

자율 주행 버스

2019. 05. 23.

작 성 자 : 김기홍 / 박두원 / 김경연 / 박준영

지도교수 : 방갑산

차 례

- I. 프로젝트 개요
- II. 시스템 구성
- III. 각 모듈별 동작 원리
- IV. 모듈별 설계
- V. 전체 시스템 설계
- VI. 제작내용 (회로도, 소스코드 등 첨부)
- VII. 결과물 설명 (결과물 관련 이미지(사진) 첨부)
- VIII. 프로젝트 수행 결과 분석
 - 1. 재학중 취득한 기초지식의 활용 내용
 - 2. 재학중 취득한 실험지식의 활용 내용
 - 3. 본 프로젝트 수행과정에서의 설계 능력 향상 내용
 - 4. 본 프로젝트 수행과정에서의 문제 해결 내용
 - 5. 본 프로젝트 수행과정에서의 실무 능력 향상 내용
 - 6. 본 프로젝트 수행과정에서의 팀원간 협동 내용
 - 7. 개발된 결과물에 대한 전시 방법 계획

< 참고 문헌 >

I. 프로젝트 개요

자율주행 자동차란 운전자의 개입 없이 주변 환경을 인식하고, 주행 상황을 판단하여, 차량을 제어함으로써 스스로 주어진 목적지까지 주행하는 자동차를 말한다.

수많은 기업들이 자율주행차량의 연구와 개발에 투자하고 있다. 메르세데스 벤츠나 현대자동차와 같은 자동차기업들 뿐만 아니라, 구글과 KT와 같은 IT기업들도 이미 자율주행차 산업에 뛰어들었다.

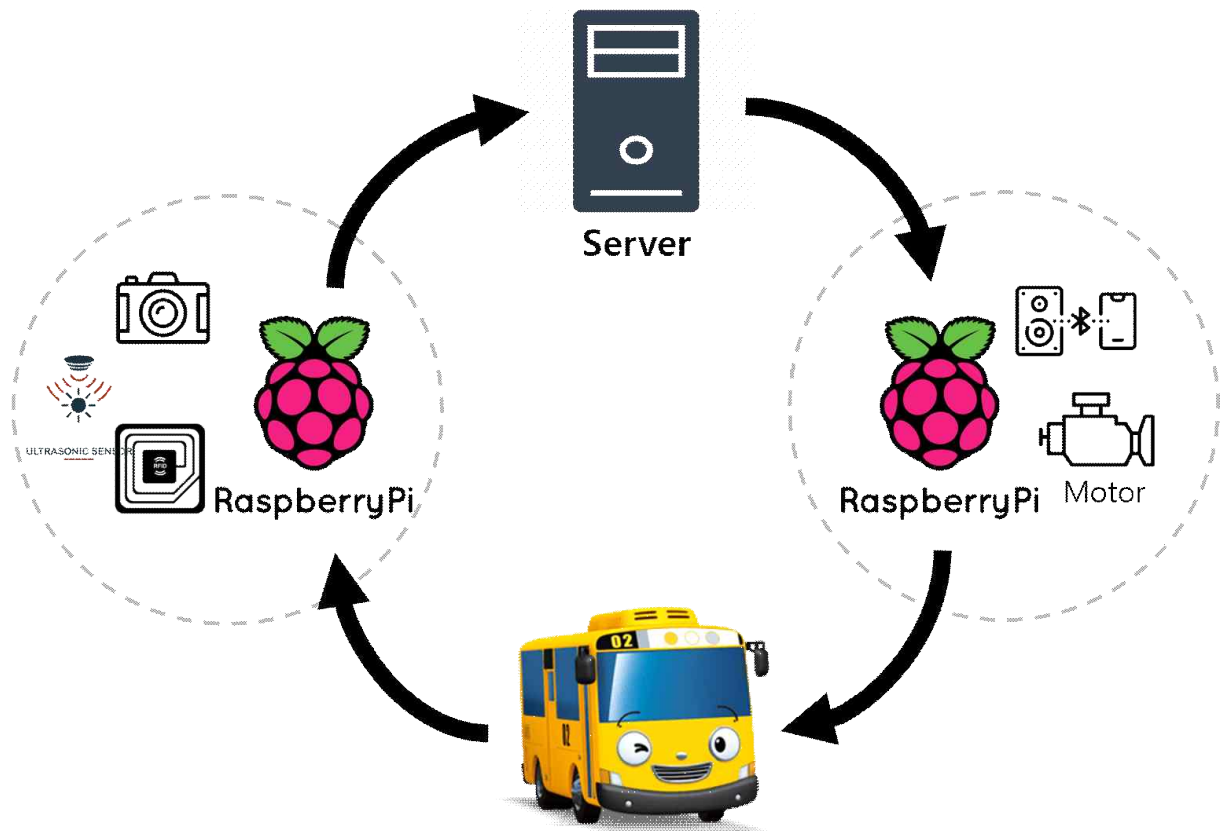
특히 2019년 4월에 최초로 5G 이동통신망이 대한민국에서 상용화되면서 자율주행 기술도 급속도로 발전될 것으로 예상된다. 기존 이동통신망에서는 지연시간이 길어 데이터를 모두 자동차에 넣어야 했지만, 5G의 등장 이후 전송시간이 단축되기 때문에 중앙 데이터베이스에서 빠른 시간 안에 정보를 주고받을 수 있기 때문에 사고 확률도 내려갈뿐더러 빅데이터 수집에도 용이하기 때문에 자율주행 기술의 발전 속도에 탄력이 붙었다. 5G의 등장으로 국내에선 KT와 SKT가 5G 기반 자율주행차량의 시장을 선점하기 위해 경쟁하고 있다.

이렇듯 자동차산업은 더 이상 자동차공학과만의 전유물이 아니다. 현재 도로에 달리고 있는 차량들만 해도 자율주행차는 아니지만, 대부분 automotive computer를 내장하고 있다. 그래서 우리 조는 전자정보공학도로서 4차 산업혁명 시대를 향해 거대한 시장을 형성하고 있는 자율주행차를 구현해 보고자 본 프로젝트를 계획하였다.


우리 조는 자율주행 자동차 중에서도 자율주행 기술을 대중교통에 접목시킨 무인 버스를 제작해 보고자 하였다. 자율주행 버스는 교통사고의 감소 및 효율적인 연료 절약 운행이 가능하게 하여 미래에는 친환경적인 대중교통을 현실화 할 것이다.

본 프로젝트를 수행함으로써 미래기술에 대한 지식을 습득하고, 실생활에 도움이 될 수 있도록 연구 및 개발을 해보았다. 카메라를 활용해 전방의 표지판을 학습하여 인식하고, 차선을 인식하여 자율주행이 가능하도록 설계해보았다. RFID를 활용하여 보다 안정적이게 정류장을 인식시키도록 설계하였고, 초음파센서를 추가하여 돌발 상황에서도 장애물을 인식하여 정지하게 하는 등 단순히 카메라 인식만으로는 위험할 수 있는 자율주행을 보완해보았다.

II. 시스템 구성



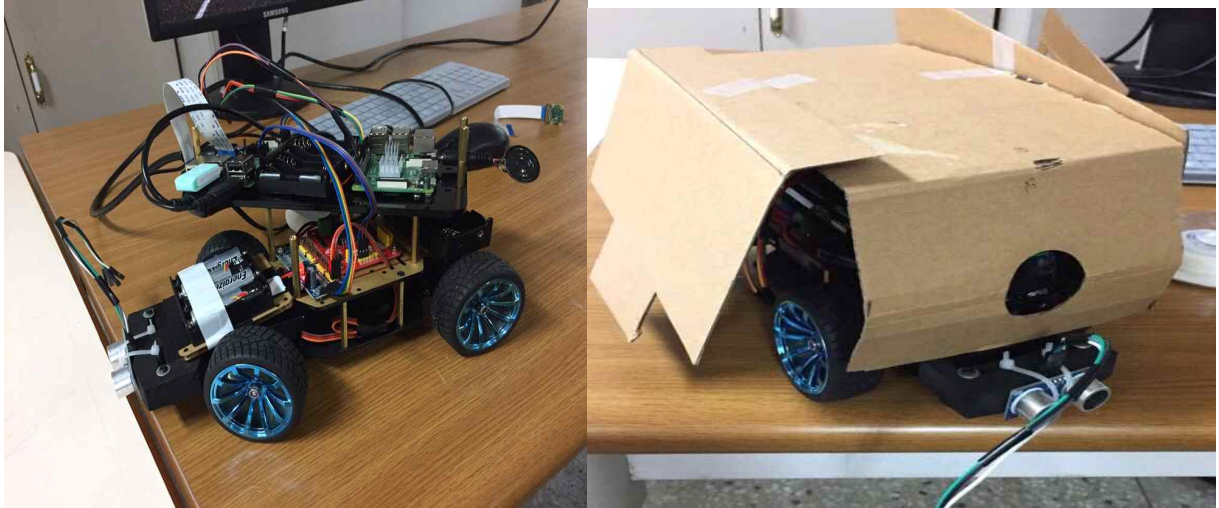
III. 각 모듈별 동작 원리

모듈	설명	사진
Raspberrypi 3 Model B	LINUX 기반의 운영체제인 Raspbian을 OS로 설치하여 사용하였고, 버스의 전반적인 제어를 담당	
Raspberrypi Camera V2	자율 주행 버스 전방 시야 확보 및 영상처리를 위한 장비	

