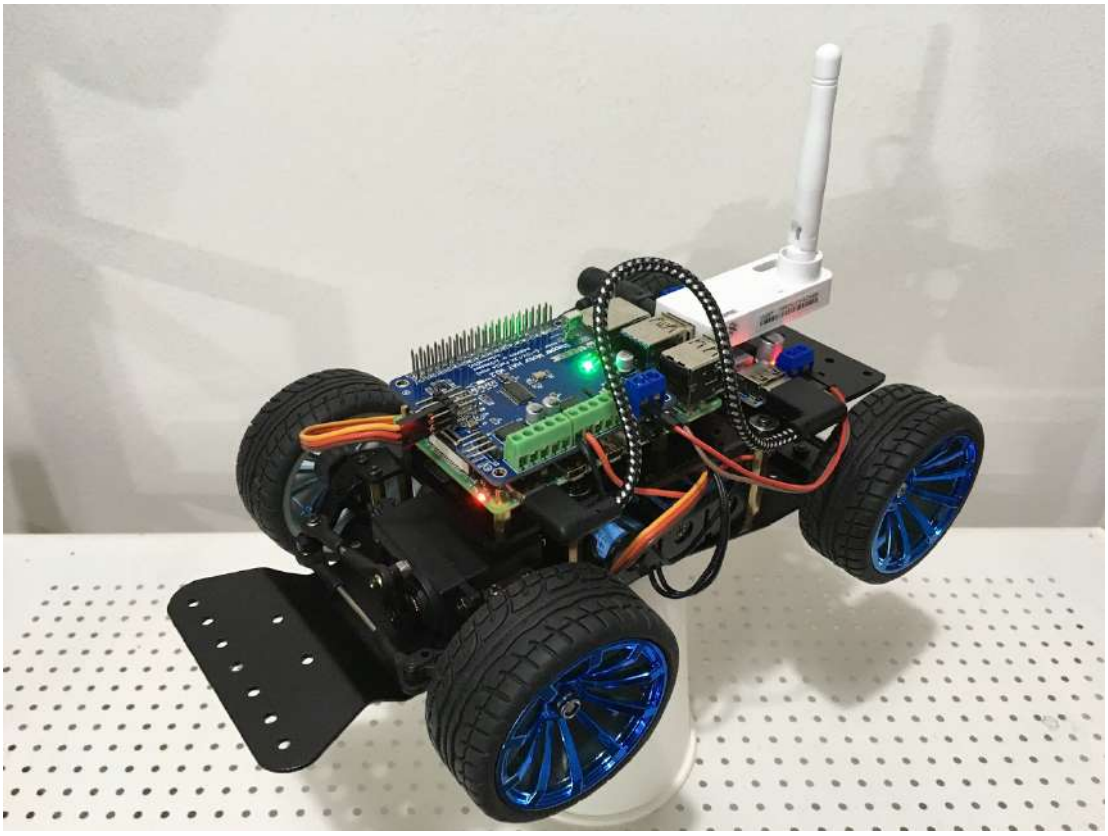
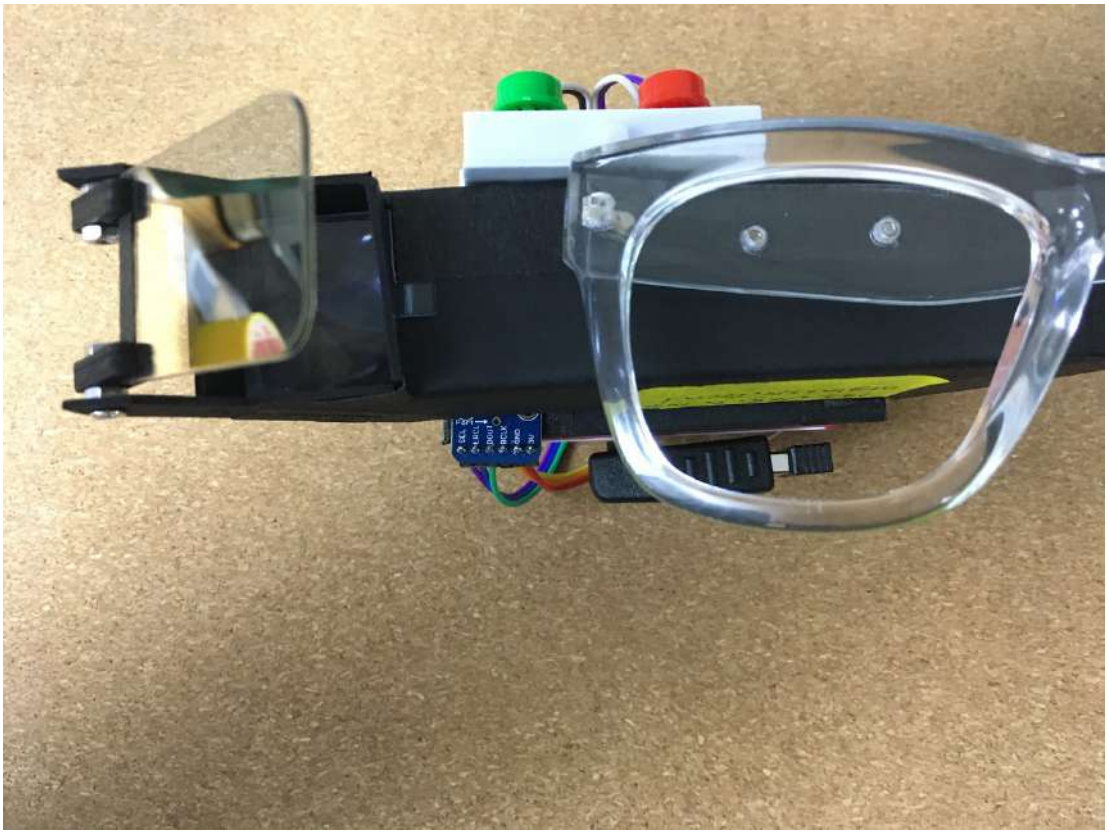
Day2. Remote-control vehicle

예시





기능명세

rc-vehicle

- 주어진 킷트를 조립하여 rc-car를 제작한다.
- Raspberry pi 4 model B 를 컨트롤러로 사용하여 ssh로 연결된 호스트컴퓨터(pc)에서 콘솔에 명령어를 입력함으로써 차량을 제어할 수 있다.
- 구현해야할 기본 명령은 '앞으로/ 뒤로/ 정지/ 빠르게/ 느리게/ 오른쪽/ 왼쪽/ 중앙' 이다.
- wifi통해 네트워크 연결되도록 한다.

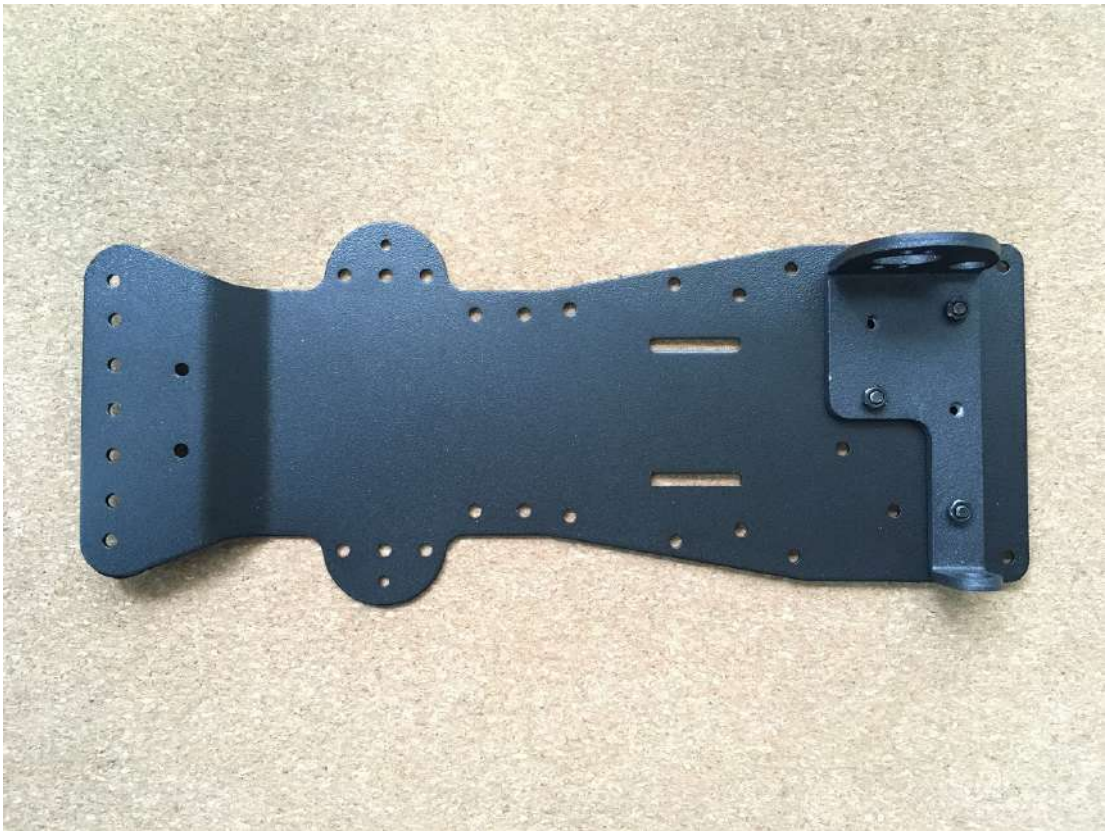
ar glass + microphone

- 제공되는 mems microphone을 raspberry pi에 설치하고 작동을 확인한다.

