임베디드 시스템 설계 프로젝트 보고서
(Embedded Systems Design)

자율주행 자동차 (카메라를 이용한 라인 트레이싱)

(Self-driving car (line tracking using camera))

과목	임베디드 시스템 설계 (Embedded Systems Design)
교수	컴퓨터과학부 이동희 교수님
분반	02
팀원 인적사항	컴퓨터과학부 2016920005 김경석 통계학과 2018580001 강창구

자율주행 자동차 (카메라를 이용한 라인 트레이싱)

(Self-driving car (line tracking using camera))

목차 (Contents)

1. 하드웨어 (Hardware)
2. 모터 제어 (Motor Control)
3. 이미지 처리 (Image Processing)
4. 하드웨어 인터페이스 (Hardware Interface)
5. 방향 (Direction)
6. 방향 결정 (Direction Determination)
    - 6.1 방향 결정 알고리즘 (Direction Determination Algorithm)
        - 6.1.1 도로 직선 탐색 알고리즘 (Lane Line Detecting Algorithm)
        - 6.1.2 벡터 계산 알고리즘 (Vector Calculating Algorithm)
        - 6.1.3 위상 차 보정 알고리즘 (Phase Difference Correction Algorithm)
        - 6.1.4 제한사항 (Restrictions)
    - 6.2 인공지능을 이용한 방향 결정 (Direction Determination with AI)
        - 6.2.1 학습 (Training)
        - 6.2.2 모델 탑재 (Model Mounting)
        - 6.2.3 성능 (Performance)
7. 부록 – 코드 (Appendix - Code)

1. 하드웨어 (Hardware)

Hardware	Quantity	Function
Raspberry Pi 4 Model B (Micro Computer)	1	Processing the Line Image and Computing next direction
Raspberry Pi Camera V2 (Camera)	1	Shooting the line Image
Arduino Uno R3 DIP Edition (Micro Controller)	1	Controlling the Motor Driver
L9110 (or L9110s) (Motor Driver)	1	Controlling Two DC Motors
Switch	1	-
Main Battery (Power for Raspberry Pi)	1	-
Sub Battery (Power for Motor Driver)	1	-
Bread Board	1	-
DC Motor	2	-
Jumper Cable	n	-

Figure 1.1

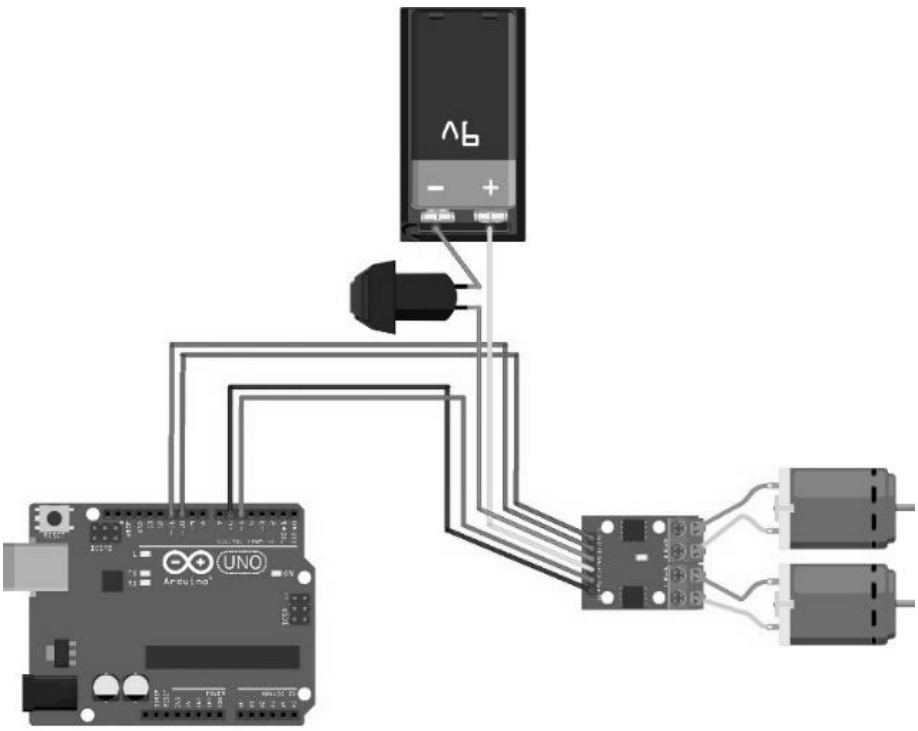


Figure 1.2

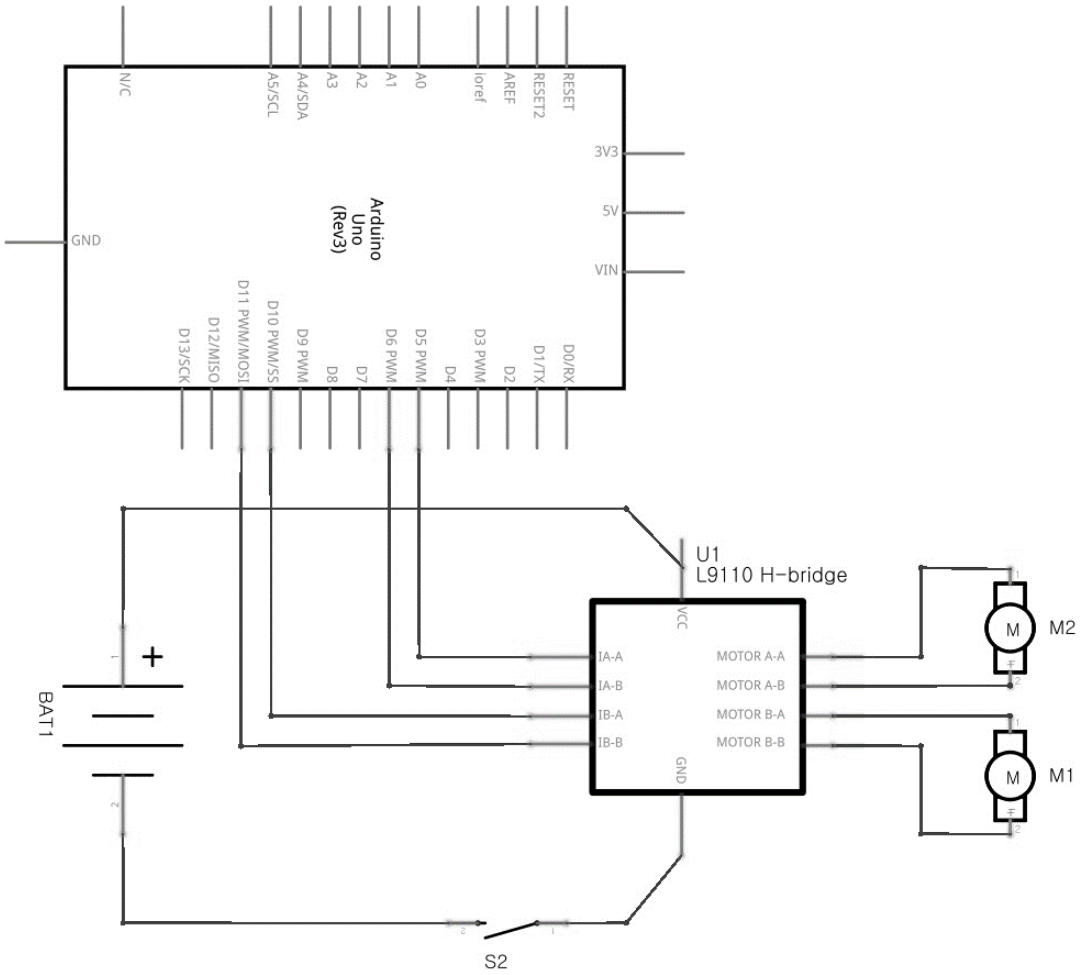


Figure 1.3

Figure 1.3는 아두이노 우노 – 모터드라이버 – DC 모터 시스템의 회로도이다. 아두이노 우노는 라즈베리 파이 4보드에 USB2.0 커넥터로 연결된다. 이 연결을 통해 아두이노 우노와 라즈베리 파이 4는 직렬 통신을 할 수 있다.

2. 모터 제어 (Motor Control)

아두이노 보드가 직접 모터의 방향과 속도를 제어하기는 어려우므로 모터드라이버를 사용하여 이를 아두이노 우노 보드 – DC 모터간 인터페이스로 활용한다. 이 시스템에는 제어 가능한 모터가 오직 두개만 존재한다. 따라서 두 바퀴의 속도를 조절하여 방향제어를 한다. 이를 위해서는 마이크로 컨트롤러 유닛인 아두이노 우노 보드와 모터드라이버가 두 DC 모터의 속도를 별도로 제어할 수 있어야 한다.

```
String speed1, speed2, delay_time;
char  sign1, sign2;
int  i, p1, p2;
int s1, s2, d;
int motorA1 = 5;
int motorA2 = 6;
int motorB1 = 9;
int motorB2 = 10;

byte DataToRead[16];

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.flush();
  pinMode(motorA1, OUTPUT);
  pinMode(motorA2, OUTPUT);
  pinMode(motorB1, OUTPUT);
  pinMode(motorB2, OUTPUT);
}

void loop() {
  for (i = 0; i < 16; i++) {
    DataToRead[i] = char(10);
  }
  if(Serial.available() > 0){
    Serial.readBytesUntil(char(10), DataToRead, 16);
    DataToRead[15] = char(10);
  }

  /* For Debugging, send string to RPi */
  /*
  for (i = 0; i < 16; i++) {
    Serial.write(DataToRead[i]);
    if (DataToRead[i] == char(10)) break;
  }
  */
  /* End of Debugging */

  for (i = 0; i < 16; i++) {
    if (DataToRead[i] == 'L') {
      p1 = i;
      break;
    }
  }

  for (i = 0; i < 16; i++) {
    if (DataToRead[i] == 'T') {
      p2 = i;
      break;
    }
  }

  speed1 = "";
  speed2 = "";

  if (DataToRead[1] == '-') {
    sign1 = '-';
    for (i = 2; (DataToRead[i] != 'L') && (i < p1); i++) {
      speed1 += char(DataToRead[i]);
    }
  }
  else {
    sign1 = '+';
    for (i = 1; (DataToRead[i] != 'L') && (i < p1); i++) {
      speed1 += char(DataToRead[i]);
    }
  }

  if (DataToRead[p1 + 1] == '-') {
    sign2 = '-';
    for (i = p1 + 2; (DataToRead[i] != char(10)) && (i < p2); i++) {
      speed2 += char(DataToRead[i]);
```

```
    }
  }
  else {
    sign2 = '+';
    for (i = p1 + 1; (DataToRead[i] != char(10)) && (i < p2); i++) {
      speed2 += char(DataToRead[i]);
    }
  }

  delay_time = "";

  for (i = p2 + 1; (DataToRead[i] != char(10)) && (i < 16); i++) {
    delay_time += char(DataToRead[i]);
  }

  s1 = speed1.toInt();
  s2 = speed2.toInt();
  d = delay_time.toInt();

  if (sign1 == '+' && sign2 == '+') {
    analogWrite(motorA1, s1);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, s2);
    analogWrite(motorB2, 0);
  }
  else if (sign1 == '+' && sign2 == '-') {
    analogWrite(motorA1, s1);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, s2);
  }
  else if (sign1 == '-' && sign2 == '+') {
    analogWrite(motorA1, 0);
    analogWrite(motorA2, s1);
    analogWrite(motorB1, s2);
    analogWrite(motorB2, 0);
  }
  else {  //sign1 == '-' && sign2 == '-'
    analogWrite(motorA1, 0);
    analogWrite(motorA2, s1);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, s2);
  }
  delay(d);

}
```

Code 2.1

Code 2.1은 아두이노 우노 보드가 모터 드라이버를 제어하는 코드이다. 이 코드는 컴파일되어 아두이노 우노 보드에 업로드된다. 아두이노 우노는 라즈베리 파이로부터 메시지를 받아 이를 파싱(parsing)하여 메시지를 해석한 뒤 내용에 맞게 모터드라이버를 제어한다.