L298 Motor Driver	DC Motor의 역전압 방지 및 전압을 제어, 정방향 또는 역방향으로 모터 회전방향과 속도를 제어	
JGA25-370 DC Motor	전, 후진을 위한 뒷바퀴 동력 전달	
16-channel 12-bit PWM PCA9685 Servo Driver	라즈베리파이로부터 PWM 값을 입력받아 RC가 앞바퀴의 각도를 조절하여 Servo Motor 제어	
MG995 Servo Motor	좌, 우 방향전환을 위한 앞바퀴 제어.	
RFID-RC522	통신을 이용해 버스 정류장 인식	
모듈	설명	사진
HC-SR04 초음파 센서	버스 전방의 물체 확인 및 물체와의 거리 측정	
JBL GO Bluetooth Speaker	차량의 운행 상태를 음성으로 안내	
노트북	AI Server 역할	

IV. 모듈별 설계

1. H/W



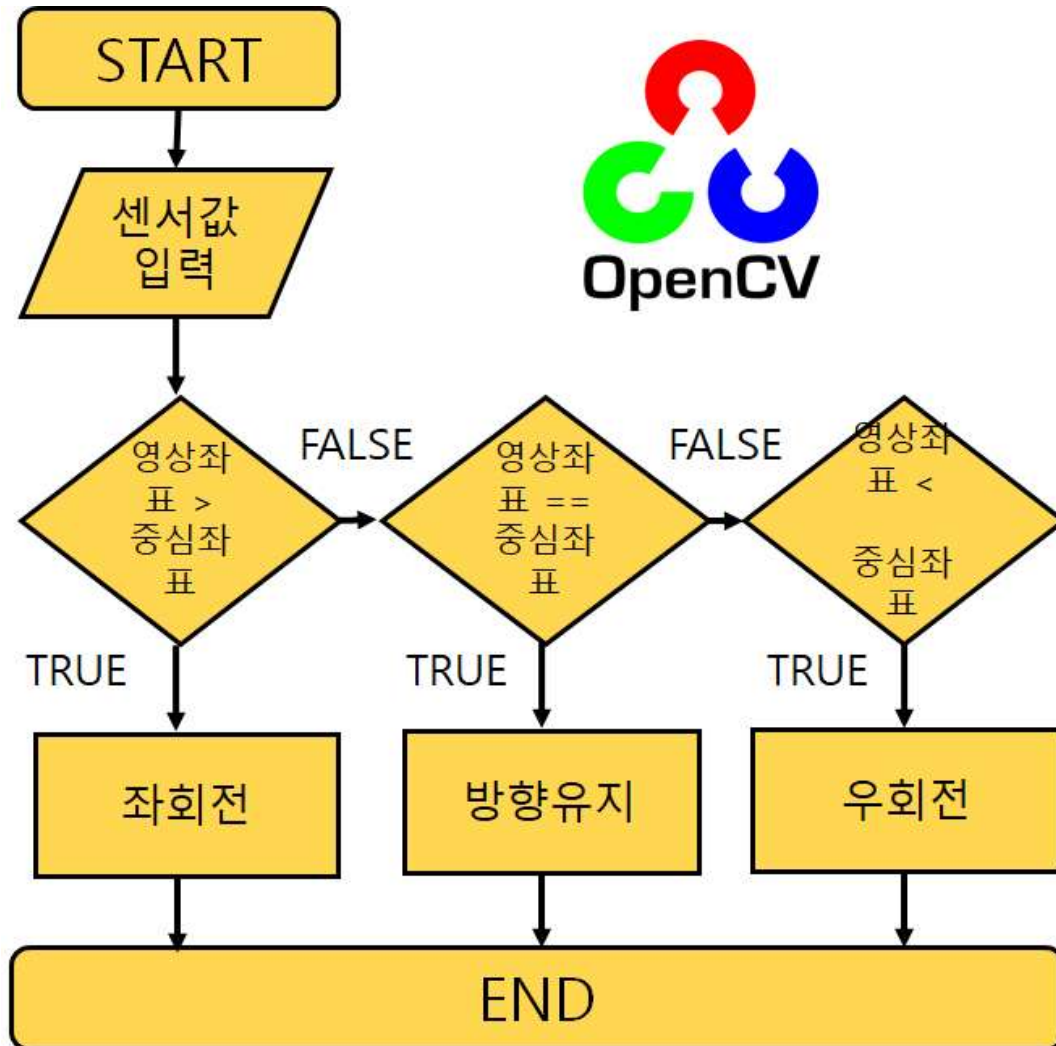
총 3단으로 구성하였다. 차량 앞면에는 초음파센서를 부착하였고, PCA9685 Servo Driver의 전원인 배터리 홀더를 설치하였다. 1단에는 차량을 구동하는 DC Motor와 Servo Motor를 장착하였고, 2단에는 모터들과 연결되는 L298 Motor Driver와 PCA9685 Servo Driver를 부착하였다. 2단 뒷면에는 L298 Motor Driver의 전원인 배터리 홀더를 부착하였다. 3단에는 파이카메라와 센서파이와 모터파이, 그리고 각 라즈베리파이에 전원을 공급하는 홀더를 장착했다.

차량의 구동을 담당한 JGA25-370 DC Motor와 조향을 담당한 MG995 Servo Motor, 그리고 전체 차체의 무게를 고려했을 때 각 부품들의 동작전압이 높을 것으로 예상되고, 보다 속도와 방향을 쉽게 제어하기 위해 DC Motor에는 Motor Driver, Servo Motor에는 Servo Driver가 필요하다고 판단했다. 모터의 속도는 모터드라이버가 PWM_Pulse Width Modulation을 사용하여 전압을 제어하여 조절할 수 있다. 전압의 크기에 비례해 속도가 증가하고 감소한다.

모터파이에서 차량을 구동하는 명령을 보내어 L298 Motor Driver가 뒷바퀴에 연결된 DC Motor를 제어하여 전진과 후진, 속도조절을 담당하게 하였다.

방향 조종은 앞바퀴가 담당한다. 두 개의 앞바퀴를 하나의 스티어링 축으로 연결하였고, 그 축을 Servo Motor에 물리게 하였다. 모터파이에서 차량을 조향하는 명령을 PCA9685 Servo Driver에 보내고 Servo Driver가 Servo Motor를 제어하게 하였다.

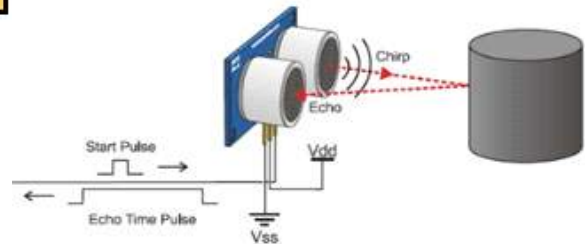
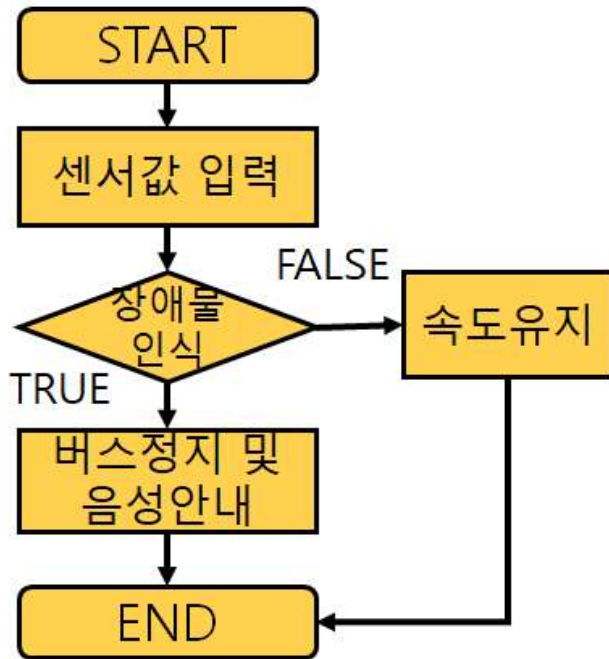
2. 자율주행



컴퓨터 영상처리 프로그래밍을 위해 OpenCV를 이용하여 차선인식을 하여 주행하도록 구현하였다. 파이카메라를 통해 촬영된 영상 값에 Garyscale 등의 필터를 적용하고, 필터링 된 영상에 Canny 알고리즘을 통해 이진화된 영상을 사용하였다.