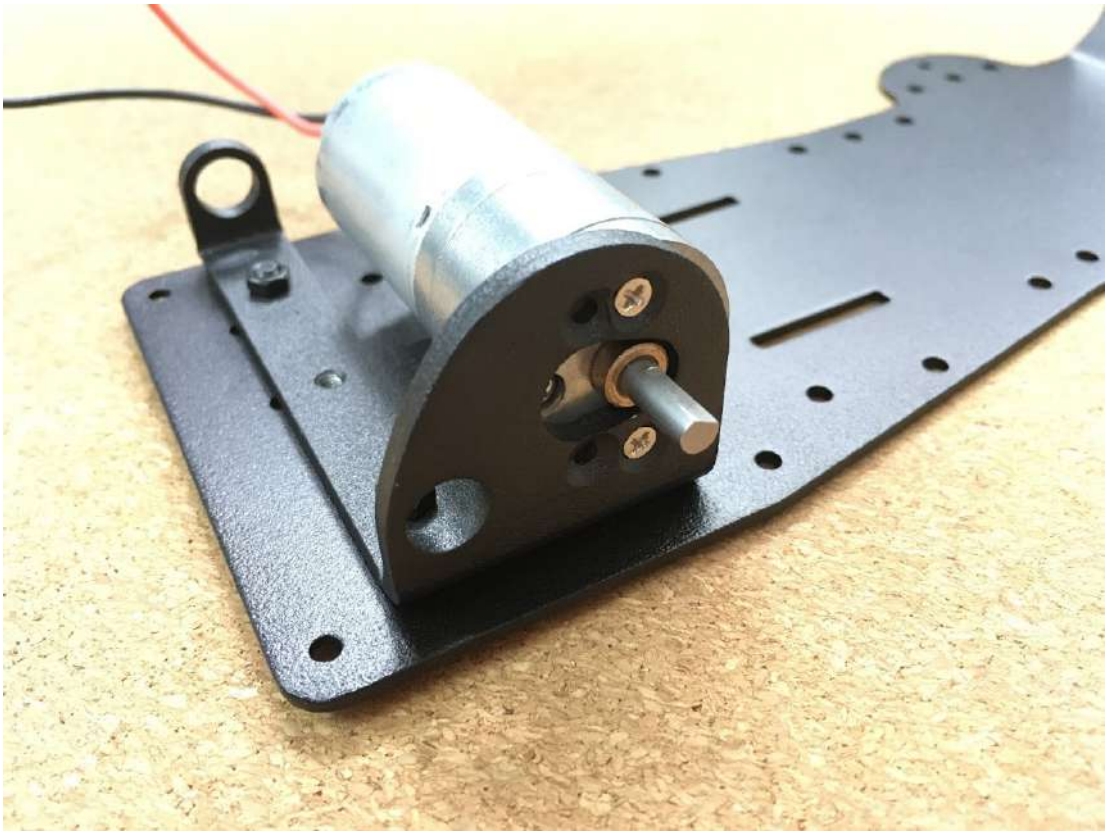
차량 조립 순서

조립 매뉴얼 참고: https://bit.ly/3dubh74

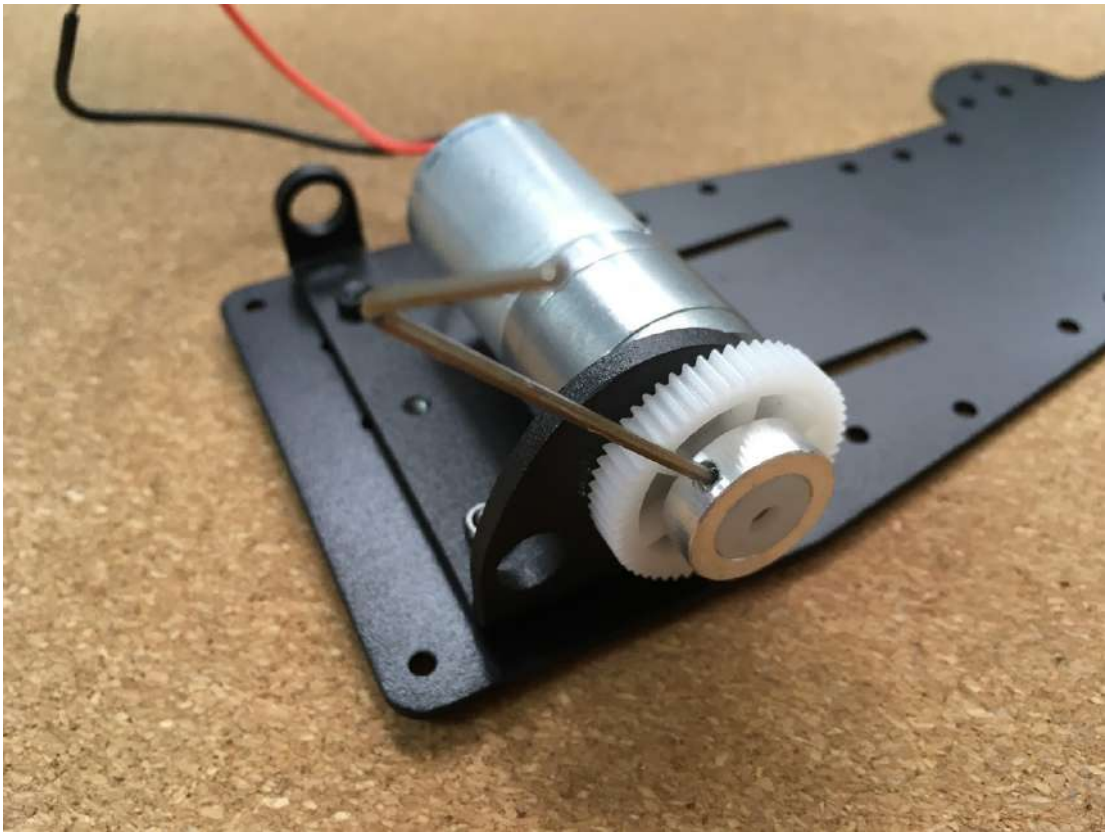
1. 모터 브라켓



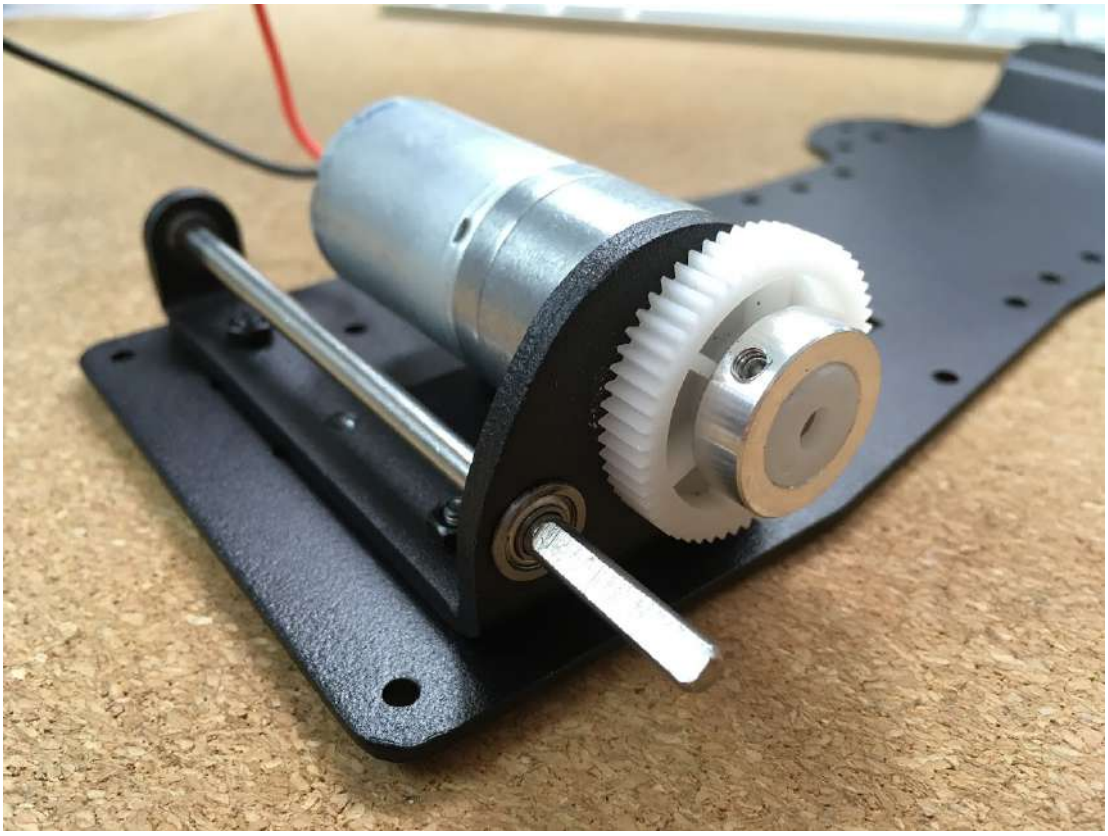
2. 모터



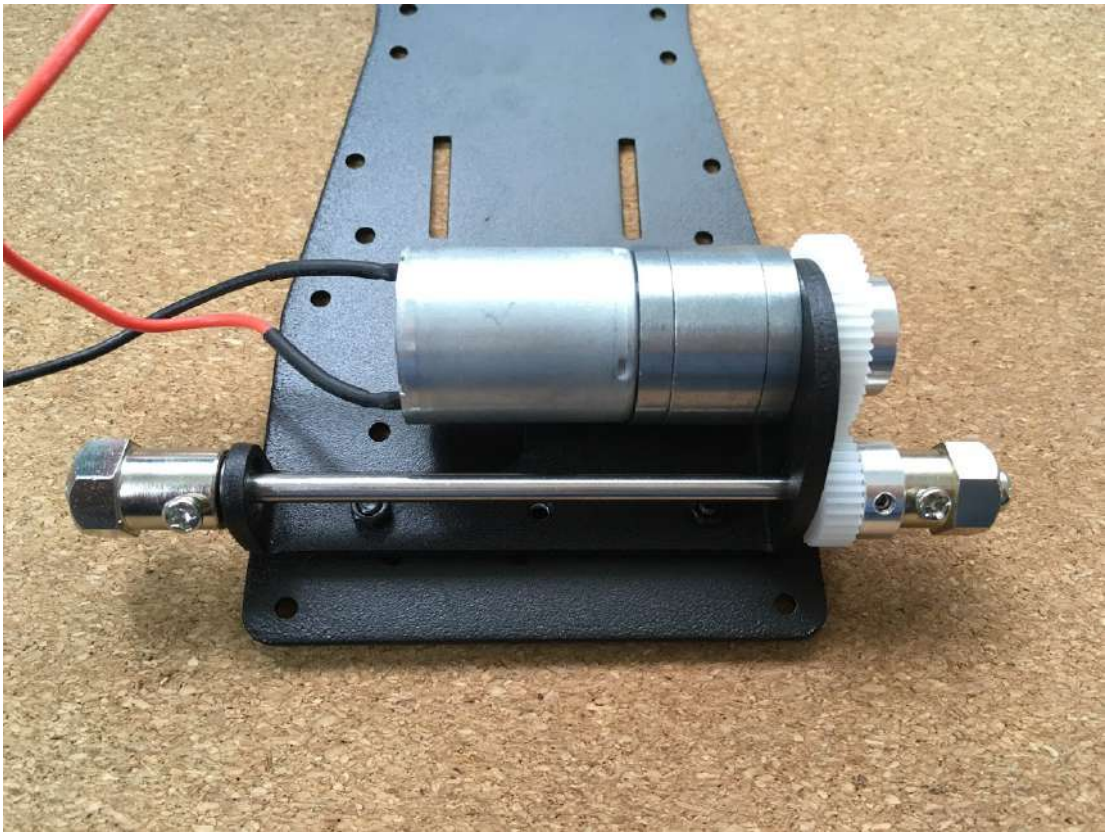
3. 기어(대)



4. 차축 + 베어링(좌,우)

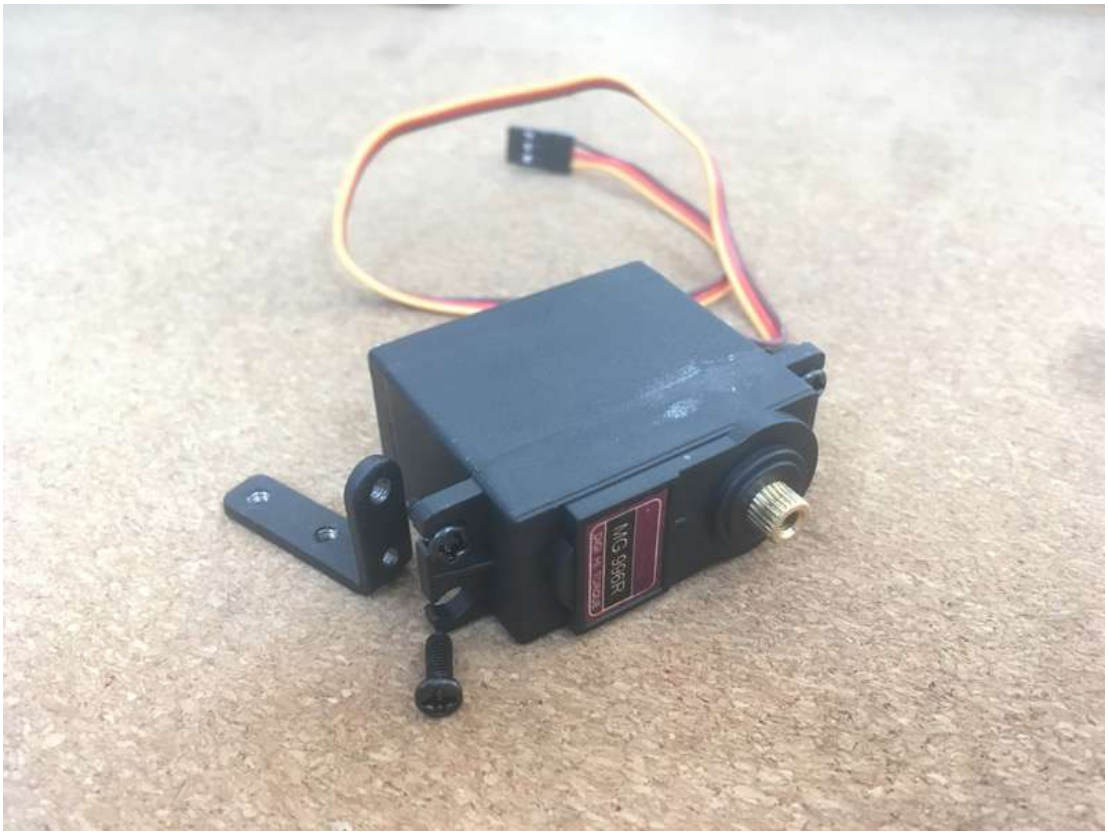


5. 기어(소) + 뒷바퀴 허브(좌우 길이 다름 주의)



