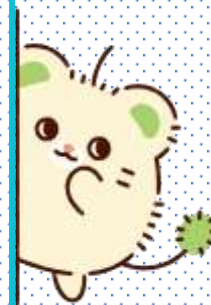




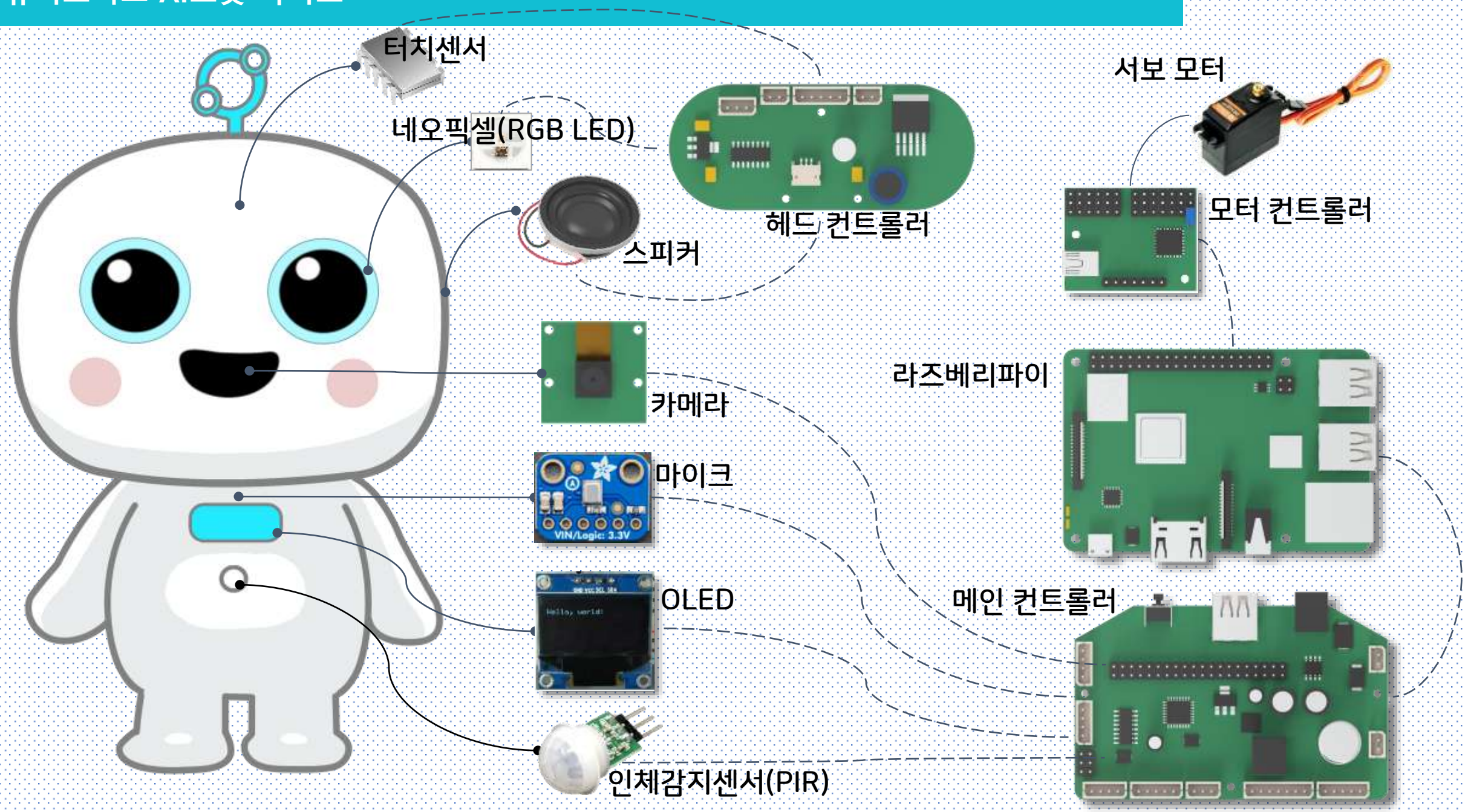
내 손으로 만드는 AI 자율행동 로봇

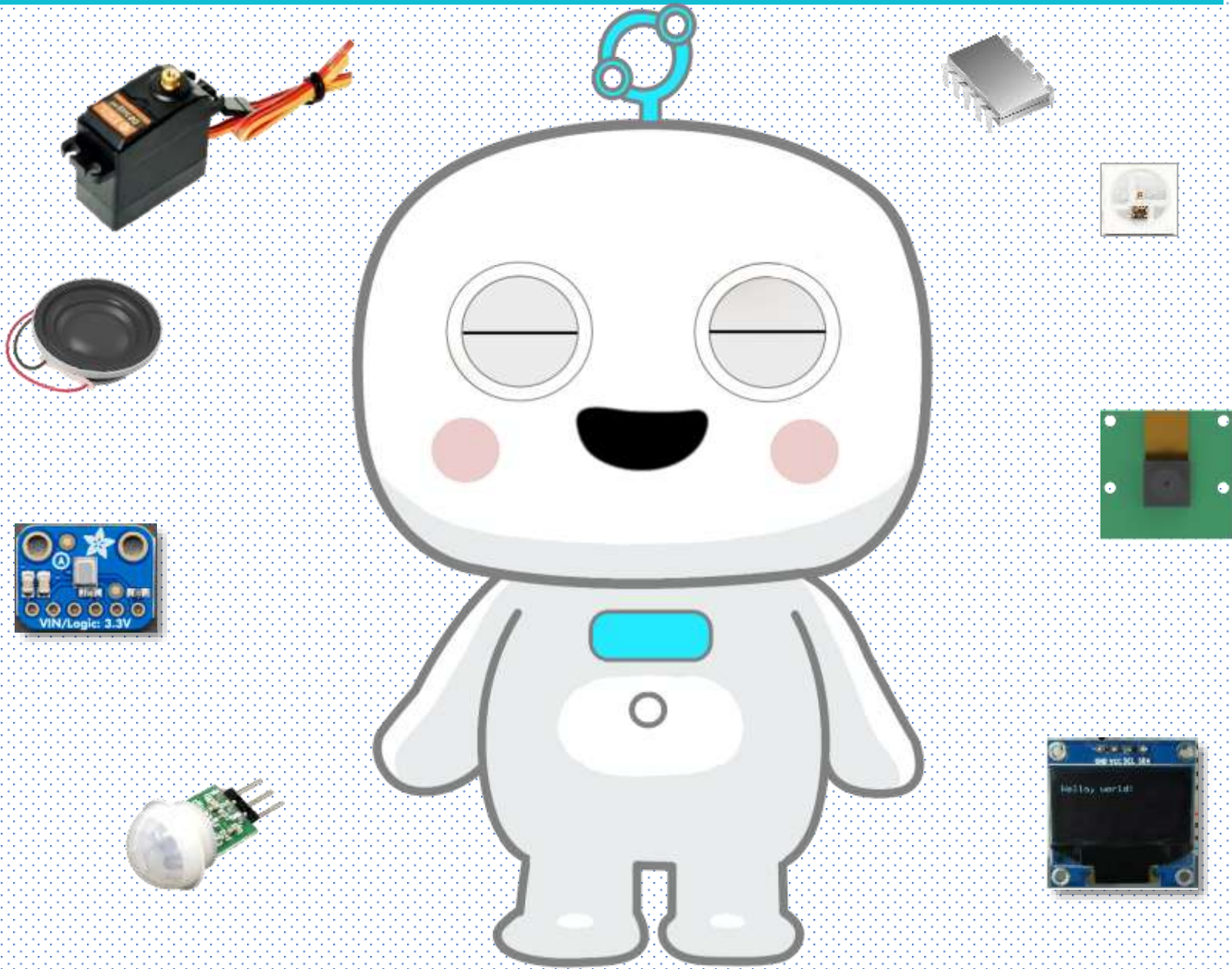


로봇 이해하기



휴머노이드 AI로봇 파이보





네오피셀

디지털신호로 LED의
빛을 조절할 수 있는 전
자부품

MIC

마이크, 소리를 전기신호로 변
환해주는 장치

OLED

전기에 자극받아 빛을 내
는 물질. 휴대폰, TV, 카메
라 화면에 쓰이는 LED

서보모터

파이보의 구동부 역할,
10개의 서보모터를 통해
움직임 구현

터치센서

인체가 닿으면 신호를
감지할 수 있는 센서

스피커

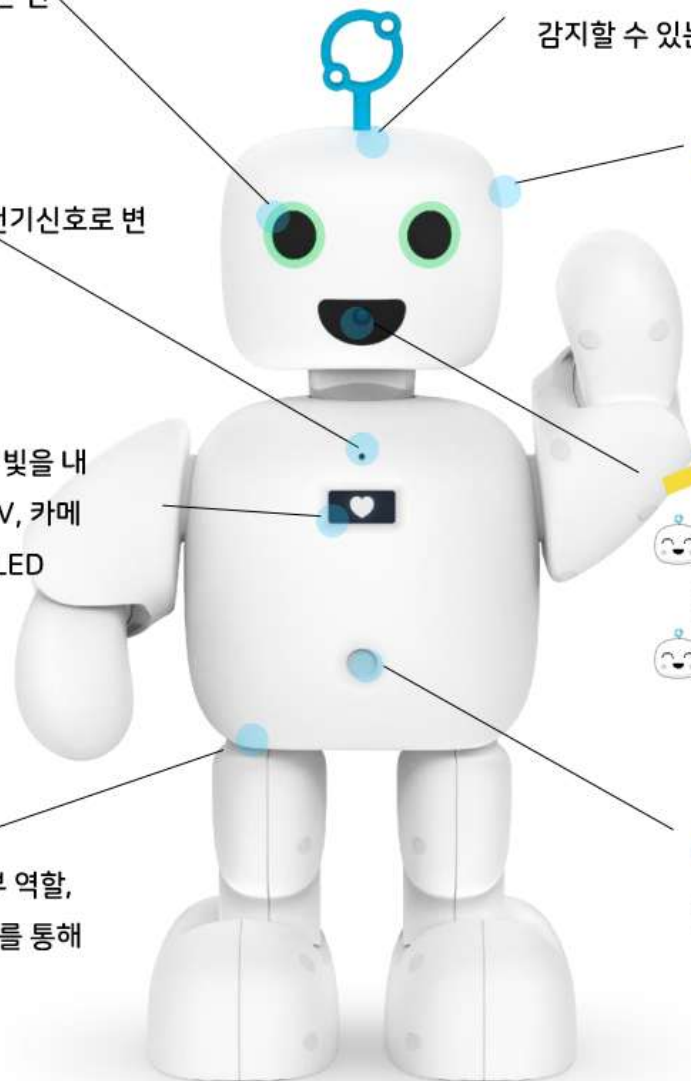
음악이나 음성합성
이 이루어질 수 있도
록 소리를 출력하는
장치

카메라

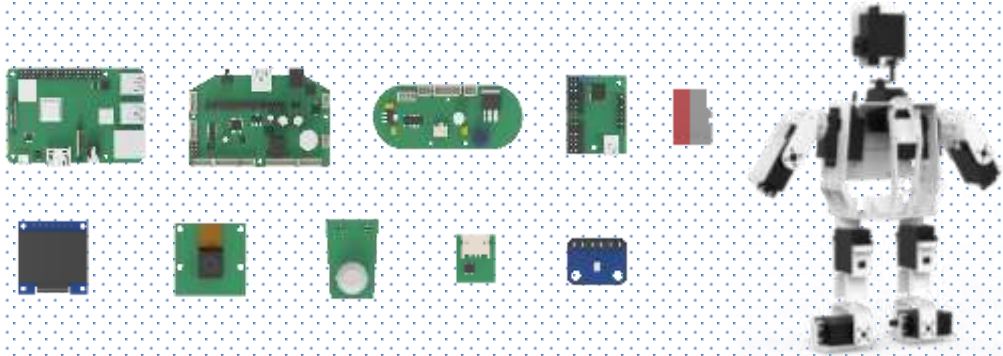
시각 정보를 읽어 전기적인
영상신호로 변환해주는 장치
컴퓨터비전 활용 가능

PIR센서

적외선을 통해 인체
를 감지하는 센서



- 로봇을 구성하는 하드웨어와 소프트웨어



하드웨어(Hardware)

로봇을 비롯한 컴퓨터/기계를 구성하는
물리적인 부품 또는 장치



소프트웨어(Software)

하드웨어를 동작하게 만들어주는
프로그래밍과 서비스 전체

- 하드웨어만 있으면 로봇이 동작할 수 있을까요?

파이보 연결하기





파이보를 꺼내 전용 충전기를 꽂는다.



파이보는 전원이 "꺼진" 상황에서만 구동부를 손으로 움직일 수 있다.
(단, 너무 세게 모터를 돌리지 않도록 유의)

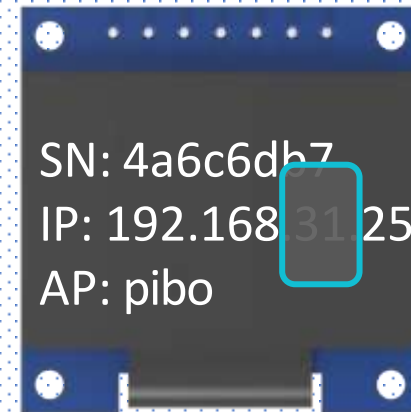
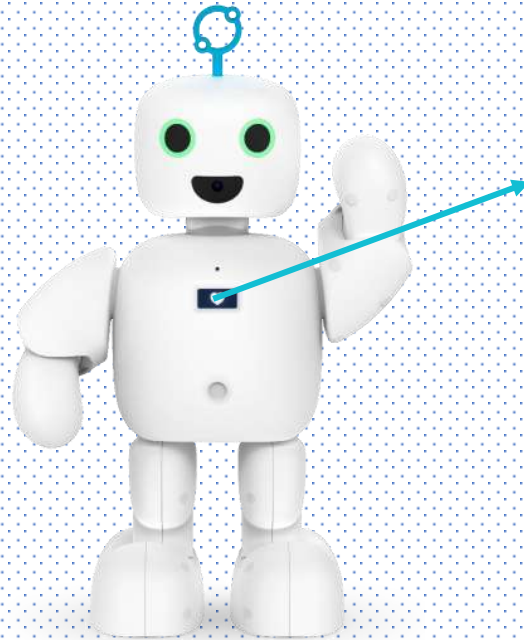
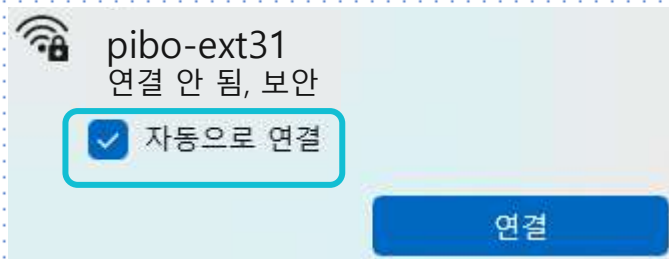


파이보의 전원은 눈에 불이 켜질 때까지 3초 이상 길게 누른다.
(파이보의 전원을 끌 때는 파이보의 눈이 **빨간색**이 될 때까지)