아두이노 우노 보드와 라즈베리 파이 간 전달되는 메시지 워드 (message word)의 형식은 Figure 2.1과 같다. M_R과 M_L은 각각 오른쪽 모터와 왼쪽 모터의 회전 방향과 회전 세기를 정한다. 이 문자열을 파싱하여 얻어진 정수의 부호가 회전 방향을 정하며, 그 절댓값이 회전 세기를 정한다. 또한 M_T는 이들이 회전하는 시간을 정한다. 예를 들어 메시지 워드 R200L-150T200은 오른쪽 모터가 정방향 회전을 하도록 200만큼의 전압을 가하고 왼쪽 바퀴가 역방향 회전을 하도록 150만큼의 전압을 가하며 이를 0.2초 동안 지속하라는 메시지이다. 이들의 전압 변화는 아두이노 보드의 PWM(Pulse Width Modulation) 기능을 사용한다.

R	M_R (-255~255)	L	M_L (-255~255)	T	M_T (0~9999)	\n

Figure 2.1

이때 아두이노 우노 보드 특성 상 Code 2.1의 `loop` 함수 한 번을 실행하는 동안은 양측 모터에 걸리는 전압은 거의 일정하게 유지된다. 즉 한 `loop`를 도는 동안 양측 모터의 회전 속도는 거의 일정하게 유지된다. 이렇게 아두이노 보드에서 `loop` 함수 한 번을 돌리는 동안 양측 바퀴속도가 일정하게 유지되는 기간을 한 시프트 텀(shift term) 이라고 부르도록 한다.

Definition 2.1

모터드라이버의 제어를 통해 양측 모터에 일정한 전압이 걸려 모터들의 회전 속도가 일정하게 유지되는 기간을 시프트 텀(shift term)이라 부른다. 이는 Code 2.1의 `loop`함수가 한 번 반복되는 기간과 거의 동일하다.

다만 한 시프트 텀에서도 왼쪽 바퀴와 오른쪽 바퀴의 회전 속도를 별도로 제어할 수 있으므로 기기의 왼쪽 속도와 오른쪽 속도가 다를 수 있다. v_L과 v_R을 각각 기기의 왼쪽 속도와 오른쪽 속도로 표기하고 w를 기기의 너비로 표기하도록 한다. 그러면 기기는 어떤 동심원을 따라 원운동을 하게 된다. r을 그 동심원의 반지름(회전 반경)으로 두면 다음이 성립한다.

Theorem 2.1

v_L과 v_R을 각각 기기의 왼쪽 속도와 오른쪽 속도로 표기하고 w를 기기의 너비로 표기하도록 한다. 그러면 기기는 어떤 동심원을 따라 원운동을 하게 된다. r을 그 동심원의 반지름 (회전 반경)으로 두면 다음이 성립한다.

$$\frac{|v_L - v_R|}{w} = \frac{max\{|v_L|, |v_R|\}}{\left|\frac{w}{2} + r\right|} = \frac{min\{|v_L|, |v_R|\}}{\left|r - \frac{w}{2}\right|}$$

$$\left|\frac{w}{2} + r\right| = \frac{w}{2} + r = \frac{w}{|v_L - v_R|} max\{|v_L|, |v_R|\}$$

$$\therefore r = \frac{w}{|v_L - v_R|} max\{|v_L|, |v_R|\} - \frac{w}{2}$$

$|v_L - v_R|$이 작을수록 회전 반경은 커지고 $|v_L - v_R|$이 클수록 회전반경은 작아진다. 즉, 두 바퀴의 속도차이가 클수록 기기가 더 큰 각도로 회전하고, 두 바퀴의 속도차이가 작을수록 기기가 더 작은 각도로 회전한다. 특히 $v_L - v_R = 0$인 경우, 즉 두 바퀴 속력이 같은 경우 회전 반경은 무한대가 되어 직진 운동을 하게 된다. 또한 동심원의 위치는 좌측 속력이 우측 속력보다 클 때 우측에 위치하고 반대로 우측 속력이 좌측 속력보다 크면 좌측에 위치한다. Figure 2.2와 Figure 2.3은 Theorem 2.1에 나타난 관계를 보여준다.

하지만 하드웨어 특성, 특히 기기의 무게, 각 바퀴에 가해지는 하중, 기기의 무게중심 등에 따라 바퀴의 속력이 달라진다. 이는 실험적으로 얻을 수밖에 없고, 이를 정확히 측정하기 어려우므로 모터드라이버가 모터에 가하는 전압만으로 기기의 왼쪽 속력 또는 오른쪽 속력을 위 방법으로 정확히 계산하기는 무리가 따른다. 따라서 이 시스템은 위 관계를 수치적으로 직접 이용하기보단 관계적으로 이용하도록 한다.

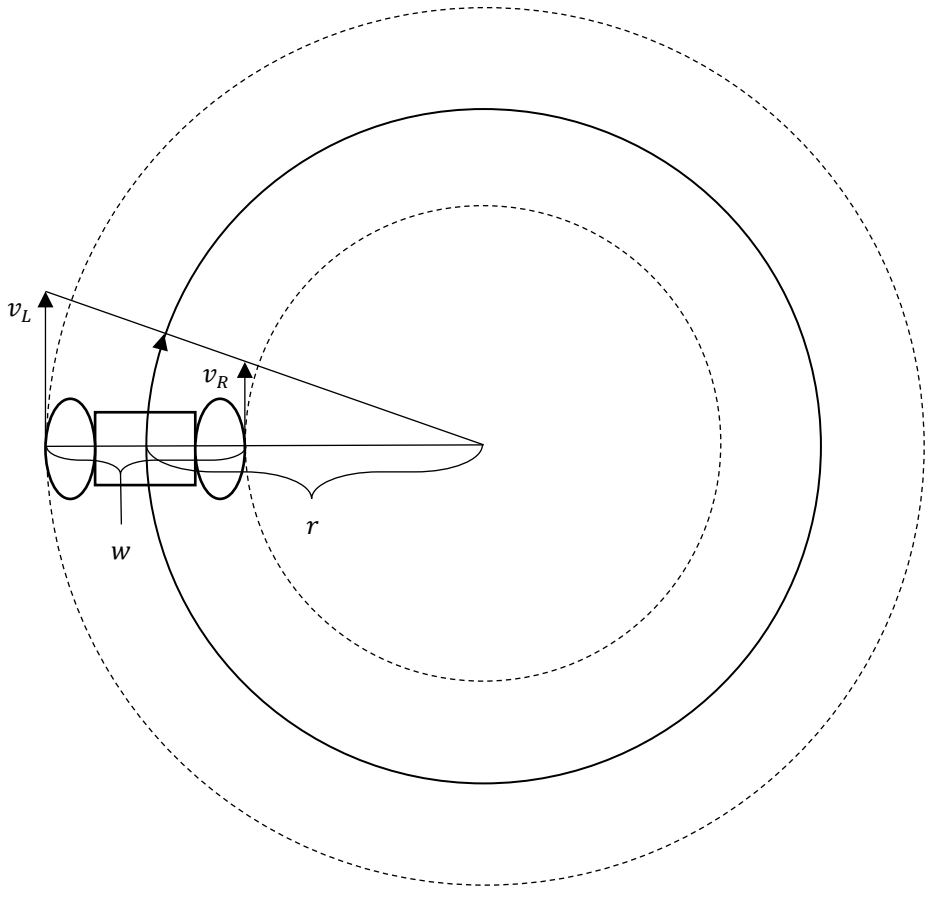


Figure 2.2

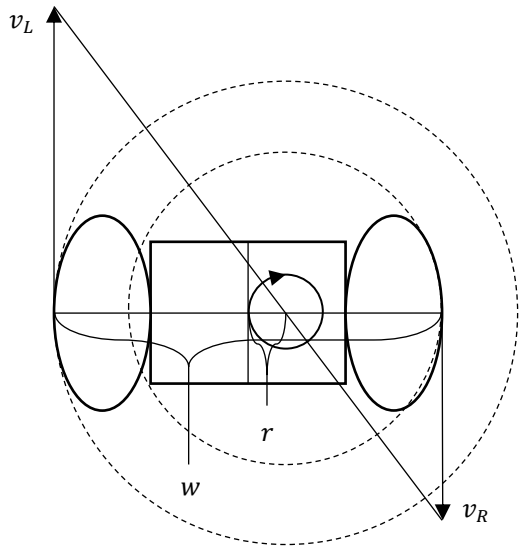


Figure 2.3

3. 이미지 처리 (Image Processing)

이 시스템은 카메라를 통해 도로(Line) 정보를 얻는다. 카메라가 이미지를 촬영하면 이를 처리하여 다음 방향을 계산한다. 이때 이미지로부터 방향을 계산하기 위해서 이미지가 프로그램이 다루기에 적절한 형태로 전처리(processing)되어야 할 필요가 있다.

먼저 이 시스템은 카메라로부터 찍은 이미지객체를 행렬형태로 변환한다. 이 행렬은 RGB값이 따로 저장되어 있는 3차원 행렬이다. 이를 흑백 이미지로 변환하여 2차원 행렬로 만든다. 이 2차원 행렬의 요소는 밝기(brightness)를 나타내게 된다. 이어서 이 요소들의 값을 반전시켜 전처리를 마무리하도록 한다. 이를 이미지 매트릭스(image matrix)라 부른다. 하드웨어 성능을 고려하여 이미지 매트릭스의 크기를 결정한다.

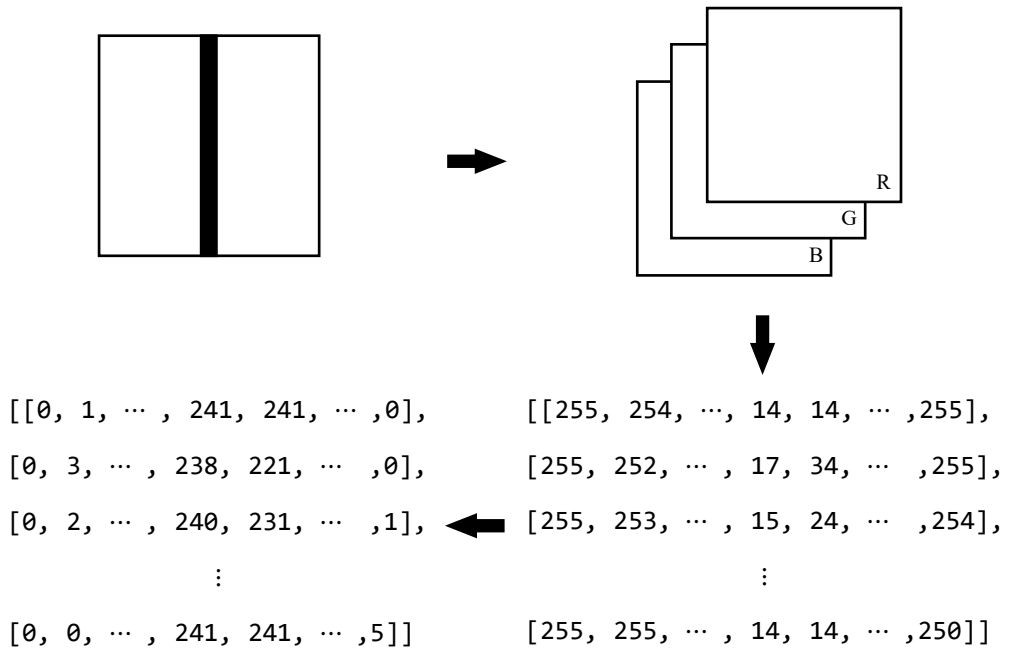


Figure 3.1

4. 하드웨어 인터페이스 (Hardware Interface)

Code 4.1은 라즈베리 파이와 기타 하드웨어 요소(라즈베리 파이 카메라, 아두이노 우노 보드) 간의 인터페이스를 구현한 코드이다. 라즈베리 파이는 직렬 통신(Serial Communication)을 통해 Figure 2.1에 나타난 메시지를 아두이노 우노 보드로 직렬통신을 사용하여 송신한다. `set_wheel_speed`, `set_right_speed`, `set_left_speed`, `stop`등의 함수가 이를 담당한다.