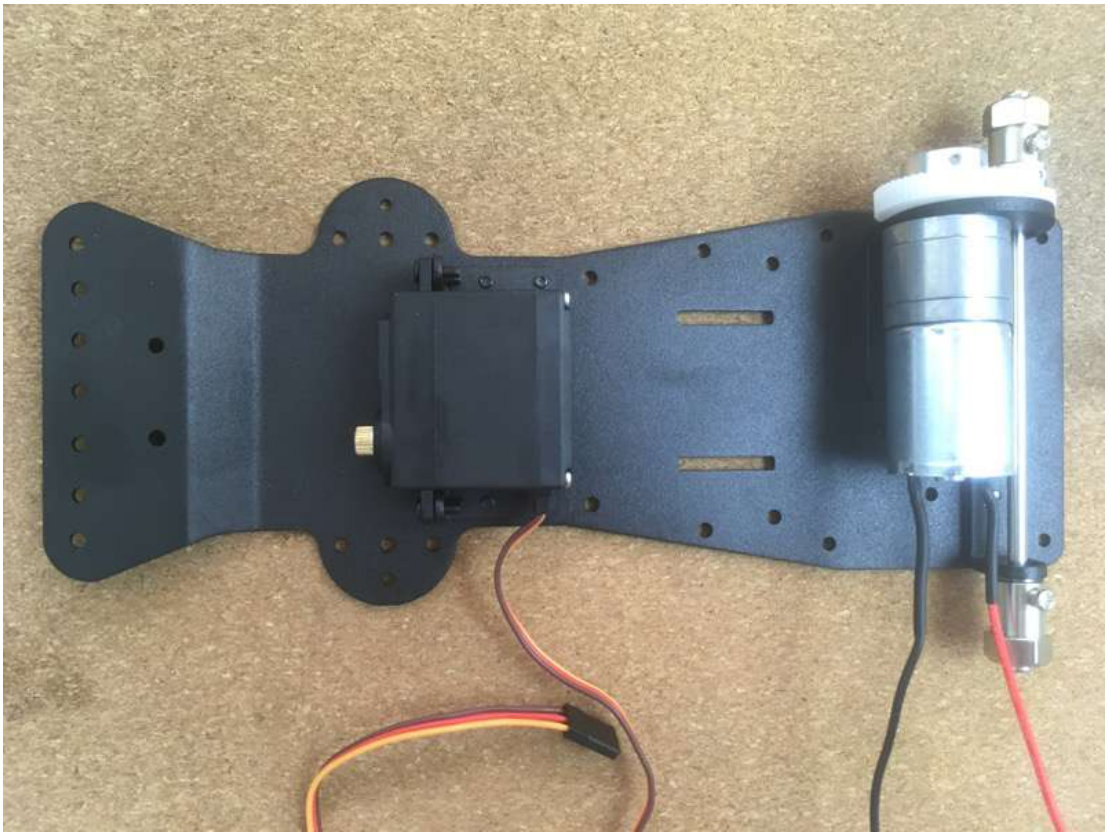
6. 서보 암(arm) 및 서보 브라켓(좌우)

Note: 서보가 중앙에 위치한 상태에서 암을 조립해야 좌/우회전이 원활하다



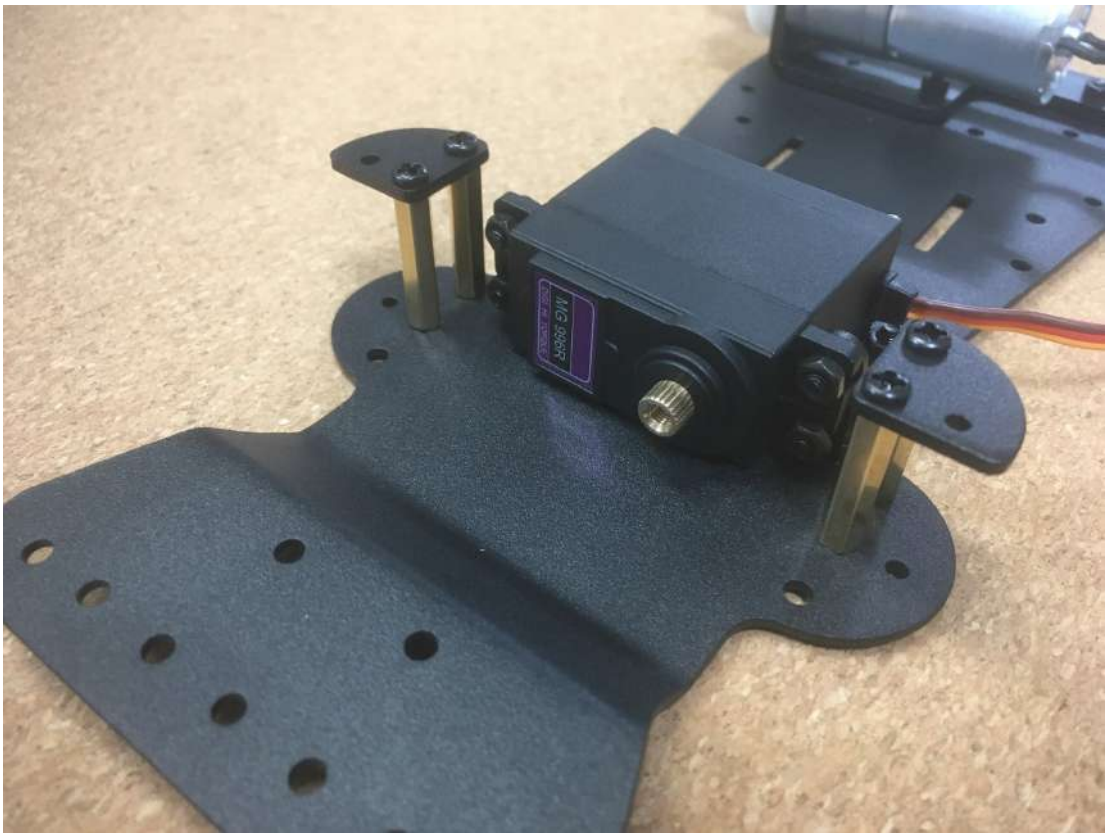
7. 바닥에 서보 브라켓 고정

아래 그림을 참고하여 m3 나사로 서보브라켓을 바닥에 결합한다.



8. 조향축 결합부 (좌,우)

6각형 기둥 중 높이 22mm(중간길이)의 것을 사용

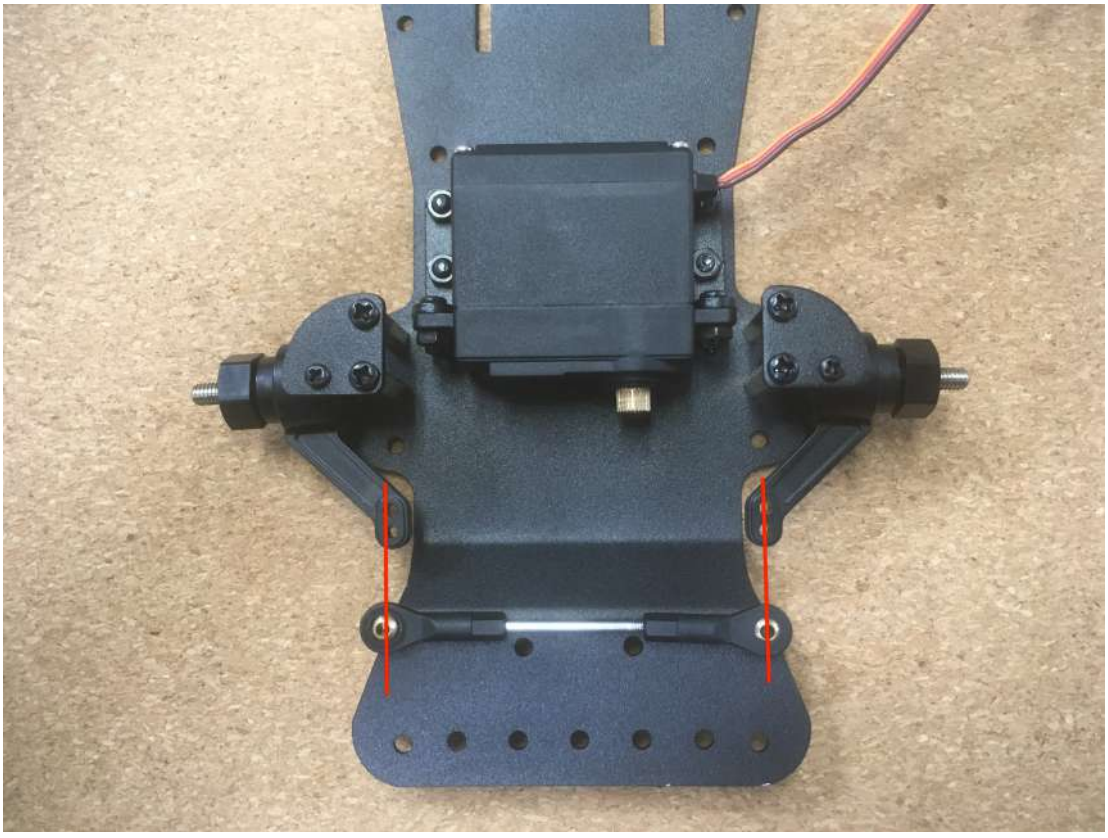


9. 앞바퀴 결합부 (너클 좌,우)

부품의 포개지는 순서에 유의

10. 너클 결합 및 타이로드(길이조정 되는 긴 막대) 길이 조정

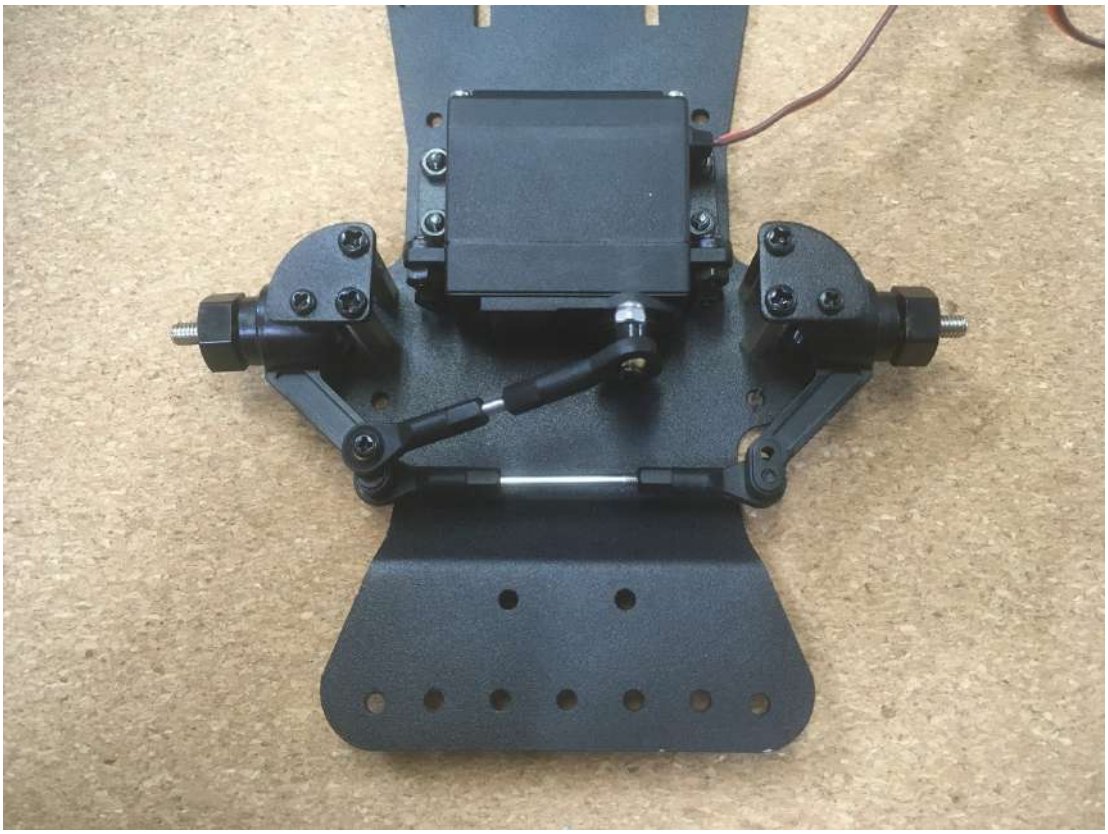
- 너클 결합에 사용되는 나사 크기 주의(m2.5)
- 타이로드의 길이는 앞바퀴를 가지런히 정렬했을 때 좌우의 너클에서 튀어나온 나사구멍 사이의 거리와 같도록 조정