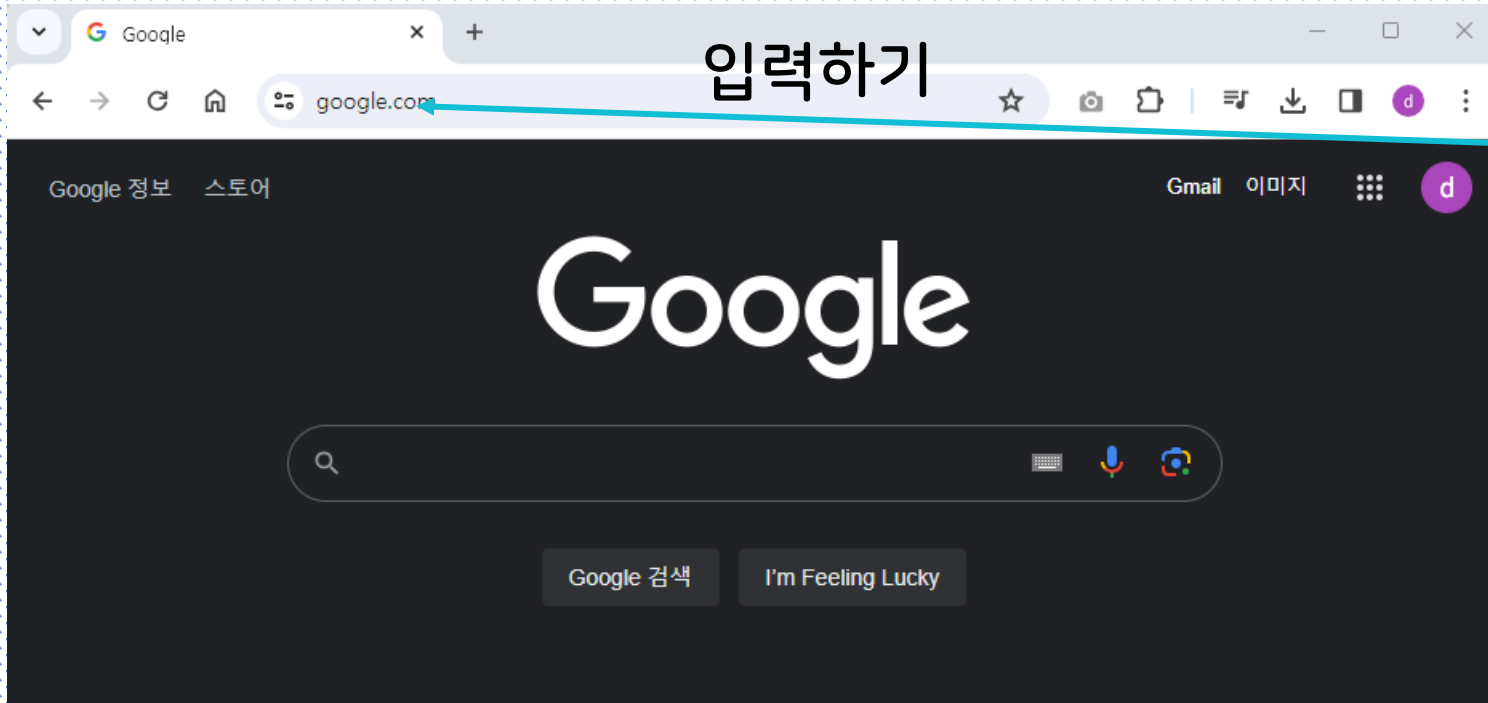


파이보의 가슴(OLED)에 시리얼 번호, IP정보가 나오면 부팅이 완료된다.

- 파이보의 부팅 및 와이파이 연결 상태 확인
- 노트북/태블릿 와이파이를 활용하여 파이보와 연결
 - 파이보 디스플레이를 확인하여 와이파이 목록에서 pibo-ext### 선택
 - 예) IP가 192.168.31.25인 경우 → 노트북 와이파이 pibo-ext31 선택
 - 연결 비밀번호 : !pibo0314
 - 와이파이 '자동으로 연결' 설정

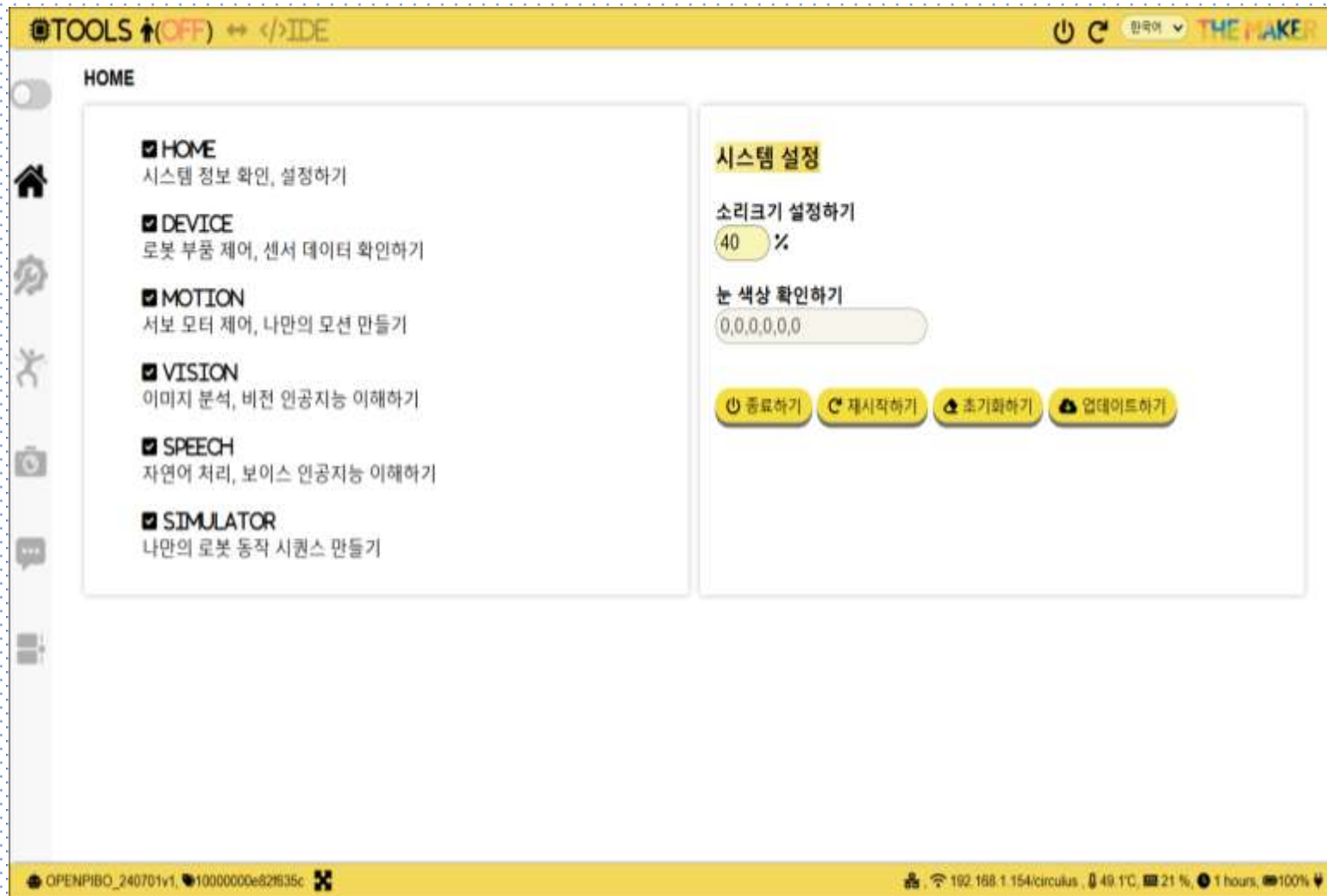


- 크롬 브라우저에서 파이보 메이커(pibo Maker) 접속

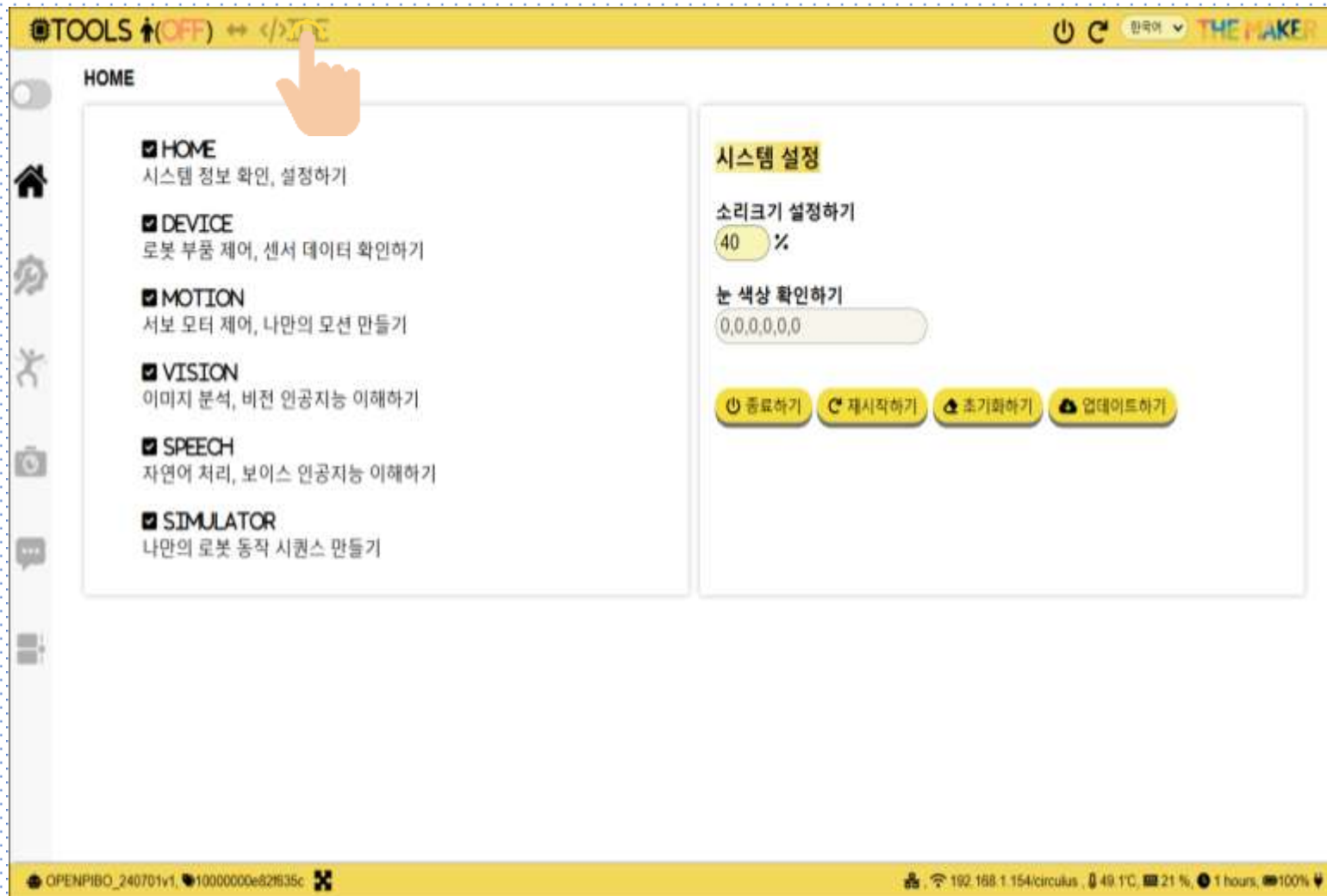


<파이보 디스플레이 정보>

- 크롬 브라우저에서 파이보 메이커(pibo Maker) 접속



- 크롬 브라우저에서 파이보 메이커(pibo Maker) 접속 – 코딩 메뉴로 이동



파이보 연결하기

The image shows the Pyboard IDE interface with several components labeled in Korean:

- 파이썬 코드 화면 테마 | 코드 창 확대**: Points to the top toolbar area.
- 블록/파이썬 가이드**: Points to the top toolbar area.
- 표준 입력**: Points to the top toolbar area.
- 파일탐색기**: Points to the file explorer on the left side.
- 결과 표시 공간**: Points to the output console on the right side.
- 코드 편집 공간 (블록코딩/파이썬)**: Points to the main code editor area.
- 이미지 뷰어, 오디오 플레이어**: Points to the bottom left area.
- 시스템 정보**: Points to the bottom right area.

The code editor displays the following Python code:

```
1 from openpiبو.device import Device
2 import time
3
4 device = Device()
5
6 # WRITE
7 device.eye_on(255,255,255)
8 time.sleep(1)
9
10 # RED BLUE
11 device.eye_on(255,0,0, 0,0,255)
12 time.sleep(1)
13
14 # OFF
15 device.eye_off()
16
```

프로그래밍과 코딩



- 사용자의 명령(요청)에 반응하고 동작하는 소프트웨어
- 주어진 명령을 해결하기 위한 처리 방법과 순서를 정리



- 프로그래를 만드는 일련의 과정(Program + ~ing)



- 컴퓨터가 우리의 명령/요청에 따라 일을 할 수 있도록 프로그램을 만드는 과정
 - 문제 인식
 - 프로그램 설계
 - 프로그램 구현
 - 테스트와 디버깅
 - 프로그램 유지보수