또한 `get_image_from_camera`는 내장된 `PiCamera` 모듈과 `PiRGBArray` 모듈을 통해 카메라가 찍은 이미지를 전처리하여 반환하는 함수이다.

```
# Copyright(c) Reserved 2020.
# Donghee Lee, University of Soul

__author__ = 'will'

import numpy as np
import cv2
import serial
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

class RC_Car_Interface:
    def __init__(self,ser):
        self.ser = ser
        self.left_wheel = 0
        self.right_wheel = 0
        self.camera = PiCamera()
        self.camera.resolution = (320,320)      ## set camera resolution to (320, 320)
        self.camera.color_effects = (128,128)   ## set camera to black and white

    def set_wheel_speed(self, right_speed, left_speed, delay_time):
        print('set right speed to ', right_speed)
        print('set left speed to ', left_speed)
        cmd = ("R%dL%dT%d\n" % (right_speed, left_speed, delay_time)).encode('ascii')
        print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)

    def set_right_speed(self, speed, delay_time):
        print('set right speed to ', speed)
        cmd = ("R%dL%dT%d\n" % (speed, 0, delay_time)).encode('ascii')
        print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def set_left_speed(self, speed, delay_time):
        print('set left speed to ', speed)
        cmd = ("R%dL%dT%d\n" % (0, speed, delay_time)).encode('ascii')
        print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def stop(self, delay_time):     # robot stop
        print('stop')
        cmd = ("R%dL%dT%d\n" % (0, 0, delay_time)).encode('ascii')
        print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def get_image_from_camera(self, image_size):
        img = np.empty((320, 320, 3), dtype=np.uint8)
        self.camera.capture(img, 'bgr')

        # 3 dimensions have the same value because camera is set to black and white
        # remove two dimension data
        img = img[:,:,0]

        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,(image_size, image_size), interpolation=cv2.INTER_AREA )
        # cv2.imshow("Image", img2)
        # cv2.waitKey(0)
        return img2
```

Code 4.1

5. 방향 (Direction)

전처리 된 이미지 매트릭스로부터 라즈베리 파이 4는 다음 방향을 계산하여 이로부터 오른쪽 바퀴의 속도와 왼쪽 바퀴의 속도를 경정하여야 한다. 방향은 기기를 중심으로 동경 (radius vector) 기준 0부터 2π 사이의 각도로 나타낼 수 있지만 이 시스템은 직진만을 고려하려 동경기준 0부터 π 사이의 각도만을 고려하도록 한다. 따라서 Figure 5.1과 같이 기기의 정면 방향을 가리키는 벡터, 즉, 이전 시프트 텀의 진행 방향을 기준으로 다음 시프트 텀의 진행 방향 θ를 $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$사이의 값으로 나타낼 수 있다. 이 시스템에선 편의성을 고려하여 $\left[-\frac{\pi}{2},\frac{\pi}{2}\right]$사이의 값을 $[-1,1]$사이의 값으로 선형 변환(linear transformation)하여 사용하도록 한다. 또는 동경을 기준으로 하여 계산된 각도 $[0,\pi]$사이의 값을 선형 변환하여 사용하도록 한다. 여기서는 $[-1,1]$사이의 값을 가지는 각도 값을 φ로 나타낸다.

$$f\colon \left[-\frac{\pi}{2},\frac{\pi}{2}\right] \to [1,-1], \qquad f(\theta)=\frac{2\theta}{\pi}, \qquad g\colon [0,\pi] \to [1,-1], \qquad g(\theta)=-\frac{2\theta}{\pi}+1$$

이 각도는 위와 같은 형태 만이 아니라 벡터 형태로도 나타낼 수 있으며 이를 텀-방향 벡터 (term-direction vector)라 칭한다. 즉, 텀-방향 벡터는 기기가 나아가야할 방향을 제시한다.

Definition 5.1

텀-방향 벡터(term-direction vector)는 기기가 나아가야할 방향을 나타내는 벡터이다. 이는 시프트 텀과 시프트 텀 사이에 오직 하나만이 존재한다.

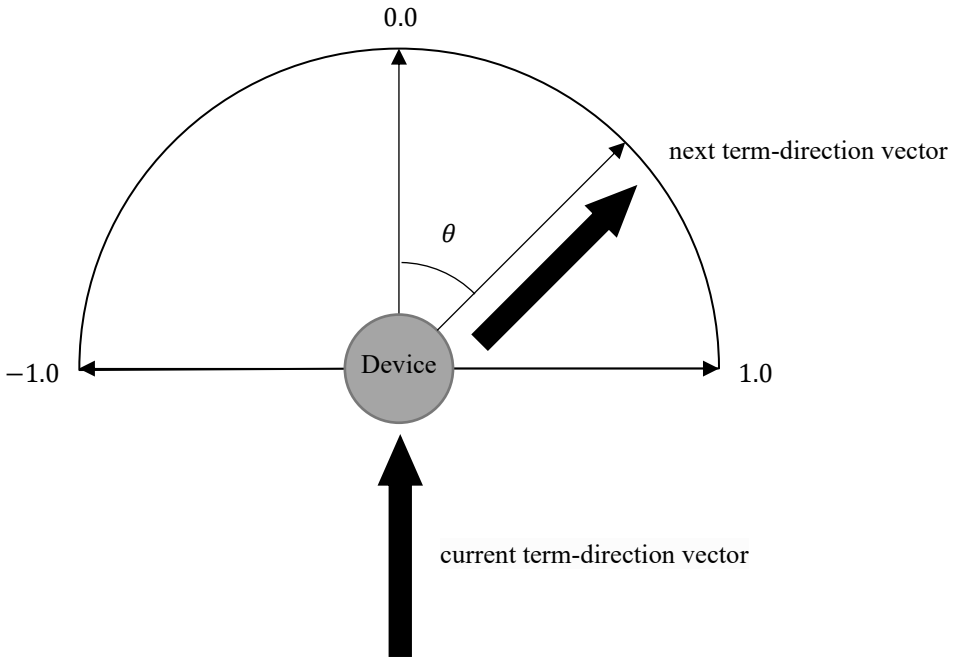


Figure 5.1

일단 $[-1, 1]$ 사이의 각도 값 φ가 계산되면 이로부터 기기가 해당 각도에 맞춰서 이동하도록 양측 바퀴의 속도를 결정해주어야 한다. 하드웨어 특성, 특히 기기의 무게, 각 바퀴에 가해지는 하중, 기기의 무게중심 등에 따라 바퀴의 속력이 달라지므로 이를 직접 계산하는 것은 어렵다. 따라서 이 시스템은 Code 5.1과 같은 알고리즘을 사용한다. Code 5.1의 `rc_car_control` 함수는 일반적인 자동차에서 핸들과 같은 역할을 하게 된다.

```
def rc_car_control(self, direction):
        sensitivity = 150
        left_power = 0.95
        right_power = 0.95

        ## calculate left and right wheel speed with direction
        if direction < -1.0:
            direction = -1.0
        if direction > 1.0:
            direction = 1.0

        print("direction: ", direction)
        if (direction <= 0.0):
            right_speed = int(((255-sensitivity) + int((1.0)*sensitivity))*right_power)
            left_speed = int(((255-sensitivity) + int((1.0 + direction)*sensitivity))*left_power)
        else:
            right_speed = int(((255-sensitivity) + int((1.0 - direction)*sensitivity))*right_power)
            left_speed = int(((255-sensitivity) + int((1.0)*sensitivity))*left_power)

        self.rc_car_cntl.set_wheel_speed(right_speed, left_speed, 150)
```

Code 5.1

Code 5.1의 `sensitivity`, `left_power`, `right_power`는 모두 실험적으로 결정하는 값이다. 하드웨어 특성이 바뀌면 위 값은 변할 수 있다. Code 5.1의 `right_speed`와 `left_speed`를 구하는 부분을 수식으로 나타내면 다음과 같다.

$$\varphi \geq 0 \rightarrow \begin{cases} M_R = [p_R \times ((255 - s) + [(1 - \varphi)s])] \\ M_L = [p_L \times ((255 - s) + s)] \end{cases}$$

$$\varphi < 0 \rightarrow \begin{cases} M_R = [p_R \times ((255 - s) + s)] \\ M_L = [p_L \times ((255 - s) + [(1 + \varphi)s])] \end{cases}$$

Code 5.1의 `direction`, `sensitivity`, `left_power`, 그리고 `right_power`는 각각 위 수식에서 φ, s, p_R, p_L에 해당한다. M_R과 M_L은 시리얼 통신으로 전해지는 -255~255사이의 정수 값이다. 각각 오른쪽 모터, 왼쪽 모터에 가해질 전압의 크기를 결정한다. 이때, s가 커지면 φ값이 같아도 M_R과 M_L의 차이는 커진다. 즉, φ값에 대해 더 민감하게 반응한다. p_R, p_L는 일종의 보정 계수로, 두 모터의 성능이 다르거나 무게중심이 한쪽에 쏠리는 등의 문제로 같은 제어신호를 보내도 다른 속도를 내는 경우를 보정하기 위한 계수이다.

6. 방향 결정 (Direction Determination)

방향을 결정하는 방법으로 두가지 방법을 고안했다.

1. 이미지 매트릭스를 직접 분석하는 프로그램을 작성하여 방향을 정하는 알고리즘 기반
2. 이미지 매트릭스와 이에 대응하는 적합한 방향 데이터를 이용하여 학습을 진행시켜 모델을 구성하는 인공지능 기반

2번, 인공지능 기반의 방법을 바로 진행하기에는 양질의 데이터를 모으는 것이 쉽지 않고, 방향 데이터가 해당 시스템의 하드웨어 특성에 적합한지를 판단하기가 어려운 문제가 있다. 따라서 본 시스템은 1번, 알고리즘 기반으로 코드를 작성한 뒤, 적합한 환경을 구성하여 코드를 여러 번 실행시켜 라인 트레이싱을 하는 동시에 데이터 수집도 자동으로 같이 수행하는 방법을 채택했다. 이 데이터를 기반으로 학습을 진행한다면 동일 하드웨어 조건에서는 정상작동을 어느정도 보장할 수 있을 것이다. 즉, 이 인공지능이 학습하는 것은 앞서 설계한 알고리즘이 된다.

6.1 방향 결정 알고리즘 (Direction Determination Algorithm)

촬영된 도로로부터 다음 진행 방향을 결정하는 가장 쉽고 직관적인 방법은 도로(라인)의 기울기를 이용하는 것이다. 이미지 자체가 충분히 국소적(local)이면 촬영되는 라인은 대부분이 거의 직선 형태일 것이고, 나타나는 곡선도 대부분 곡률이 작아 직선으로 일차 근사하는 것이 가능할 것이다. 따라서 이 알고리즘은 촬영되는 도로는 곡률이 크거나 복잡한 곡선은 없다고 가정하고 곡선을 직선으로 근사하여 처리한다.

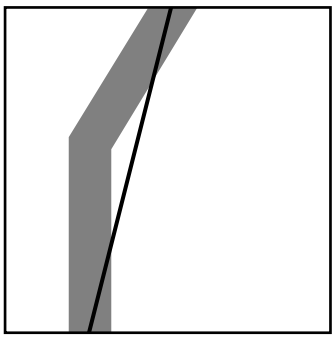


Figure 6.1

Figure 6.1과 같이 촬영된 도로를 직선형태로 근사하면 그 기울기를 보다 쉽게 다룰 수 있다. 이렇게 도로를 근사하여 얻은 직선을 도로 직선(lane line)이라 부른다.

Definition 6.1

촬영한 2차원이미지에서 도로는 넓이가 있는 영역의 형태를 가진다. 이 영역으로부터 어떤 알고리즘을 통해 도로의 대략적인 방향과 위치를 직선형태로 나타낸 것을 도로 직선(lane line)이라 부른다.

Definition 6.2

직선 l의 방향 벡터(direction vector) $\boldsymbol{d}$는 l이 원점(O)을 지나도록 평행 이동된 직선 l'의 차원(dimension)이 1인 기저(basis)의 원소가 될 수 있는 벡터이다.

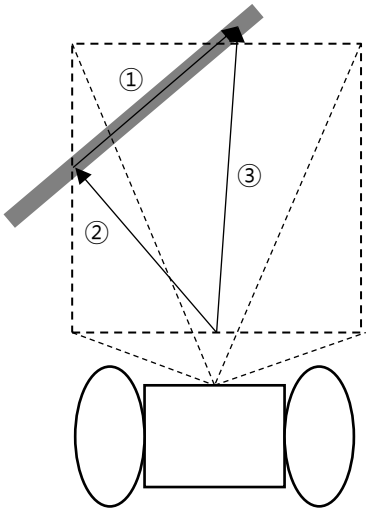
도로 직선을 구하는 알고리즘은 이후에 다루며 별 다른 언급이 없는 한, 본문에서 방향 벡터는 도로 직선의 방향 벡터를 의미한다.

하지만 방향 벡터 만으로는 충분하지 않다. 도로 직선의 기울기만 아니라 기기의 도로 직선에 대한 상대 위치도 고려를 해야 한다. 이는 기기의 도로 직선으로부터의 수직 방향 거리로 고려하는 것이 자연스러우며 이는 직선의 방향 벡터와 수직인 벡터, 즉 법선 벡터(normal vector)를 이용하는 것이 자연스럽다.