11. 조향장치 연결

- 앞바퀴가 가지런해지도록 두개의 로드(rod)의 길이를 조절.
- 스크류 m2.5

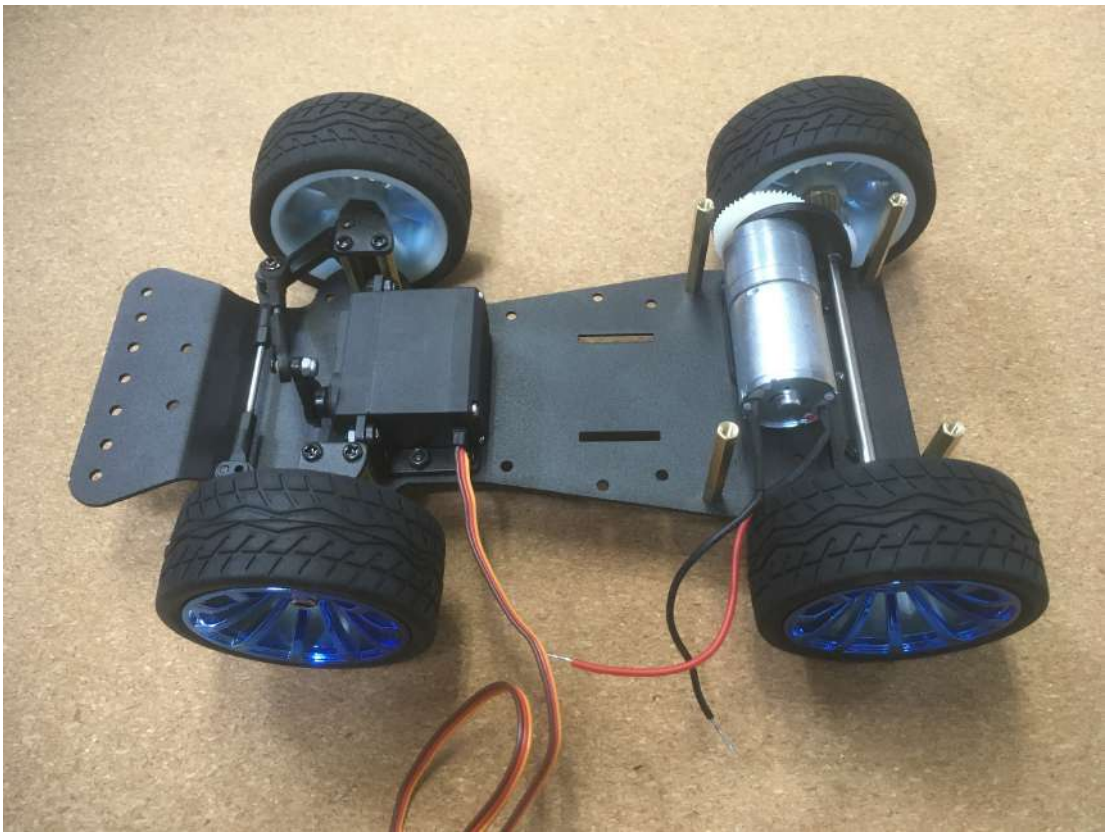
- 서보와 짧은 커넥팅 로드 연결시에는 m2.5 풀림방지 넛트 사용. *Note: 서보 위치가 중앙에 오도록 조정 후에 서보 암을 결합토록 한다.*



12. 바퀴 조립

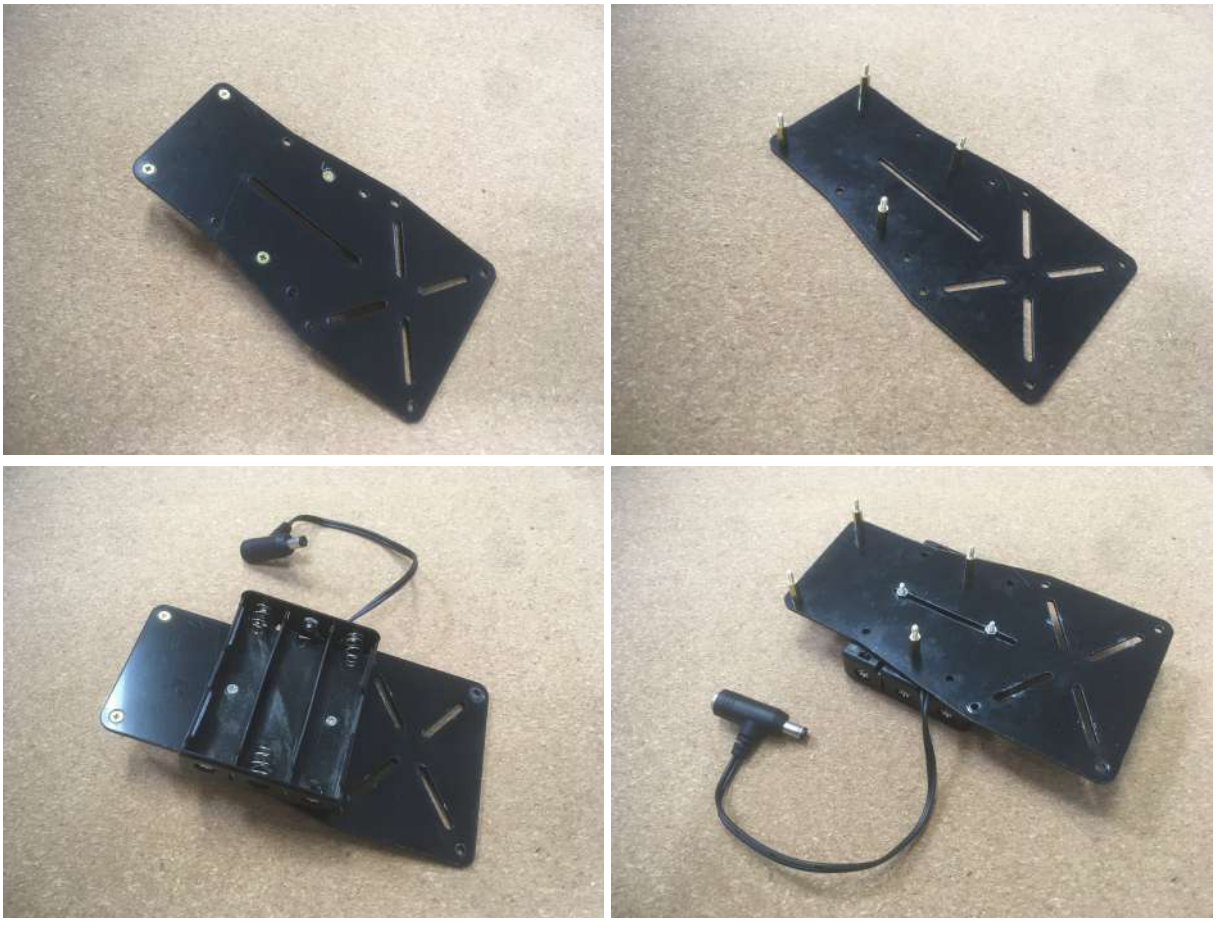
13. 상판 기둥 조립

아래 이미지 참고하여 35mm 서포트(긴 것)을 4개 조립한다.



14. 데크 & 배터리 케이스 조립

데크에 2.5mm pcb 서포트 4개를 끼운 후, m2.5 스크류를 사용해 배터리 케이스를 데크 하부에 부착한다.

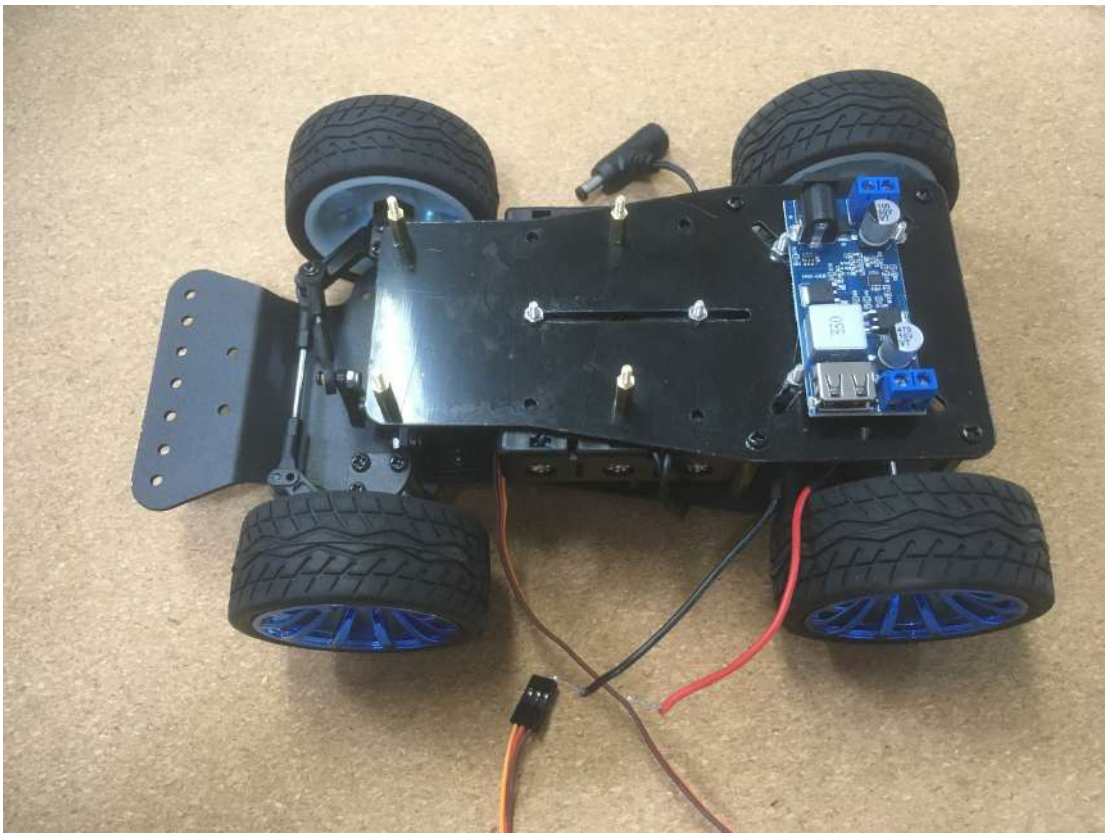


15. 전원 공급 모듈 부착

전원 모듈을 상판에 부착한다. m2.5 스크류나 양면테이프를 사용한다.

16. 상판조립

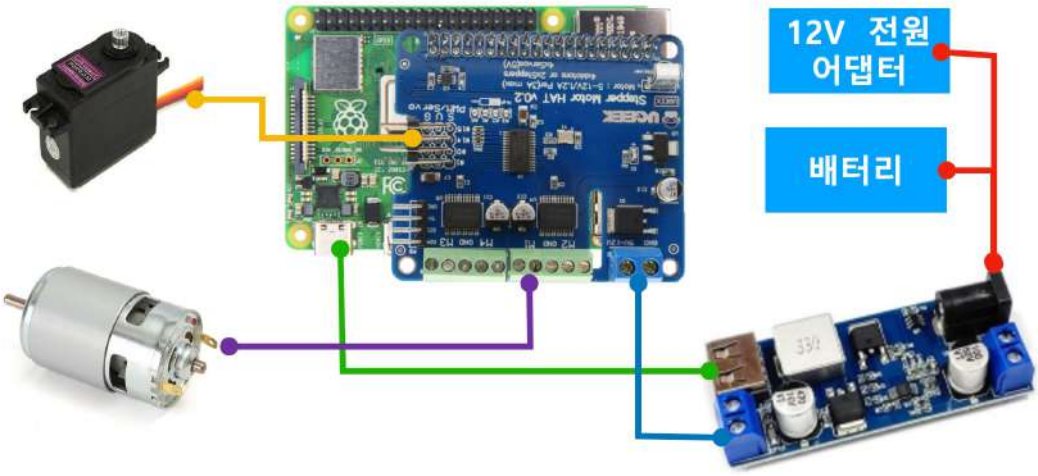
데크는 그림과 같이 4개의 서포트 위에 고정된다.