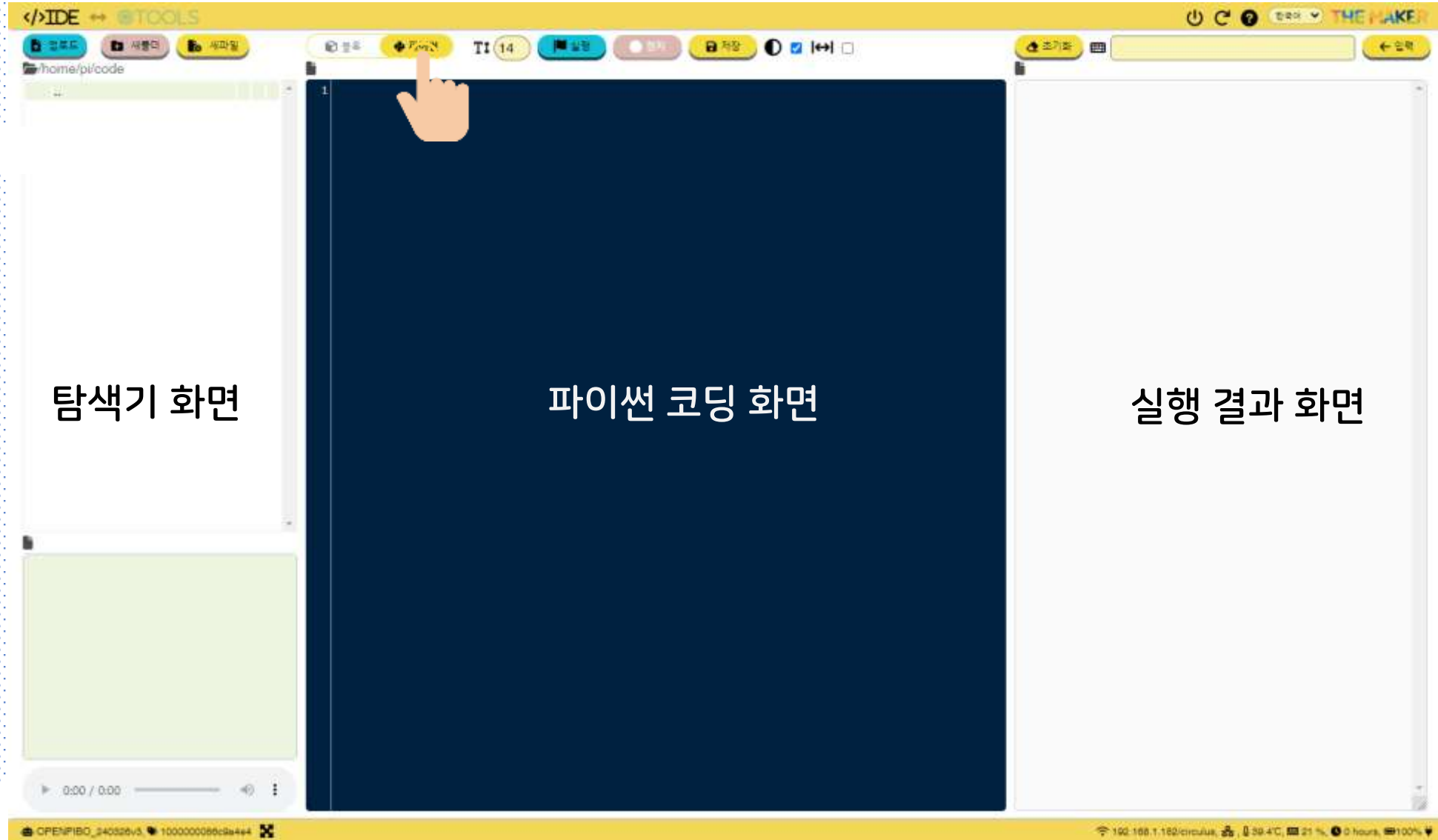
- 프로그래밍 과정 중 필요한 데이터/값을 컴퓨터가 이해할 수 있는 형태로 바꾸는 활동
 - 문제 인식
 - 프로그램 설계
 - 프로그램 구현
 - 테스트와 디버깅
 - 프로그램 유지보수

실습활동

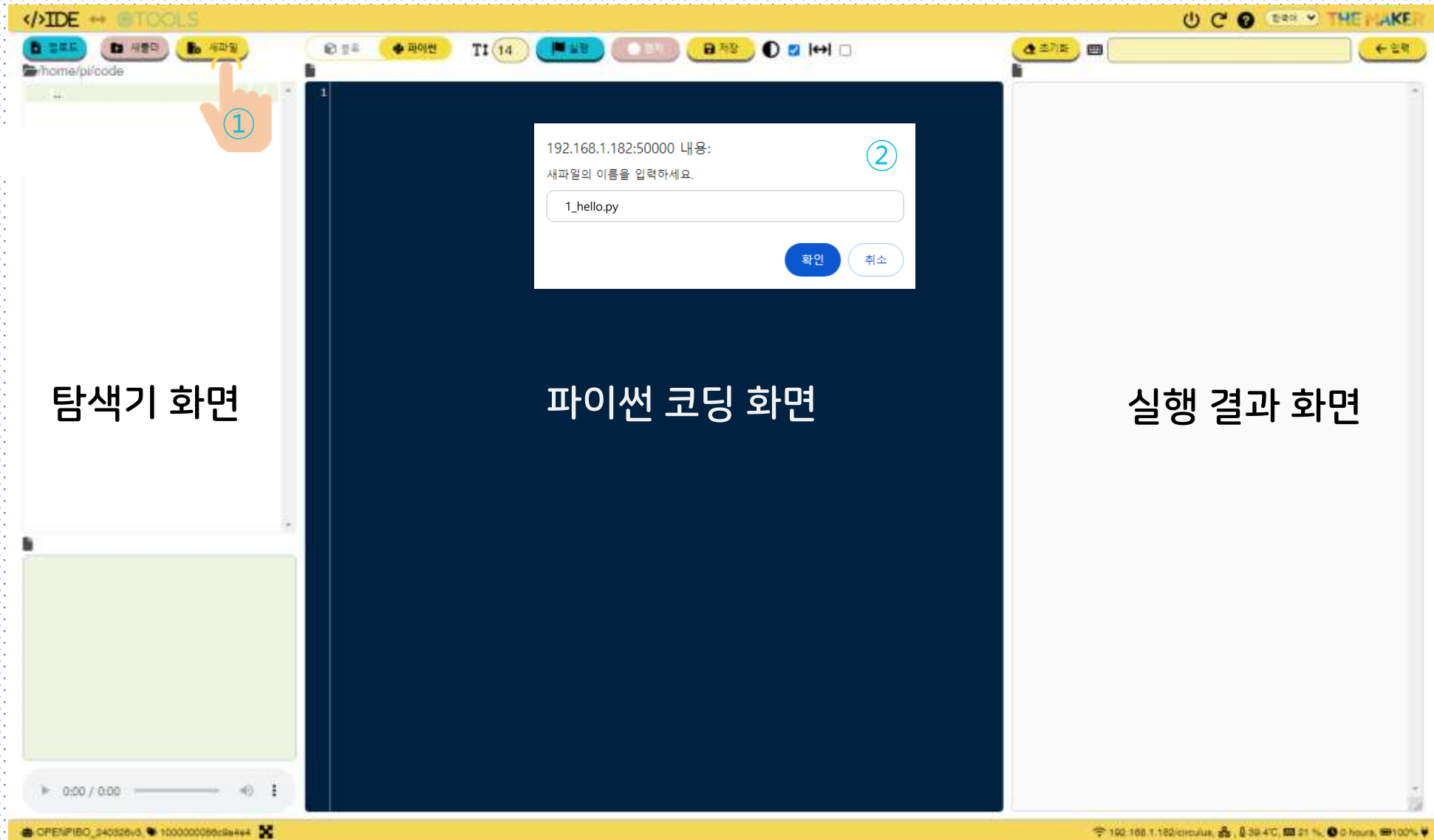
화면에 "안녕"을
출력해요!



- 파이보에게 명령을 내리기 위한 새 파일 생성



- 파이보에게 명령을 내리기 위한 새 파일 생성



- 파이보 메이커 IDE 출력창에 “안녕”을 출력합니다.

print() 안의 문자, 숫자 등을 출력창에 출력

```
print('안녕')  
  
print(123)  
  
print(2024, '년', 5, '월')
```


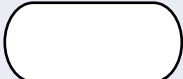

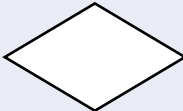


```
안녕  
123  
2024 년 5 월  
  
종료됨.
```

- 알고리즘

- 문제를 해결하기 위해 수행해야 할 과제를 순서에 맞게 나열한 절차

- 순서도

- 프로세스 과정을 순차적으로 표현한 흐름도

형태	명칭	설명
	흐름선	프로세스의 실행 순서를 나타낸다.
	터미널	하위 프로세스나 프로그램의 시작과 끝을 나타낸다.
	처리	데이터의 값, 형태, 장소를 변경하는 한 세트의 실행을 표현한다.
	판단	프로그램이 실행되는 두 가지 경로 중에 하나를 결정하는 조건부 실행을 나타낸다. 예/아니오 또는 참/거짓을 표현한다.
	입력/출력	데이터를 입력하거나 결과를 출력하는 경우와 같이 데이터의 입력과 출력을 나타낸다.
	출력	입력한 값의 출력을 나타낸다.

실습활동

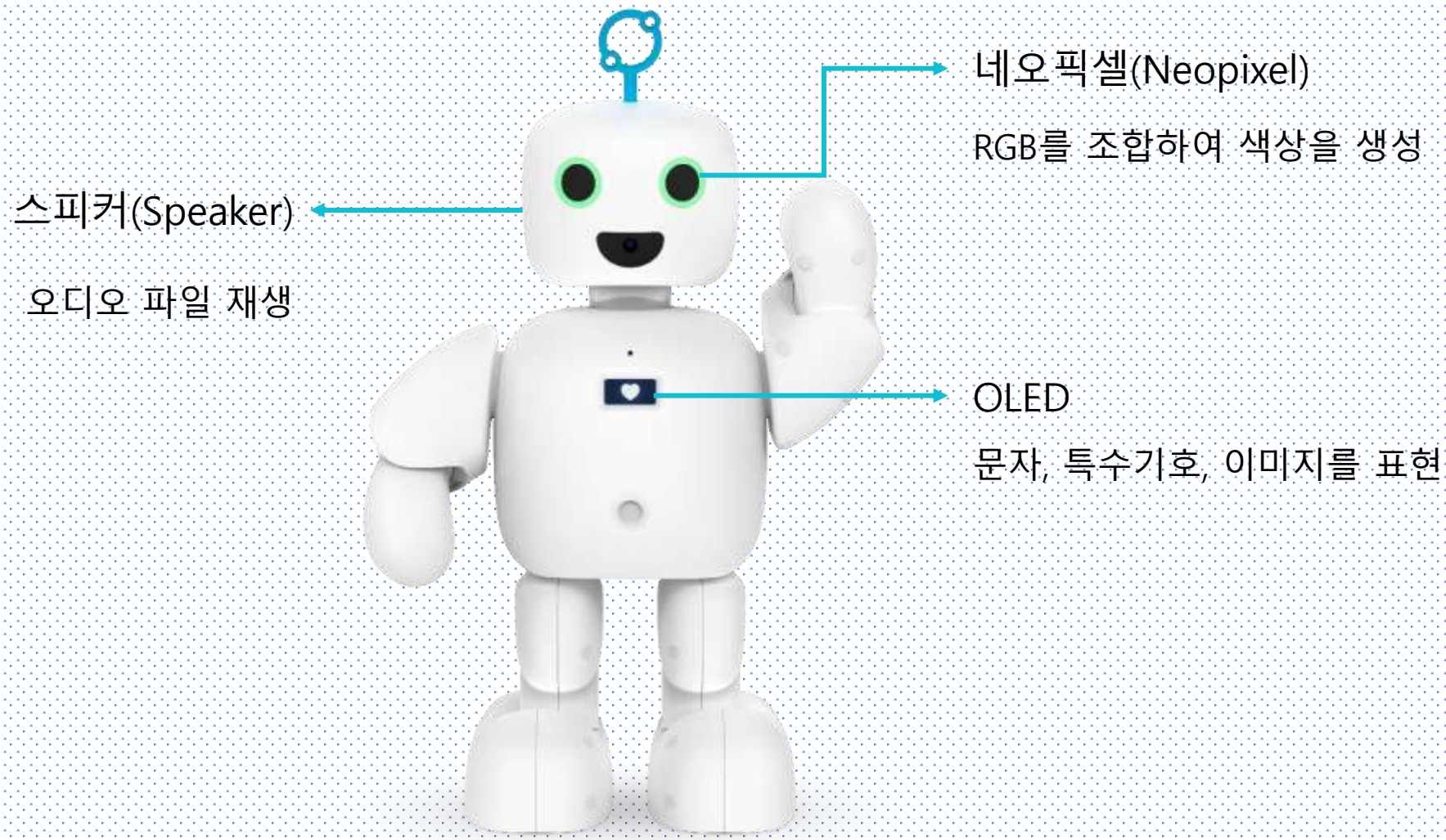
00000 과정을
순서대로 완성해요!



나를 소개하는 파이보

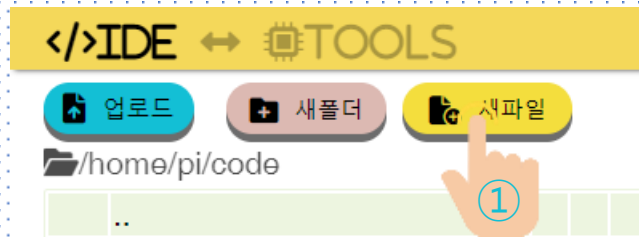


- 로봇의 전자부품을 제어하여 나만의 파이보를 만들 수 있어요!



- 처음 만난 친구들에게 파이보를 활용해 나를 소개합니다.
 - 네오픽셀 : 내가 좋아하는 색상을 표시해요.
 - OLED : 내 이름 또는 별명을 표시해요.
 - 스피커 : 파이보의 음성으로 내 소개를 해요.

- 새 파일을 생성합니다.

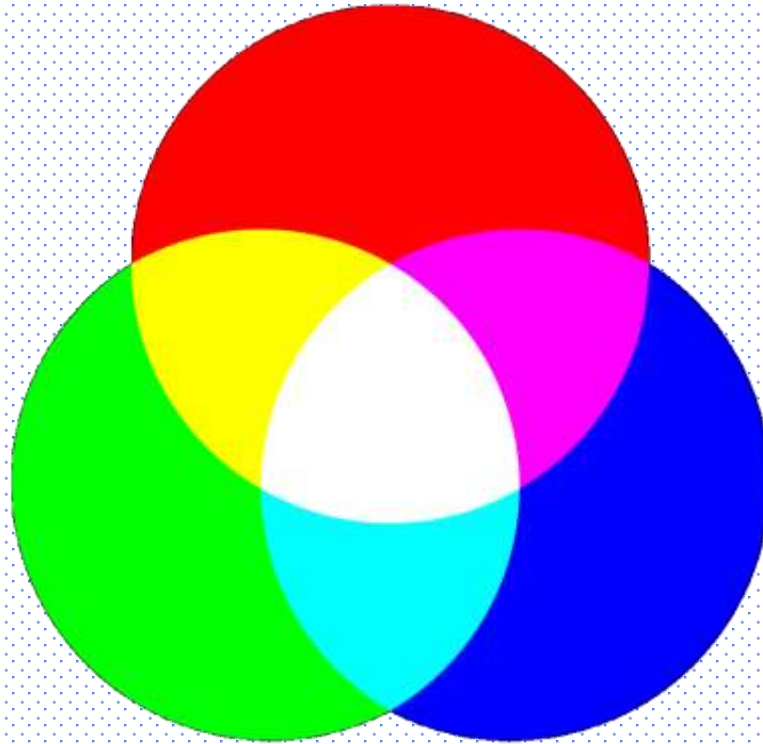


192.168.1.182:50000 내용: ②

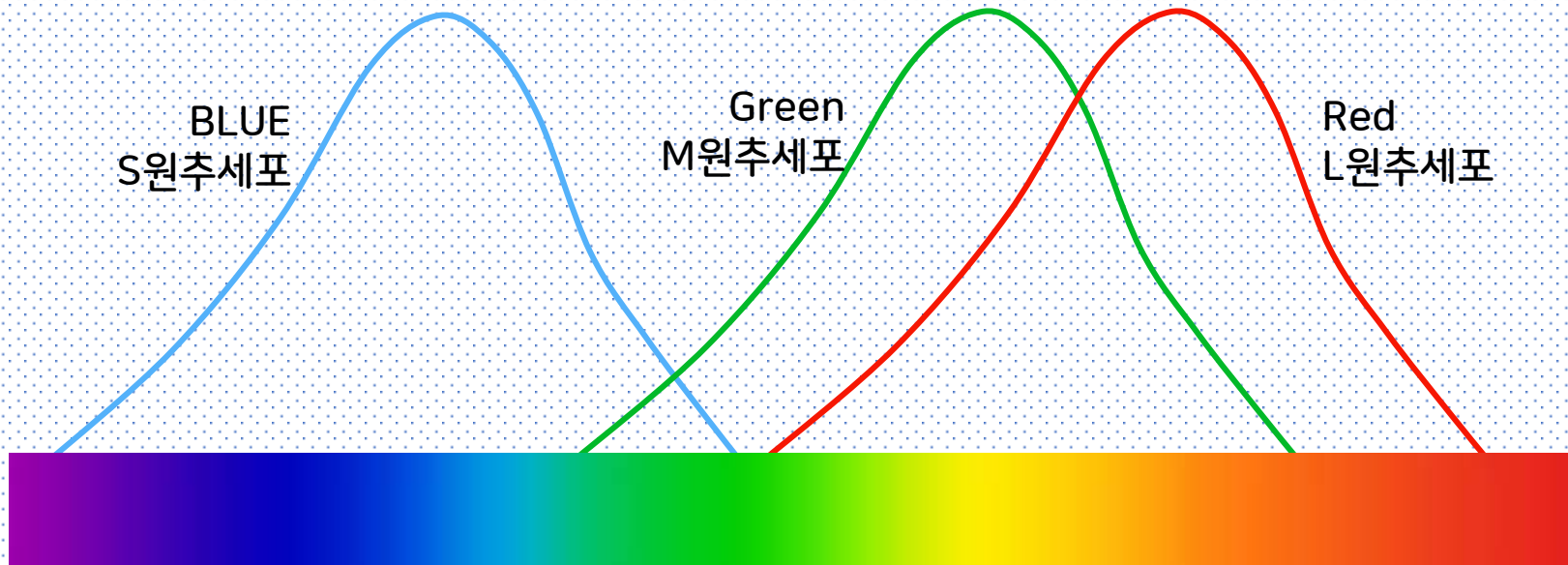
새파일의 이름을 입력하세요.

