threading.Lock()

    t_serial = threading.Thread(target=serial_thread)
    t_serial.start()

    t_detection = threading.Thread(target=detection_thread)
    t_detection.start()

    t_emergency = threading.Thread(target=emergency_lights)
    t_emergency.start()

    t_music = threading.Thread(target=music_thread)
    t_music.start()

    car = SDcar.Drive()

    main()

    # 종료 처리
    is_running = False
```

2.4. 테스트 및 결과

2.4.1. 자율주행 테스트

실내 테스트 트랙(4m x 3m 규모, 흰색 바닥에 검은색 라인)에서 자율주행 성능을 평가했습니다. 기본 주행 테스트는 트랙 3바퀴 연속 주행을 통한 안정성 검증과 함께, 다양한 출발 위치와 방향에서의 주행 성능을 종합적으로 평가했습니다. 총 20회의 테스트를 진행한 결과, 모든 시도에서 성공적인 주행을 달성하여 100%의 완벽한 성공률을 기록했습니다. 특히 주목할 만한 점은, 모터의 속도 파라미터값을 100까지 상향 조정했을 때도 회전 구간에서 미세한 불안정성이 관찰되었으나 전반적인 자율주행 기능은 정상적으로 작동했다는 것입니다. 주행 중 간혹 차선을 밟는 경우가 발생했으나, 이러한 상황에서도 차량이 신속하게 정상 궤도로 복귀하며 안정적인 주행 경로를 유지하는 것을 확인할 수 있었습니다.

[그림 2] 실내 테스트 트랙

2.4.2. 객체 인식 및 긴급 정지 테스트

객체 인식 시스템의 성능을 평가하기 위해 총 다섯 개의 물체를 대상으로 테스트를 진행했습니다. 물체별 10회의 테스트 결과, 노트북이 가장 높은 객체 인식 정확도를 보였습니다.

물체	노트북	스마트폰	텀블러	키보드	자동차 이미지
정확도 (%)	76.28	52.18	38.40	67.92	71.79

전반적으로 객체의 종류와 특성에 따라 인식률의 차이가 크게 나타났습니다. 노트북의 경우 76.28%의 높은 정확도를 보였는데, 이는 크기가 크고 일정한 형태를 가진 특성 때문으로 분석됩니다. 자동차 이미지와 키보드도 각각 71.79%, 67.92%로 비교적 양호한 인식률을 보였습니다. 반면 스마트폰(52.18%)과 텀블러(38.40%)는 상대적으로 낮은 인식률을 보였는데, 이는 객체의 크기가 작고 반사율이 높은 표면 특성이 영향을 미친 것으로 판단됩니다.

긴급 정지 기능의 성능을 테스트한 결과, 객체 인식 후 차량 정지까지 약 2초의 반응 시간이 소요되었습니다. 이는 당초 목표했던 반응 속도보다는 다소 늦은 수치이나, 긴급 정지 기능 자체는 안정적으로 작동하는 것을 확인했습니다. 반응 속도가 지연되는 주된 원인은 자율주행, 객체 인식, 음악 재생 등 다양한 기능이 동시에 수행되면서 발생하는 높은 연산 부하로 분석됩니다. 이는 하드웨어 성능 개선을 통해 반응 속도를 향상시킬 수 있을 것으로 예상됩니다.

2.4.3. 부가 기능 테스트

자율주행차의 기본적인 기능 외에 실제 차량의 특성을 반영한 다양한 부가 기능들을 구현했습니다.

1) 비상등 시스템
   i) 저속 주행 시 실제 차량의 비상등 작동을 모사하여 구현
   ii) 자율주행 모드 활성화 시 좌우 LED를 0.5초 간격으로 동기화하여 점멸
   iii) 멀티스레딩을 통해 주행에 영향을 주지 않고 독립적으로 동작

2) 경적 시스템
    i) 장애물 감지 시 자동으로 작동하는 안전 경고 시스템
    ii) PWM 제어를 통한 659.25Hz 주파수의 경고음 출력
    iii) 0.1초 간격으로 두 번 반복하여 경고음 생성