허프변환 함수를 이용하여 차선으로 예상되는 선을 검출하여 검출된 선을 각도에 따라 좌, 우측 차선으로 구분하여 최종적인 차선을 검출하며 좌, 우측 차선의 중앙값을 기반으로 차량수행을 하도록 하였다. 영상좌표가 중심좌표보다 크면 좌회전, 영상좌표와 중심좌표가 같으면 방향유지, 영상좌표가 중심좌표보다 작으면 우회전을 수행하도록 구현하였다.

3. 장애물 인식

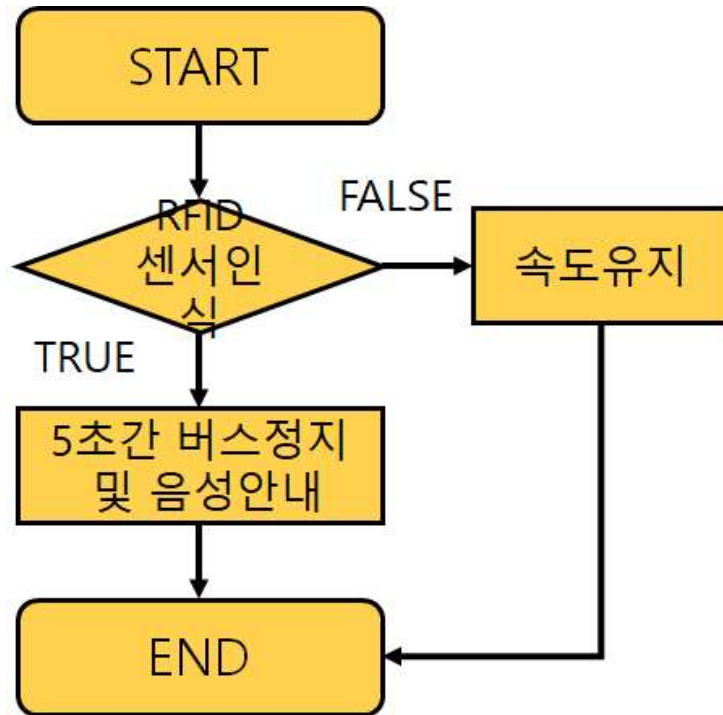


초음파 센서를 이용하여 전방의 장애물 및 외벽을 인식하면 버스가 장애물이 사라질 때까지 정지하도록 구현하였다.

초음파 센서는 초음파를 이용하여 범위 내에 사물을 감지할 수 있는 센서이다. 작동원리로는 Transmitter에서 초음파를 방출하고, Receiver에서 부딪혀 되돌아오는 초음파를 받아들여 발신시점과 수신시점의 시간차를 계산하여 초음파 센서로부터 전방의 물체까지의 떨어진 거리를 측정한다.

실제 교통 상황에서 주행 중 충돌은 안전과 직결된 상황이기 때문에 처리속도가 빨라야 하는 만큼 초음파 센서는 센서파이에 GPIO로 유선 연결되어 있다. 초음파 센서가 프로그래밍으로 설정해주는 거리 이내에 사물을 장애물로 인식하면 센서파이로 정보를 보내고, 센서파이는 통신을 통해 모터파이에게 정보를 보내 DC모터를 정지하도록 제어시킨다.

4. 버스 정류장 인식



RFID 모듈을 이용하여 버스정류장을 인식하며 버스정류장 인식 시 버스가 5초간 정차하도록 구현하였다.

RFID는 무선주파수를 이용하여 RFID 태그의 ID를 식별하는 장치로 전자태그라고도 불린다. 작동원리로는 RFID 태그에 부착된 IC칩에 저장되어 있는 데이터를 무선 주파수를 이용하여 RFID 리더기가 비접촉식 방법으로 판독해 내서 데이터를 처리한다.

우리 조는 차량 바닥에 RFID-RC522 리더기를 설치하고 센서파이의 GPIO에 유선 연결 하였다. 도로에 RFID태그(카드형)을 두어 버스가 지나갈 때 인식하도록 하였고, 인식 시 통신을 이용해 정보를 보내 모터 제어를 실행한다.

V. 전체 시스템 설계

본 자율주행버스 프로젝트는 주변 상황에 대한 인식(이하 센서파이)과 인식에 대한 판단(AI Server), 판단을 통한 차량 제어(이하 모터파이)를 수행하도록 하여 자율주행 버스를 구현하였다.

기본적으로 Raspberrypi Camera V2가 인식한 차량 전방의 차선을 따라 주행할 것이다. 차선을 주행하는 것 보다 우선적으로 HC-SR04 초음파센서가 장애물을 인식하면 정지하고 장애물이 사라지면 다시 주행을 하도록 한다. 버스 정류장 구간에서는 버스의 장착된 RFID-RC522 리더기와 버스 정류장이 통신하면 버스가 5초간 정차하고, 시간이 지나면 다시 주행하게 된다. 이렇게 인식파이에 연결된 파이카메라, 초음파센서, RFID가 주변 환경에 대하여 인식한 정보를 소켓 통신을 이용하여 AI Server로 보내준다.

AI Server에서는 앞서 말한 센서파이에서 수신한 정보를 바탕으로 알맞은 구동을 하도록 연산을 하고 결과 값을 역시 소켓 통신을 이용하여 모터파이에 명령한다. Raspberry Pi 장비만으로는 앞서 인식한 정보들을 처리하고 연산하는 데에 시간이 오래 걸리기 때문에, 성능이 좋은 데스크톱이나 랩톱을 사용하여 차량의 인식속도가 차량의 위치 변화 속도를 따라잡게 하기 위함이다.

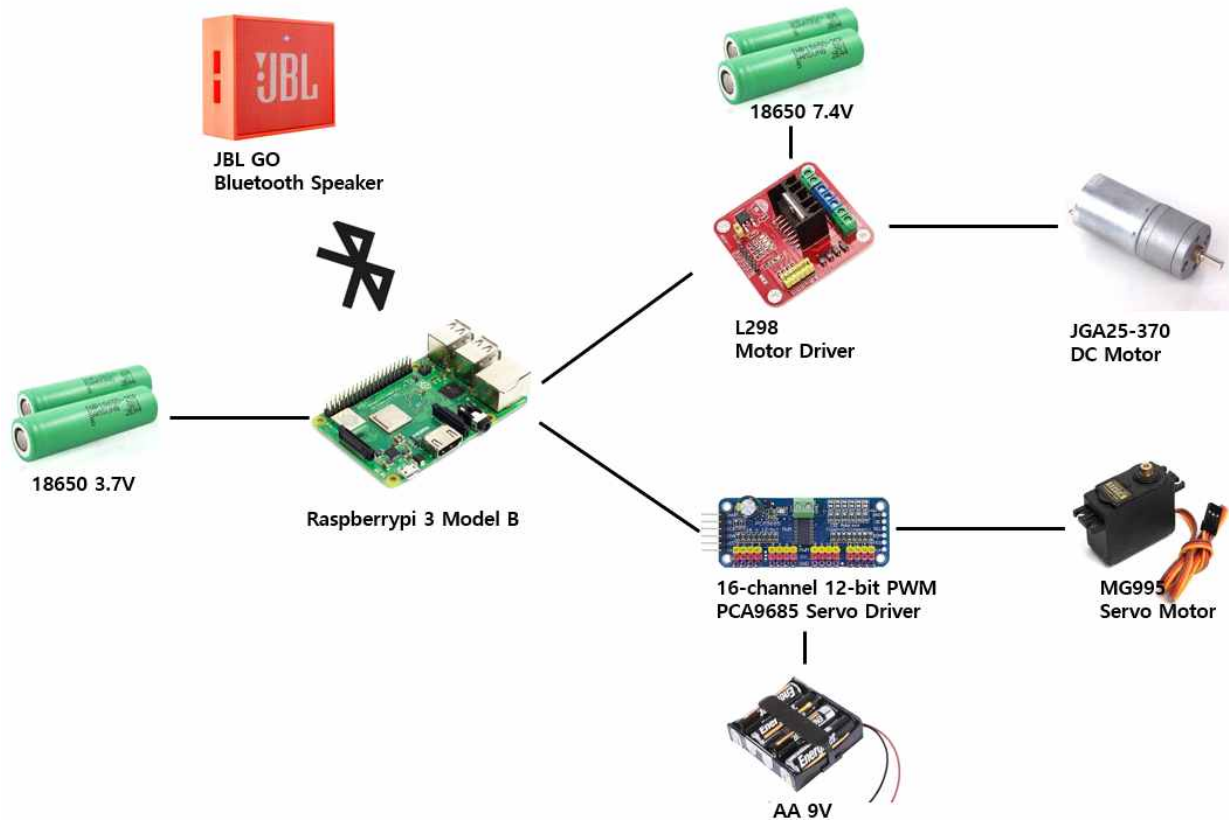
마지막으로 모터파이는 모터를 제어하여 주행해야할 차선에 알맞게, 그리고 정지해야할 상황에 알맞게 방향과 속도를 조정하게 하도록 설계하였고 블루투스 스피커를 활용해 음성안내를 추가해보았다.

차량 이동으로 인해 다시 바뀐 주변 환경을 센서파이가 인식하며 위의 일련의 과정을 반복함으로써 버스를 주행한다.