Definition 6.3

위상 벡터(phase vector) $\boldsymbol{p}$는 기기의 위치(P)와 도로 직선 l의 수직거리(vertical distance)만큼의 크기를 지니며, 도로 직선의 방향벡터와는 수직(orthogonal)인 관계에 있으며. 점 $P+\boldsymbol{p}$가 직선 l위에 존재하는 벡터이다.

이 알고리즘은 위상 벡터(phase vector) $\boldsymbol{p}$와 특정 값으로 일관된 크기를 가지는 방향 벡터(direction vector) $\boldsymbol{d}$의 합성(composition)을 이용하여 기기가 나아가야할 방향, 즉 텀-방향 벡터(term-direction vector)를 계산한다. Figure 6.2는 이 과정을 나타낸다. Figure 6.2의 ①번 벡터는 도로 직선의 방향 벡터이며 ②번 벡터는 도로 직선에 대한 기기의 위상 벡터이다. ③번 벡터는 이들의 합성으로 기기가 나아가야할 방향을 제시해주는 텀-방향 벡터(term-direction vector)이다.



① Calculating the sloop of the line (direction vector)
② Calculating the distance from the line (phase vector)
③ Composition of the direction vector and the phase vector

Figure 6.2

6.1.1 도로 직선 탐색 알고리즘 (Lane Line Detecting Algorithm)

위 작업을 위해선 가장 먼저 적절한 도로 직선을 찾을 수 있어야 한다. 직선은 서로 다른 두 점이 주어지면 유일하게 결정될 수 있다. 따라서 이를 수행하기 위해 이미지 매트릭스로부터 직접 가능한 경우를 고려하여 직선을 결정하는 두 인덱스(index) `[a][b]`와 `[c][d]`를 반환하는 함수를 작성하였다.

먼저 이를 위해서 이미지 매트릭스는 좀더 정교하게 처리되어야 한다.

```
for i in range(self.image_size):
    for j in range(self.image_size):
        if img[i][j] < 220:
            img[i][j] = 0
        else:
            img[i][j] = 255
```

Code 6.1

Code 6.1은 이미지 매트릭스로부터 역치(threshold)를 두어 한 번 더 정제(refine)를 하는 코드이다. 역치(Code 6.1에서는 220)을 넘지 못하면 0으로 이보다 같거나 크면 255로 처리한다.

```
def find_critical_points(img, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = -1, -1, -1, -1, -1, -1, -1, -1
    s = imgSize - 1
    for i in range(imgSize):
        if img[i][0] == 255:
            a1 = i
            break
    for i in range(imgSize):
        if img[s - i][0] == 255:
            a2 = s - i
            break
    for i in range(imgSize):
        if img[i][s] == 255:
            b1 = i
            break
    for i in range(imgSize):
        if img[s - i][s] == 255:
            b2 = s - i
            break
    for i in range(imgSize):
        if img[0][i] == 255:
            c1 = i
            break
    for i in range(imgSize):
        if img[0][s - i] == 255:
            c2 = s - i
            break
    for i in range(imgSize):
        if img[s][i] == 255:
            d1 = i
            break
    for i in range(imgSize):
        if img[s][s - i] == 255:
            d2 = s - i
            break
    return float(a1), float(a2), float(b1), float(b2), float(c1), float(c2), float(d1), float(d2)
```

Code 6.2

Code 6.2는 직선을 결정하는데 쓰이는 두 인덱스 `[a][b]`와 `[c][d]`를 찾는데 유용하게 쓰일 점들을 반환한다. 이들은 이미지의 테두리에 있으며 존재하지 않을 수도 있다. 이들은 영역 형태의 도로가 이미지 테두리 어느 부분에 걸쳐 존재하는지를 정한다. 따라서 이들은 존재하지 않을 수도 있고, 존재하지 않는 경우 값을 `-1`로 둔다. Figure 6.3은 이들을 탐색하는 방향을 보여준다.

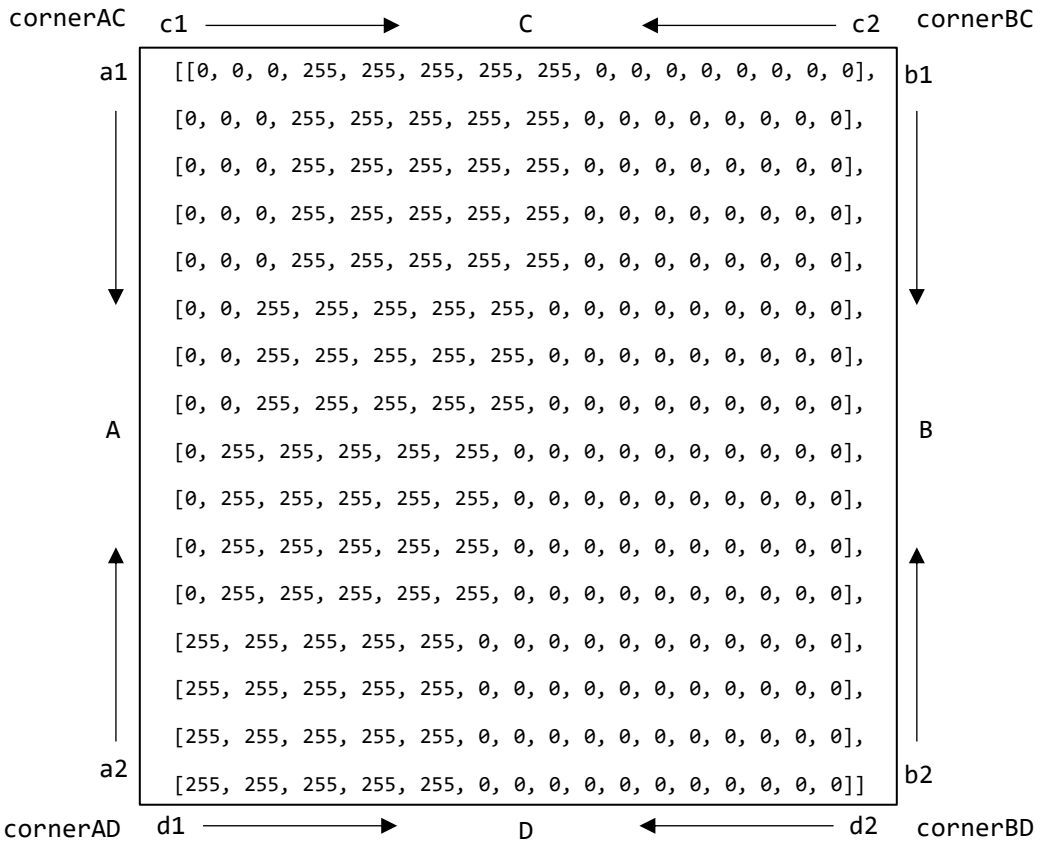


Figure 6.3

```
def cornerAC (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if a1 == 0.0 and c1 == 0.0:
        return True
    else:
        return False

def cornerBC (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if c2 == imgSize - 1 and b1 == 0:
        return True
    else:
        return False

def cornerAD (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if a2 == imgSize - 1 and d1 == 0:
        return True
    else:
        return False

def cornerBD (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if d2 == imgSize - 1 and b2 == imgSize - 1:
        return True
    else:
        return False
```

Code 6.3

```
def borderA (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if a1 == -1 and a2 == -1:
        return False
    else:
        return True

def borderB (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if b1 == -1 and b2 == -1:
        return False
    else:
        return True

def borderC (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if c1 == -1 and c2 == -1:
        return False
    else:
        return True

def borderD (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if d1 == -1 and d2 == -1:
        return False
    else:
        return True
```

Code 6.4

Code 6.3과 Code 6.4는 각각 Figure 6.3에 표시된 이미지 모서리 AC, BC, AD, BD와 이미지 테두리 A, B, C, D에 도로 영역이 존재하는지를 판단하는 함수이다. 이를 이용하여 도로의 대략적인 개형을 추론할 수 있다.

```
import random as rd

def linepoints (critical_points, imgSize, lineWidth): # lineWidth is determined by experiment
    p = critical_points
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    s = imgSize - 1
    w = lineWidth/2.0

    # 4
    if borderA(p) and borderB(p) and borderC(p) and borderD(p):
        if cornerAC(p, imgSize) and cornerBD(p, imgSize) and not cornerAD(p, imgSize) and not corner
BC(p, imgSize):
            return a1/2, c1/2, (s + b2)/2, (s + d2)/2
        elif not cornerAC(p, imgSize) and not cornerBD(p, imgSize) and cornerAD(p, imgSize) and corn
erBC(p, imgSize):
            return b1/2, d1/2, (s + a2)/2, (s + c2)/2
        elif not cornerAC(p, imgSize) and cornerBD(p, imgSize) and cornerAD(p, imgSize) and cornerBC
(p, imgSize):
            return b1, s, s, d1
        elif cornerAC(p, imgSize) and not cornerBD(p, imgSize) and cornerAD(p, imgSize) and cornerBC
(p, imgSize):
            return 0, c2, a2, 0
        elif cornerAC(p, imgSize) and cornerBD(p, imgSize) and not cornerAD(p, imgSize) and cornerBC
(p, imgSize):
            return 0, c1, b2, s
        elif cornerAC(p, imgSize) and cornerBD(p, imgSize) and cornerAD(p, imgSize) and not cornerBC
(p, imgSize):
            return a1, 0, s, d2
        else:
```

```
            return rd.randint(0, s), rd.randint(0, s), rd.randint(0, s), rd.randint(0, s)

    # 3
    elif not borderA(p) and borderB(p) and borderC(p) and borderD(p):
        return 0, c1, s, d1
    elif borderA(p) and not borderB(p) and borderC(p) and borderD(p):
        return 0, c2, s, d2
    elif borderA(p) and borderB(p) and not borderC(p) and borderD(p):
        return a1, 0, b1, s
    elif borderA(p) and borderB(p) and borderC(p) and not borderD(p):
        return a2, 0, b2, s

    # 2
    elif not borderA(p) and not borderB(p) and borderC(p) and borderD(p):
        return 0, (c1 + c2)/2, s, (d1 + d2)/2
    elif borderA(p) and borderB(p) and not borderC(p) and not borderD(p):
        return (a1 + a2)/2, 0, (b1 + b2)/2, s
    elif borderA(p) and not borderB(p) and not borderC(p) and borderD(p):
        return (a1 + a2)/2, 0, s, (d1 + d2)/2
    elif not borderA(p) and borderB(p) and borderC(p) and not borderD(p):
        return 0, (c1 + c2)/2, (b1 + b2)/2, s
    elif borderA(p) and not borderB(p) and borderC(p) and not borderD(p):
        return 0, (c1 + c2)/2, (a1 + a2)/2, 0
    elif not borderA(p) and borderB(p) and not borderC(p) and borderD(p):
        return (b1 + b2)/2, s, s, (d1 + d2)/2

    # 1
    elif borderA(p) and not borderB(p) and not borderC(p) and not borderD(p):
        return a1, 0, a2, 0
    elif not borderA(p) and borderB(p) and not borderC(p) and not borderD(p):
        return b1, s, b2, s
    elif not borderA(p) and not borderB(p) and borderC(p) and not borderD(p):
        return 0, c1, 0, c2
    elif not borderA(p) and not borderB(p) and not borderC(p) and borderD(p):
        return s, d1, s, d2

    # 0
    else:
        return rd.randint(0, s), rd.randint(0, s), rd.randint(0, s), rd.randint(0, s)
```

Code 6.5

Code 6.5는 Code 6.3과 Code 6.4의 함수들을 이용하여 대략적인 도로영역의 형태를 추론하고 이를 토대로 도로 직선을 결정하는 두 인덱스 `[a][b]`와 `[c][d]`를 튜플(tuple)형태로 반환한다. `lineWidth`는 도로의 두께를 의미하며 이는 실험적으로 결정하여야 한다. 이는 이미지의 해상도(`imgSize`), 즉 이미지 매트릭스의 크기에 독립적으로 작동하도록 작성하였다.

Figure 6.5는 이 알고리즘을 사용했을 때, 정제된 이미지 매트릭스로부터 점 `a1`, `a2`, `b1`, `b2`, `c1`, `c2`, `d1`, `d2`가 결정되고 이로부터 도로 직선을 결정하는 두 인덱스 `[a][b]`와 `[c][d]`가 결정되는 한 예시를 보여준다.

