임베디드시스템설계 자율주행차 제작 보고서

2016920029 유시온
2016920056 최문기

1. 아두이노

아두이노 프로그램에 사용되는 주요 함수는 다음과 같습니다.

1-1. set_motor_ctrl(unsigned char cmd, int s)

```
void set_motor_ctrl(unsigned char cmd, int s) {
// direction
  // 0: forward, 1: reverse
  if (cmd == 'L') {
    // set direction
    m_a_dir = 1;  // reverse
    m_b_dir = 0;  // forward
    // set speed
    m_a_spd = s;
    m_b_spd = s;
  }
  else if (cmd == 'R') {
    // set direction
    m_a_dir = 0;
    m_b_dir = 1;
    // set speed
    m_a_spd = s;
    m_b_spd = s;
  }
  else if (cmd == 'F') {
    // set direction
    m_a_dir = 0;
    m_b_dir = 0;
    // set speed
    m_a_spd = s;
    m_b_spd = s;
  }
  else if (cmd == 'X') {
    m_a_dir = 0;
    m_b_dir = 0;
    m_a_spd = 0;
    m_b_spd = 0;
  }
}
```

set_motor_ctrl은 명령을 해석해서 모터 방향과 속도를 결정하는 함수입니다. 매개변수로 오는 cmd에 따라 모터의 방향을 결정하고 s에 따라 모터의 회전 속도를 저장합니다. 전역변수로 선언되어 있는 모터 방향과 속도 변수를 수정합니다.

1-2. motor_drive()

```
void motor_drive()
{
  if(m_b_dir == 1)
  {
    digitalWrite(MOTOR_B_a, LOW);     //motorB+ LOW
    analogWrite(MOTOR_B_b, m_b_spd);  //motorB- spd
  }
  else
  {
    analogWrite(MOTOR_B_a, m_b_spd);  //motorB+ spd
    digitalWrite(MOTOR_B_b, LOW);     //motorB- LOW
  }
  if(m_a_dir == 0)
  {
    digitalWrite(MOTOR_A_a, LOW);     //motorA+ LOW
    analogWrite(MOTOR_A_b, m_a_spd);  //motorA- spd
  }
  else
  {
    analogWrite(MOTOR_A_a, m_a_spd);  //motorA+ spd
    digitalWrite(MOTOR_A_b, LOW);     //motorA- LOW
  }

}
```

motor_drive는 전역변수로 저장되어 있는 모터의 방향과 속도를 확인하여 적절한 핀에 신호를 보내주는 역할을 합니다. 모터의 모든 핀은 PWM을 지원하는 번호로 연결되어 있으므로 analogWrite를 이용해 0~255까지의 값을 줄 수 있습니다.

1-3. loop()

```
void loop() {
  DataToRead[5] = '\n';
  Serial.readBytesUntil(char(13), DataToRead, 5);

  cmd = DataToRead[0];
  Speed = "";
  for (i = 1; (DataToRead[i] != '\n') && (i < 6); i++) {
    Speed += DataToRead[i];
  }
  s = Speed.toInt();
  set_motor_ctrl(cmd, s);
  motor_drive();
}
```

loop 함수는 Serial 입력으로 들어오는 명령을 해석하여 set_motor_ctrl, motor_drive를 호출합니다.

2. 라즈베리파이

라즈베리파이 프로그램에 사용되는 주요 코드는 다음과 같습니다.

2-1. self_driving.py

drive()를 무한루프하여 실행합니다. drive()에서는 이미지를 한 장 촬영한 후 DNN_Driver에서 predict한 후에 그 값을 기준으로 좌회전, 직진, 우회전을 결정합니다.

```
# Copyright Reserved (2020).
# Donghee Lee, Univ. of Seoul
#
__author__ = 'will'

from rc_car_interface import RC_Car_Interface
from tf_learn import DNN_Driver
import numpy as np
import time
import cv2
import serial
ser = serial.Serial('/dev/serial/by-id/usb-Arduino_Srl_Arduino_Uno_75439313737351415 2F0-if00', 9600)

class SelfDriving:

    def __init__(self):
        self.rc_car_cntl = RC_Car_Interface()   # 인터페이스
        self.dnn_driver = DNN_Driver()          # 신경망
        self.direction = 0                      # 방향
        self.dnn_driver.tf_learn()              # 학습

    def rc_car_control(self, direction):
        cmd = ''
        if (-0.5 < direction < 0.5):
            cmd = "F200"
        elif (0.5 <= direction):
            cmd = "R255"
        elif (direction <= -0.5):
            cmd = "L255"
        cmd = cmd + '\n'
        ser.write(cmd.encode('ascii'))

    def rc_car_stop(self):
        cmd = 'F000' + '\n'
        ser.write(cmd.encode('ascii'))

    def drive(self):
        while True:
            img = self.rc_car_cntl.get_image_from_camera()
            self.direction = self.dnn_driver.predict_direction(img)      # -1 or 0 or 1
            self.rc_car_control(self.direction)
            if self.direction != 0:
                time.sleep(0.1)
            else:
                time.sleep(0.2)
            self.rc_car_stop()
            time.sleep(0.03)

if __name__ == "__main__":
    SelfDriving().drive()
```