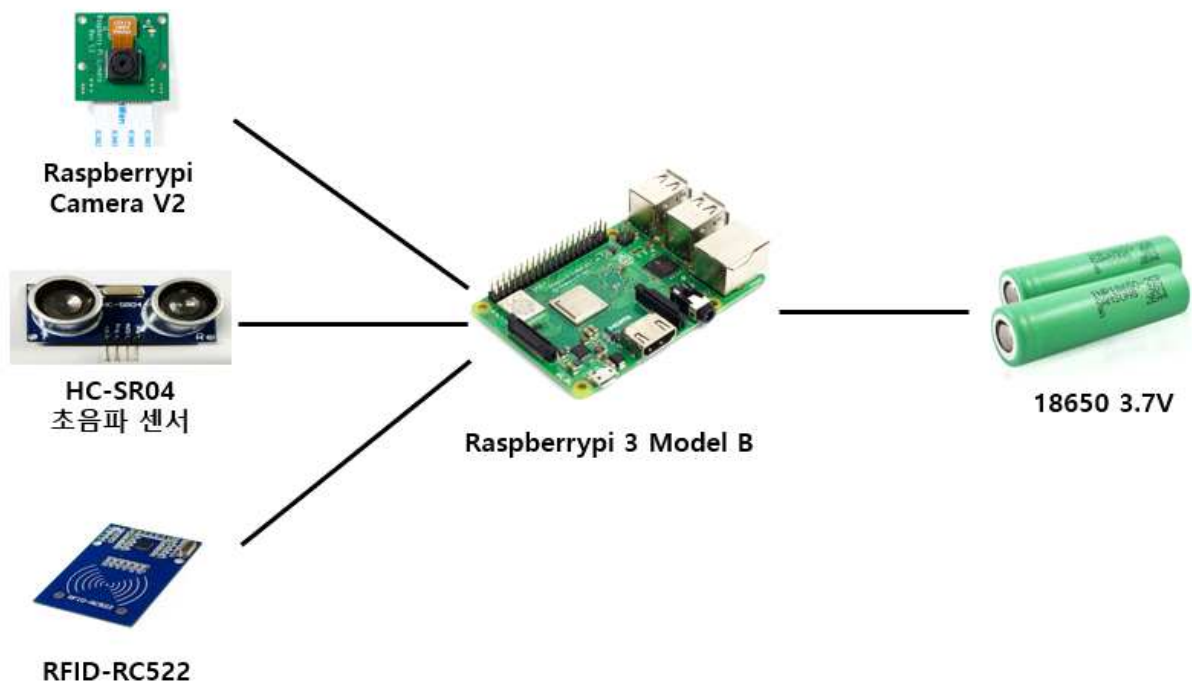
VI. 제작내용 (회로도, 소스코드 등 첨부)