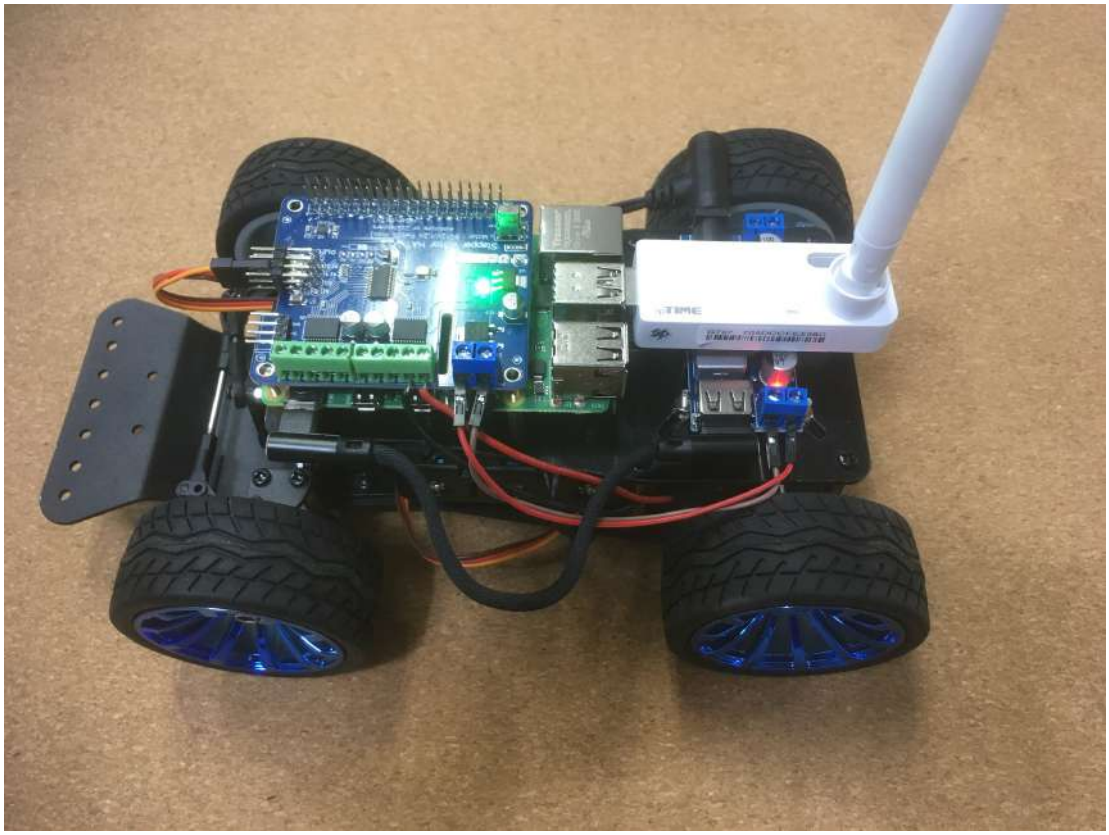


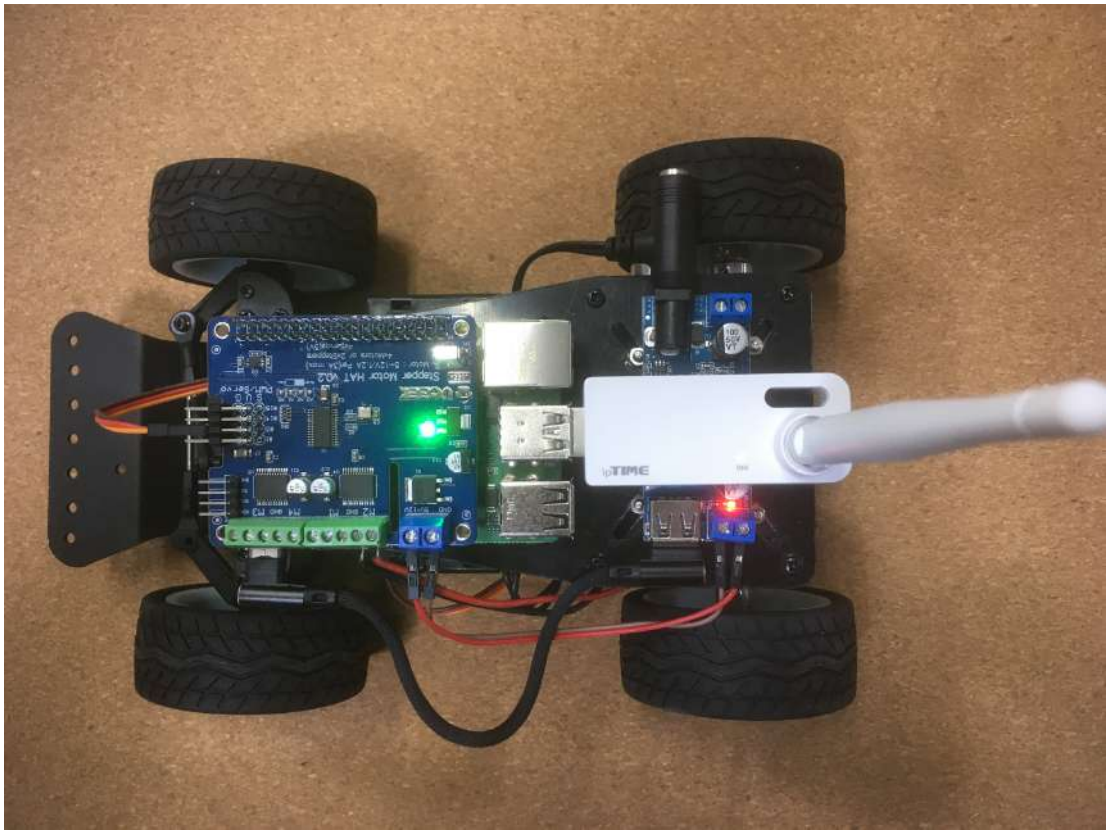
17. 라즈베리파이 및 모터 컨트롤러 부착

- 라즈베리파이와 모터 컨트롤러를 스택하여 데크에 결합한다.

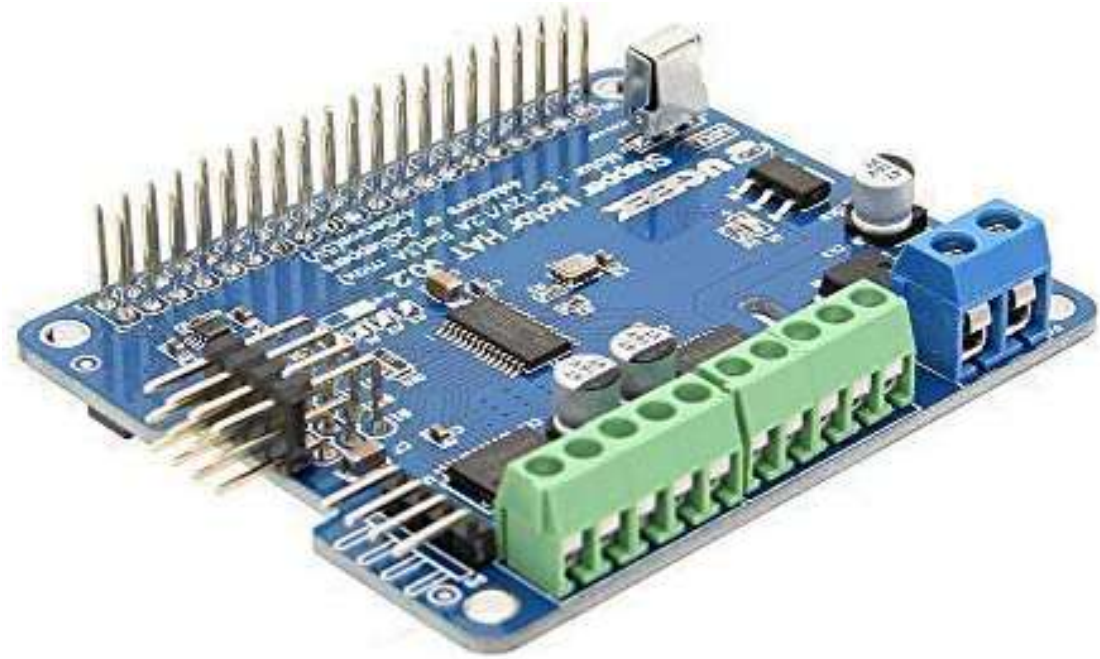
- 배선을 잘 정리해 연결한다.







raspberry pi motor HAT v0.2



참고: http://raspberrypiwiki.com/Robot_Expansion_Board

- i2c control
- 4x DC motor 연결. 256단계 speed 제어.
- 2x Stepper motor 연결. (unipolar or bipolar)
- 4 H-Bridges: TB6612 chipset provides 1.2A per bridge with thermal shutdown protection, internal kickback protection diodes.
- 5~12VDC 외부전원
- 최대 32층으로 stackable
- 각 모터당 공급가능한 전류량은 1.2A, 순간최대 3A.

rpi에 연결

rpi의 소켓에 잘 꽂음으로써 i2c 와 파워 모두 배선완료된다. 모터에 공급할 외부 전원을 연결한다. (5~12V) *Note: rpi를 통해 모터에 전원 공급되지는 않으며, 역으로 모터 전원 연결해도 rpi에 전원 공급되지는 않는다. 2개의 개별적인 전원이 필요하다.*

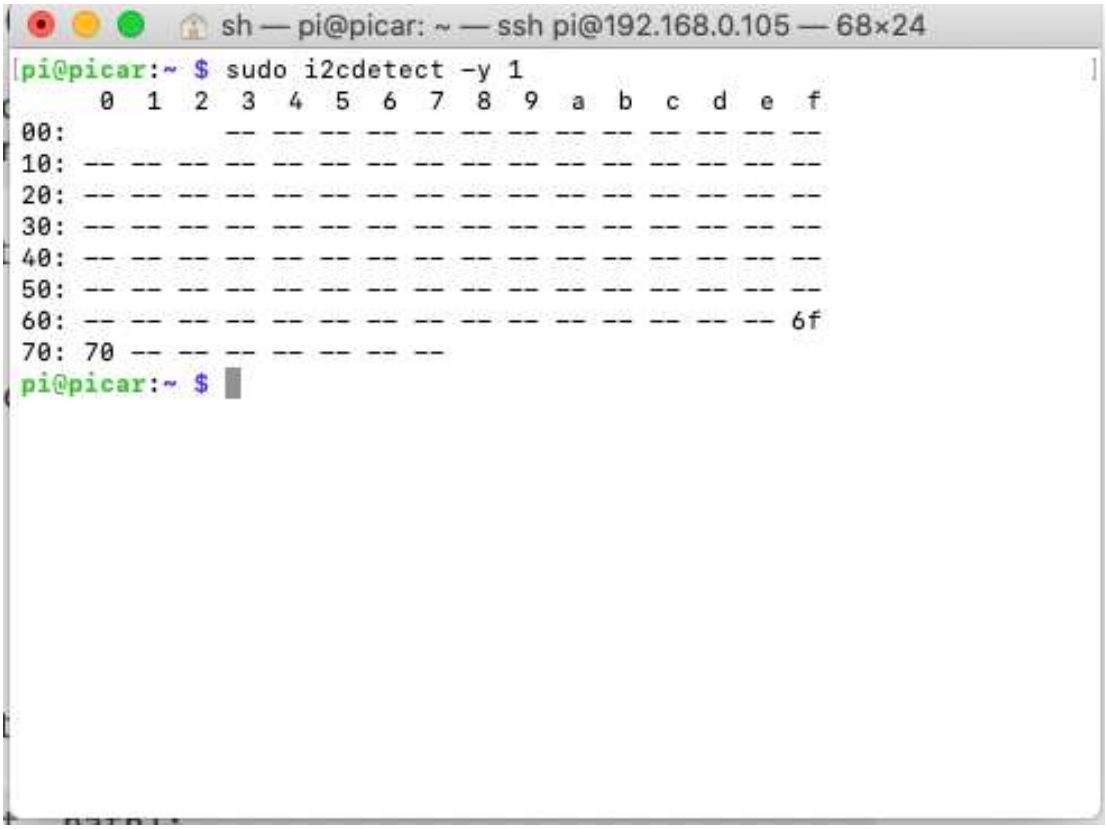
i2c 활성화하기

1. 먼저, 라즈베리가 i2c 통신을 사용하도록 설정해 주어야 한다.

```
$ sudo raspi-config
...
interface options > i2c >켜기
```