2-2. tf_learn.py

이미지를 보고 방향을 예측하는 인공신경망 model 입니다. tf_learn을 실행하면 **CNN 분류 모델**을 생성하고 데이터를 이용하여 fit합니다. predict_direction(img)을 실행하면 예측값을 return합니다. 예측값을 좌회전(-1) 또는 직진(0) 또는 우회전(1)입니다.

```
# Copyright Reserved (2020).
# Donghee Lee, Univ. of Seoul
#
__author__ = 'will'

from keras.models import Sequential
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Dropout
from keras.layers import Flatten
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf

outputs = 1

from get_image_data import *

class DNN_Driver():
    def __init__(self):
        self.trX = None
        self.trY = None
        self.teX = None
        self.teY = None
        self.model = None
```

```
    def tf_learn(self):
        self.trX, self.trY = get_training_data()
        self.teX, self.teY = get_test_data()

        seed = 0
        np.random.seed(seed)
        tf.random.set_seed(seed)

        # Hyperparameters
        nb_classes = 3  # 클래스 수 결정
        nb_epochs = 15  # epoch 결정

        # channel 차원 추가
        self.trX = self.trX.reshape(self.trX.shape[0], 16, 16, 1)
        self.teX = self.teX.reshape(self.teX.shape[0], 16, 16, 1)

        # One-hot encoding
        enc = OneHotEncoder()
        enc.fit(self.trY.reshape(-1, 1))
        trY_onehot = enc.transform(self.trY.reshape(-1, 1)).toarray()
        teY_onehot = enc.transform(self.teY.reshape(-1, 1)).toarray()

        # build model
        self.model = Sequential()
        self.model.add(Conv2D(32, (3, 3), padding='same', input_shape=self.trX.shape[1:],
activation='tanh'))
        self.model.add(MaxPooling2D(pool_size=(2, 2)))
        self.model.add(Dropout(0.25))

        self.model.add(Conv2D(64, (3, 3), padding='same', activation='tanh'))
        self.model.add(MaxPooling2D(pool_size=(2, 2)))
        self.model.add(Dropout(0.25))

        self.model.add(Flatten())
        self.model.add(Dense(1024, activation='relu'))
        self.model.add(Dropout(0.5))
        self.model.add(Dense(nb_classes, activation='softmax'))

        self.model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
['accuracy'])

        # learning
        self.model.fit(self.trX, trY_onehot, epochs=nb_epochs, batch_size=1)
        print('=' * 100)
        return

    def predict_direction(self, img):
        img = img.reshape(1, 16, 16, 1)
        ret = self.model.predict(img)
        ret = np.argmax(ret[0]) - 1
        return ret
```

2-3. rc_car_interface.py

get_image_from_camera()를 통해서 picamera를 이용한 사진을 찍습니다. 더 빠른 촬영을 위해서 PiVideoStream 객체에서 이미지를 불러왔으며(뒤에서 설명) 또한 RC카의 이동은 더 빠른 반응을 위해서 self_driving.py에 바로 넣고 rc_car_interface.py에는 넣지 않았습니다.

```
# Copyright(c) Reserved 2020.
# Donghee Lee, University of Soul
#
__author__ = 'will'

import numpy as np
import cv2
from picameraStream import PiVideoStream

class RC_Car_Interface():

    def __init__(self):
        self.pividео = PiVideoStream()
        self.pividео.start()

    def get_image_from_camera(self):
        img = self.pividео.read()
        img = img[:,:,0] # 흑백
        threshold = int(np.mean(img))*0.5
        # Invert black and white with threshold
        ret, img2 = cv2.threshold(img.astype(np.uint8), threshold, 255,
cv2.THRESH_BINARY_INV)
        img2 = cv2.resize(img2,(16,16), interpolation=cv2.INTER_AREA )
        return img2
```