확인 취소

- 파이보의 눈 색상을 만들어요.
 - 빛의 삼원색(Red, Green, Blue)을 활용합니다.
 - 빛의 삼원색 합성해보기



- 왜 빨강, 초록, 파랑일까요?
 - 빛의 삼원색은 사람이 느낄 수 있는 세 가지 색상
 - 색감을 감지하는 인간의 원추세포가 RGB로 구성되어 있음



- 파이보의 눈 색상을 만들어요.

```
from openpibo.device import Device  
device = Device()  
device.eyе_on(255,255,255)
```

} 파이보의 전자부품 제어를 위한 초기 설정

→ 눈 색상 켜기

- 파이보의 눈 색상을 만들어요.

```
device.eye_on(255,255,255)
```

```
device.eye_on(255,255,255,0,0,0)
```

```
device.eye_off()
```

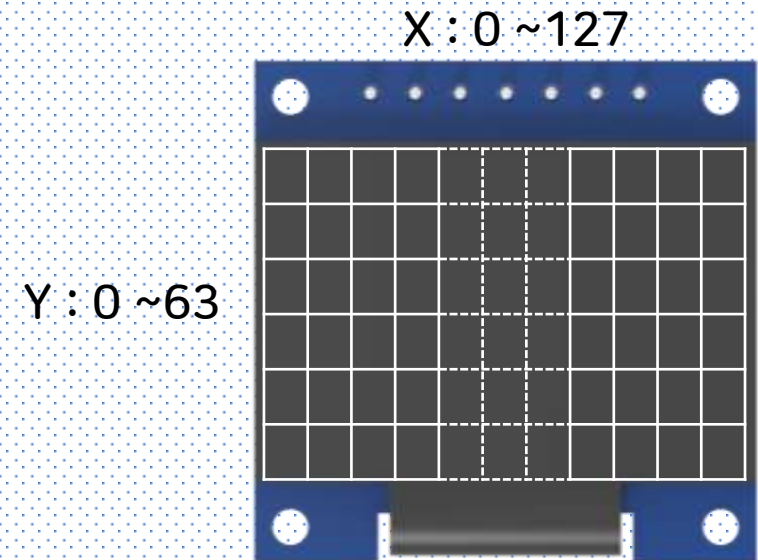
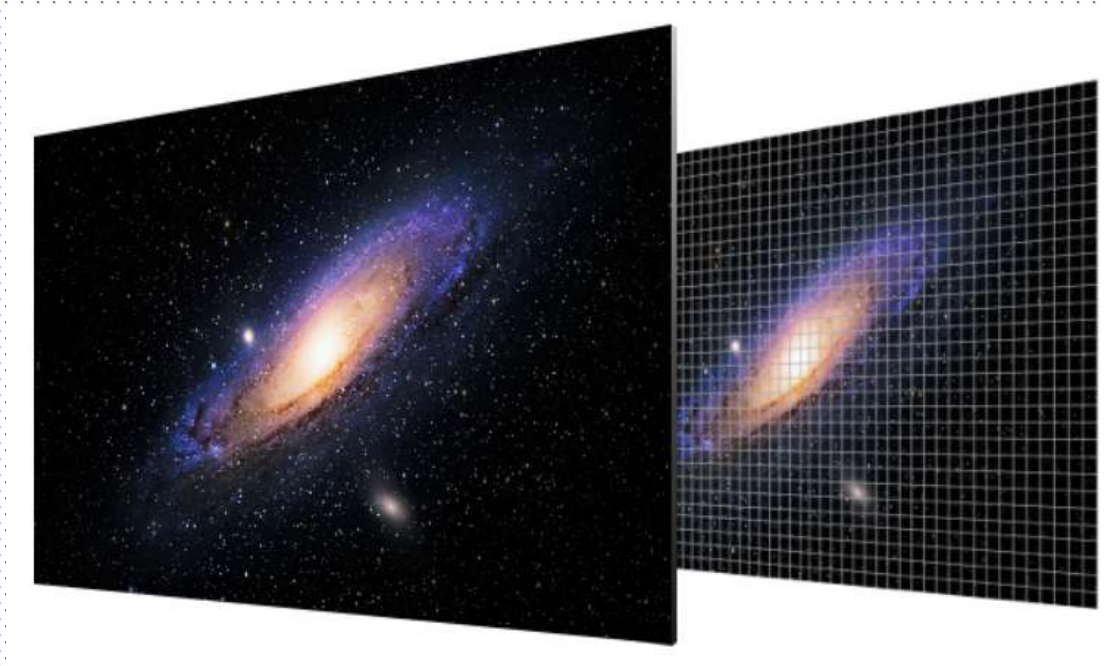
R, G, B를 조합하여 눈 색상을 만들 수 있습니다.

R, G, B를 조합하여 좌, 우 눈 색상을 만들 수 있습니다.

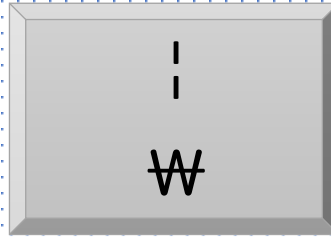
눈 색상을 끕니다.

```
== device.eye_on(0,0,0)
```

- OLED에 내 이름(별명)을 적어요.
 - OLED는 전류가 흐르면 스스로 빛을 내는 전자회로 소자
 - 픽셀(Pixel, 화면을 구성하는 단위) 각각의 밝기 제어 가능
 - 문자(한글, 영어, 숫자, 특수기호), 이미지, 도형 표시 가능



- OLED에 내 이름(별명)을 적어요.



★ 줄 바꾸기와 띄어쓰기로
글씨 위치를 조정할 수 있어요! ★

파이보WnHi



파이보Wn Hi



- OLED에 내 이름(별명)을 적어요.

```
from openpibo.oled import Oled  
  
oled = Oled()  
  
oled.set_font(size=20)  
oled.draw_text((0,0), '안녕하세요')  
oled.show()
```

파이보의 OLED 제어를 위한 초기 설정

파이보의 OLED 제어
(글씨 크기, 글씨 쓰기, 출력하기)

- OLED에 내 이름(별명)을 적어요.

```
oled.set_font(size=20)
```

```
oled.draw_text((0,0), '안녕하세요')
```

```
oled.show()
```



OLED에 표시할 문자 크기를 설정합니다.

OLED에 표시할 문자의 시작 위치와 내용을 입력합니다.

입력한 위치와 내용을 OLED에 표시합니다.

★ show()를 사용하지 않으면
입력한 내용을 출력하지 않습니다.

- OLED에 이미지를 출력해요.

```
oled.draw_image('/home/pi/openpibo-files/image/expression/smile.jpg')
```

OLED에 표시할 이미지를 준비합니다.

```
oled.show()
```

입력한 위치와 내용을 OLED에 표시합니다.

★ show()를 사용하지 않으면
입력한 내용을 출력하지 않습니다.



* 이미지 파일

1. IDE의 파일 탐색기의 "/home/pi/openpibo-files/image" 확인
2. 개별 업로드 (jpg, png)

- 스피커를 통해 내 소개를 말해요.

```
from openpibo.speech import Speech  
from openpibo.audio import Audio
```

```
speech = Speech()  
audio = Audio()
```

파이보의 음성합성, 오디오 출력을 위한 초기 설정

```
speech.tts(string='안녕하세요, 파이보입니다.', filename='voice.mp3', voice='main')  
audio.play('voice.mp3', 80)
```

음성합성 및 오디오 출력하기

- 스피커를 통해 내 소개를 말해요. – 음성 합성하기

```
speech.tts(string='안녕하세요, 파이보입니다.', filename='voice.mp3', voice='main')
```

↓
생성할 음성 내용을 입력합니다.

↓
음성을 저장할 위치와
파일 이름을 입력합니다.
★ 파일 입력만 입력한 경우
실행한 파일과 같은 위치에 저장합니다.

↓
음성 목소리를
선택합니다.

★ espeak, gtts, main, boy, girl, man1, woman1

- 스피커를 통해 내 소개를 말해요. – 오디오 출력하기

```
audio.play('voice.mp3', 80)
```

출력할 오디오 파일 경로와 이름, 볼륨 크기를 입력합니다.

★ 오디오 파일 경로를 입력하지 않으면
실행한 파일과 같은 위치에 있는 파일을 실행합니다.

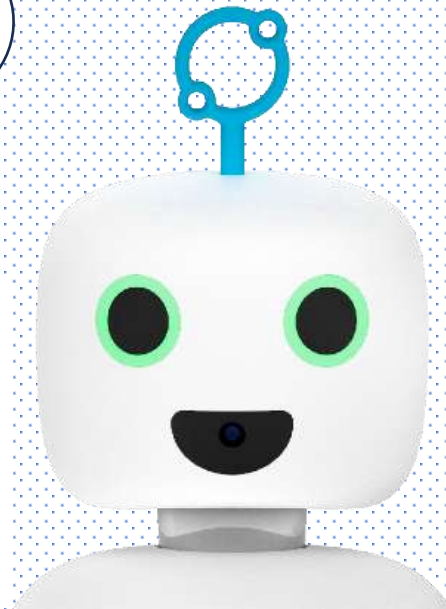
AI 자율행동 로봇

Part 1. 로봇 댄스 대회



- 팀을 구성하고 1분 이내 댄스 대회를 운영합니다.

우리는 신인 아이돌
파이브(pibo+ive)

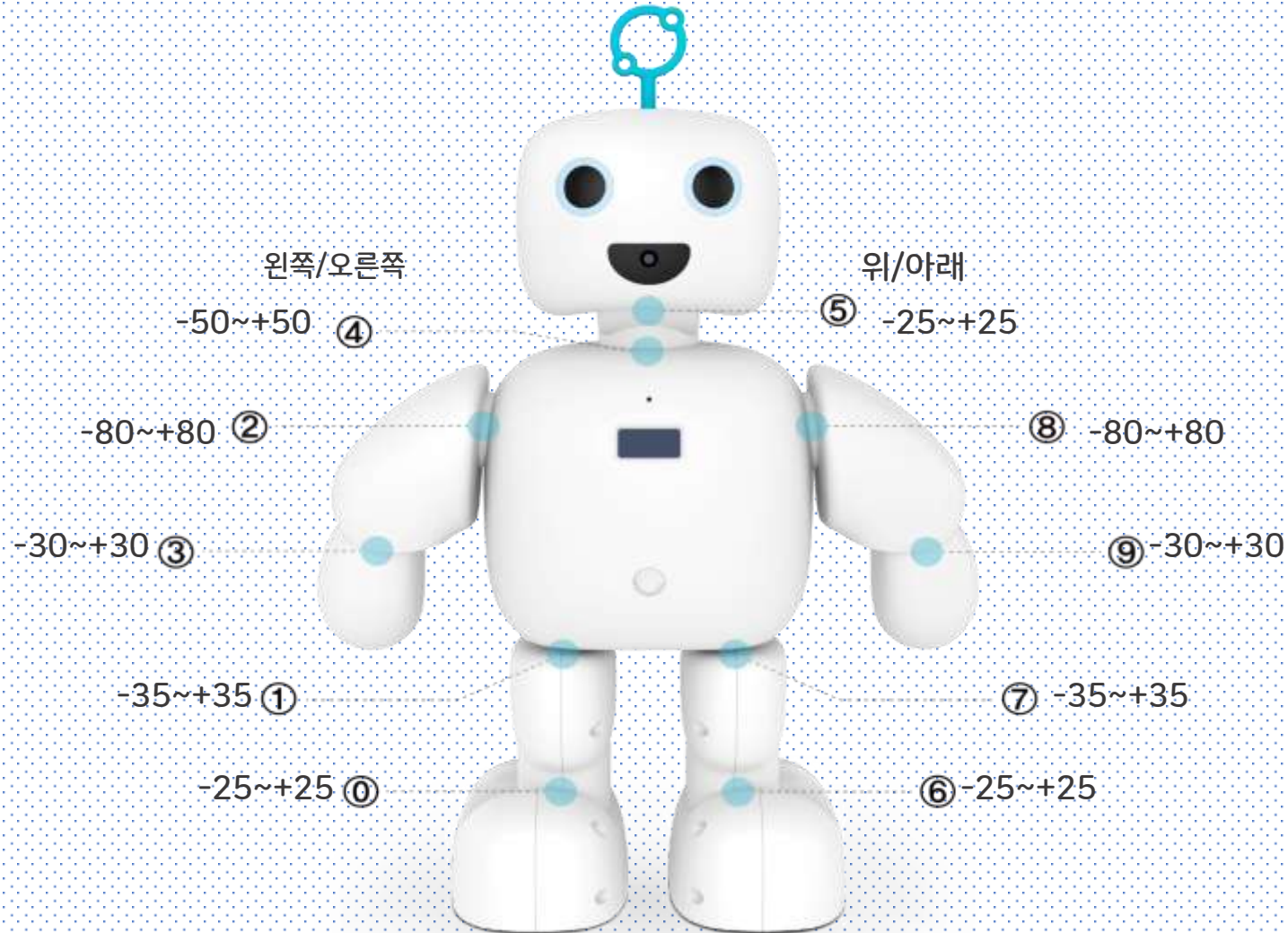


- 팀 빌딩을 위한 활동

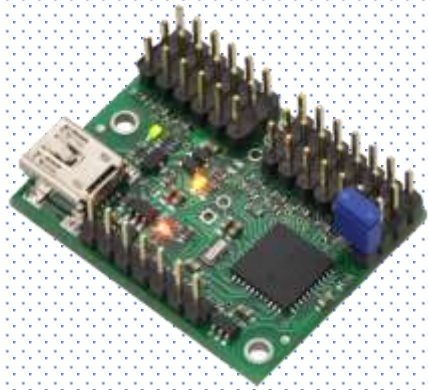
팀을 구성하기 위한 활동을 진행합니다.
특화형은 서로 모르는 친구들이 올 확률이 높기 때문에
아이스브레이킹과 팀 빌딩을 통해
분위기를 풀어주고
남은 수업을 잘 할 수 있도록 지원해주세요.