1. H/W 전체 회로도

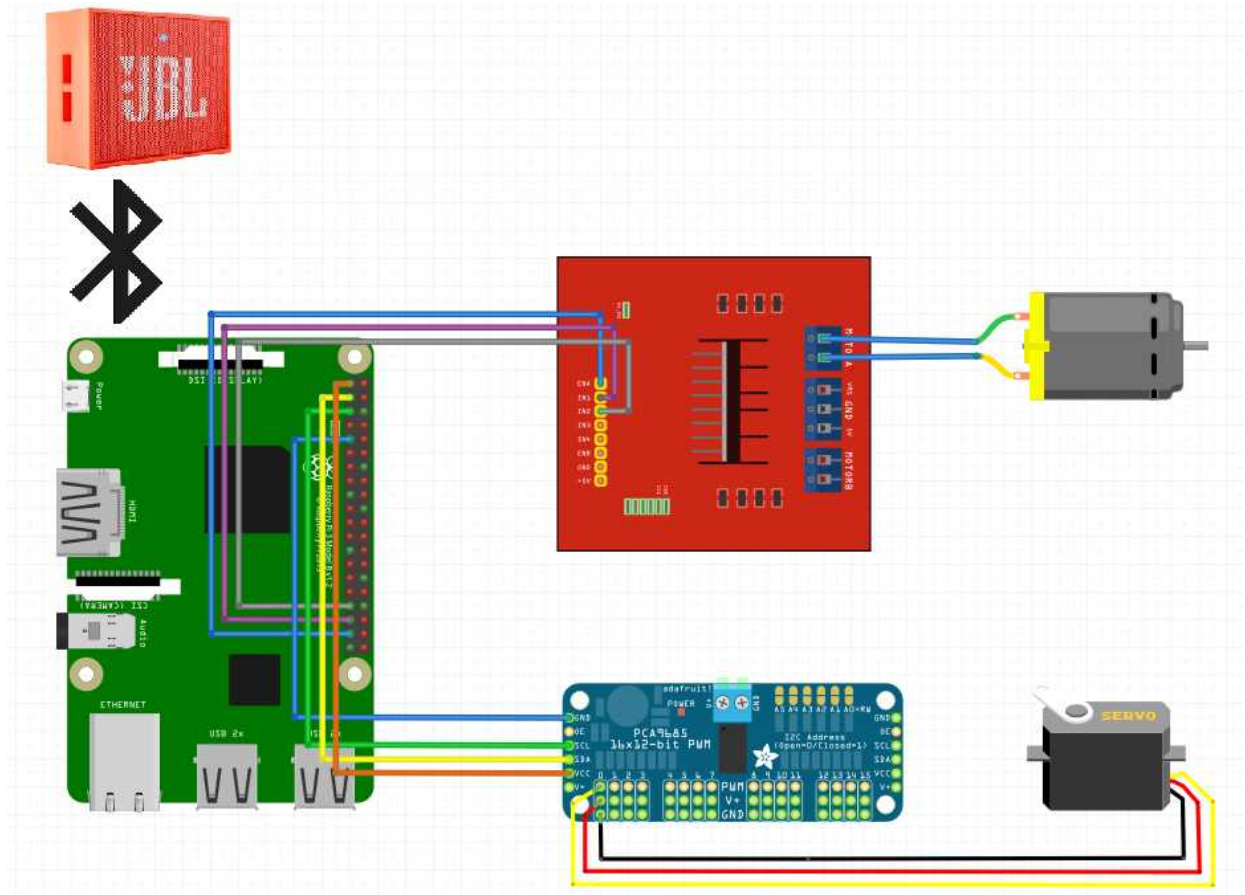
1) Motor Pi



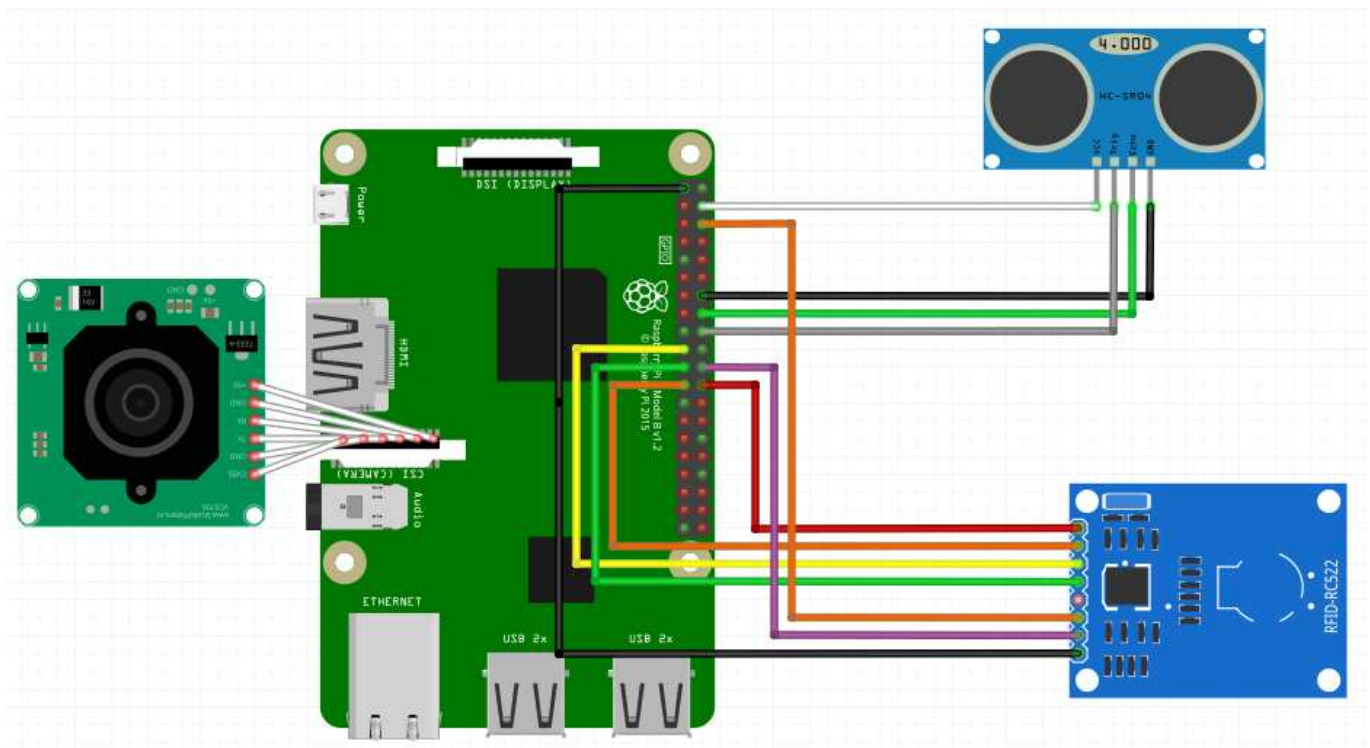
2) Sensor Pi



2. MotorPi 회로도



3. SensorPi 회로도



소스코드

<line_strem.py>

```
import cv2
```

```
import socket
```

```
ip = 'HOST IP'
```

```
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
server_address = (ip, 2000)
```

```
sock.connect(server_address)
```

```
width, height = 480, 360
```

```
cap = cv2.VideoCapture(0)
```

```
while cap.isOpened():
```

```
    ret, frame = cap.read()
```

```
    frame = cv2.resize(frame, (width, height))
```

```
    #frame = cv2.flip(frame, -1) # 영상 뒤집기
```

```
    try:
```

```
        frame, angle = detect_lane(frame)
```

```
        print(angle)
```

```
        sock.send(angle.encode())
```

```
    except:
```

```
        pass
```

```
    cv2.imshow('detect', frame)
```

```
    if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
        break
```

```
cv2.destroyAllWindows()
```

```
cap.release()
```

```

<RFID.py>
from __future__ import division
import RPi.GPIO as GPIO
import socket
import time
import pygame
import MFRC522
from multiprocessing import Process, Queue

pygame.mixer.init(17000)
pygame.mixer.music.load('arrive.mp3')
pygame.mixer.music.set_volume(1)

ip = 'HOST IP'
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (ip, 1001)
sock.connect(server_address)
print("sensor connected...")

MIFAREReader = MFRC522.MFRC522()
while True:
    (status, TagType) = MIFAREReader.MFRC522_Request(MIFAREReader.PICC_REQIDL)
    if status == MIFAREReader.MI_OK:
        print("도착했습니다")
        pygame.mixer.music.play()
        while pygame.mixer.music.get_busy():
            continue
        sock.send("stop".encode())
        time.sleep(3)
    else :
        sock.send("departure".encode())
        time.sleep(3)
GPIO.cleanup()

```

<ultra.py>

```
from __future__ import division
import RPi.GPIO as GPIO
import socket, time
import pygame
```

```
ip = 'HOST IP'
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server_address = (ip, 1002)
sock.connect(server_address)
print("sensor connected...")
pygame.mixer.init(17000)
pygame.mixer.music.load('stop.mp3')
pygame.mixer.music.set_volume(1)
```

```
GPIO.setmode(GPIO.BOARD)
```

```
try :
```

```
    trig = 18
    echo = 16
    GPIO.setup(trig, GPIO.OUT)
    GPIO.setup(echo, GPIO.IN)
    print("초음파 센서")
```

```
    while True:
```

```
        GPIO.output(trig, False)
        time.sleep(0.5)
        GPIO.output(trig, True)
        time.sleep(0.00001)
        GPIO.output(trig, False)
```

```
        while GPIO.input(echo) == 0:
```

```
            pulse_s = time.time()
```

```
        while GPIO.input(echo) == 1:
```

```
            pulse_e = time.time()
```

```
duration = pulse_e - pulse_s
distance = duration * 17000
distance = round(distance, 2)
if (distance < 15):
    sock.send("stop".encode())
    print("장애물이 감지되었습니다!")
    pygame.mixer.music.play()
    while pygame.mixer.music.get_busy():
        continue
else:
    sock.send("departure".encode())
    print("거리 : ", distance)

except :
    GPIO.cleanup()
```



```

<lanedetect.py>
# -*- coding: utf-8 -*-
import cv2
import numpy as np
import socket
from time import sleep

# 영상 회전 함수
def rotate(frame, angle):
    matrix = cv2.getRotationMatrix2D((width/2, height/2), angle, 1) # 회전 중심
    dst = cv2.warpAffine(frame, matrix, (width, height))
    return dst

# ROI
def ROI(frame, vertices, color3 = (255, 255, 255), color1 = 255):
    mask = np.zeros_like(frame) # img 크기와 같은 이미지(배열)

    if len(frame.shape) > 2: # 컬러 이미지(3채널)일 때
        color = color3
    else: # 흑백 이미지(1채널)일 때
        color = color1

    # mask에 vertices 점들로 이뤄진 다각형 부분을 color로 채움
    cv2.fillPoly(mask, vertices, color)

    # 이미지와 ROI를 합성
    ROI_img = cv2.bitwise_and(frame, mask)
    return ROI_img

# 선 그리기 함수
def draw_line(frame, lines, color = [0, 0, 255], thickness = 10):
    cv2.line(frame, (lines[0], lines[1]), (lines[2], lines[3]), color, thickness)

# 대표선 추출 함수

```

```

def get_fitline(frame, f_lines):
    lines = np.squeeze(f_lines)
    lines = lines.reshape(lines.shape[0] * 2, 2) # 좌표값으로 변환
    vx, vy, x, y = cv2.fitLine(lines, cv2.DIST_L2, 0, 0.01, 0.01)

    x1, y1 = int(((frame.shape[0] - y) / vy * vx + x) , int(frame.shape[0] - 20)
    x2, y2 = int(((frame.shape[0] / 2 + 100) - y) / vy * vx + x) ,
int(frame.shape[0] / 2 + 100 - 20)

    result = [x1, y1, x2, y2]
    #cv2.circle(img, (x1, y1), 5, (0, 0, 255), -1) # -1이면 내부 채워짐
    return result, x, y

def intersaction(L_line, R_line):
    m1 = (L_line[3] - L_line[1]) / (L_line[2] - L_line[0])
    m2 = (R_line[3] - R_line[1]) / (R_line[2] - R_line[0])

    #line1 = m1(x - L_line[0]) + L_line[1]
    #line2 = m2(x - R_line[0]) + R_line[1]

    x = int((R_line[1] - L_line[1] + L_line[0] * m1 - R_line[0] * m2) / (m1 - m2))
    y = int(m1 * (x - L_line[0]) + L_line[1])
    return (x, y)

def detect_lane(img):
    gray_img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) # 흑백 이미지로 변환
    #cv2.imshow("gray", gray_img)

    blur_img = cv2.GaussianBlur(gray_img, (3, 3), 0) # Blur 효과
    #cv2.imshow("blur", blur_img)

    kernel = np.ones((5,5), np.uint8)
    dilation_img = cv2.dilate(blur_img, kernel, 1) # 팽창 연산

```

```

#cv2.imshow("dilation", dilation_img)

canny_img = cv2.Canny(dilation_img, 200, 300) # 외곽선 검출
#cv2.imshow("canny", canny_img)

vertices = np.array([(0, height * 3 / 2), (0, height - 30), (width, height - 30),
(width, height * 3 / 2)], dtype = np.int32)
ROI_img = ROI(canny_img, vertices) # ROI 설정
img = cv2.polylines(img, vertices, True, (255, 0 ,0), 5)

line_arr = cv2.HoughLinesP(ROI_img, 1, np.pi / 180, 10, 12, 10) # 직선 검출
line_arr = np.squeeze(line_arr) # 불필요한 값 정리

'''
print(line_arr)
for i in range(0, len(line_arr)):
    l = line_arr[i, :]
    cv2.line(img, (l[0], l[1]), (l[2], l[3]), (0, 0, 255), 4)
    cv2.circle(img, (l[0], l[1]), 5, (0, 255, 0), -1)
cv2.imshow("hough", img)
'''

slope_degree = np.arctan2(line_arr[:, 1] - line_arr[:, 3], line_arr[:, 0] - line_arr[:,
2]) * 180 / np.pi # 기울기 구하기 arctan(y/x)

# 수평 기울기 제한
line_arr = line_arr[np.abs(slope_degree) < 170]
slope_degree = slope_degree[np.abs(slope_degree) < 170]

# 수직 기울기 제한
line_arr = line_arr[np.abs(slope_degree) > 120]
slope_degree = slope_degree[np.abs(slope_degree) > 120]

L_lines, R_lines = line_arr[(slope_degree > 0), :], line_arr[(slope_degree < 0), :]