2-4. get_image_data.py

데이터를 불러오고 학습 데이터와 테스트 데이터로 나눠주는 코드입니다. **데이터는 모두 직접 라즈베리파이 카메라로 찍어서 모두 직접 라벨링했습니다.(516장)**

```
__author__ = 'will'

import pickle
import numpy as np

data = pickle.load( open( "trainingfinal516.p", "rb" ), encoding="latin1" )
n_images = len(data)
test, training = data[0:int(n_images/3)], data[int(n_images/3):]

def get_training_data():
    trX = np.array([a[2] for a in training], dtype=np.float64)
    trY = np.zeros((len(training)),dtype=np.float)
    for i, data in enumerate(training):
        trY[i] = float(data[0])
    return trX, trY

def get_test_data():
    teX = np.array([a[2] for a in test], dtype=np.float64)
    teY = np.zeros((len(test)),dtype=np.float)
    for i, data in enumerate(test):
        teY[i] = float(data[0])
    return teX,teY
```

2-5. picameraStream.py

더 효율적으로 picamera를 이용하는 소스코드입니다. 이 소스코드의 함수를 이용하면 다음 프레임 나올 때까지 block돼서 cpu 손실이 없다고 합니다. 다음 링크로 소스코드를 대체합니다.
https://github.com/jrosebr1/imutils/blob/master/imutils/video/pivideostream.py

3. 드라이빙 베이스 제작

다음과 같은 과정을 순서대로 거쳐서 드라이빙 베이스(주행 자동차)가 완성되었습니다.

1. 먼저 모터를 사용하여 주행하면 제대로 직진하지 못하는 문제가 있었습니다. 여러 시행착오를 겪어 관찰한 직진하지 못하는 문제는 세가지입니다.

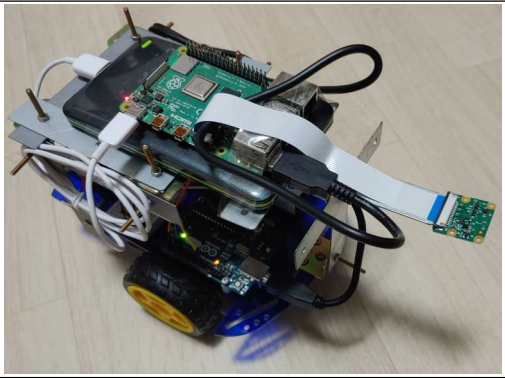
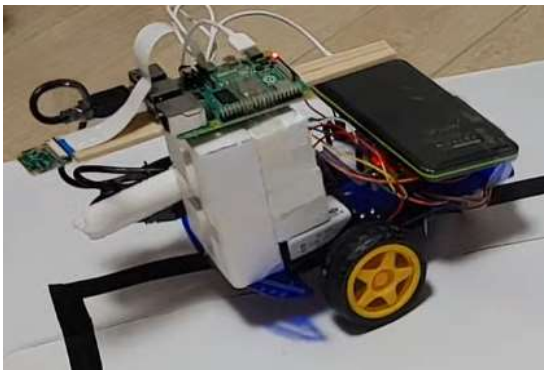
A. 모터의 출력이 양쪽이 다르다.
- 이는 교수님으로부터 모터를 교체받고 납땜하여 장착하니 해결되었습니다.

B. 바퀴가 바깥 고무와 안쪽 플라스틱 사이에서 겉돈다.
- 이 또한 교수님으로부터 바퀴도 같이 받아서 해결되었습니다.

C. 뒷 바퀴가 꺾여있으면 직진하지 못하거나 커브를 돈다.
- 이전에 쓰던 뒷 바퀴가 윤활제를 넣어도 회전력이 약해서 이것 또한 교체했습니다. 하지만 바퀴가 꺾여있을 때 직진하지 않고 약간 커브를 도는 것은 해결이 되지 못했습니다. 모터의 힘이 약합니다.

2. 그리고 철물점에서 라즈베리파이와 카메라를 고정할 재료를 사서 프레임을 제작했습니다.