제공하는 물품 : 전지, 사인펜, 색연필

- 서보모터를 이해하고 각각을 제어합니다.
- 10개의 서보모터를 이용하여 다양한 모션 생성



- 파이보에는?



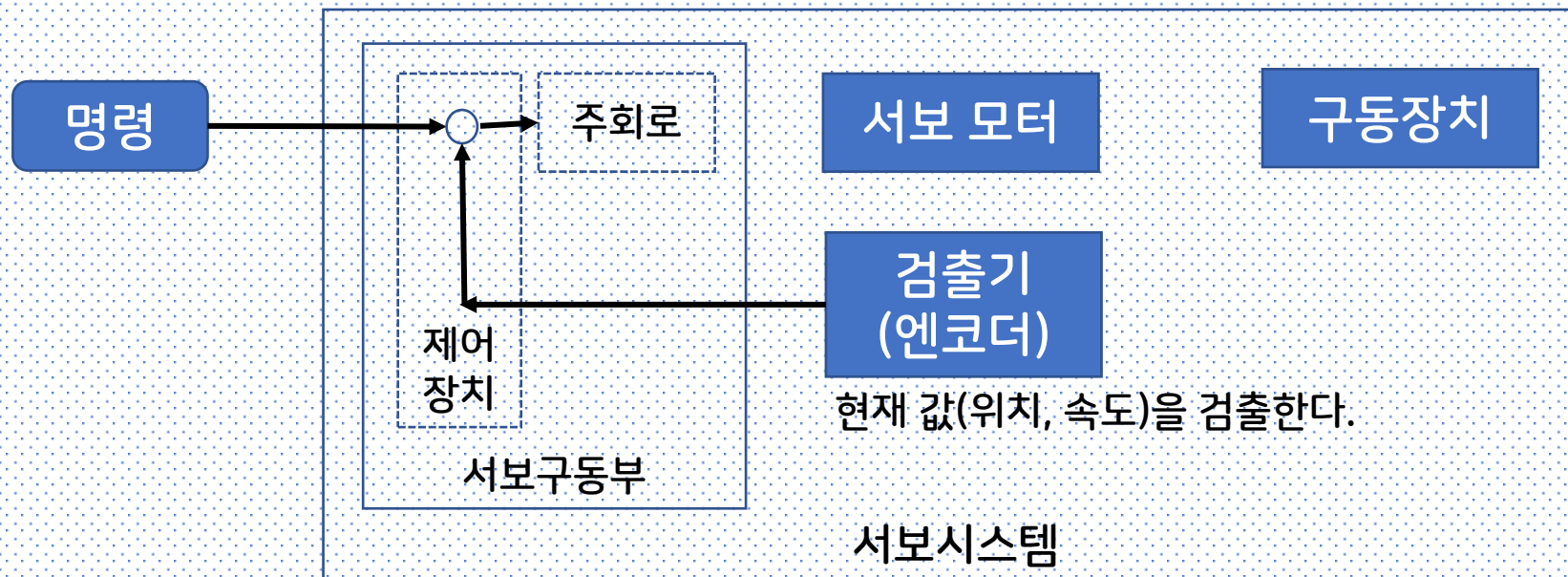
모터 컨트롤러
속도와 가속도 제어



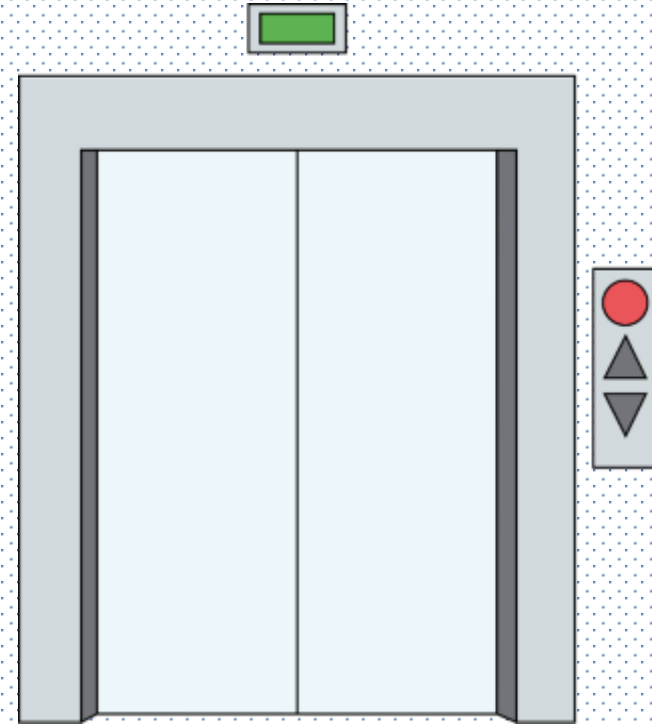
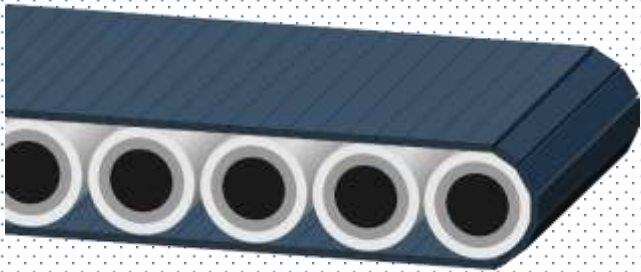
서보 모터

- 서보모터란?

- 라틴어 'Servus(서버스)'에서 유래
- '추종한다', '따른다'는 의미로 명령을 따르는 모터
- 구동시스템을 구축하고 있는 모터를 사용하여 위치와 속도를 제어
- 전기 에너지 → 역학 에너지(운동 에너지, 위치 에너지)



- 서보모터의 위치 제어
 - 지정한 위치로 정확하게 이동 또는 정지
 - 예) 엘리베이터, 컨베이어 벨트



- 서보모터의 속도 제어

- 크기, 무게가 변동해도 일정한 속도로 움직이게 하기 위해 속도를 가감
- 예) 에스컬레이터



- 참고 영상



The screenshot shows a robot control interface. On the left is a vertical toolbar with icons for home, settings, a stick figure (highlighted with a red box), camera, chat, and a dashboard. Above the toolbar is a yellow 'TOOLS' button with a robot icon and '(ON)' status. To the right of the toolbar is a 'MOTION' section with a yellow toggle switch (highlighted with a red box). The main area features a 3D robot model with sliders for various joints: M5(Head Tilt) at 0, M4(Head Pan) at 0, M2(Right Arm) at -80, M8(Left Arm) at 80, M3(Right Hand) at 0, M9(Left Hand) at 0, M1(Right Leg) at 0, M7(Left Leg) at 0, M0(Right Foot) at 0, and M6(Left Foot) at 0. Above the robot is a '원래자세' (Original Pose) button (highlighted with a red box) and a '시간' (Time) field set to 0 with a '+ 추가하기' (+ Add) button (highlighted with a red box). Red arrows point from these elements to instructional text on the right.

TOOLS (ON)

MOTION

원래자세

시간: 0 + 추가하기

차렷 자세로 되돌아감

1 연결된 서보모터의 슬라이더를 움직여 동작을 먼저 설정

2 서보모터 이동을 완료할 시간 설정

1, 2의 과정을 반복하여 원하는 동작을 만듦

TOOLS (ON) </>IDE

한국어 THE MAKER

MOTION

원래자세 시간: 0 + 추가하기

반복: 1

▶ 실행하기

■ 정지하기

☒ 모두 지우기

3 동작 반복 횟수 설정 후 실행

시간	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9
1 초	0	0	-80	0	0	-15	0	0	80	0
2 초	0	0	-80	0	21	15	0	0	80	0
2.5 초	0	0	-80	0	-38	0	0	0	80	0
3 초	0	0	0	0	-38	0	0	0	0	0
5 초	0	0	0	30	-38	-20	0	0	0	-30

더블 클릭하면 원하는 한 줄만 삭제 가능

4 영어로 나만의 모션이름 정하기!

모션이름: nmotion

✓ 등록하기

↓ 불러오기

- 삭제하기

☒ 모두 지우기

📄 내보내기

📁 가져오기

/home/pi/mymotion.json

welcome

원하는 모션 예제 클릭, 수정하여 사용 가능, 모션 예제명을 적어 불러오기 후 실행 가능

예제: forward, backward, left, right, welcome, foot, happy, sad, clapping, wave1, wave2, dance1, dance2

TOOLS (ON) </>IDE

한국어 THE MAKER

MOTION

원래자세

시간: 0

+ 추가하기

반복: 1

▶ 실행하기

■ 정지하기

☒ 모두 지우기

시간	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9
1 초	0	0	-80	0	0	-15	0	0	80	0
2 초	0	0	-80	0	21	15	0	0	80	0
2.5 초	0	0	-80	0	-38	0	0	0	80	0
3 초	0	0	0	0	-38	0	0	0	0	0
5 초	0	0	0	30	-38	-20	0	0	0	-30

모션이름: nmotion

등록하기

↓ 불러오기

- 삭제하기

☒ 모두 지우기

내보내기

가져오기

/home/pi/mymotion.json

welcome

예제: forward, backward, left, right, welcome, foot, happy, sad, clapping, wave1, wave2, dance1, dance2

5

내가 만든 모션을 내보내 팀원과 공유

6

팀원이 만든 모션을 내 파이보에 저장

- 음악을 선택해요! – 파이보 내장 음악 선택



TOOLS (ON) ↔ </>IDE

DEVICE

눈 색상 조정하기

오른쪽

빨강 초록 파랑

0 0 0

왼쪽

빨강 초록 파랑

0 0 0

저장하기

오디오 사용하기

마이크

파일 \$~/myaudio/mic.wav

5 초 녹음하기 다시듣기 ...

음악재생 \$~/openpibo-files/audio

음악 선택 재생하기 정지하기

선택

내 오디오에 업로드 classic trot rock exercise ballad nature funk 업로드

저장 위치 확인

파이보에 저장된 음악을 선택

- 음악을 선택해요! – 파이보에 음악을 저장

TOOLS (ON) ↔ </>IDE

DEVICE

눈 색상 조정하기

오른쪽

빨강 초록 파랑

0 0 0

왼쪽

빨강 초록 파랑

0 0 0

저장하기

오디오 사용하기

마이크

파일 \$~/myaudio/mic.wav

5 초 녹음하기 다시듣기 ...

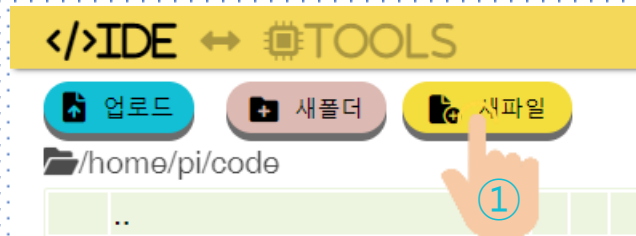
음악재생 \$~/openpibo-files/audio

음악 선택 재생하기 정지하기

내 오디오에 업로드하기 \$~/myaudio 업로드

음악 저장 위치 확인

- 파이썬을 활용하여 댄스를 실행할 수 있습니다. - IDE로 이동
- 새 파일을 생성합니다.



192.168.1.182:50000 내용: ②

새파일의 이름을 입력하세요.

3_dance.py

확인 취소

- 파이썬을 활용하여 댄스를 실행할 수 있습니다.

```
from openpibo.motion import Motion
from openpibo.audio import Audio
import time
```

```
motion = Motion()
audio = Audio()
```

```
print(motion.get_motion())
print(motion.get_motion(path='/home/pi/mymotion.json'))
```

```
audio.play('/home/pi/openpibo-files/audio/music/exercise.mp3', 80)
```

```
motion.set_mymotion('test', 3)
time.sleep(1)
motion.set_motion('wave1', 1)
```

```
audio.stop()
motion.set_motion('stop')
```

파이보의 서보모터, 오디오 제어를 위한 초기 설정

내장 또는 내 모션 목록 조회
선택한 음악 실행

모션 실행 및 제어

- 파이보에 내장된 모션을 사용할 수 있어요.

```
motion.set_motion('clapping1', 1)
```

반가움

clapping1, clapping2,
handshaking, bow,
greeting, welcome

머리 움직이기

head_h, spin_h,
yes_h, no_h

신남

wave1, wave2, wave3,
wave4, wave5, wave6,
happy1, happy2,
happy3, excite1,
excite2

침착함

stop, lookup, sleep,
breath1, breath2,
breath3

걸기

forward1,
forward2,
backward1,
backward2

리액션

cheer1, cheer2,
cheer3, think1,
wake_up1, wake_up2,
hand1, hand2, hand3,
hand4, handup_r,
handup_l, look_r,
look_l, speak1,
speak2

댄스

dance1, dance2,
dance3, dance4,
dance5

지루함

boring1, boring2,
sad1, sad2, sad3

다리 움직이기

left, left_half, right,
right_half, foot1,
foot2

- 파이썬을 활용하여 댄스를 실행할 수 있습니다

```
motion.get_motion()  
motion.get_motion(path='/home/pi/mymotion.json')
```

파이보에 저장된 내가 만든 모션 목록을 확인할 수 있습니다.

- ★ print()와 함께 사용하여 모션 목록을 출력
- ★ 파이보에 내장된 모션 목록 확인 : motion.get_motion()

```
motion.set_mymotion('test', 3)
```

실행할 내가 만든 모션 이름과 실행 횟수를 입력합니다.

```
motion.set_motion('wave1', 1)
```

파이보에 내장된 모션 중 실행할 모션 이름과 실행 횟수를 입력합니다.

```
motion.stop()
```

모션을 정지하고 차렷 자세를 합니다.

2_intro.py에서 사용한 네오픽셀, 디스플레이를 추가하여 좀 더 꾸밀 수 있습니다.

음악 외에 음성을 함께 재생할 수도 있습니다.