```

```
if len(L_lines) == 0 or len(R_lines) == 0 : # 양쪽 차선 기울기 같거나 한쪽 차선만 인식될 때
```

```
    lines = line_arr[:, None]
    fit_line, x, y = get_fitline(img, lines)
    draw_line(img, fit_line)
    angle = np.arctan(fit_line)
    angle = mean(angle)
```

```
else :
```

```
    L_lines, R_lines = L_lines[:, None], R_lines[:, None]
```

```
    # 대표선 구하기
```

```
    left_fit_line, lx, ly = get_fitline(img, L_lines)
    right_fit_line, rx, ry = get_fitline(img, R_lines)
```

```
    # 대표선 그리기
```

```
    draw_line(img, left_fit_line)
    draw_line(img, right_fit_line)
```

```
    inter_point = intersaction(left_fit_line, right_fit_line) # 소실점
```

```
    center_point = int(lx + (rx - lx) / 2), int(max(ly, ry)) # 양쪽 차선 중심점
```

```
    #cv2.circle(img, inter_point, 5, (0, 0, 255), -1) # -1이면 내부 채워짐
```

```
    #cv2.circle(img, center_point, 5, (0, 0, 255), -1)
```

```
    cv2.line(img, inter_point, center_point, (0, 255, 0), 5)
```

```
    angle = np.arctan2(center_point[1] - inter_point[1], center_point[0] - inter_point[0]) * 180 / np.pi # 기울기 측정
```

```
if inter_point[0] < center_point[0]:
```

```
    angle = int(angle) - 90
```

```
else :
```

```
    angle = - (int(angle) - 90)
```

```
return img, angle
```

HOST=""

PORT=2000

```
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
```

```
s.bind((HOST,PORT))
```

```
s.listen(1)
```

```
print('Socket now listening..')
```

#연결, conn에는 소켓 객체, addr은 소켓에 바인드 된 주소

```
conn,addr=s.accept()
```

```
while True:
```

```
    # client에서 받은 stringData의 크기 (==(str(len(stringData))).encode().ljust(16))
```

```
    length = recvall(conn, 16)
```

```
    stringData = recvall(conn, int(length))
```

```
    data = np.fromstring(stringData, dtype = 'uint8')
```

```
    #data를 디코딩한다.
```

```
    frame = cv2.imdecode(data, 1)
```

```
    cv2.imshow('frame',frame)
```

```
    try:
```

```
        angle = detect_lane(frame)
```

```
        print(angle)
```

```
        cv2.imshow('result',frame)
```

```
        s.send(angle.encode())
```

```
    except:
```

```
        print("error")
```

```
        pass
```

```
    if cv2.waitKey(1) & 0xFF == ord('q'):
```

```
        break
```

```
    cv2.waitKey(1)
```

```
<motor_control.py>
# -*- coding: utf-8 -*-
from __future__ import division
import Adafruit_PCA9685
import RPi.GPIO as GPIO
import socket
import threading
from time import sleep
import pygame

ip1 = '192.168.0.19' # 차선
ip2 = '192.168.0.6' # RFID, 초음파

# 모터 상태
STOP = 0
FORWARD = 1
BACKWARD = 2

# PIN 입출력 설정
OUTPUT = 1
INPUT = 0

ENA = 26 # 37 pin
IN1 = 19 # 35 pin
IN2 = 13 # 33 pin

# GPIO 모드 설정
GPIO.setmode(GPIO.BCM)
GPIO.setup(ENA, GPIO.OUT)
GPIO.setup(IN1, GPIO.OUT)
GPIO.setup(IN2, GPIO.OUT)

DC = GPIO.PWM(ENA, 100)
DC.start(0)
```

```

pwm = Adafruit_PCA9685.PCA9685()
pwm.set_pwm_freq(60) # 주파수 변경

# DC모터 제어 함수
def setMotorContor(DC, IN1, IN2, speed, state):
    DC.ChangeDutyCycle(speed) # PWM으로 제어

    if state == FORWARD: # 전진
        GPIO.output(IN1, GPIO.HIGH)
        GPIO.output(IN2, GPIO.LOW)

    elif state == BACKWARD: # 후진
        GPIO.output(IN1, GPIO.LOW)
        GPIO.output(IN2, GPIO.HIGH)

    elif state == STOP: # 정지
        GPIO.output(IN1, GPIO.LOW)
        GPIO.output(IN2, GPIO.LOW)

def setMotor(speed, state):
    setMotorContor(DC, IN1, IN2, speed, state)

angle = [241, 287, 322, 355, 400, 423, 446, 492]
CENTER = 394

Bus_stop = False
Obstacle = False

def from_line():
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (ip1, 1000)
    print("line socket listening...")

```

```
sock.bind(server_address)
```

```
sock.listen(1)
```

```
try:
```

```
    client, address = sock.accept()
```

```
    print("Line Connected")
```

```
    while True:
```

```
        data = client.recv(4)
```

```
        try:
```

```
            angle = int(data)
```

```
        except:
```

```
            continue
```

```
        if Bus_stop or Obstacle : # 버스 정류장, 장애물 인식시 정지
```

```
            setMotor(50, STOP)
```

```
        else:
```

```
            if abs(angle) < 10 : # 직진
```

```
                pwm.set_pwm(0, 0, CENTER)
```

```
            elif angle < -30 :
```

```
                pwm.set_pwm(0, 0, angle[0])
```

```
            elif -30 < angle < -20 :
```

```
                pwm.set_pwm(0, 0, angle[1])
```

```
            elif -20 < angle < -10 :
```

```
                pwm.set_pwm(0, 0, angle[2])
```

```
            elif 10 < angle < 20 :
```

```
                pwm.set_pwm(0, 0, angle[3])
```

```
            elif 20 < angle < 30 :
```

```
                pwm.set_pwm(0, 0, angle[4])
```

```
            elif 30 < angle :
```

```
                pwm.set_pwm(0, 0, angle[5])
```

```
            setMotor(50, FORWARD)
```

```
except:
```

```
    print("close line")
```

```
    GPIO.cleanup()
```

```
    exit(0)
```



```

def from_RFID():
    global Bus_stop, Obstacle
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (ip2, 1001)
    print("Sensor socket listening...")
    sock.bind(server_address)
    sock.listen(1)

    try:
        client, address = sock.accept()
        while True:
            data = client.recv(4)
            print(data)
            if data == "departure" and not Obstacle:
                setMotor(50, FORWARD)
                Bus_stop = False
            elif data == "arive":
                setMotor(50, STOP)
                sleep(5)
                Bus_stop = True
    except:
        print("close RFID")
        sock.close()
        GPIO.cleanup()
        exit(1)

def from_Ultra():
    global Obstacle, Bus_stop
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = (ip2, 1002)
    print("ultrar socket listening...")
    sock.bind(server_address)

```

```
sock.listen(1)
```

```
try:
```

```
    client, address = sock.accept()
```

```
    while True:
```

```
        data = client.recv(4)
```

```
        print(data)
```

```
        if data == "departure" and not Bus_stop:
```

```
            setMotor(50, FORWARD)
```

```
            Obstacle = False
```

```
        elif data == "stop":
```

```
            setMotor(50, STOP)
```

```
            sleep(5)
```

```
            Obstacle = True
```

```
except:
```

```
    print("close ultra")
```

```
    sock.close()
```

```
    GPIO.cleanup()
```

```
    exit(1)
```

```
LINE = threading.Thread(target=from_line)
```

```
RFID = threading.Thread(target=from_RFID)
```

```
ULTRA = threading.Thread(target=from_Ultra)
```

```
LINE.start()
```

```
RFID.start()
```

```
ULTRA.start()
```

```
LINE.join()
```

```
RFID.join()
```

```
ULTRA.join()
```

```
pwm.set_pwm(0, 0, 0)
```

```
GPIO.cleanup()
```

VII. 결과물 설명 (결과물 관련 이미지(사진) 첨부)

1. H/W



완성된 차량의 사진이다. 외형은 쉽게 구할 수 있는 종이박스로 차량의 규격을 측정하여 만들어 보았다.

처음에는 모든 장비에 가장 평범한 1.5V AA건전지를 각 장비의 동작전압에 맞게 홀더를 구비하여 전원을 공급하려고 하였다. PCA9685 Servo Driver와 MG995 Servo Motor를 구동시키는 데에는 문제가 없었다. 1.5V AA건전지 4개를 9V 홀더에 장착시켜 작동시키게 하였다.

L298 Motor Driver와 JGA25-370 DC Motor가 연결된 부분은 AA건전지로는 알맞은 동작 전류의 세기를 충족시키지 못하여 3.7V 18650 리튬 이온 배터리를 사용하였다. 동작전압이 높은 편이기에 2개를 7.4V 배터리 홀더에 장착시켜 연결하였다.

2개의 Raspberry Pi 장비 역시 AA건전지를 사용하지 못했다. 알맞은 규격의 배터리 홀더가 없었고, 역시 AA건전지로는 동작하지 않을 것이라 예상했다. Raspberry Pi의 경우 마이크로 5핀 단자를 전원 단자로 사용하여, 18650 1개가 들어가면서 출력을 마이크로 5핀으로 줄 수 있는 배터리 홀더를 구하여 각각 장착하였다.