3) 음악 재생 기능
    i) Coldplay의 'Viva la Vida'를 8비트 변환하여 구현
    ii) PWM 제어를 통한 다양한 음계 출력
    iii) 독립 스레드에서 실행되어 주행에 영향을 주지 않음
4) 블루투스 통신
    i) HM-10 모듈을 통한 9600bps 속도의 안정적인 통신
    ii) 자율주행/객체인식/음악재생의 ON/OFF 제어
    iii) 프로그램 종료 등 시스템 제어 기능 구현

속도 제어의 경우, 다양한 기능의 동시 처리로 인한 연산 부하를 고려하여 직진 시 40, 회전 시 20으로 제한하였습니다. 이는 안정적인 주행과 시스템 성능의 균형을 고려한 설정입니다.

각 부가 기능은 멀티스레딩으로 구현하여 메인 주행 기능과 독립적으로 동작하도록 했습니다. 이는 시스템의 반응성을 높이고 각 기능 간의 간섭을 최소화하기 위한 설계 선택이었습니다.

2.4.4. 종합 평가

모든 테스트를 종합적으로 평가한 결과, 전체 시스템은 한 차례의 오류도 없이 매우 높은 안정성을 보였습니다. 특히 자율주행의 핵심 기능인 차선 인식과 객체 인식이 안정적으로 작동했으며, 다양한 부가 기능들도 성공적으로 통합되어 작동했습니다. 다만 객체 인식 속도 부분에서 개선의 여지가 있음을 확인했습니다.

3. 결론

3.1. 프로젝트 요약

본 프로젝트에서는 카메라 기반의 딥러닝을 활용한 자율주행차를 구현했습니다. 먼저 라즈베리파이와 모터, 카메라 등의 하드웨어를 조립하여 자율주행차 플랫폼을 제작했습니다. 그 후 본격적인 자율주행 시스템 개발을 위해 다음과 같은 단계로 프로젝트를 진행했습니다.

1) 자율주행을 위한 데이터 수집 시스템 구축
    i) 차량을 수동으로 조종하며 전진, 좌회전, 우회전 키 입력 시마다 해당 순간의 카메라 이미지가 자동으로 저장되도록 했습니다. 이때 파일명에 조향각 정보를 포함시켜 데이터 레이블링을 자동화했습니다.
2) 수집된 데이터를 바탕으로 CNN 기반 자율주행 모델 학습
    i) 이 모델은 카메라 이미지를 입력받아 적절한 조향각을 예측하도록 설계되었으며, 전진/좌회전/우회전 세 가지 클래스를 분류하는 방식으로 구현했습니다. Backend AI 클라우드 환경에서 학습을 진행했고, 모델의 성능을 검증하며 최적화 작업을 수행했습니다.
3) OpenCV DNN을 활용한 실시간 물체 감지 기능추가
    i) SSD MobileNet 모델을 사용하여 주행 중 전방의 장애물을 감지할 수 있도록 했으며, 쓰레드 기반 병렬 처리를 통해 자율주행과 물체 감지가 동시에 원활하게 동작하도록 구현했습니다. 특히 물체가 감지되었을 때 긴급 정지하는 안전 기능을 추가하여 실제 주행 환경에서의 활용성을 높였습니다.
4) 사용자 편의성을 높이기 위한 다양한 부가 기능 구현
    i) 블루투스 통신을 통한 원격 제어 기능을 추가했으며, 자율주행 모드 전환 시 비상등이 자동으로 작동하도록 했습니다. 또한 장애물 감지 시 경고음을 발생시키는 부저 기능과 Coldplay의 'Viva la Vida'를 8비트 음원으로 변환하여 재생하는 음악 재생 기능도 구현했습니다. 이러한 기능들은 모두 멀티스레딩을 통해 독립적으로 동작하도록 설계되어, 주행 안전성을 해치지 않으면서도 사용자 경험을 향상시킬 수 있었습니다.

최종적으로 개발된 시스템은 정해진 트랙에서 안정적인 자율주행이 가능했으며, 장애물 감지 시 적절히 대응하는 것을 확인했습니다. 본 프로젝트를 통해 딥러닝 기반 자율주행의 기본 원리를 실습하고, 실제 하드웨어에 적용하는 과정에서 발생하는 다양한 문제들을 해결하는 경험을 쌓을 수 있었습니다.

3.2. 한계점 및 개선 방안

본 프로젝트에서 구현한 자율주행 시스템은 다음과 같은 한계점을 가지고 있습니다.