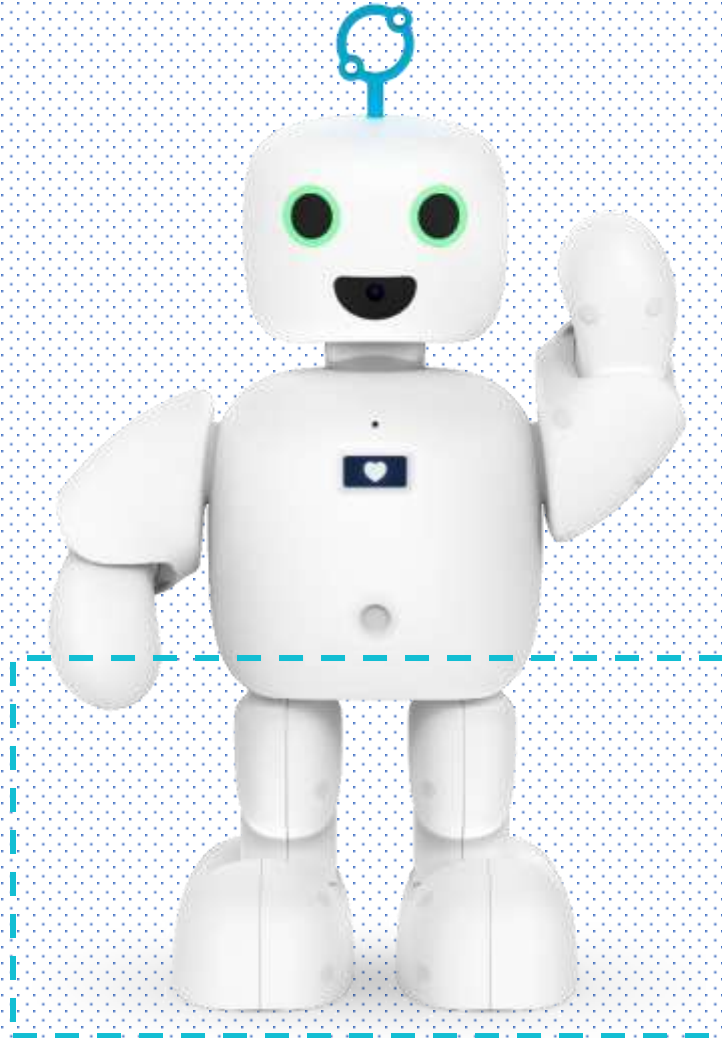
아이들이 만든 결과물을 촬영하여 공유해주세요.

AI 자율행동 로봇

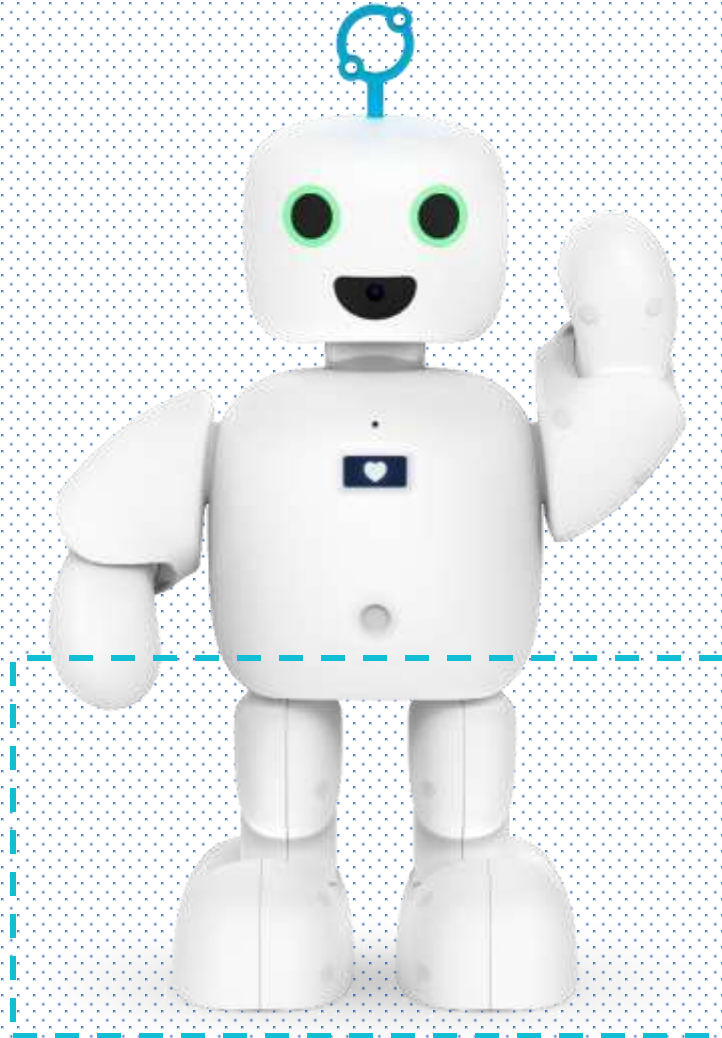
Part 2. 앞으로 전진하는 파이보



- 이족보행 휴머노이드 로봇 파이보
 - 두 다리를 이용하여 앞, 뒤, 좌, 우로 걷기 가능



- 이족보행 휴머노이드 로봇 파이보
 - 사람이 걷는 모습과 무엇이 다른가요?



- 걷기 모션을 실행하여 파이보가 걷는 모습을 관찰합니다.

```
motion.set_motion('forward1', 1)
```

걷기

forward1,
forward2,
backward1,
backward2

다리 움직이기

left, right

침착함

stop

- 모션 툴에서 걷기 모션을 확인하고 수정해봅니다.

TOOLS ON IDE THE MAKER

MOTION

↑ 관리자세 시간: 0 + 추가하기

M5(Head Tilt) 0

M4(Head Pan) 0

M2(Right Arm) -80 M0(Left Arm) 80

M3(Right Hand) 0 M0(Left Hand) 0

M1(Right Leg) 0 M7(Left Leg) 0

M0(Right Foot) 0 M0(Left Foot) 0

반복: 1 ▶ 실행하기 ■ 정지하기 : 모두 지우기

시간	M0	M1	M2	M3	M4	M5	M6	M7	M8	M9
0 초	0	0	-70	-25	0	0	0	0	70	25
0.3 초	10	0	-70	-25	0	0	20	0	70	25
0.6 초	10	0	-80	-25	20	0	20	-30	60	25
0.9 초	10	-30	-80	-25	20	0	20	-30	60	25
1.2 초	10	-30	-80	-25	20	0	0	-30	60	25
1.5 초	-20	-30	-80	-25	20	0	-10	-30	60	25
1.8 초	-20	30	-60	-25	-20	0	-10	-30	80	25
2.1 초	-20	30	-60	-25	-20	0	-10	30	80	25
2.4 초	0	30	-60	-25	-20	0	0	30	80	25

모션이름: 등록하기 불러오기 삭제하기 : 모두 지우기 : 내보내기 가져오기

~/mymotion.json

welcome, welcome_test

예제 forward, backward, left, right, welcome, foot, happy, sad, clapping, wave1, wave2, dance1, dance2

1. 모션 툴 예제에서 forward 모션을 선택합니다.
2. forward 모션을 실행합니다.
3. forward 모션을 나만의 스타일로 수정합니다.
4. forward 모션을 내 모션에 저장합니다.

저장 이름 : forward

- ★ 기존에 파이보에 저장된 forward와 내가 저장한 forward는 다른 모션으로 인식합니다.
기존에 파이보에 저장된 forward 모션 실행 : `motion.set_motion('forward1', 1)`
내가 저장한 forward 모션 실행 : `motion.set_mymotion('forward', 1)`

실습활동

내가 만든 forward 모션을
활용한 걷기 대회하기



- 어떤 파이보가 결승점에 제일 빨리 도착할까요?

친구들이 수정한 forward 모션을 이용하여
어떤 로봇이 제일 빨리 결승점에 도착하는지 시합합니다.

1. 마스킹 테이프를 이용하여 시작점과 결승점을 표시합니다.
2. 프로그래밍을 따로 하지 않고 모션 툴에서 모션을 실행하여 진행합니다.
3. 수업 참여 인원에게 따라 한번에 혹은 팀을 나눠 대회를 진행합니다.



AI 자율행동 로봇

Part 3. 이미지를 분류하는 파이보

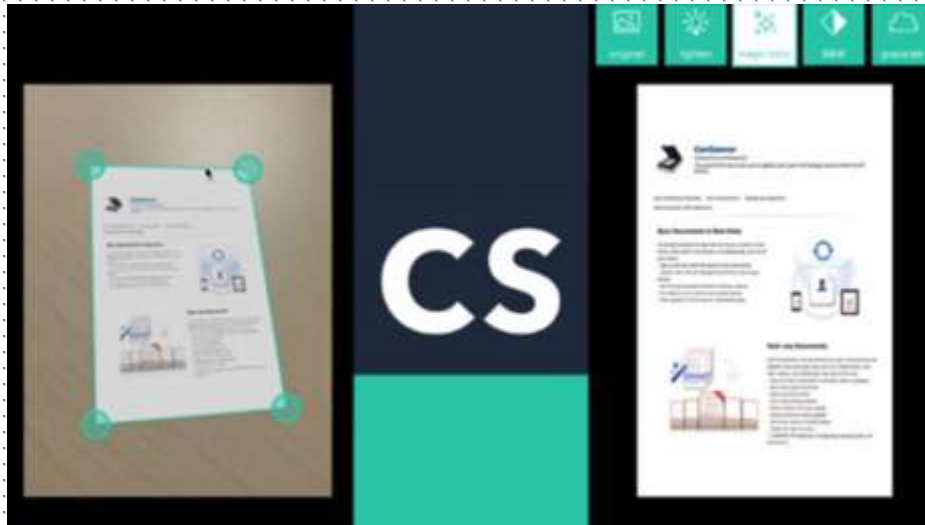


• 컴퓨터 비전

- 카메라를 통해 영상을 캡처하고 컴퓨터를 사용하여 의미 있는 정보를 추출하고 이해



9821480865132823066470938
4460955058223172535940812
8481117450284102701938521
1055596446229489549303819
6442881097566593344612847



- 컴퓨터 비전 활용 사례
자율주행 자동차



- TOOLS의 비전 메뉴에서 다양한 컴퓨터 비전을 체험할 수 있습니다.
- 토글이 'on' 상태인지 확인합니다.

VISION

카메라

사진저장하기 사진 10장 저장하기

카테고리

카메라 위치 조정 17.0

Vision 사용하기

기능 설정
카테고리

티쳐블머신 모델 \$~/mymodel: 업로드

마커길이: 5 cm

결과

기능 설정

카테고리

카테고리

흑백

윤곽선

만화

스케치(흑백)

스케치(컬러)

선명도개선

흐리게

QR코드

얼굴분석

얼굴분석(랜드마크)

사물인식

이미지분류

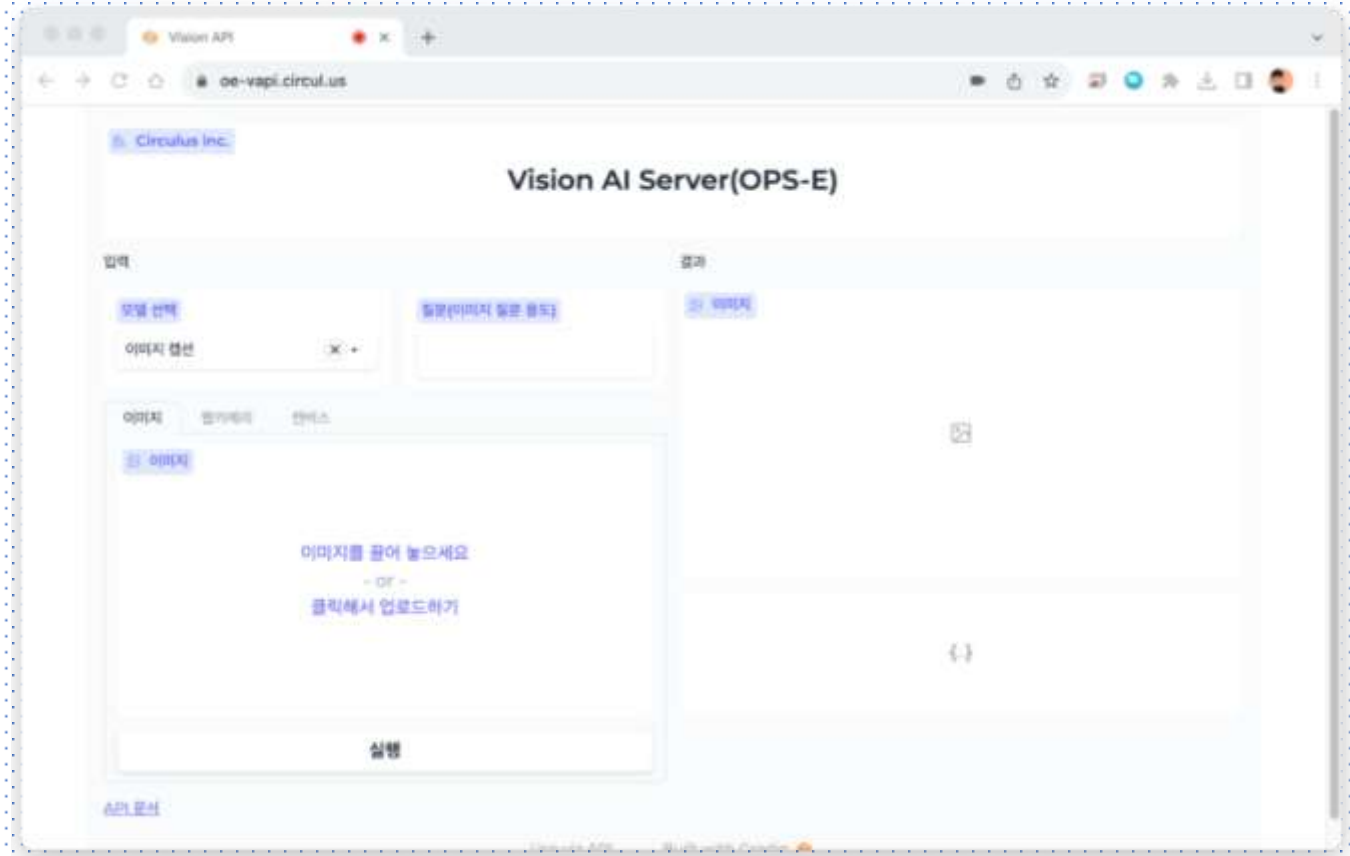
포즈인식

티쳐블머신

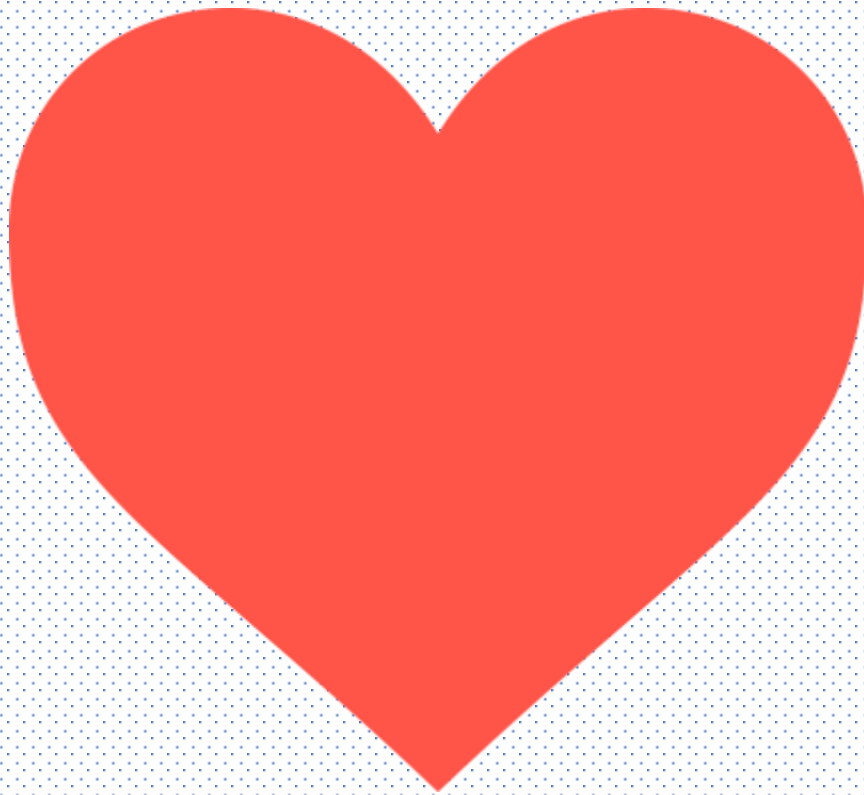
사물 트래킹

마커인식

- 좀 더 다양한 컴퓨터 비전 체험을 하려면?
- 크롬 또는 엣지 등 브라우저에 <https://oe-vapi.circul.us> 를 입력하여 접속
- https://docs.google.com/presentation/d/17hLPDdeH_0x7Z0SZq3yHamC8zj7xD1Fy/edit#slide=id.p3