앞 IV. 모듈별 설계에서의 설명대로 총 3단으로 구성하였다. 차량 앞면에는 초음파 센서를 부착하였고, PCA9685 Servo Driver의 전원인 배터리 홀더를 설치하였다. 1단에는 차량을 구동하는 DC Motor와 Servo Motor를 장착하였고, 2단에는 모터들과 연결되는 L298 Motor Driver와 PCA9685 Servo Driver를 부착하였다. 2단 뒷면에는 L298 Motor Driver의 전원인 18650 배터리 홀더를 부착하였다. 3단에는 파이카메라와 센서파이와 모터파이, 그리고 각 라즈베리파이에 전원을 공급하는 홀더를 장착했다.

2. 자율주행

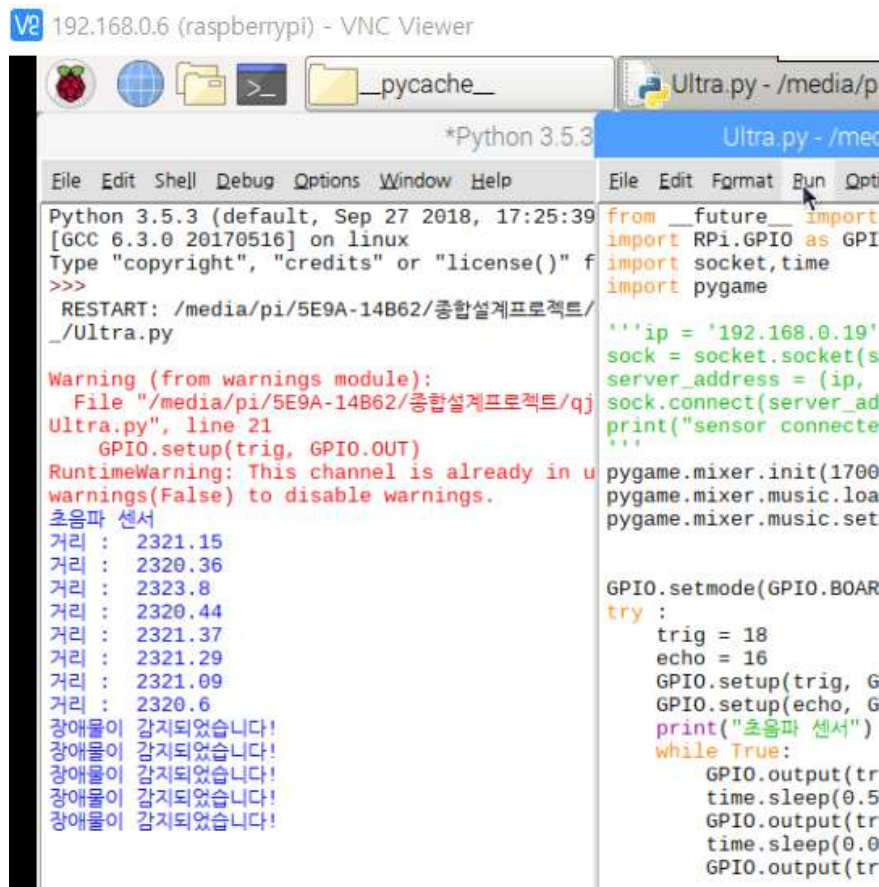
트랙 + 운전하는 사진

트랙을 따라 차선을 인식하여 주행하는 버스의 모습이다.

여러 가지 소재 (박스, 방수천, 폴리에스터 판넬 등)에서 트랙을 구성해보았고, 차선의 종류도 여러 종류의 테이프를 사용해보았다. 밑바닥의 재질에 따라 바닥이 울어서 형광등 빛이 너무 심하게 반사되거나 차선과 같은 패턴이 생겨서 차선을 인식하는 것이 힘들었다. 여러 시도를 거쳐 최종적으로 빔 프로젝터 스크린으로 바닥 재질을 결정하였고, 하얀색 절연테이프로 차선을 만들어서 실제 환경(콘크리트 바닥과 페인트 차선)과 유사하게 구성해보았다.

차선을 구현할 때 중요했던 것은 커브 길에서의 기울기 정도였다. 너무 완만한 기울기로 구현하면 많은 공간을 차지하였고, 너무 급격한 기울기로 구현하면 차량을 운행할 수 없었다. 알맞은 기울기를 찾기 위해 차량에 펜을 고정하고, 구동 명령만 내려 측정해보았다. 좌회전과 우회전 상황 각각 Servo Motor의 회전각을 조정하며 최대로 꺾일 수 있는 각도와 커브를 돌 수 있는 적당한 각도를 측정했다.

3. 장애물 인식



```
Python 3.5.3 (default, Sep 27 2018, 17:25:39)
[GCC 6.3.0 20170516] on linux
Type "copyright", "credits" or "license()" for more
>>>
RESTART: /media/pi/5E9A-14B62/중합설계프로젝트/_/Ultra.py

Warning (from warnings module):
  File "/media/pi/5E9A-14B62/중합설계프로젝트/qj
Ultra.py", line 21
    GPIO.setup(trig, GPIO.OUT)
RuntimeWarning: This channel is already in u
warnings(False) to disable warnings.
초음파 센서
거리 : 2321.15
거리 : 2320.36
거리 : 2323.8
거리 : 2320.44
거리 : 2321.37
거리 : 2321.29
거리 : 2321.09
거리 : 2320.6
장애물이 감지되었습니다!
장애물이 감지되었습니다!
장애물이 감지되었습니다!
장애물이 감지되었습니다!
장애물이 감지되었습니다!

from __future__ import
import RPi.GPIO as GPI
import socket,time
import pygame

'''ip = '192.168.0.19'
sock = socket.socket(s
server_address = (ip,
sock.connect(server_ad
print("sensor connecte

pygame.mixer.init(1700
pygame.mixer.music.loa
pygame.mixer.music.set

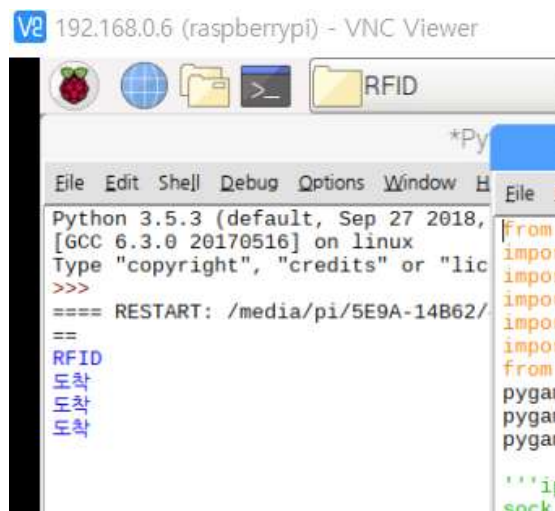
GPIO.setmode(GPIO.BOAR
try :
    trig = 18
    echo = 16
    GPIO.setup(trig, G
    GPIO.setup(echo, G
    print("초음파 센서")
    while True:
        GPIO.output(tr
        time.sleep(0.5
        GPIO.output(tr
        time.sleep(0.0
        GPIO.output(tr
```

정지하는데 까지 제동시간이 있기 때문에 제동거리까지 고려하였다. 직접 줄자와

물체를 놓고 차량을 구동시켜서 여러 번의 측정을 해보았다. 결과적으로 전방 15cm 이내에 있는 물체를 감지하면 버스가 피해야할 장애물로 인식시키도록 설정하였고 15cm 이하의 거리에 물체가 포착되면 정지가 가능하다.

15cm라고 설정한 거리와 관계없이 초음파 센서는 계속해서 초음파를 발신하고 수신하여 거리를 측정한다. HC-SR04 초음파 센서는 최대 4m 측정이 가능하고, 파이썬 코드를 실행해가며 작업해본 결과 평소 약2m의 거리를 늘 측정하고 있다.

4. 버스 정류장 인식



RFID 태그와 리더기의 위치를 어떻게 할 것인지 고민하였다. 우리 조가 사용하는 RFID-RC522의 경우 공진주파수가 13.56MHz 이었는데, 이는 2cm 이하의 거리에서 리더기가 태그를 인식하는 수치였다. 초반 계획에는 리더기의 위치를 버스의 옆 면에, 태그의 위치를 버스 정류장의 표지판에 설치하려고 하였는데 2cm의 인식거리로는 불가능한 방법이었다. RFID 전자 부품의 경우에도 결정한 부품 외에는 시중에서 쉽게 구할 수 없기 때문에, 버스의 밑면과 정류장 근처에 도로 바닥에 제동거리까지 생각하여 올바른 정류장의 위치에 정지하도록 설치

VIII. 프로젝트 수행 결과 분석

1. 재학 중 취득한 기초지식의 활용 내용

- 회로이론: 전반적인 회로구성 방법
- 컴퓨터 구조 및 마이크로프로세서: 마이크로프로세서의 기초적인 지식
- 무선 통신 공학: 소켓 통신을 활용한 각각의 라즈베리파이 연결

2. 재학 중 취득한 실험지식의 활용 내용

- 전기전자회로 실험 및 설계: 회로 설계 및 구성
- 마이크로프로세서실험: 전원공급기, L298n 의 사용지식 활용
- 오픈소스 로보틱스: 초음파 센서 사용 경험 지식 활용
- 임베디드 시스템 실험: 리눅스의 기초적인 명령문을 활용