3. 하지만 직접 주행하려고 보니 최대 출력을 주어도 모터의 힘이 약해서 종종 움직이지 못하는 문제가 발생했습니다. 무게을 줄이기 위해서 다시 프레임을 모두 제거하고 스티로폼을 이용했습니다.

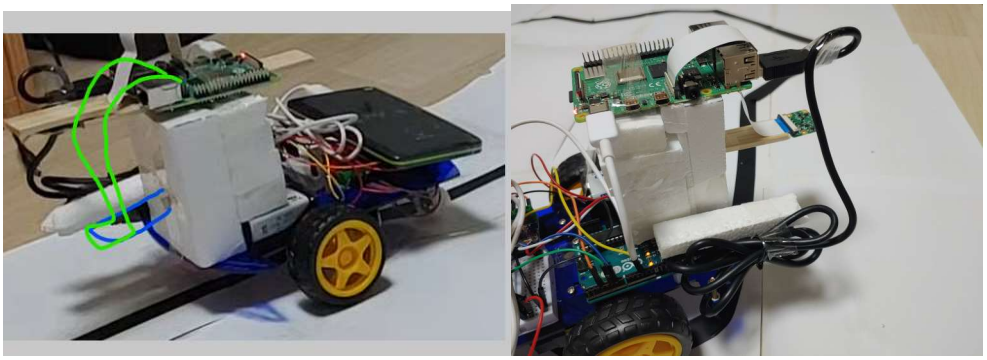
	
이전 드라이빙 베이스	개선 드라이빙 베이스

4. 모터의 힘이 약하다보니 최소 주행속도가 너무 빨랐습니다. 트랙을 너무 쉽게 벗어나기 때문에 딜레이 후 잠시 정지하는 코드를 추가했습니다. 결과적으로 속도 조절에는 성공하였습니다.

이전 코드	개선 코드

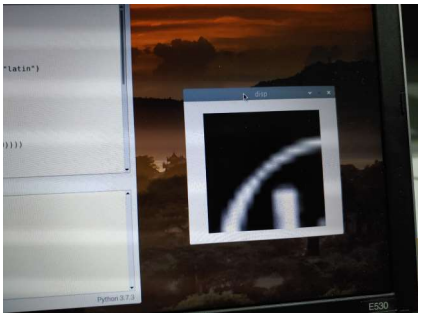
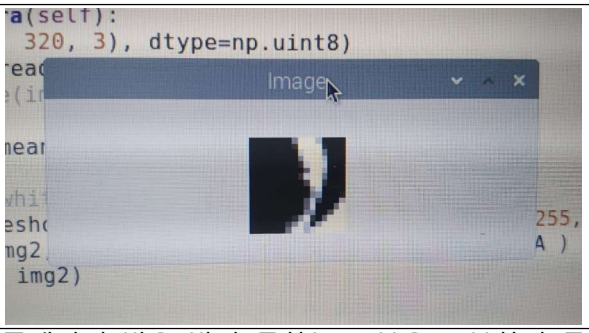
`self.rc_car_control(self.direction)`
`time.sleep(0.001)`
	`self.rc_car_control(self.direction)`
	`if self.direction != 0:`
	`  time.sleep(0.1)`
	`else:`
	`  time.sleep(0.2)`
	`self.rc_car_stop()`
	`time.sleep(0.03)`

5. 바퀴가 너무 트랙을 벗어나는 문제가 있었습니다. 라인 트레이서인데 커브에서는 라인 안에서보다는 라인 밖에서 주행하는 경우가 많아 카메라의 촬영 위치를 조정했습니다.



▲ 카메라 위치 조정

6. 카메라를 조정하니 그림자가 문제였습니다. 밝기의 평균값을 threshold로 하다보니 그림자와 트랙을 구분하지 못하는 문제가 발생했습니다. 그래서 플래시를 옆에 부착했지만 이번에는 너무 밝아서 플래시의 빛을 받지 못하는 부분과 트랙을 구분하지 못했습니다.

	
▲그림자를 구분하지 못함	▲플래시의 빛을 받지 못하는 부분을 구분하지 못함

7. 최종적으로 플래시를 비추는 것보다 그림자를 균일하게 덮으면 트랙을 제대로 구분할 수 있을 것이라 판단했으며 실제로 트랙을 잘 구분해내었습니다. 최종 드라이빙 베이스는 위 모델에서 A4용지로 빛을 가린 모델입니다.

▲ A4용지로 덮은 후 촬영한 트랙 이미지