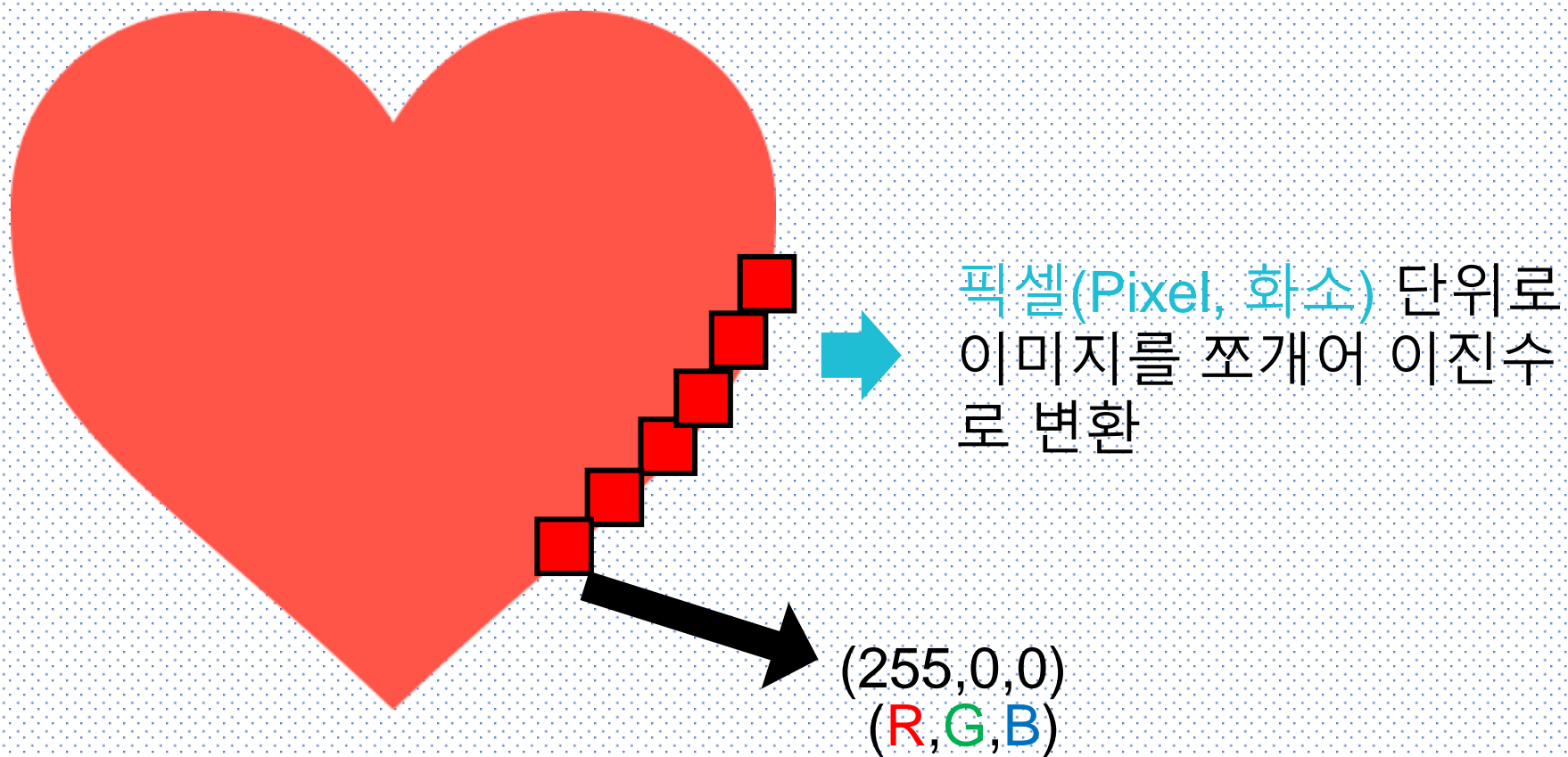


- 인공지능의 이미지 학습방법
 - 컴퓨터가 이해할 수 있는 0과 1로 이미지를 데이터화

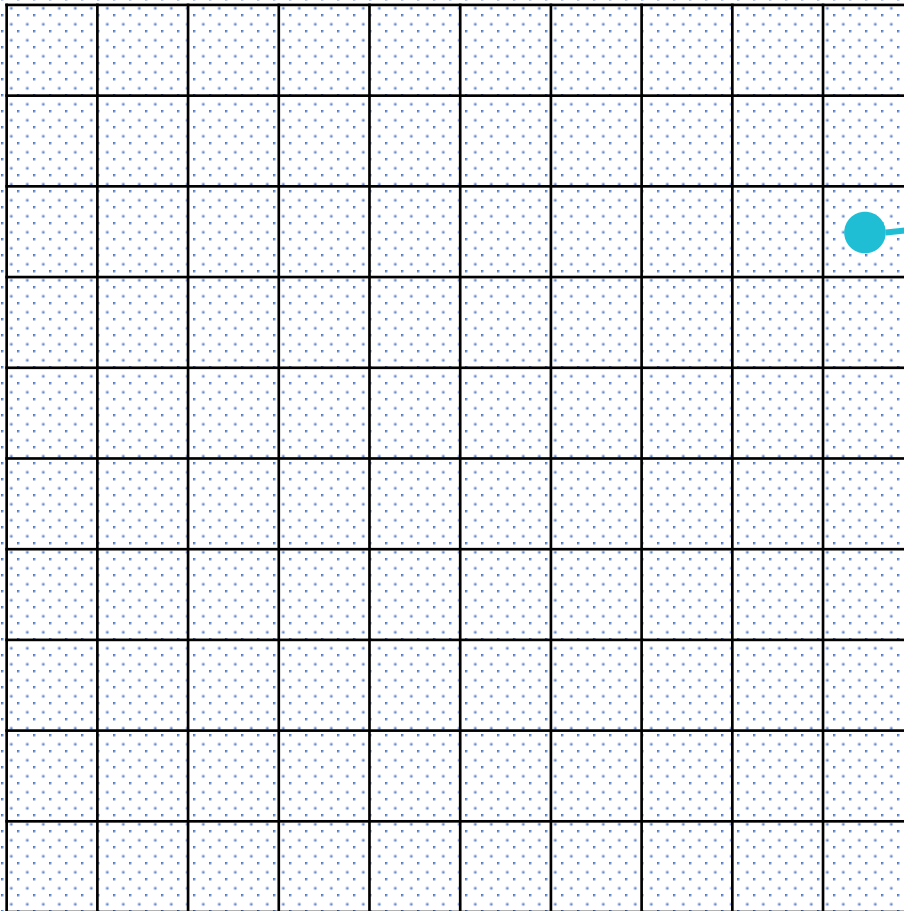


```
101111010101010101  
110000101010100001  
110010101010100000  
.....
```

- 인공지능의 이미지 학습방법
 - 컴퓨터가 이해할 수 있는 0과 1로 이미지를 데이터화



- 이미지(Image) : 픽셀의 2차원(가로x세로) 모음

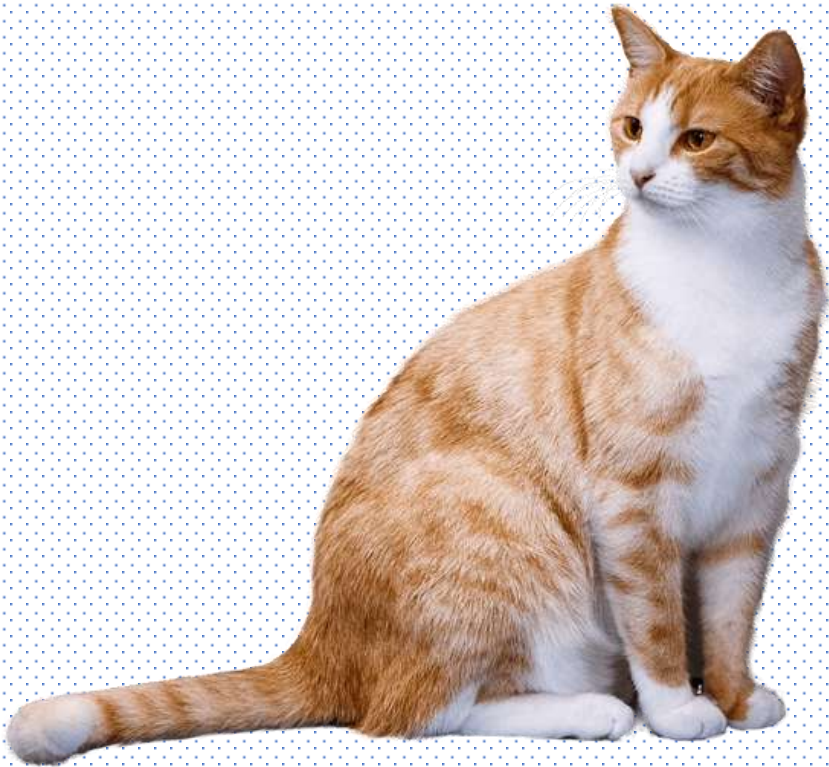


픽셀(pixel)
(BGR) \rightarrow (255,255,255)

- 인공지능의 이미지 학습방법
 - 컴퓨터가 이해할 수 있는 0과 1로 이미지를 데이터화



- 사람은 다음 사진을 보고 어떻게 고양이라고 판단할까요?

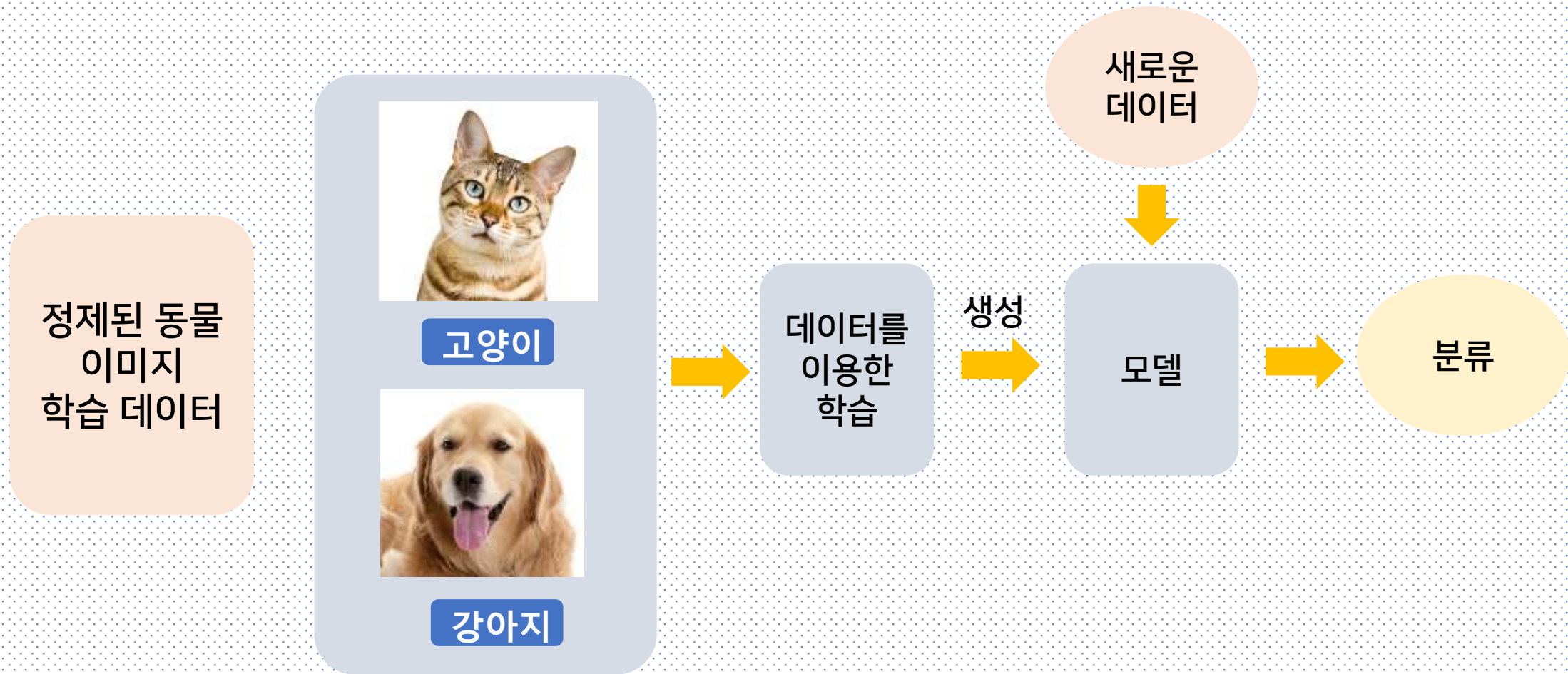


고양이 **특징(Features)** 뾰족한 귀, 무늬, 눈, 코, 입 위치, 꼬리 등 특징을 찾아 고양이의 모습으로 판단합니다.

우리는 먼저 학습과 경험을 통해 사진이 고양이라는 것을 알 수 있습니다.

컴퓨터도 고양이 사진을 보고 고양이라고 판단할 수 있을까요?

- 컴퓨터가 이미지를 인식/분류할 수 있도록 학습을 시켜봅니다.



- 티처블 머신을 사용하여 자율주행에 활용할 다양한 표지판을 학습하고 분류합니다.



- 티처블 머신을 사용하여 자율주행에 활용할 다양한 표지판을 학습하고 분류합니다.

Class 1



Class 2



TRAIN MODEL



MY PROJECT



MY PROJECT



1 모으기

예시를 수집하여 컴퓨터가 학습하기를 원하는 클래스 또는 카테고리로 그룹화하세요.

2 학습 시키기

모델을 학습시키세요. 그런 다음 모델이 새로운 예시를 올바르게 분류하는지 즉시 테스트해 보세요.

3 내보내기

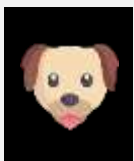
프로젝트(사이트, 앱 등)에 대한 모델을 내보냅니다. 모델을 다운로드하거나 온라인에서 호스팅할 수 있습니다.