3. 본 프로젝트 수행과정에서의 설계 능력 향상 내용

- 라즈베리 파이를 이용한 회로 구성
- 아두이노를 이용한 회로 구성 (초반에 아두이노를 설계하였지만 이후 제외하였다.)
- OpenCV를 활용한 차선인식과 자율주행
- 라즈베리파이와 라즈베리파이, 그리고 PC 간 소켓통신을 구현
- RFID 기기의 회로 구성 및 원리 이해
- 초음파 센서의 회로 구성 및 원리 이해

4. 본 프로젝트 수행과정에서의 문제 해결 내용

- 회로 구성 미숙으로 인한 라즈베리파이 파손

무선으로 진행해야 하는 프로젝트이니 만큼, 각 장치의 전원 역시 무게가 적게 나가면서 알맞은 전압과 전류를 출력해 줄 수 있는 전원을 찾고 있었다. 장치에 관해 충분히 공부하는 과정이 없이, 전자제품 상가에서 판매직원만 믿고 잘못된 방식으로 전원의 회로를 구성하여 진행하던 중 기기가 불타는 상황이 있었다. 라즈베리파이는 같은 제품으로 교체하였으며, 문제점을 인지하고 휴대가 용이하면서 라즈베리파이에 알맞은 전원을 공급해줄 수 있는 배터리 쉴드를 찾게 되었다.

- 각 장치의 전원 공급 문제

초반 설계 과정에서는 각 장치의 동작 전압을 찾고, 전원 공급기로 그에 맞는 전압을 설정하여 작동시켰다. 그 후 시중에서 쉽게 구할 수 있는 1.5V AA건전지로 각 장치에 전압을 맞추어 전원을 공급하였는데 PCA9685 Servo Driver와 MG995를 제외한 장치들이 작동하지 않았다. 인터넷에서는 (RC카 관련 사이트, 전자부품 판매 사이트, 프로젝트에 참고한 보고서 등) 취미로 즐기는 RC카를 제외하면 전자부품을 활용한 RC카와 그 부품에 대한 정보를 쉽게 얻을 수 없었다. 정영모 교수님과 호광춘 교수님께 질문을 드리고 전류의 문제라는 것을 인지하게 되었으며, 더 높은 전류를 출력해줄 수 있는 배터리(18650 리튬 이온 배터리)를 찾아보았고, 다시 각 부품에 맞게 설계하였다.

- Servo Motor 조작 미숙으로 인한 장치 파손

처음 RC카 키트에 포함된 Servo Motor는 MG996 Servo Motor였다. 차체의 조립과정에서 1차적으로 모터에 물리적인 힘을 가하여 손상이 많이 있었고, 2차적으로 조립 상에서의 한계 각도를 생각하지 못하고 무리하게 앞바퀴를 조향하여서 결국 파손되었다. 세운상가에서 전 모터와 유사한 MG995 Servo Motor를 구입하여 교체하였다. RC카 관련 부품들은 대부분 예민하고 고장이 잦기 때문에 더 조심해서 다루게 되었다. 비슷한 상황으로 초음파센서 HC-SR04도 잦은 전면 충돌로 인해 파손되었다가 교체하였다.

-라즈베리파이와 파이카메라의 한계

차선인식을 위한 영상처리를 할 때에 카메라의 인식속도와 라즈베리파이의 연산 처리속도가 차량의 이동속도를 따라가지 못하여 커브 구간에서 주행 시 차량이 차선 밖으로 이탈하였다. 파이카메라는 같은 제품으로 교체하였는데 처리속도가 한층 빨라진 점으로 보아 카메라 부품 자체의 결함이 있었던 것으로 생각된다.

라즈베리파이의 연산속도를 극복하기 위해 사용 중인 센서파이와 모터파이 중간 연산 과정을 수행하도록 노트북을 Server로 구축해 활용하였다. 마지막으로 출력되는 화면의 크기를 작게 하고 해상도 낮게 조정하여 인식과 처리의 속도를 더 빠르게 구현할 수 있었다.

-Wifi 환경의 문제점

학교에서 제공해주는 공용 Wifi로는 라즈베리파이 간의 통신과 라즈베리파이와 서버로 사용하는 노트북의 연결 속도에 한계를 느꼈다. 학과사무실에 요청하여 Wifi 기기를 대여하였다.

-스피커의 활용

작고 콤팩트한 스피커를 라즈베리파이에 직접 부착하여 음성안내를 시도하려고 하였지만 소리가 너무 작았다. 앰프의 필요성을 알게 되었는데 라즈베리파이에 직접 납땜을 해야만 한다는 정보를 얻게 되어 블루투스 스피커를 통한 음성안내로 바꾸게 되었다. 라즈베리파이 같은 경우 핵심적인 부품이기도 하고 다른 부품들에 비해 고가이기 때문에 납땜의 위험성을 감수할 수 없었다.

5. 본 프로젝트 수행과정에서의 실무 능력 향상 내용

- 분업과 협업

졸업 프로젝트는 오랜 시간이 걸리고 난이도도 다른 프로젝트보다 높은 수준이었다. 특히 우리 조는 조 편성을 상당히 늦게 하였기 때문에 빠르고 효과적인 업무의 처리가 필요했다. 그러기 위해 회의 과정을 통해 업무의 목표를 명확히 하고 분담을 하여 처리하였다. 각자 주된 역할을 맡아 진행하면서도, 부재하는 인원의 발생이나 혼자서는 부담되는 업무를 처리하기 위해 각 업무에 보조자를 두어 2인 1조로 진행

하여 협동력을 향상시킬 수 있었다.

-오류의 수정

하드웨어인 차량 자체, 소프트웨어인 라즈비안과 파이썬 코드, 마지막으로 네트워크까지 포함된 프로젝트이기 때문에 한 가지의 오류가 발생해도 어떤 부분에서 고쳐나가야 할지 생각해야할 경우의 수가 굉장히 많았다. 앞서 언급한 전원의 문제, 장치의 파손 문제 등 해결 후에 보고서를 작성하였기 때문에 간단해 보이지만 그 이면에는 고민의 과정이 굉장히 컸다. 이로 인해 엔지니어로서의 고충을 알게 되었고 많은 경험을 할 수 있었다.

-소통의 중요성

프로젝트가 완성되어 갈수록 분업보단 협업의 시간이 늘어갔다. 많은 부분에서 분업을 하여서 각자 맡은 업무를 합치는 과정에서 인원이 1명만 빠져도 많이 힘들어지는 경우가 발생하였다. 이를 극복하기 위해서 실습실에서 다른 조원들이 꼭 알아야 할 내용들을 메모하기 시작하였고, 모두가 모이는 시간대에는 회의를 하는 과정에서 그 내용을 이해하기 쉽게 설명하였다. 단합심이 증대되는 기회였다.

6. 본 프로젝트 수행과정에서의 팀원 간 협동 내용

하드웨어 개발, 재료 조달, 예산 관리, 각종 제출 자료 작성, 소프트웨어 개발 환경 구축, 파이썬을 이용한 프로그래밍 등 동시에 진행할 수 있는 분업이 가능한 부분들이 있었으며 트랙과 외형 제작, 파이카메라를 이용한 차선인식 과정, 초음파 센서를 이용한 장애물 감지, RFID를 이용한 버스정류장 인식 등 두 사람 이상 있어야 하는 협업이 필요한 부분도 있었다. 각 도움이 필요한 파트를 서로 도와가면서 작업을 진행함으로써 작업 시간을 줄이고 효과적인 결과를 도출해낼 수 있었다.

상호평가 표					
대상자 평가자	김기홍	박두원	김경연	박준영	총합 (100점)
김기홍	10	5	45	40	100
박두원	10	5	45	40	100
김경연	10	5	45	40	100
박준영	10	5	45	40	100
합산	40	25	180	160	400

7. 개발된 결과물에 대한 전시 방법 계획

- 버스의 주행 트랙을 설치한다.
- 장애물 및 버스정류장이 없는 상태에서 차선인식 주행을 확인한다.
- 장애물 및 버스정류장을 설치한 상태에서 버스의 동작을 확인한다.

- 주행 중 초음파 센서가 장애물을 발견하면 버스는 정지한다.
- 장애물이 사라지면 버스는 다시 주행한다.
- 버스정류장을 발견하면 RFID가 인식하여 잠시 정차한다.
- 정류장에서의 정차 시간이 지나면 버스는 다시 주행한다.

< 참고 문헌 >

1. python으로 배우는 OpenCV 프로그래밍 (김동근, 2018)
2. 파이썬을 이용한 머신러닝, 딥러닝 실전 개발 입문 (쿠지라 히코우즈쿠에, 2017)
3. OpenCV를 활용한 컴퓨터 비전 프로그래밍 (로버트 라가니에, 2015)
4. 라즈베리파이 활용백서 : 실전 프로젝트 2.0 (이재상 / 표윤석, 2013)
5. OpenCV로 배우는 영상 처리 및 응용 (정성환 / 배종욱, 2017)