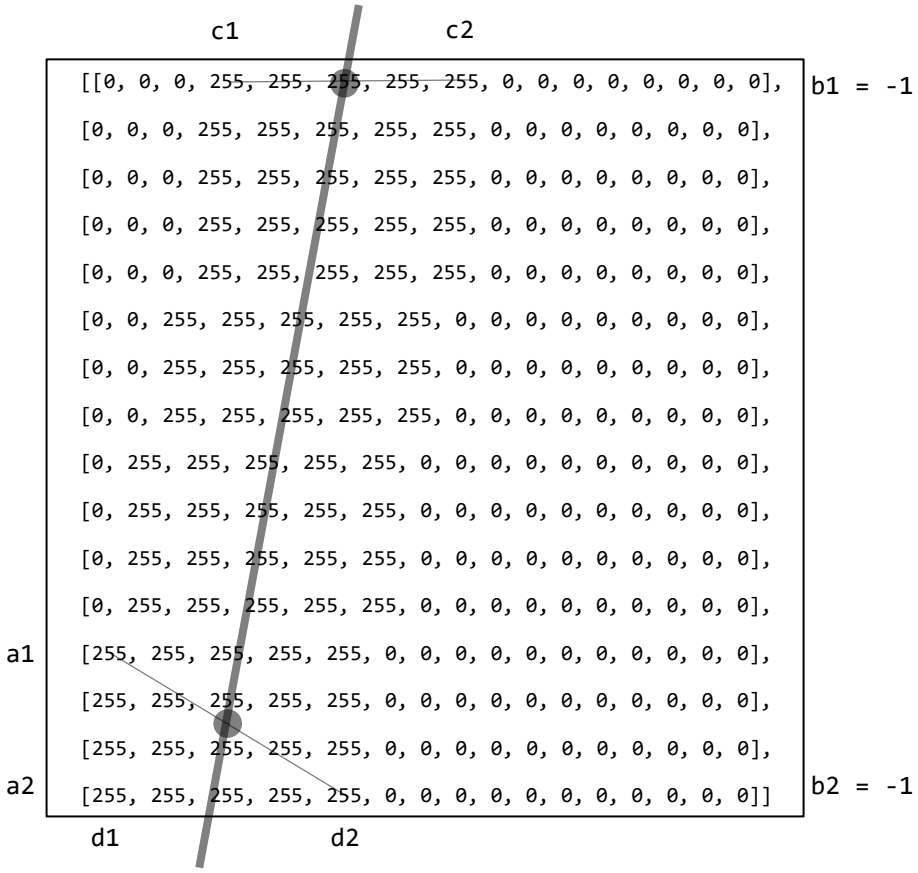


Figure 6.5

6.1.2 벡터 계산 알고리즘 (Vector Calculating Algorithm)

도로 직선이 탐색되면 본격적으로 방향 벡터와 위상 벡터를 계산할 수 있다. 이때 일반적인 수식사용을 용이하게 하게 위해 이미지 매트릭스의 인덱스를 직교 좌표계에 대응시켜 계산할 수 있다. 이미지 사이즈가 16×16일 때, 2차원 이미지 매트릭스 `img`의 img[15][0] 원소가 직교좌표 상 원점 O에 대응되고 img[0][15]의 원소가 점 (15,15)에 대응되도록 한다. Figure 6.6은 이를 보여준다. 두 인덱스 `[a][b]`와 `[c][d]`는 직교좌표상의 두 좌표 $(b, 2p-a), (d, 2p-c)$에 대응된다.

두 좌표 $(b, 2p-a), (d, 2p-c)$가 결정하는 직선 l의 방정식은 다음과 같다.

$$l: (c-a)(x-d) - (b-d)y + (b-d)(2p-c) = 0$$

d는 직선 l과 기기의 위치 $(p, 0) = \left(\frac{\beta-1}{2}, 0\right)$의 수직거리이다. β는 이미지 매트릭스의 열의 크기 또는 행의 크기를 나타낸다. 이미지 매트릭스의 크기가 16×16이면 $(p, 0) = \left(\frac{\beta-1}{2}, 0\right) = \left(\frac{16-1}{2}, 0\right) = (7.5, 0)$이다.

$$d = \frac{|p(c-a) - (b-d)0 + (b-d)(2p-c) + d(a-c)|}{\sqrt{(b-d)^2 + (c-a)^2}}$$

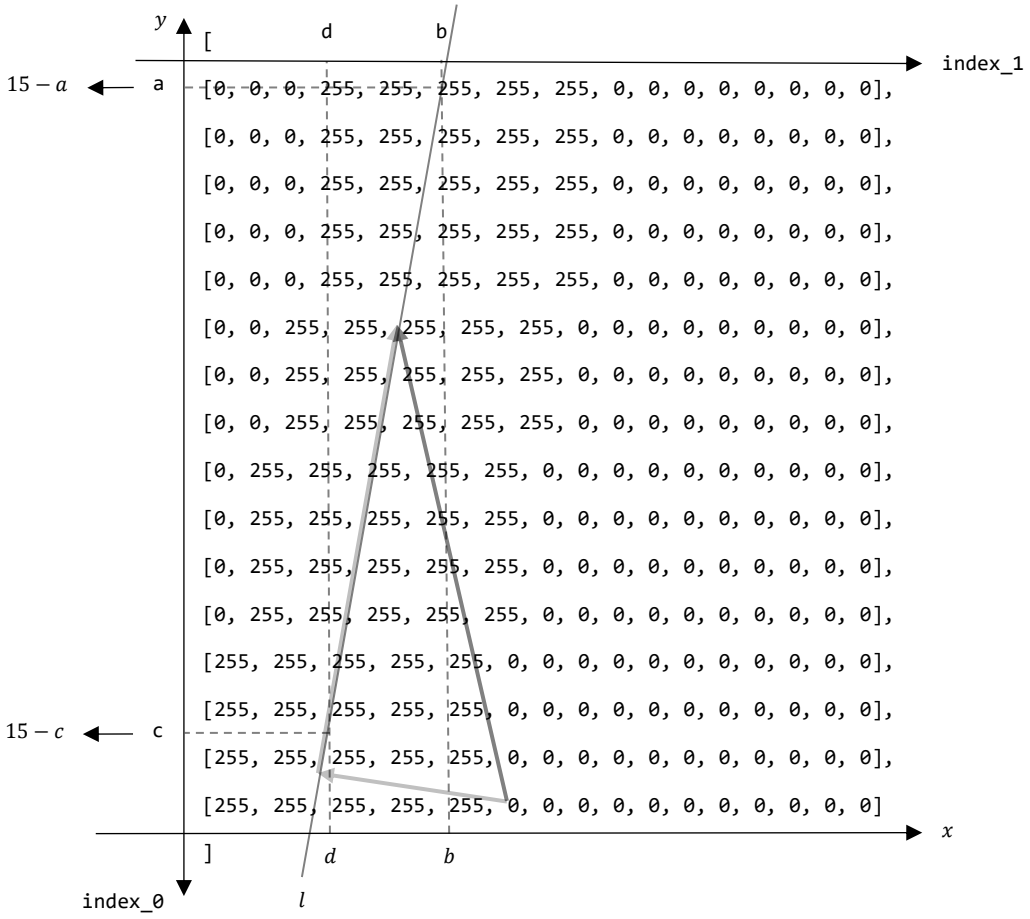


Figure 6.6

Theorem 6.1

직선의 방정식 $l: (c-a)(x-d) - (b-d)y + (b-d)(2p-c) = 0$ 으로부터 방향벡터 $\boldsymbol{d}$는 다음과 같다. $C \in \mathbb{R}$는 이미지 매트릭스 크기에 따라 달라지는 값으로 실험적으로 결정된다.

$$(direction\ vector) = \boldsymbol{d} = \boldsymbol{s_1}(b-d, c-a)\frac{C}{\sqrt{(b-d)^2 + (c-a)^2}}$$

$$\boldsymbol{s_1} = \begin{cases} 1 & (a \le c) \\ -1 & (a > c) \end{cases}$$

Theorem 6.2

직선의 방정식 $l: (c-a)(x-d) - (b-d)y + (b-d)(2p-c) = 0$과 기기 위치 $(p, 0)$로부터 계산되는 위상 벡터 $\boldsymbol{p}$는 다음과 같다.

$$(phase\ vector) = \boldsymbol{p} = s_2(c-a, d-b)\ \frac{|p(c-a) + (b-d)(2p-c) + d(a-c)|}{(b-d)^2 + (c-a)^2}$$

$$\boldsymbol{s_2} = \begin{cases} 1 & (p(c-a) + (b-d)(2p-c) + d(a-c) \le 0) \\ -1 & (p(c-a) + (b-d)(2p-c) + d(a-c) > 0) \end{cases}$$

이어서 이들의 합성으로 텀-방향 벡터 $\boldsymbol{v_{a,b,c,d}}$를 얻는 것이 가능하다.

$$(term-direction\ vector) = (direction\ vector) + (phase\ vector) = \boldsymbol{d} + \boldsymbol{p} = v_{a,b,c,d}$$

이를 바탕으로 다음과 같이 y축과 이루는 각도 $\theta_{a,b,c,d} \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]$를 구하는 것도 가능하다.

Theorem 6.3

$$v_{a,b,c,d} = \boldsymbol{d} + \boldsymbol{p}$$

$$\boldsymbol{s_3} = \begin{cases} 1 & (v_{a,b,c,d}[0] \geq 0) \\ -1 & (v_{a,b,c,d}[0] < 0) \end{cases}$$

$$\theta_{a,b,c,d} = \boldsymbol{s_3} cos^{-1} \frac{v_{a,b,c,d} \cdot (0,1)}{|v_{a,b,c,d}|}$$

이어서 $f(\theta) = \frac{2\theta}{\pi}$ 선형변환을 통해 $[-1, 1]$ 사이의 방향 값을 얻을 수 있다.

```
def determine_direction(a, b, c, d, imgSize):
    fix = 10
    a = float(a)
    b = float(b)
    c = float(c)
    d = float(d)
    p = (imgSize-1)/2

    if a <= c:
        sign1 = 1
    else:
        sign1 = -1

    if (c-a)*p+(b-d)*(2*p-c)+d*(a-c) <= 0:
        sign2 = 1
    else:
        sign2 = -1

    direction_vector = sign1*np.array((b-d, c-a))*fix/math.sqrt((b-d)**2 + (c-a)**2)
    phase_vector = sign2*np.array((c-a,d-b))*abs((c-a)*p+(b-d)*(2*p-c)+d*(a-c))/((b-d)**2 + (c-
a)**2)

    # Composition of the direction vector and phase vector
    v = direction_vector + phase_vector

    # Calculate direction
    cos = v[1]/math.sqrt(v[0]**2 + v[1]**2)
    if v[0] >= 0:
        sign3 = 1
    else:
        sign3 = -1
    theta = sign3*math.acos(cos)
    direction = 2*theta/math.pi
    return direction
```

Code 6.6

6.1.3 위상 차 보정 알고리즘 (Phase Difference Correction Algorithm)

카메라와 기기의 중심은 물리적 간격이 존재한다. 따라서 이미지에는 기기의 중심이 정확히 나타나지 않는다. 즉, 기기는 현재 위치에 대한 정확한 방향벡터와 위상벡터를 계산하기는 어렵다. 이 간격을 보정하기 위해 기기는 다음과 같이 이전 텀-방향 벡터를 이용할 수 있다.

$$f(\theta_{current}) = (1-\alpha)f(\theta_{a,b,c,d}) + \alpha f(\theta_{previous}), \qquad \alpha \in (0,1)$$

여기서 α는 위상 차 보정 계수(Phase Difference Correction Coefficient)라 부르도록 하며 실험적으로 결정되는 값이다.

6.1.4 제한사항 (Restrictions)

이 알고리즘은 이미지 매트릭스를 한 번 더 정제하여 사용한다. 정제된 이미지 매트릭스가 적합하지 않으면 제대로 작동하지 않는다. 예를 들어 반사광에 의해 검은색 라인이 밝게 인식되거나 그림자로 인해 흰색 바탕이 어둡게 인식되는 경우 정도에 따라 이미지 매트릭스 정제가 제대로 이루어지지 않을 수 있다. 이렇게 주위 환경 요소(조명)가 이미지 매트릭스의 원소 값에 유의미한 영향을 주는 것을 이미지 노이즈(image noise)라 부르기로 한다. 따라서 적절 조명 환경이 제공되지 않은 경우, 이미지 노이즈가 많아지고 이 알고리즘은 신뢰도가 낮아진다.

6.2 인공지능을 이용한 방향 결정 (Direction Determination with AI)

알고리즘을 활용한 방법은 이미지 노이즈의 영향을 많이 받는다. 따라서 이를 직접 활용하는 대신 인공지능을 통해 위 알고리즘을 학습하는 방법을 채택하였다. 즉, 이 인공지능이 학습하는 것은 앞서 설계한 알고리즘이 된다.