2. rpi에 연결된 i2c 디바이스를 검색해보자.

```
$ sudo i2cdetect -y 1
```



위의 경우를 보면 두개의 i2c 디바이스가 **address 0x6F** 과 0x70(아마도 라즈베리 내부적으로 사용중...)에서 발견되었다.

motor HAT 라이브러리 다운로드

라이브러리 다운로드:

```
$ git clone https://github.com/ssafy-embedded-project/Raspi-MotorHAT-
python3.git
```

라이브러리 3개 파일(Raspi_MotorHAT.py, Raspi_PWM_Servo_Driver.py, Raspi_I2C.py)을 현재 작업디렉토리에 넣어둔다.

```
$ cd Raspi-MotorHAT-python3
$ cp Raspi_* ~/
```

motor HAT 라이브러리 - DC motor 사용법

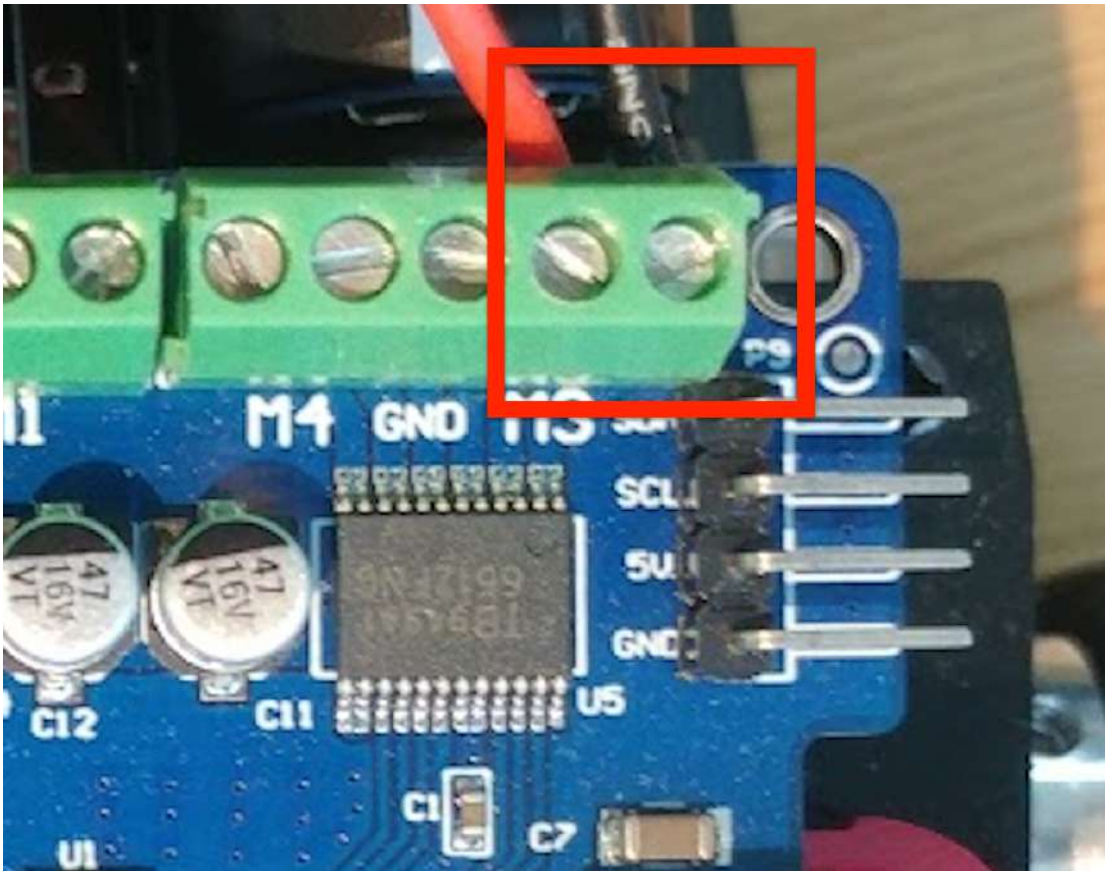
1. Raspi_MotorHAT을 임포트한다. ( )

```
from Raspi_MotorHAT import Raspi_MotorHAT, Raspi_DCMotor
```

2. 모터햇 오브젝트를 만든다.

```
mh = Raspi_MotorHAT(addr=0x6f)
```

3. 모터를 연결한다. 여기서는 M3 에 연결했다.




```
myMotor = mh.getMotor(3)
```

4. 모터 스피드를 설정한다. 0~255 모터가 운행중 속도 세팅 변경하면 바로 적용된다.

```
myMotor.setSpeed(150)
```

5. 모터를 작동시킨다.

```
myMotor.run(Raspi_MotorHAT.FORWARD)
```

값	방향
Raspi_MotorHAT.FORWARD	전진
Raspi_MotorHAT.BACKWARD	후진
Raspi_MotorHAT.RELEASE	정지

6. 스크립트 종료시 모터 작동 멈추는 코드 추가해준다. 파이썬 코드가 끝나거나, 리눅서 커널이 먹통이 되어도 DCMotor 는 자동으로 멈추지 않는다.

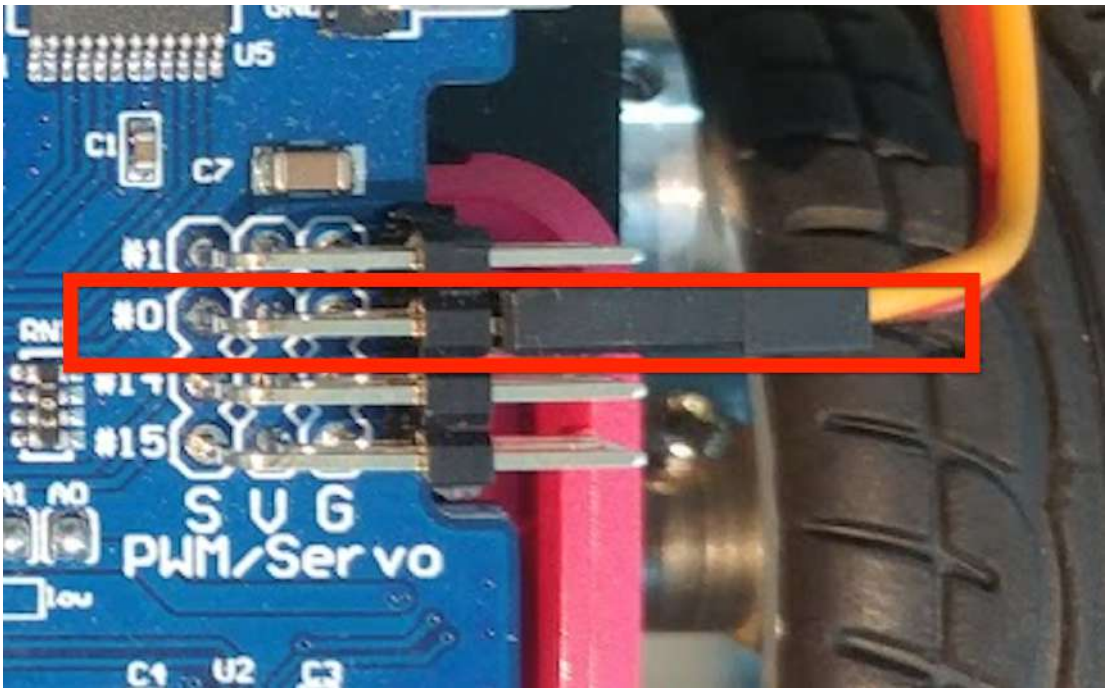
```
import time
time.sleep(10)
```

```
myMotor.run(Raspi_MotorHAT.RELEASE)
```

motor HAT 라이브러리 - servo 사용법

PCA9685 드라이버를 사용하고 있다. 서보조작은 물론 pwm을 통한 led제어에도 사용할 수 있다.
참고: https://www.adafruit.com/product/815

서보를 4개까지 연결할 수 있다. #0,#1,#14,#15 채널(라즈베리의 핀번호와 무관한 서보컨트롤러가 사용하는 번호)에 연결된다. 여기서는 #0 채널에 서보를 연결하였다.



1. 서보 컨트롤러 오브젝트 앞서 'mh = Raspi_MotorHAT(0x6F)' 문장에서 Raspi_MotorHAT object를 만드는 순간 내부적으로 이미 서보 컨트롤러 오브젝트를 `mh._pwm`라는 이름으로 사용할 수 있다.

Note: Raspi_MotorHAT 모듈은 내부적으로 i2c로 pwm서보를 컨트롤 할 수 있는 `Raspi_PWM_Servo_Driver` 모듈을 사용한다. 참고: https://github.com/adafruit/Adafruit_Python_PCA9685

```
from Raspi_MotorHAT import Raspi_MotorHAT

mh = Raspi_MotorHAT(0x6F) # motor-HAT i2c주소
servo = mh._pwm
```

2. pwm frequency 설정하기 서보컨트롤에서는 50Hz 혹은 60Hz가 일반적이다. (여기에서는 50Hz 사용.)

```
servo.setPWMFreq(50)
```

3. pwm 신호 보내기 (특정 각도로 서보 움직이기) `Raspi_MotorHAT._pwm.setPWM(channel, on, off)` 함수로 pwm 신호를 보낼 수 있다.

```
```

```
# 서보 왕복운동
while (True):
  servo.setPWM(0, 0, servoMin)
  time.sleep(1)
  servo.setPWM(0, 0, servoMax)
  time.sleep(1)
```

이 때, on 과 off는 1 주기(즉, 1/50 sec)중 언제 pulse를 시작하고 언제 끝낼 것인가 하는 것인데, PCA9685는 한 주기를 4096등분(12bit) 할 수 있는 분해능을 갖고 있다. 한마디로 (on,off) 가 (0, 2048)이면 50% duty, (0, 410)이면 10% duty라고 보면된다. (on은 일반적으로 0으로 한다.)

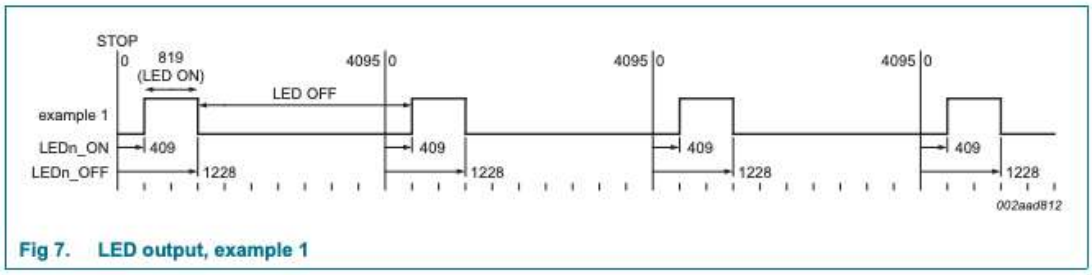


Fig 7. LED output, example 1

일반적인 서보에서 1 millisec signal = 5% duty = 0˚, 2 millisec signal = 10% duty = 180˚이므로, `Raspi_MotorHat._pwm.setPWM(channel, on, off)`에서,

channel	on	off	서보 각도(대략)
0,1,14,15	0	200	0˚
	0	300	90˚
	0	400	180˚

- 서보의 가동범위와 여러가지 특성은 모델마다 모두 다르다. 반드시 직접 테스트해보자.