첫째, 데이터 수집이 제한된 환경에서 이루어져 다양한 주행 상황에 대한 대응력이 부족합니다. 특히 조명 조건이 변하거나 예상치 못한 장애물이 나타나는 경우 안정적인 주행이 어려울 수 있습니다. 이를 개선하기 위해서는 다양한 환경에서의 주행 데이터를 추가로 수집하고, 데이터 증강(augmentation) 기법을 활용하여 모델의 일반화 성능을 향상시킬 필요가 있습니다.

둘째, 현재의 물체 감지 시스템은 처리 속도가 다소 느려 실시간 대응에 제약이 있습니다. 물체 감지에 약 2초의 지연시간이 발생하는데, 이는 실제 주행 환경에서 안전성을 저해할 수 있는 요소입니다. 이를 해결하기 위해서는 실시간 처리에 부담을 주는 연산량을 줄일 필요가 있습니다. 예를 들어, 영상의 해상도를 낮추거나 물체 감지 주기를 조절하는 방법을 적용할 수 있습니다. 또한 프레임 스킵 기법을 도입하여 처리해야 할 데이터량 자체를 줄이는 방법도 고려할 수 있습니다. 이러한

최적화를 통해 현재의 하드웨어 구성에서도 충분히 실시간성을 확보할 수 있을 것으로 기대됩니다. 또한 현재의 라즈베리파이보다 처리 성능이 뛰어난 하드웨어로 업그레이드하는 것도 하나의 방안이 될 수 있습니다.

셋째, 차선 인식과 주행 경로 계획이 단순한 형태로 구현되어 있어 복잡한 주행 환경에서의 적응력이 부족합니다. 현재 구현한 프로젝트는 자율주행이라기보다, 라인트레이싱에 더 가깝습니다. 이를 개선하기 위해 더 정교한 경로 계획 알고리즘을 도입하고, 차선 변경이나 교차로 통과와 같은 고급 주행 기능을 추가해볼 수 있습니다.

3.3. 고찰

본 프로젝트를 통해 자율주행의 핵심 기술들을 실제로 구현해보며, 다양하고 의미 있는 경험을 할 수 있었습니다. 먼저 딥러닝 모델의 성능이 데이터의 품질과 분포에 크게 좌우된다는 점을 직접 체감했습니다. 초기에는 단순히 많은 양의 데이터를 수집하는 것이 중요하다고 생각했으나, 실제로는 그렇지 않았습니다. 오히려 다양한 상황에 대한 데이터의 균형 있는 분포가 더욱 중요한 요소였습니다. 이러한 경험은 실제 자율주행 시스템 개발에서 데이터 수집 전략의 중요성을 깨닫게 해준 좋은 계기가 되었습니다.

다음으로, 실시간 처리의 중요성과 그 구현의 어려움을 경험했습니다. 카메라를 통한 영상 처리, 객체 인식, 주행 제어 등 여러 기능이 동시에 실행되면서 시스템의 반응 속도가 현저히 저하되는 문제가 발생했습니다. 이를 해결하기 위해 멀티스레딩을 도입하고 처리 주기를 조절하는 등 다양한 최적화 작업을 진행했습니다. 비록 완벽한 해결책을 찾지는 못했지만, 이러한 시도들을 통해 실제 자율주행 시스템 개발에서 고려해야 할 중요한 요소들을 파악할 수 있었습니다.

또한, 하드웨어의 제약을 극복하는 과정에서 많은 것을 배웠습니다. 라즈베리파이의 제한된 연산 능력 안에서 최적의 성능을 끌어내기 위해 다양한 시도를 했고, 이 과정에서 하드웨어와 소프트웨어의 균형 잡힌 설계가 얼마나 중요한지 깨달았습니다. 특히 자율주행 시스템의 안전성 확보가 얼마나 중요하고 어려운 문제인지 실감했습니다. 단순히 객체를 인식하고 정지하는 기능을 구현하는 데에도 수많은 고려사항이 있었는데, 이는 실제 도로에서의 자율주행이 얼마나 복잡한 문제인지를 잘 보여주는 예시였습니다.

무엇보다도 이번 강의를 통해 얻은 가장 큰 수확은 Linux 운영체제에 대한 이해도를 높일 수 있었다는 점입니다. 이전에는 Linux에 대해 크게 관심이 없었고 실무에서의 활용도도 잘 알지 못했습니다. 하지만 프로젝트를 진행하며 다양한 Linux 명령어를 직접 사용해보고, 그 유용성을 체감할 수 있었습니다. 또한 Git을 처음 사용해보는 과정을 통해 실제 개발 현장에서 사용되는 도구들을 경험해볼 수 있어 매우 유익했습니다.

이번 프로젝트는 비록 소규모였지만, 자율주행 기술의 기본 원리를 이해하고 실제 구현 경험을 쌓는 데 매우 의미 있는 기회였습니다. 더불어 실무에서 활용할 수 있는 다양한 개발 도구와 환경을 경험해볼 수 있어 더욱 값진 시간이었습니다.