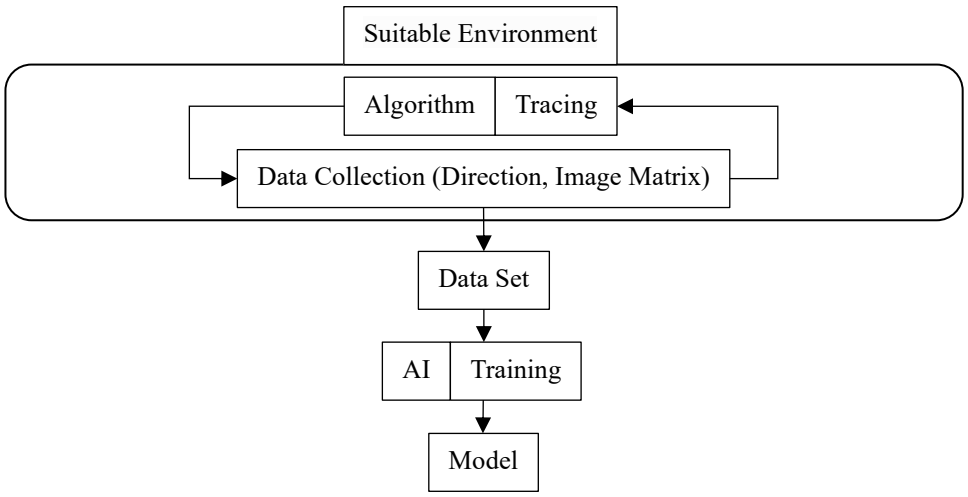


Figure 6.7

먼저 알고리즘이 돌아가기에 적합한 환경을 구축한다. 이미지 노이즈가 적게 생기는 조명 아래에서 주행을 반복하고 도로 촬영 시 마다 이미지 매트릭스와 계산된 방향 정보를 저장한다. 이것이 데이터가 된다. 학습과정에서 방향 정보는 데이터의 타겟(target) 또는 레이블(label)로 다뤄진다. 입력으로 이미지 매트릭스를 받으면 방향 정보를 출력하는 모델을 구축한다.

인공지능 라이브러리는 텐서플로우(tensorflow) 2.3.0을 사용한다. 학습은 라즈베리 파이4에서 직접 수행하는 방법이 아니라 비교적 성능이 좋은 PC에서 수행한 뒤 학습된 모델을 .h5 파일 또는 .tflite 파일로 저장하여 라즈베리 파이4에서 로드하는 방법을 사용한다.

6.2.1 학습 (Training)

학습 데이터로 정사각 행렬 형태의 1채널 라즈베리파이 카메라를 통해 직접 촬영한 이미지가 주어지며, 타겟 값은 위에서 제시한 방법에 따라 0도~180도 사이의 각도 또는 -1에서 1사이의 값으로 주어진다. 또한 scikit-learn 패키지의 train_test_split 함수를 이용해 학습 데이터와 평가 데이터는 2:1의 비율로 나눈다. 본문은 16×16 사이즈의 이미지 매트릭스를 입력으로 받는 모델을 상정한다.

케라스 버전 2.3.0의 model, layer등의 서브모듈을 이용해 딥러닝 방향예측을 위한 모델을 구성하며, 구성한 모델은 아래와 같다. 두 모델 모두 라즈베리파이 내에서 방향 예측 시 발생하는 오버헤드를 줄이기 위해 모델을 최대한 가볍게 구성하였다. 두 모델 모두 MSE loss 함수를 사용하였고, optimizer 로 adam을 사용하였다. 또한 mini-batch의 크기는 10, epoch는 100으로 두고 학습을 진행하였다.

A. 2D Convolution 층으로 구성

```
model = Sequential()
model.add(Conv2D(4, (2, 2), padding='same', activation='relu', input_shape=(16, 16, 1)))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(8, (2, 2), padding='same', activation='relu'))
model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dense(1))
```

Code 6.7

높이와 너비가 각각 16, 채널 수가 1인 변환된 입력 이미지에 대하여, 채널 수가 4, 커널의 크기가 2인 Conv2d 층, 커널의 크기가 2인 Maxpooling2d 층, 채널 수가 8, 커널의 크기가 2인 Conv2d 층을 순서대로 쌓아 activation map의 크기가 점차 작아지도록 설계하였다. 또한 위에 나열한 층들을 통과 후에는 64개의 노드로 구성된 Fully-Connected 층을 통과하도록 구성하였다. 층 별 파라미터 수는 아래와 같다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 16, 16, 4)         20
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 8, 8, 4)           0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 8, 8, 8)           136
_________________________________________________________________
flatten (Flatten)            (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 33,053
Trainable params: 33,053
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.8

학습 결과, 학습 데이터에 대한 MSE loss 값은 3.0647, 평가 데이터에 대한 loss는 19.2744를 기록하였다. epoch에 따른 loss 변화는 아래와 같다.

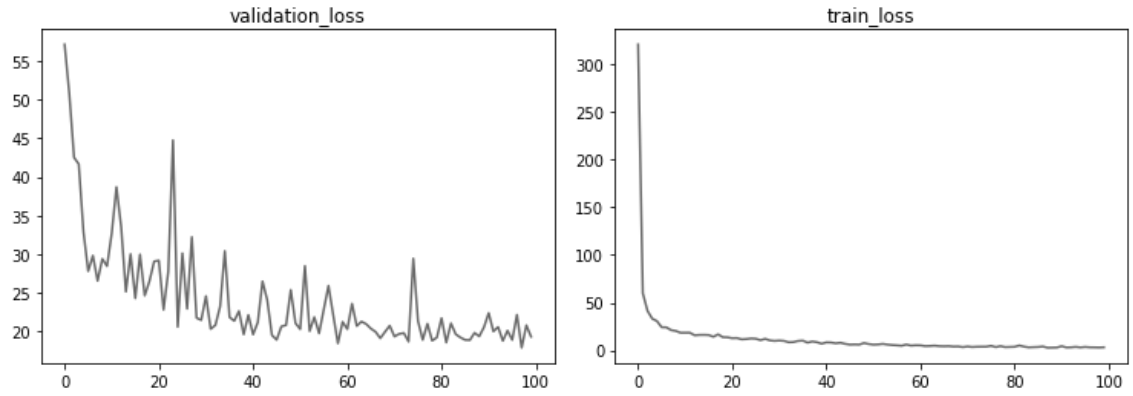


Figure 6.9

B. Fully-Connected 층으로 구성

```
model = Sequential()
model.add(Dense(512, input_dim=np.shape(trX)[1], activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1))
```

Code 6.8

입력층의 노드 수가 256개(16 * 16)인 Fully-Connected 층 위에 노드 수가 64개, 노드 수가 1개인 층을 순서대로 쌓고, 데이터가 모델을 통과한 후 MSE loss 함수에서 loss 값을 계산하도록 구성하였다. 층 별 파라미터 수는 아래와 같다.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_7 (Dense)              (None, 64)                32832
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 65
=================================================================
Total params: 164,481
Trainable params: 164,481
Non-trainable params: 0
_________________________________________________________________
```

Figure 6.10

학습 결과, 학습 데이터에 대한 MSE loss 값은 3.3039, 평가 데이터에 대한 loss는 26.3571을 기록하였다. epoch에 따른 loss 변화는 아래와 같다.

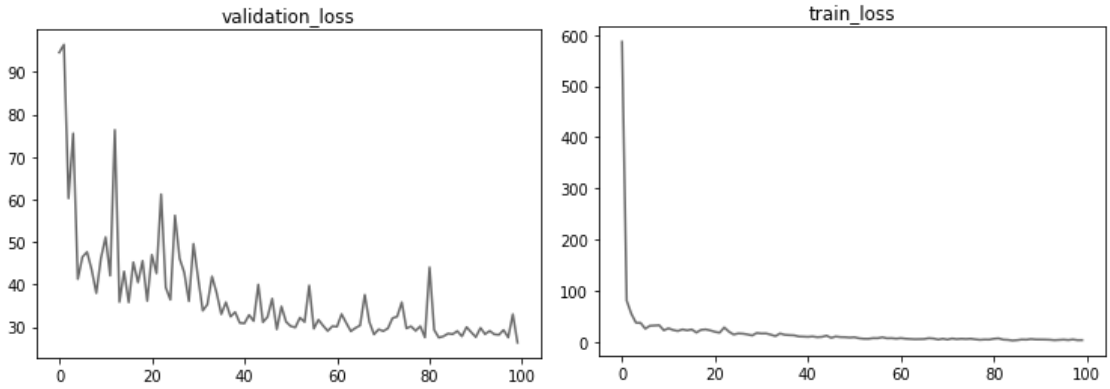


Figure 6.11

두 모델을 비교하였을 때, CNN층 2개로 구성할 모델이 Fully-Connected 층 2개로 구성한 모델보다 파라미터 수가 1/5 수준임에도, validation loss 값이 더 작게 나온 것을 확인할 수 있다. Fully-Connected 로 구성한 모델의 경우 Batch Nomalization을 추가하여 학습을 더 빠르게 진행해보고자 하였으나, 큰 차이는 없음을 확인하였다.

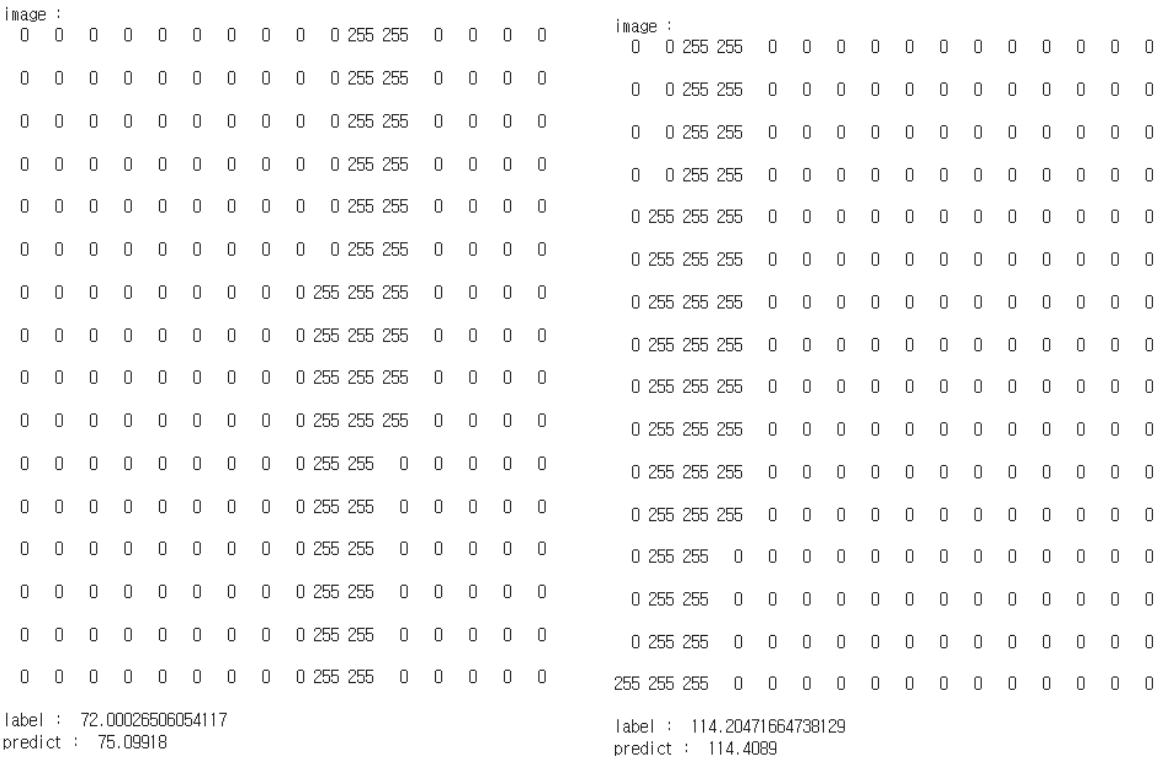


Figure 6.12

아래는 CNN층으로 구성한 모델을 이용하여 이미지에 대한 방향(각도)를 예측한 결과 예시이다. label은 앞에서 알고리즘을 이용하여 계산한 각도이고, predict는 딥러닝 모델을 이용하여 계산한 각도이다. 비슷한 직선임에도 위치에 따라서 다르게 예측하도록 학습된 것을 확인할 수 있다.

6.2.2 모델 탑재 (Model Mounting)

텐서플로우에서는 학습한 모델을 파일로 저장하여 다른 프로그램에서 로드하는 기능을 지원한다. 라즈베리 파이4는 학습된 모델을 로드하여 이 모델을 통해 방향을 얻는다. 모델을 저장하는 방법은 .h5 파일로 저장하거나 .tflite 파일로 저장할 수 있다.

모델을 .h5 파일로 저장한 뒤 텐서플로우와 케라스를 사용하여 라즈베리 파이4 에서 모델을 로드하는 경우, 학습된 모델의 크기(capacity)가 너무 커서 라즈베리 파이4의 성능으로는 방향 계산에 너무 많은 오버헤드(overhead)가 생길 수 있다. 그러나 텐서플로우에서 지원하는 저전력 ARM 아키텍처를 사용하는 모바일 디바이스 또는 마이크로 컴퓨터를 위한 텐서플로우모듈인 텐서플로우 라이트(tensorflow lite)모듈을 사용하여 모델을 .tflite 파일로 저장한 뒤 로드하는 방법을 사용하였을 때는 방향 계산에 가해지는 오버헤드가 10분의 1수준으로 줄었다. 따라서 텐서플로우 모듈을 직접 사용하기보단 텐서플로우 라이트 모듈을 사용하고 이에 맞는 모델을 사용하도록 한다.

6.2.3 성능 (Performance)

탑재된 텐서플로우 모델을 단순히 예측하는 것 일지라도 라즈베리 파이4의 환경은 캐퍼시티가 큰 모델을 단순히 예측만 하는 작업을 하기에도 상당히 제한된 환경이다. GPU를 통한 계산이나 병렬처리 측면에서 매우 제한되며 메모리 공간도 제약이 많다. 따라서 CPU자원을 최대한 활용하기 위해 오버클럭(Overclock)을 사용한다. /boot/config.txt를 관리자권한으로 수정한다. [pi4] 항목에 다음을 입력하여 안전한 범위에서 오버클럭하여 사용할 수 있다.