---

input()

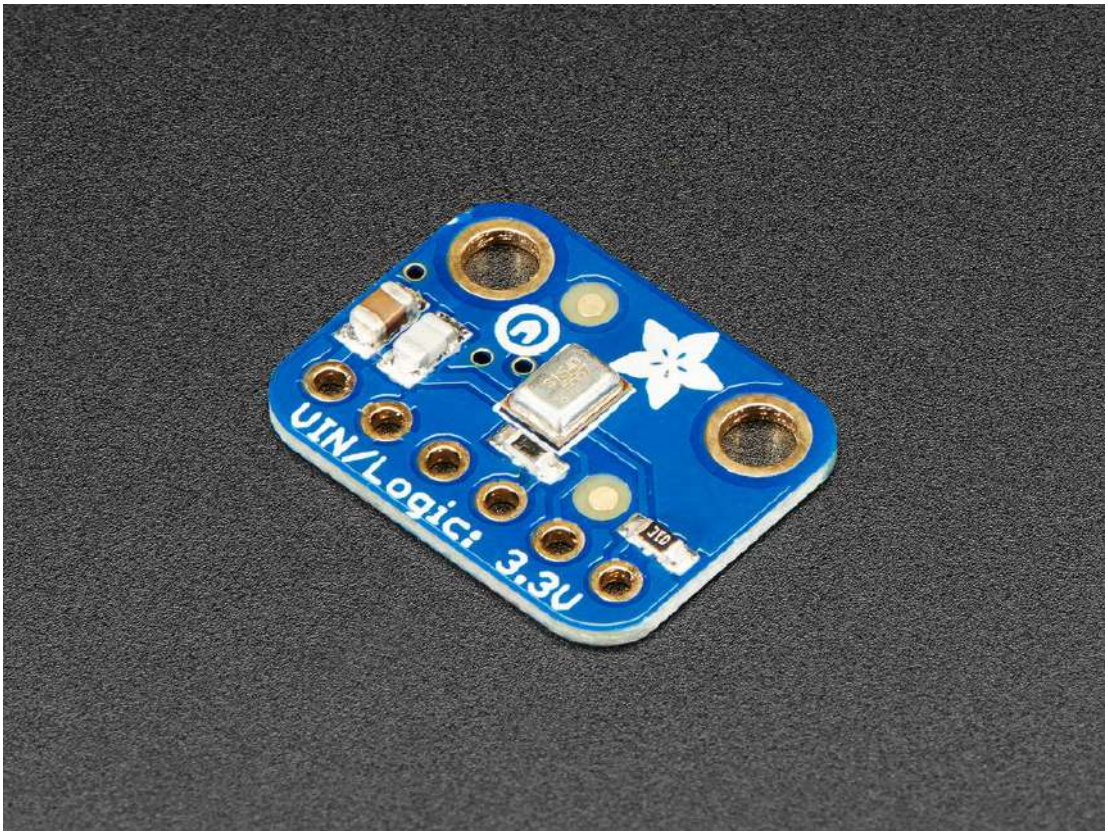
파이썬 내장함수. 표준 입력장치로부터 문자열을 입력받는다. 참고: https://docs.python.org/ko/3/library/functions.html

```
# python3 interpreter

>>> birthday = input("생일을 입력하세요:")
생일을 입력하세요:8월6일
>>> birthday
'8월6일'
```

---

i2s microphone



참고: https://learn.adafruit.com/adafruit-i2s-mems-microphone-breakout

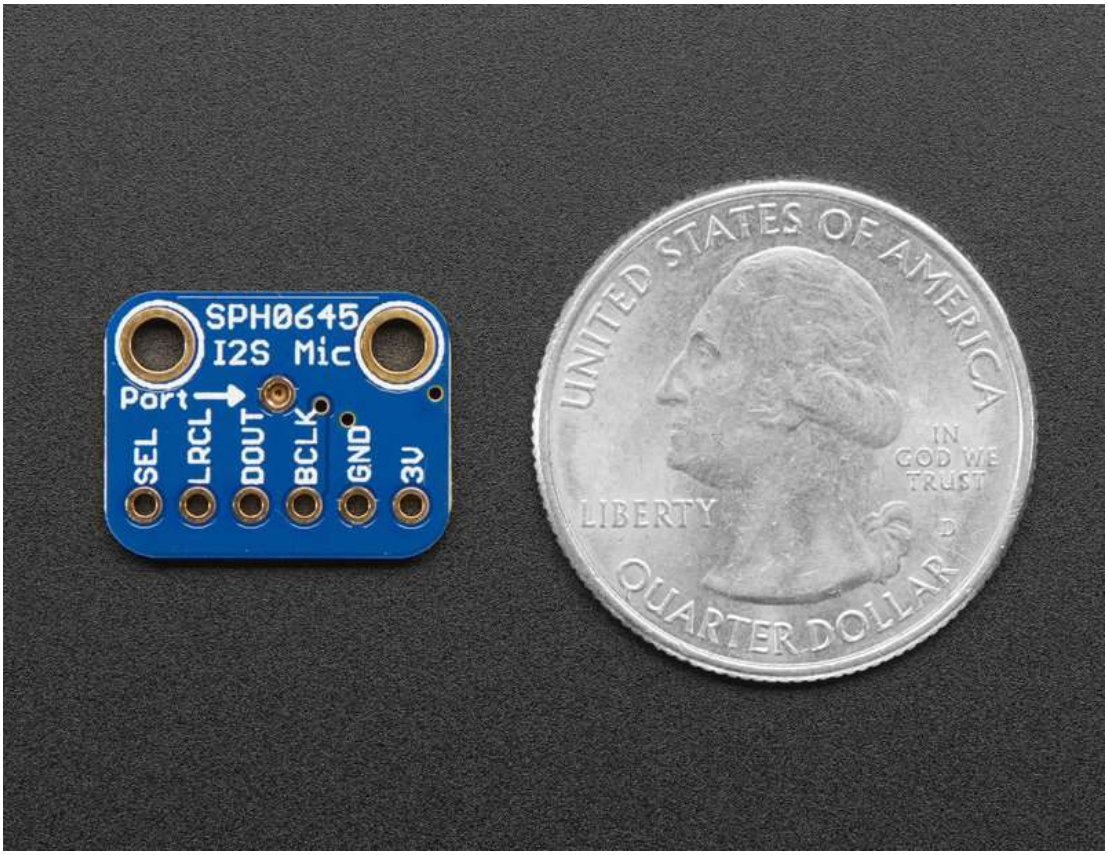
i2s

i2s 는 회로 위에서 사운드 신호처리를 위한 디지털 신호 전송 프로토콜. cd player회로에서 콤포넌트 사이에 데이터를 주고받기위해 i2s가 널리 쓰인다. 디지털 신호이므로 잡음 발생할 여지가 없다는 점이 강점이다. 참고: https://en.wikipedia.org/wiki/I%C2%B2S

Adafruit SPH0645lm4h i2s mems microphone

- sph0645lm4h mems digital microphone을 사용한다.
- mems 기술 사용하여 매우 작고,
- i2s 인터페이스를 사용한다.
- 작동전압 1.6v ~ 3.6v

사용시 주의할 점은 뒷면의 'port->' 라고 씌여진 부분이 소리가 들어가는 구멍이므로 방향을 착각하지 않도록 해야한다.



라즈베리 파이에 i2s mic는 디바이스 모듈(드라이버)가 기본 빌트인되어있지 않기 때문에 우리가 직접 모듈을 설치 해 주어야 한다. 따라서 마이크를 사용하기 위해서는 아래와 같은 과정이 필요하다.

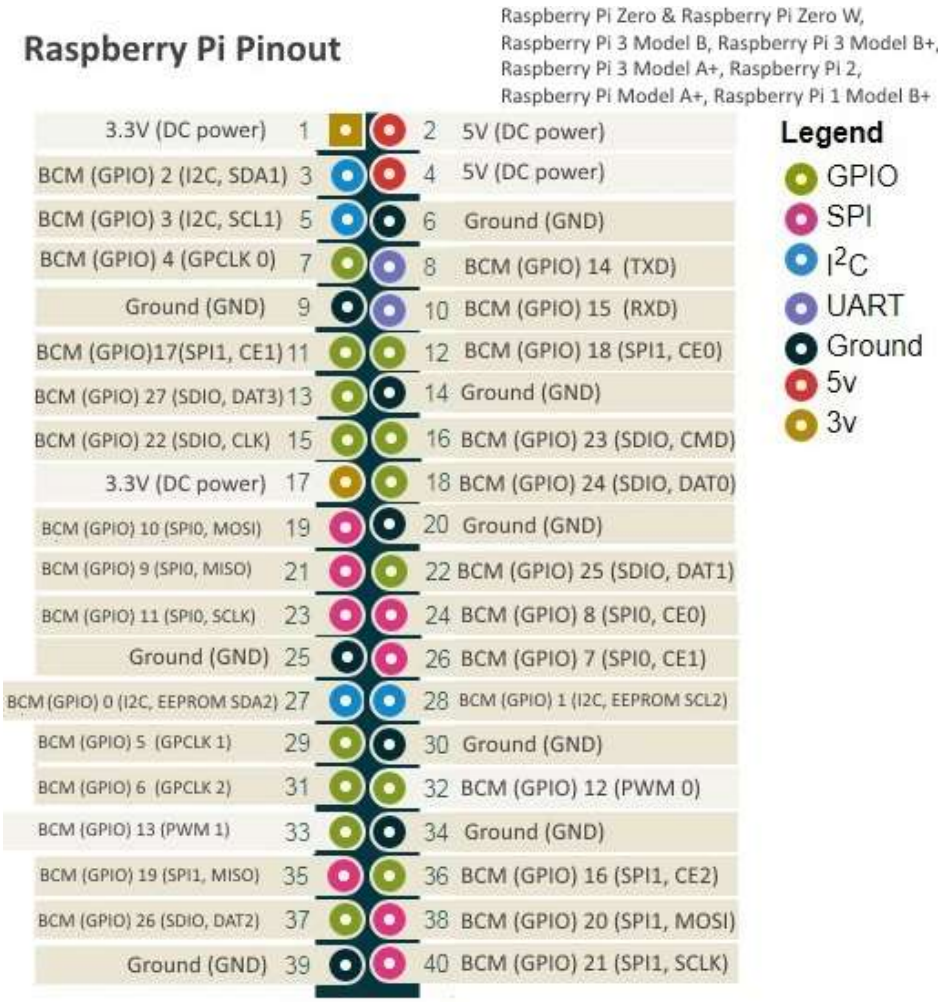
1. 하드웨어 연결
2. 디바이스 모듈 설치
3. 디바이스 트리 오버레이 설치

하드웨어 연결

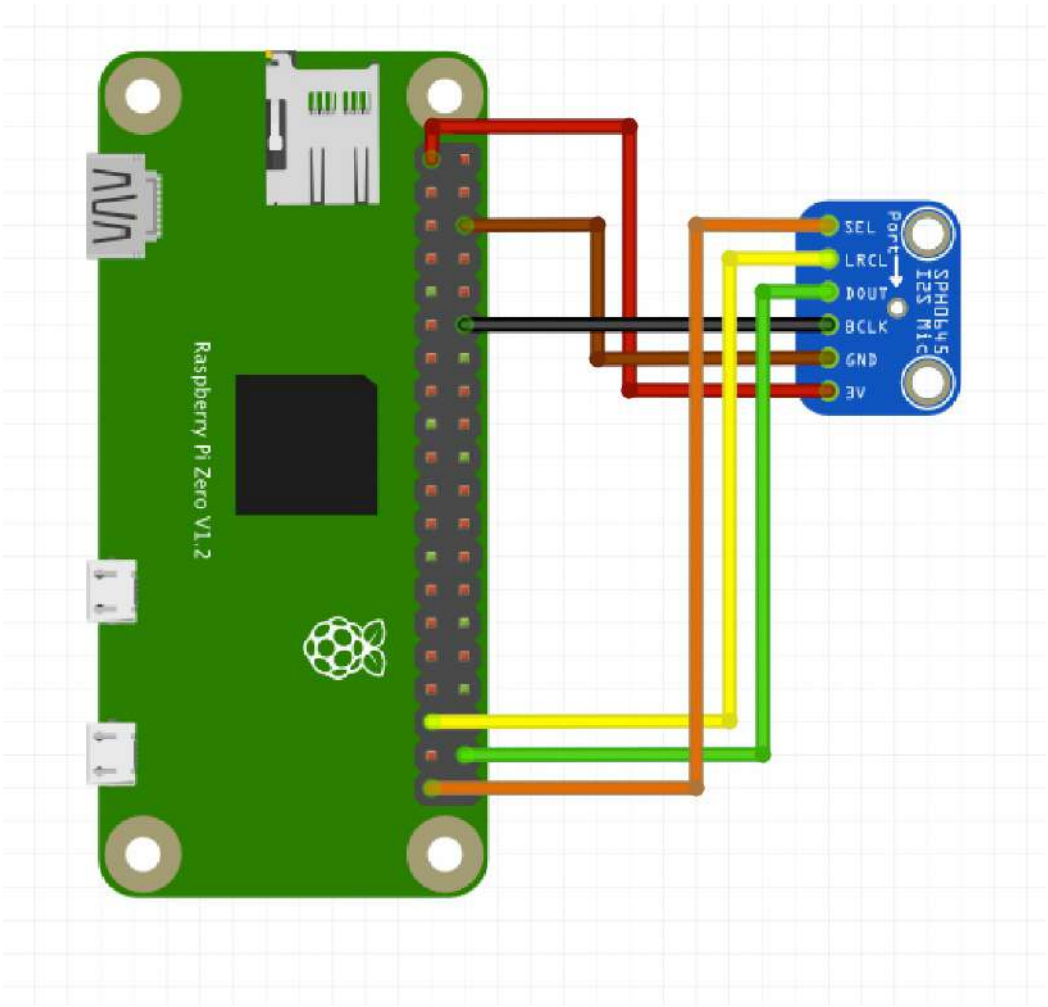
라즈베리파이의 PCM핀을 사용해야한다.

sph0645 microphone pin	**raspberry pi pin**	**설명**
3v	3.3v	전원선. 1.6~3.6v 범위에서 작동.
GND	GND	그라운드
BCLK	bcm18	clock. 마스터에서 보내는 이 신호에 맞추어 데이터가 오간다.
DOUT	bcm20	마이크로부터의 sound data
LRCLK (WS)	bcm19	스테레오를 구현하기 위해 dout은 규칙적으로 좌/우 신호를 번갈아가며 내보내는데, 그 주기를 이 핀으로 컨트롤 한다.(L/R clock)
SEL	GND(좌)(우측 마이크라면 3.3v에 연결)	channel select. 우측마이크인지 좌측마이크인지 결정. LRCLK가 Low면 좌측, High에 연결되어있으면 우측 마이크로 본다.

raspberry pi GPIO 핀 배열표


Raspberry Pi Pinout
Raspberry Pi Zero & Raspberry Pi Zero W,
Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+,
Raspberry Pi 3 Model A+, Raspberry Pi 2,
Raspberry Pi Model A+, Raspberry Pi 1 Model B+
3.3V (DC power) 1 2 5V (DC power)
BCM (GPIO) 2 (I2C, SDA1) 3 4 5V (DC power)
BCM (GPIO) 3 (I2C, SCL1) 5 6 Ground (GND)
BCM (GPIO) 4 (GPCLK 0) 7 8 BCM (GPIO) 14 (TXD)
Ground (GND) 9 10 BCM (GPIO) 15 (RXD)
BCM (GPIO)17(SPI1, CE1) 11 12 BCM (GPIO) 18 (SPI1, CE0)
BCM (GPIO) 27 (SDIO, DAT3) 13 14 Ground (GND)
BCM (GPIO) 22 (SDIO, CLK) 15 16 BCM (GPIO) 23 (SDIO, CMD)
3.3V (DC power) 17 18 BCM (GPIO) 24 (SDIO, DAT0)
BCM (GPIO) 10 (SPI0, MOSI) 19 20 Ground (GND)
BCM (GPIO) 9 (SPI0, MISO) 21 22 BCM (GPIO) 25 (SDIO, DAT1)
BCM (GPIO) 11 (SPI0, SCLK) 23 24 BCM (GPIO) 8 (SPI0, CE0)
Ground (GND) 25 26 BCM (GPIO) 7 (SPI0, CE1)
BCM (GPIO) 0 (I2C, EEPROM SDA2) 27 28 BCM (GPIO) 1 (I2C, EEPROM SCL2)
BCM (GPIO) 5 (GPCLK 1) 29 30 Ground (GND)
BCM (GPIO) 6 (GPCLK 2) 31 32 BCM (GPIO) 12 (PWM 0)
BCM (GPIO) 13 (PWM 1) 33 34 Ground (GND)
BCM (GPIO) 19 (SPI1, MISO) 35 36 BCM (GPIO) 16 (SPI1, CE2)
BCM (GPIO) 26 (SDIO, DAT2) 37 38 BCM (GPIO) 20 (SPI1, MOSI)
Ground (GND) 39 40 BCM (GPIO) 21 (SPI1, SCLK)
Legend
GPIO
SPI
I2C
UART
Ground
5v
3v

마이크 결선


Raspberry Pi Zero V1.2
SEL
LRCL
DOUT
BCLK
GND
3V
SPH0645
I2S MIC

디바이스 모듈 설치

- 미리 컴파일 돼어있는 마이크 디바이스의 디바이스 모듈을 다운로드한다.

```
$ git clone https://github.com/ssafy-embedded-project/mems-mic-
deviceModule.git
...
```

- 해당 디바이스 모듈을 `/lib/modules/$(uname -r)` 위치에 복사하고 `depmod -a` 명령으로 dependency 를 정리한다.

```
$ cd mems-mic-deviceModule/ics43432-pi0
$ sudo cp ics43432.ko /lib/modules/$(uname -r)
$ sudo depmod -a
```

디바이스 트리 오버레이 설치

디바이스트리 오버레이는 추가되는 디바이스만 디바이스트리에 선택적으로 추가해 '오버레이' 하도록 한다. 참고: https://www.raspberrypi.org/documentation/configuration/device-tree.md

- `.dtbo` 오버레이를 `/boot/overlays`에 설치한다.

```
$ sudo cp i2s-soundcard.dtbo /boot/overlays
```

- config.txt 에 부팅시 디바이스 사용함을 명시한다.

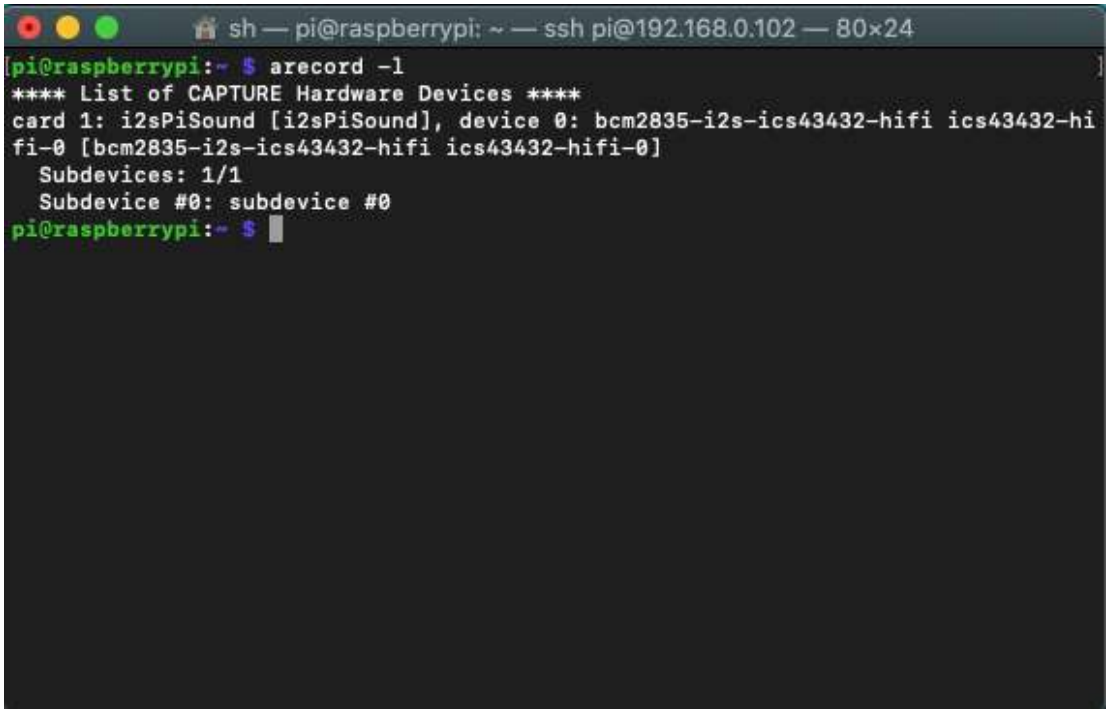
```
$ sudo nano /boot/config.txt

... 아래 내용 uncomment한다.
dtparam=i2s=on
dtparam=audio=on

... 아래 내용 추가한다.
dtoverlay=i2s-soundcard,alsaname=i2sPiSound
```

- 재부팅 후 arecord 프로그램으로 확인해보면 마이크가 설치된 것을 볼 수 있다.

```
$ sudo reboot
...
$ arecord -l
```



녹음 테스트

마이크 작동하는지 10초 녹음 테스트해보자. 스피커는 연결돼있지 않으므로 pc로 옮겨서 들어본다.

```
$ arecord -c1 -r48000 -fS16_LE -twav -d10 -vv test.wav
```

arecord 사용법 참고: http://manpages.ubuntu.com/manpages/trusty/man1/aplay.1.html

---

pyaudio

pyaudio는 오디오 I/O을 위한 closs-platform 라이브러리 PortAudio의 파이썬 구현이다. 오디오 데이터의 획득, 기록, 교환등에 비교적 간편하게 사용할 수 있다. 참고: https://people.csail.mit.edu/hubert/pyaudio/docs/

설치

```
$ sudo apt-get install portaudio19-dev
$ pip3 install pyaudio
```

마이크 입력 기본 코드

마이크 입력으로부터 바이너리 스트림 raw data를 획득한다.

```
# pyaudio-mic-stream.py

import pyaudio
import queue
import time
```

```
# 디지털 사운드 인코딩 특성
RATE = 16000 # Hz
CHUNK = int(RATE/10) # 연속되는 음성 데이터를 끊어 처리하기 위한 버퍼크기. 100ms

# 음성데이터 스트림
class MicrophoneStream:
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk
        self._buff = queue.Queue()  # 마이크로 입력받은 오디오 데이터를 chunk 단위로
queue에 저장한다.
        self.closed = True  # audio스트림 열려있는지 닫혀있는지

    # 클래스 생성 (스트림 시작)될 때
    def __enter__(self):
        self._audio_interface = pyaudio.PyAudio()
        self._audio_stream = self._audio_interface.open(    #
pyaudio.open()은 pyaudio.Stream object를 리턴.
            format = pyaudio.paInt16,  #16bit 다이나믹 레인지
            channels = 1,   # mono
            rate = self._rate,  # sampling rate
            input = True,   # 마이크로부터 입력되는 스트림임 명시
            frames_per_buffer = self._chunk,
            stream_callback = self._fill_buffer,    # 버퍼가 chunk만큼 꽉 찼을
때 실행할 콜백함수 등록 (non-blocking)
        )
        self.closed = False
        return self

    # 스트림 끝날 때
    def __exit__(self, type, value, traceback):
        self._audio_stream.stop_stream()
        self._audio_stream.close()
        self.closed = True
        self._buff.put(None)
        self._audio_interface.terminate()

    # 버퍼 찼을 때 콜백함수. pyaudio.Stream에서 호출되는 콜백은 4개 매개변수 갖고, 2개값
리턴한다. pyaudio문서 참고.
    def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
        self._buff.put(in_data) # data를 queue에 넣기
        return None, pyaudio.paContinue

    # python generator, 한 라운드의 루프마다 현재 버퍼의 내용을 모아서  byte-stream을
생산함.
    def generator(self):
        while not self.closed:
            chunk = self._buff.get() # 큐에서 데이터 가져오기
            if chunk is None:
                return

            # 큐에 더이상 데이터가 없을 때 까지 data에 이어붙임
            data = [chunk]
            while True:
                try:
                    chunk = self._buff.get(block=False)
                    if chunk is None:
```

```
                        return
                    data.append(chunk)
                except queue.Empty:
                    break

            yield b''.join(data)    # byte-stream

def main():
    # mic로부터 오디오스트림 생성
    MicStream = MicrophoneStream(RATE, CHUNK)
    with MicStream as stream:
        audio_generator = stream.generator()

        for val in audio_generator:
            print(val)  # raw data 출력

if __name__ == '__main__':
    main()
```

Note: portAudio관련하여 다수의 에러 메시지가 발생한다. 프로젝트 진행에는 무리가 없으므로 무시하도록 한다.

Note: 3주차 google cloud platform에 계정을 개설하고 활용하기 위해서는 신용카드(체크카드)가 필요하다. 다음 시간에 준비토록 한다. (결제는 되지 않습니다.)

추가과제 (optional)

- 차량 제어를 위한 셸을 제작해본다.
- 버튼입력을 통해 직관적/ 즉각적으로 차량 작동되도록 한다.