```
over_voltage=2
arm_freq=1750
gpu_freq=600
initial_turbo=30
temp_limit=75
```

Code 6.9

7. 부록 - 코드 (Appendix - Code)

7.1 아두이노 우노 코드

```
String speed1, speed2, delay_time;
char  sign1, sign2;
int  i, p1, p2;
int s1, s2, d;
int motorA1 = 5;
int motorA2 = 6;
int motorB1 = 9;
int motorB2 = 10;

byte DataToRead[16];

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  Serial.flush();
  pinMode(motorA1, OUTPUT);
  pinMode(motorA2, OUTPUT);
  pinMode(motorB1, OUTPUT);
  pinMode(motorB2, OUTPUT);
}

void loop() {
  for (i = 0; i < 16; i++) {
    DataToRead[i] = char(10);
  }
  if(Serial.available() > 0){
    Serial.readBytesUntil(char(10), DataToRead, 16);
    DataToRead[15] = char(10);
  }

  /* For Debugging, send string to RPi */
  /*
  for (i = 0; i < 16; i++) {
    Serial.write(DataToRead[i]);
    if (DataToRead[i] == char(10)) break;
  }
  */
  /* End of Debugging */
```

```
  for (i = 0; i < 16; i++) {
    if (DataToRead[i] == 'L') {
      p1 = i;
      break;
    }
  }

  for (i = 0; i < 16; i++) {
    if (DataToRead[i] == 'T') {
      p2 = i;
      break;
    }
  }

  speed1 = "";
  speed2 = "";

  if (DataToRead[1] == '-') {
    sign1 = '-';
    for (i = 2; (DataToRead[i] != 'L') && (i < p1); i++) {
      speed1 += char(DataToRead[i]);
    }
  }
  else {
    sign1 = '+';
    for (i = 1; (DataToRead[i] != 'L') && (i < p1); i++) {
      speed1 += char(DataToRead[i]);
    }
  }

  if (DataToRead[p1 + 1] == '-') {
    sign2 = '-';
    for (i = p1 + 2; (DataToRead[i] != char(10)) && (i < p2); i++) {
      speed2 += char(DataToRead[i]);
    }
  }
  else {
    sign2 = '+';
    for (i = p1 + 1; (DataToRead[i] != char(10)) && (i < p2); i++) {
      speed2 += char(DataToRead[i]);
    }
  }

  delay_time = "";

  for (i = p2 + 1; (DataToRead[i] != char(10)) && (i < 16); i++) {
    delay_time += char(DataToRead[i]);
  }

  s1 = speed1.toInt();
  s2 = speed2.toInt();
  d = delay_time.toInt();

  if (sign1 == '+' && sign2 == '+') {
    analogWrite(motorA1, s1);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, s2);
    analogWrite(motorB2, 0);
  }
  else if (sign1 == '+' && sign2 == '-') {
    analogWrite(motorA1, s1);
    analogWrite(motorA2, 0);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, s2);
  }
  else if (sign1 == '-' && sign2 == '+') {
    analogWrite(motorA1, 0);
    analogWrite(motorA2, s1);
    analogWrite(motorB1, s2);
    analogWrite(motorB2, 0);
  }
  else {  //sign1 == '-' && sign2 == '-'
    analogWrite(motorA1, 0);
```

```
    analogWrite(motorA2, s1);
    analogWrite(motorB1, 0);
    analogWrite(motorB2, s2);
  }
  delay(d);

}
```

Code 7.1 (Motor Control)

7.2 라즈베리 파이 알고리즘 자율주행 코드 + 데이터 수집

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

from rc_car_interface import RC_Car_Interface
import numpy as np
import time
import cv2
import serial
import math
import find_line as fl
import pickle
import copy

class SelfDriving:

    def __init__(self, ser):
        self.rc_car_cntl = RC_Car_Interface(ser)
        self.rc_car_cntl.set_left_speed(0, 150)
        self.rc_car_cntl.set_right_speed(0, 150)

        self.velocity = 0
        self.direction = 0
        self.image_size = 32

    def rc_car_control(self, direction):
        sensitivity = 150
        left_power = 1.0
        right_power = 1.0

        ## calculate left and right wheel speed with direction
        if direction < -1.0:
            direction = -1.0
        if direction > 1.0:
            direction = 1.0

        #print("direction: ", direction)
        if (direction <= 0.0):
            right_speed = int(((255 - sensitivity) + int((1.0)*sensitivity))*right_power)
            left_speed = int(((255 - sensitivity) + int((1.0 + direction)*sensitivity))*left_power)
        else:
            right_speed = int(((255 - sensitivity) + int((1.0 - direction)*sensitivity))*right_power
)
            left_speed = int(((255 - sensitivity) + int((1.0)*sensitivity))*left_power)

        self.rc_car_cntl.set_wheel_speed(right_speed, left_speed, 150)

    def drive(self):
        previous_direction = 0
        save_image_num = 5000
        alpha = 0.8
        list_image = []
        save_image = True
        run = True
        itr = 0
```

```
        while run:
            try:
                primitive_img = self.rc_car_cntl.get_image_from_camera(self.image_size)
                #print(primitive_img)

                # Image refine
                img = copy.copy(primitive_img)
                for i in range(self.image_size):
                    for j in range(self.image_size):
                        if img[i][j] < 220:
                            img[i][j] = 0
                        else:
                            img[i][j] = 255

                # Initialize
                direction = 0
                a = self.image_size - 1
                b = 0
                c = self.image_size - 1
                d = self.image_size - 1

                critical_points = fl.find_critical_points(img, self.image_size)

                # Calculate (a, b) and (c, d)
                (a, b, c, d) = fl.linepoints(critical_points, self.image_size, 7)
                if (a == c and b == d):
                    b = 0
                    d = self.image_size - 1

                # Find direction
                direction = fl.determine_direction(a, b, c, d, self.image_size)
                primitive_direction = direction

                # Phase correction with momentum
                direction = alpha * direction + (1.0 - alpha) * previous_direction

                # Update the previous direction
                previous_direction = direction
                print(primitive_direction)

                # Save image
                if save_image:
                    list_image.append([primitive_direction, 0, primitive_img])
                if len(list_image) >= save_image_num:
                    break

                # Control the wheels
                self.rc_car_control(direction)

                itr = itr + 1
                if itr == 500:
                    run = False

            except KeyboardInterrupt:
                print("KeyboardInterrupt")
                break

        self.rc_car_cntl.stop(150)
        cv2.destroyAllWindows()

        pickle.dump(list_image, open("new_32_trainingdata_3.p", "ab"))

try:
    ser =  serial.Serial('/dev/ttyACM0',9600)
except:
    ser = serial.Serial('/dev/ttyACM1',9600)

a = SelfDriving(ser)
a.drive()
```

Code 7.2 (Self Driving, `self_driving.py`)

```
import random as rd
import math
import numpy as np

def find_critical_points(img, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = -1, -1, -1, -1, -1, -1, -1, -1
    s = imgSize - 1
    for i in range(imgSize):
        if img[i][0] == 255:
            a1 = i
            break
    for i in range(imgSize):
        if img[s - i][0] == 255:
            a2 = s - i
            break
    for i in range(imgSize):
        if img[i][s] == 255:
            b1 = i
            break
    for i in range(imgSize):
        if img[s - i][s] == 255:
            b2 = s - i
            break
    for i in range(imgSize):
        if img[0][i] == 255:
            c1 = i
            break
    for i in range(imgSize):
        if img[0][s - i] == 255:
            c2 = s - i
            break
    for i in range(imgSize):
        if img[s][i] == 255:
            d1 = i
            break
    for i in range(imgSize):
        if img[s][s - i] == 255:
            d2 = s - i
            break
    return float(a1), float(a2), float(b1), float(b2), float(c1), float(c2), float(d1), float(d2)

def cornerAC (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if a1 == 0.0 and c1 == 0.0:
        return True
    else:
        return False

def cornerBC (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if c2 == imgSize - 1 and b1 == 0:
        return True
    else:
        return False

def cornerAD (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if a2 == imgSize - 1 and d1 == 0:
        return True
    else:
        return False

def cornerBD (critical_points, imgSize):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if d2 == imgSize - 1 and b2 == imgSize - 1:
        return True
    else:
        return False

def borderA (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if a1 == -1 and a2 == -1:
        return False
```

```
    else:
        return True

def borderB (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if b1 == -1 and b2 == -1:
        return False
    else:
        return True

def borderC (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if c1 == -1 and c2 == -1:
        return False
    else:
        return True

def borderD (critical_points):
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    if d1 == -1 and d2 == -1:
        return False
    else:
        return True

def linepoints (critical_points, imgSize, lineWidth): # lineWidth is determined by experiment
    p = critical_points
    a1, a2, b1, b2, c1, c2, d1, d2 = critical_points
    s = imgSize - 1
    w = lineWidth/2.0

    # 4
    if borderA(p) and borderB(p) and borderC(p) and borderD(p):
        if cornerAC(p, imgSize) and cornerBD(p, imgSize) and not cornerAD(p, imgSize) and not corner
BC(p, imgSize):
            return a1/2, c1/2, (s + b2)/2, (s + d2)/2
        elif not cornerAC(p, imgSize) and not cornerBD(p, imgSize) and cornerAD(p, imgSize) and corn
erBC(p, imgSize):
            return b1/2, d1/2, (s + a2)/2, (s + c2)/2
        elif not cornerAC(p, imgSize) and cornerBD(p, imgSize) and cornerAD(p, imgSize) and cornerBC
(p, imgSize):
            return b1, s, s, d1
        elif cornerAC(p, imgSize) and not cornerBD(p, imgSize) and cornerAD(p, imgSize) and cornerBC
(p, imgSize):
            return 0, c2, a2, 0
        elif cornerAC(p, imgSize) and cornerBD(p, imgSize) and not cornerAD(p, imgSize) and cornerBC
(p, imgSize):
            return 0, c1, b2, s
        elif cornerAC(p, imgSize) and cornerBD(p, imgSize) and cornerAD(p, imgSize) and not cornerBC
(p, imgSize):
            return a1, 0, s, d2
        else:
            return rd.randint(0, s), rd.randint(0, s), rd.randint(0, s), rd.randint(0, s)

    # 3
    elif not borderA(p) and borderB(p) and borderC(p) and borderD(p):
        return 0, c1, s, d1
    elif borderA(p) and not borderB(p) and borderC(p) and borderD(p):
        return 0, c2, s, d2
    elif borderA(p) and borderB(p) and not borderC(p) and borderD(p):
        return a1, 0, b1, s
    elif borderA(p) and borderB(p) and borderC(p) and not borderD(p):
        return a2, 0, b2, s

    # 2
    elif not borderA(p) and not borderB(p) and borderC(p) and borderD(p):
        return 0, (c1 + c2)/2, s, (d1 + d2)/2
    elif borderA(p) and borderB(p) and not borderC(p) and not borderD(p):
        return (a1 + a2)/2, 0, (b1 + b2)/2, s
    elif borderA(p) and not borderB(p) and not borderC(p) and borderD(p):
        return (a1 + a2)/2, 0, s, (d1 + d2)/2
    elif not borderA(p) and borderB(p) and borderC(p) and not borderD(p):
        return 0, (c1 + c2)/2, (b1 + b2)/2, s
    elif borderA(p) and not borderB(p) and borderC(p) and not borderD(p):
```

```
        return 0, (c1 + c2)/2, (a1 + a2)/2, 0
    elif not borderA(p) and borderB(p) and not borderC(p) and borderD(p):
        return (b1 + b2)/2, s, s, (d1 + d2)/2

    # 1
    elif borderA(p) and not borderB(p) and not borderC(p) and not borderD(p):
        return a1, 0, a2, 0
    elif not borderA(p) and borderB(p) and not borderC(p) and not borderD(p):
        return b1, s, b2, s
    elif not borderA(p) and not borderB(p) and borderC(p) and not borderD(p):
        return 0, c1, 0, c2
    elif not borderA(p) and not borderB(p) and not borderC(p) and borderD(p):
        return s, d1, s, d2

    # 0
    else:
        return rd.randint(0, s), rd.randint(0, s), rd.randint(0, s), rd.randint(0, s)

def determine_direction(a, b, c, d, imgSize):
    fix = 16
    a = float(a)
    b = float(b)
    c = float(c)
    d = float(d)
    p = (imgSize-1)/2

    if a <= c:
        sign1 = 1
    else:
        sign1 = -1

    if (c-a)*p+(b-d)*(2*p-c)+d*(a-c) <= 0:
        sign2 = 1
    else:
        sign2 = -1

    direction_vector = sign1*np.array((b-d, c-a))*fix/math.sqrt((b-d)**2 + (c-a)**2)
    phase_vector = sign2*np.array((c-a,d-b))*abs((c-a)*p+(b-d)*(2*p-c)+d*(a-c))/((b-d)**2 + (c-
a)**2)

    # Composition of the direction vector and phase vector
    v = direction_vector + phase_vector

    # Calculate direction
    cos = v[1]/math.sqrt(v[0]**2 + v[1]**2)
    if v[0] >= 0:
        sign3 = 1
    else:
        sign3 = -1
    theta = sign3*math.acos(cos)
    direction = 2*theta/math.pi
    return direction
```

Code 7.3 (Algorithm, `fine_line.py`)

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

import numpy as np
import cv2
import serial
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

class RC_Car_Interface:
```

```
    def __init__(self,ser):
        self.ser = ser
        self.left_wheel = 0
        self.right_wheel = 0
        self.camera = PiCamera()
        self.camera.resolution = (320,320)      ## set camera resolution to (320, 320)
        self.camera.color_effects = (128,128)   ## set camera to black and white

    def set_wheel_speed(self, right_speed, left_speed, delay_time):
        #print('set right speed to ', right_speed)
        #print('set left speed to ', left_speed)
        cmd = ("R%dL%dT%d\n" % (right_speed, left_speed, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)

    def set_right_speed(self, speed, delay_time):
        #print('set right speed to ', speed)
        cmd = ("R%dL%dT%d\n" % (speed, 0, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def set_left_speed(self, speed, delay_time):
        #print('set left speed to ', speed)
        cmd = ("R%dL%dT%d\n" % (0, speed, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def stop(self, delay_time):     # robot stop
        #print('stop')
        cmd = ("R%dL%dT%d\n" % (0, 0, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def get_image_from_camera(self, image_size):
        img = np.empty((320, 320, 3), dtype=np.uint8)
        self.camera.capture(img, 'bgr')

        ## 3 dimensions have the same value because camera is set to black and white
        ## remove two dimension data
        img = img[:,:,0]

        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,(image_size, image_size), interpolation=cv2.INTER_AREA )
        # cv2.imshow("Image", img2)
        # cv2.waitKey(0)
        return img2
```

Code 7.4 (RC Car Interface, `rc_car_interface.py`)

7.3 라즈베리 파이 인공지능 자율주행 코드

```
# 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
import math

class SelfDriving:

    def __init__(self, ser):
        self.rc_car_cntl = RC_Car_Interface(ser)
        self.dnn_driver = DNN_Driver()

        self.rc_car_cntl.set_left_speed(0, 150)
        self.rc_car_cntl.set_right_speed(0, 150)

        self.velocity = 0
        self.direction = 0
        self.image_size = 32

        self.dnn_driver.tf_load("model.tflite")


    def rc_car_control(self, direction):
        sensitivity = 150
        left_power = 0.95
        right_power = 0.95

        ## Calculate left and right wheel speed with direction
        if direction < -1.0:
            direction = -1.0
        if direction > 1.0:
            direction = 1.0

        #print("direction: ", direction)
        if (direction <= 0.0):
            right_speed = int(((255 - sensitivity) + int((1.0)*sensitivity))*right_power)
            left_speed = int(((255 - sensitivity) + int((1.0 + direction)*sensitivity))*left_power)
        else:
            right_speed = int(((255 - sensitivity) + int((1.0 - direction)*sensitivity))*right_power
)
            left_speed = int(((255 - sensitivity) + int((1.0)*sensitivity))*left_power)

        self.rc_car_cntl.set_wheel_speed(right_speed, left_speed, 150)


    def drive(self):
        previous_direction = 0
        alpha = 0.85
        while True:
            img = self.rc_car_cntl.get_image_from_camera(self.image_size)

            # Image formatting
            img = img[..., np.newaxis].astype(np.float32)

            # Predict with single image
            direction = self.dnn_driver.predict_direction(img)

            # Linear transformation from [0,180] to [1,-1]
            direction = ((-1)/90)*direction + 1

            # Phase correction with momentum
            direction = alpha*direction + (1-alpha)*previous_direction

            # Update the previous direction
            previous_direction = direction

            # Control the wheels
            self.rc_car_control(direction)

        self.rc_car_cntl.stop(150)
        cv2.destroyAllWindows()
```

```
try:
    ser =  serial.Serial('/dev/ttyACM0',9600)
except:
    ser = serial.Serial('/dev/ttyACM1',9600)

a = SelfDriving(ser)
a.drive()
```

Code 7.5 (Self Driving, `self_driving.py`)

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

import numpy as np
import cv2
import serial
import time
from picamera.array import PiRGBArray
from picamera import PiCamera

class RC_Car_Interface:

    def __init__(self,ser):
        self.ser = ser
        self.left_wheel = 0
        self.right_wheel = 0
        self.camera = PiCamera()
        self.camera.resolution = (320,320)      ## set camera resolution to (320, 320)
        self.camera.color_effects = (128,128)   ## set camera to black and white

    def set_wheel_speed(self, right_speed, left_speed, delay_time):
        #print('set right speed to ', right_speed)
        #print('set left speed to ', left_speed)
        cmd = ("R%dL%dT%d\n" % (right_speed, left_speed, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)

    def set_right_speed(self, speed, delay_time):
        #print('set right speed to ', speed)
        cmd = ("R%dL%dT%d\n" % (speed, 0, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def set_left_speed(self, speed, delay_time):
        #print('set left speed to ', speed)
        cmd = ("R%dL%dT%d\n" % (0, speed, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def stop(self, delay_time):     # robot stop
        #print('stop')
        cmd = ("R%dL%dT%d\n" % (0, 0, delay_time)).encode('ascii')
        #print("My cmd is %s" % cmd)
        self.ser.reset_output_buffer()
        self.ser.write(cmd)


    def get_image_from_camera(self, image_size):
        img = np.empty((320, 320, 3), dtype=np.uint8)
        self.camera.capture(img, 'bgr')
```

```
        ## 3 dimensions have the same value because camera is set to black and white
        ## remove two dimension data
        img = img[:,:,0]

        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,(image_size, image_size), interpolation=cv2.INTER_AREA )
        # cv2.imshow("Image", img2)
        # cv2.waitKey(0)
        return img2
```

Code 7.6 (RC Car Interface, `rc_car_interface.py`)

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

import tflite_runtime.interpreter as tflite

import numpy as np

class DNN_Driver():
    def __init__(self):
        self.model = None
        self.interpreter = None

    def tf_load(self, path):
        self.interpreter = tflite.Interpreter(model_path = path)
        self.interpreter.allocate_tensors()

    def predict_direction(self, img):
        input_details = self.interpreter.get_input_details()
        output_details = self.interpreter.get_output_details()
        input_shape = input_details[0]["shape"]
        img = img.reshape((1,)+img.shape)
        self.interpreter.set_tensor(input_details[0]["index"], img)
        self.interpreter.invoke()
        ret = self.interpreter.get_tensor(output_details[0]["index"])
        return ret
```

Code 7.7 (AI model, `tf_learn.py`)

7.3 텐서플로우 학습 코드

```
import pickle
import math
import numpy as np
from sklearn.model_selection import train_test_split

data = []
for i in range(1, 4):
    objects = []

    with  open(f"new_32_trainingdata_{i}.p", "rb") as file:
        while True:
            try:
                objects.append(pickle.load(file))
            except (pickle.UnpicklingError, EOFError) as e:
                break
```

```
        for i in range(len(objects)):
            data.extend(objects[i])

## except degree of 180
data_except = []
for i in range(len(data)):
    if data[i][0] == -1: pass
    else: data_except.append(data[i])

print(len(data), len(data_except))

training, test = train_test_split(data_except, test_size=0.33, random_state=42)

def get_training_data(use_cnn=True):
    if use_cnn:
        trX = np.array([a[2][..., np.newaxis].astype(np.int16) for a in training])
        trY = np.array([(a[0] - 1) *(-90) for a in training])
    else:
        trX = np.array([np.reshape(a[2],a[2].shape[0]**2) for a in training])
        trY = np.array([(a[0] - 1) *(-90) for a in training])
    return trX, trY

def get_test_data(use_cnn=True):
    if use_cnn:
        teX = np.array([a[2][..., np.newaxis].astype(np.int16) for a in test])
        teY = np.array([(a[0] - 1) *(-90) for a in test])
    else:
        teX = np.array([np.reshape(a[2],a[2].shape[0]**2) for a in test])
        teY = np.array([(a[0] - 1) *(-90) for a in test])
    return teX, teY

trX,trY = get_training_data(use_cnn=True)
teX,teY = get_test_data(use_cnn=True)
```

Code 7.8 (Data Set Refine)

```
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten, BatchNormalization, Dropout, ReLU
from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
import tensorflow as tf
import pickle

outputs = 1
use_cnn= True

# from get_image_data import *

trX,trY = get_training_data(use_cnn)
teX,teY = get_test_data(use_cnn)

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

if use_cnn:
    model = Sequential()
    model.add(Conv2D(8, (2, 2), strides=(2, 2), padding='same', activation='relu', input_shape=(32,
32, 1)))
    model.add(Dropout(0.1))
    model.add(BatchNormalization())

    model.add(Conv2D(16, (2, 2), strides=(2, 2), padding='same', activation='relu'))
    model.add(Dropout(0.1))
    model.add(BatchNormalization())

    model.add(Conv2D(32, (2, 2), padding='same', activation='relu'))
```

```
    model.add(Dropout(0.1))
    model.add(BatchNormalization())

    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.1))
    model.add(BatchNormalization())

    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.1))
    model.add(BatchNormalization())

    model.add(Dense(1))

model.summary()

optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(loss='mean_squared_error', optimizer=optimizer)

hist = model.fit(trX, trY, epochs=150, batch_size=100,
                 validation_data=(teX, teY))

Y_prediction = model.predict(teX).flatten()

with open("predict.txt", 'w') as file:
    for i in range(Y_prediction.shape[0]):
        label = teY[i]
        pred = Y_prediction[i]
        print("label:{:.3f}, pred:{:.3f}".format(label, pred))
        file.write("label:{:.3f}, pred:{:.3f}\n".format(label, pred))
model.summary()

model.save("new_pt_model_cnn.h5")
```

Code 7.9 (Model Training and Save the Model)

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

with open("model.tflite", 'wb') as f:
    f.write(tflite_model)
```

Code 7.10 (Model Conversion from .h5 to .tflite)