- 티처블 머신을 사용하여 자율주행에 활용할 다양한 표지판을 학습하고 분류합니다.



많은 양의 데이터

- 동일한 이미지에 대한 다양한 형태의 이미지가 많을수록 GOOD!

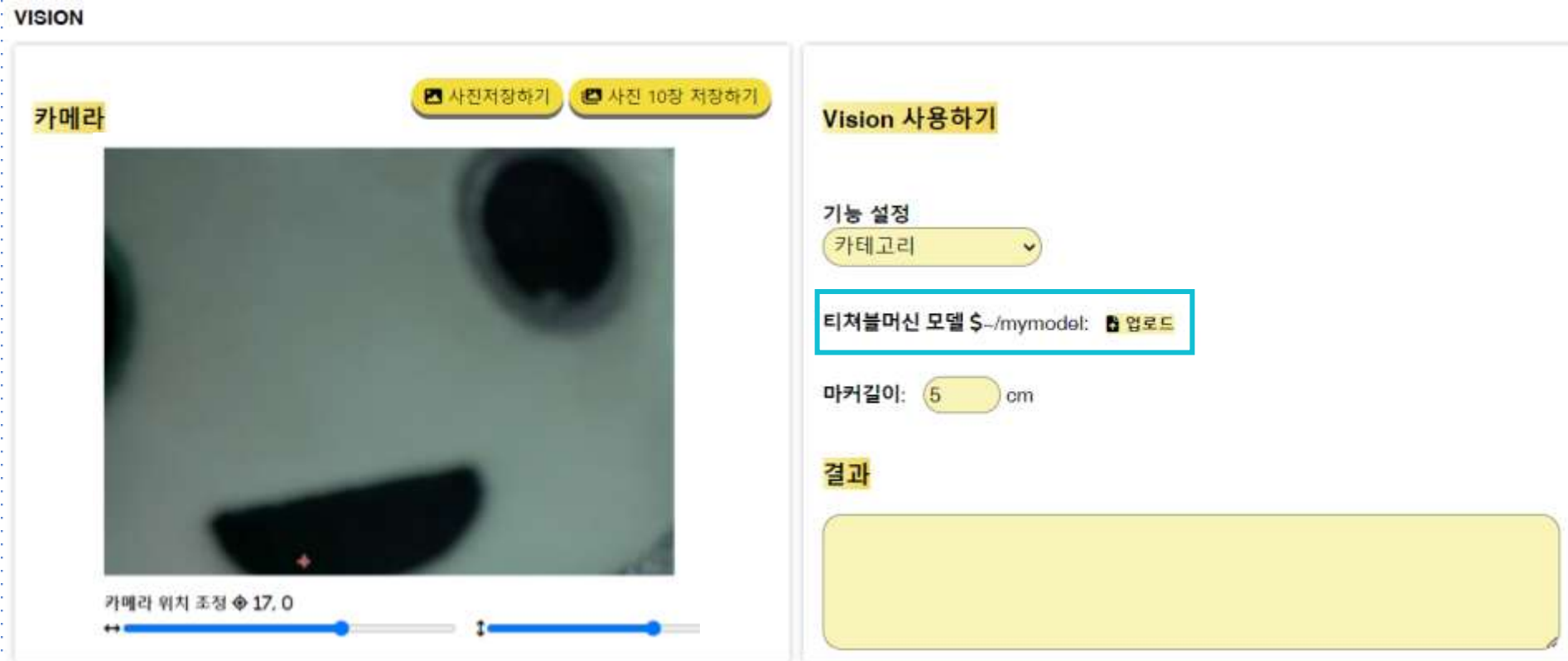


좋은 질의 데이터

- 정해진 조건에 맞는 이미지가 많을 수록 GOOD!

티처블머신으로 생성한 모델은 별도로 제공합니다.
수업시간에는 노트북을 이용해서 티처블머신에서 모델생성 및 분류를 해보고
파이보에는 저희가 제공하는 모델을 업로드하여 실습합니다.

- 파이보에 모델을 업로드하여 분류를 실습합니다.



- 이미지를 촬영하고, 가공해봅니다.

```
from openpibo.vision import Camera
```

```
camera = Camera()
```

```
image = camera.read() # 이미지 촬영하기
```

```
# 화면의 (100,100), (300,300) 위치에 (255,255,0) 색상, 두께 2의 사각형 그리기
```

```
image = camera.rectangle(image, (100,100), (300,300), (255,0,0), 2)
```

```
# 화면의 0, 0 위치에 글자크기 30인 "안녕하세요" 문자 쓰기
```

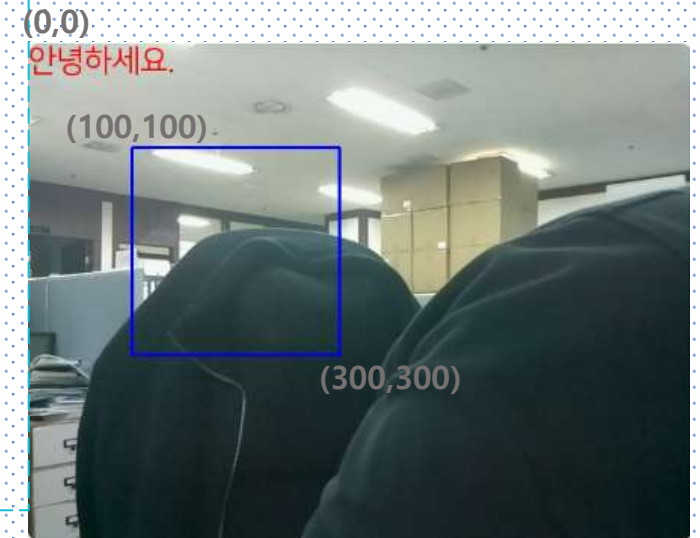
```
image = camera.putTextPIL(image, '안녕하세요.', (0, 0), 30, (0, 0, 255))
```

```
# image 변수를 test.jpg 저장
```

```
camera.imwrite("test.jpg", image)
```

```
# IDE 뷰어에 표시
```

```
camera.imshow_to_ide(image)
```



- 파이보 카메라와 티쳐블머신 모델을 통해 이미지를 분류합니다.

```
from openpibo.vision import TeachableMachine
from openpibo.vision import Camera

tm = TeachableMachine()
camera = Camera()

MODEL_DIR = '/home/pi/mymodel/'
tm.load(MODEL_DIR+'model_unquant.tflite', MODEL_DIR+'labels.txt')
img = camera.read()

result = tm.predict(img)
print(result[0])
```

티쳐블머신에서 학습한 모델로 이미지를 분류합니다.

★ Tools > Vision 탭에서 모델을 업로드 해주세요

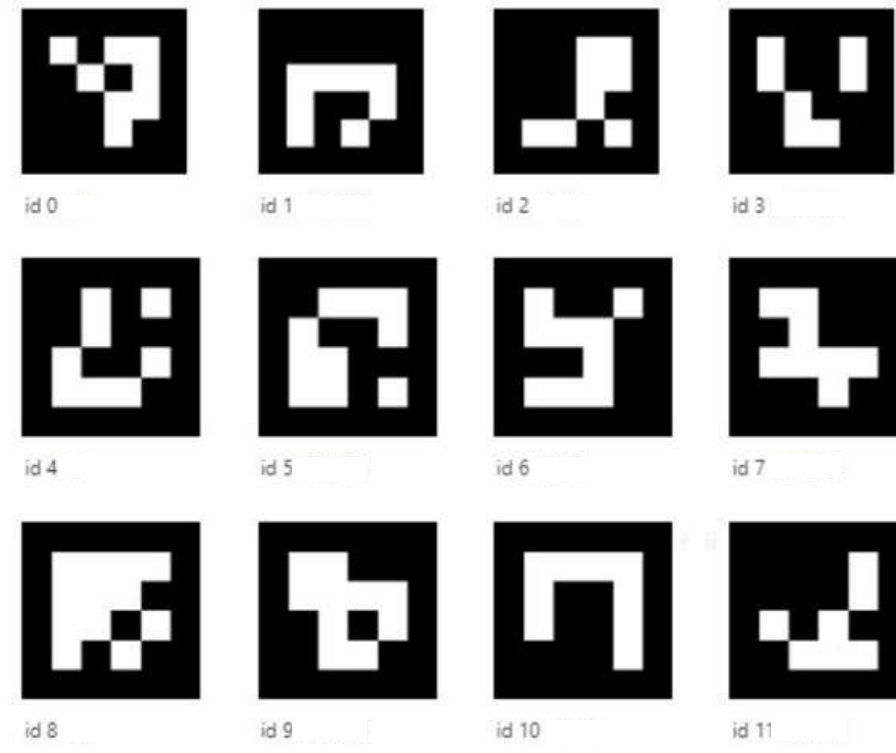


AI 자율행동 로봇

Part 4. 마커를 인식해
움직이는 파이보



- 마커란(Marker)란?
 - 미리 지정해둔 특정 표시를 인식하면 사전에 정의해둔 반응을 실행
 - 정사각형의 코드 이미지
 - 명암 대비가 명확하여 특징점을 추출하기 쉬움



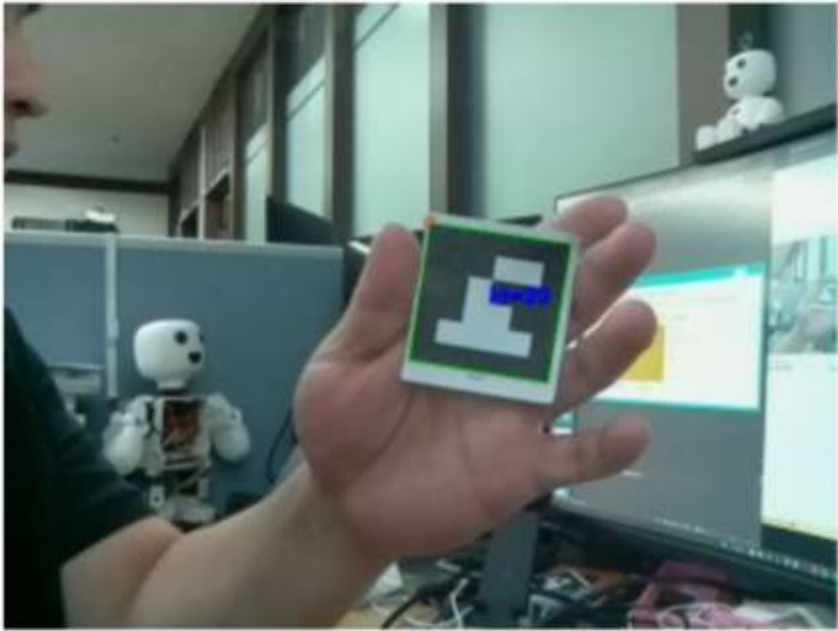
- 파이보 카메라를 통해 마커를 인식합니다.

TOOLS **ON** ↔ </>IDE 한국어 THE MAKER

VISION

카메라

사진 저장하기 사진 10장 저장하기



카메라 위치 조정 0, 0

Vision 사용하기

기능 설정
마커인식

티처블머신 모델 \$~/mymodel: 업로드

마커길이: 8.5 cm 마커의 실제 길이

결과
(29)-29.1cm

마커 번호 / 마커와 카메라의 거리

- 파이보 카메라를 통해 마커를 인식합니다.

```
from openpibo.vision import Camera
from openpibo.vision import Detect

MARKER_LENGTH = 8.5
camera = Camera()
detect = Detect()

image = camera.read()
items = detect.detect_marker(image, MARKER_LENGTH)

for item in items['data']:
    _id = item['id']
    cx, cy = item['center']
    distance = item['distance']
    print(_id, (cx,cy), distance)

camera.imshow_to_ide(items['img'])
```

마커 실제 길이 8.5cm

29 (431, 185) 29.4

종료됨.

마커 번호
마커 중심 좌표
마커와 카메라와의 실제거리(cm)

AI 자율행동 로봇

Part 4. AI 자율주행 로봇 파이보



- AI 자율주행 대회 규칙
 - 자율주행 맵 시작~결승전까지 한 바퀴를 돌아서 들어옵니다.
 - 팀별로 최대 10장의 마커를 사용할 수 있습니다.
 - 최단기간에 결승전을 통과한 팀이 1등을 수상합니다.
- 예제 소스 설명
 - 7_automove.py
 - 지정한 마커 1개를 확인하면서, 직선으로 보행
 - 8_automove_ext.py
 - automove.py의 기능에 더해서 마커와의 거리가 짧아지면, 회전하여, 맵 1바퀴 회전 보행
 - 마커 4개를 선정하여, 순서대로 인식
 - 9_automove_ext2.py
 - automove_ext.py 기능에 더해서, 표지판 이미지 인식 (티쳐블머신 모델 활용)
 - 마커를 찾지 못했을 때, 머리를 좌측/우측으로 이동하여, 1번 더 체크하는 기능 추가



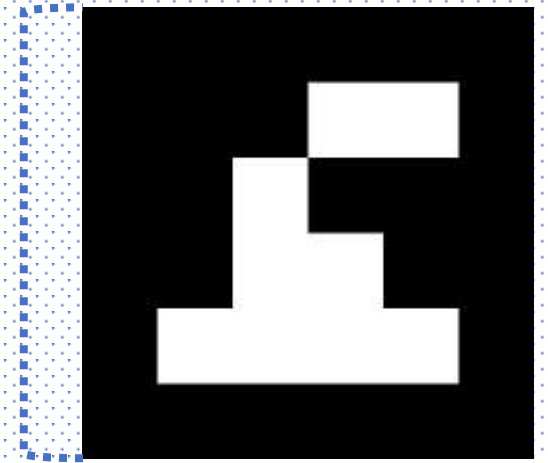
```
from openpibo.vision import Camera  
from openpibo.vision import Detect  
from openpibo.motion import Motion  
import time
```

```
MARKER_LENGTH = 8.5  
MARKER_ID = 29
```

```
camera = Camera()  
detect = Detect()  
motion = Motion()
```

vision / motion 패키지 import

vision / motion 초기화



MARKER_LENGTH: 마커의 실제 길이(cm) -카드 기준 8.5cm
MARKER_ID: 마커 번호

마커 인식을 활용한 직선 주행 예시

```
def get_dirX(x):  
    if x > 420:  
        dirX = "오른쪽"  
    elif x < 220:  
        dirX = "왼쪽"  
    else:  
        dirX = "가운데"  
    return dirX
```

```
def engine(data, _id):  
    distance, dX = None, None  
    for item in data:  
        if item['id'] == _id:  
            distance = item['distance']  
            dX = get_dirX(item['center'][0])  
            break  
    return dX, distance
```

- 마커의 위치 확인함수
입력: 마커 중심좌표의 x값
출력: 마커 위치

- 마커의 위치/거리 확인함수
입력: 마커 결과, 마커 번호
출력: 마커 위치 / 거리

* 인식된 마커 중 지정된

automove.py



★ 이미지에서 마커가 항상 CENTER에 위치하면, 마커를 따라 일직선으로 가는 것!!

```
while True:
    motion.set_motion('stop')
    time.sleep(2)

    image = camera.read()
    items = detect.detect_marker(image, MARKER_LENGTH)
    camera.imshow_to_ide(items['img'])
    dX, distance = engine(items['data'], MARKER_ID)

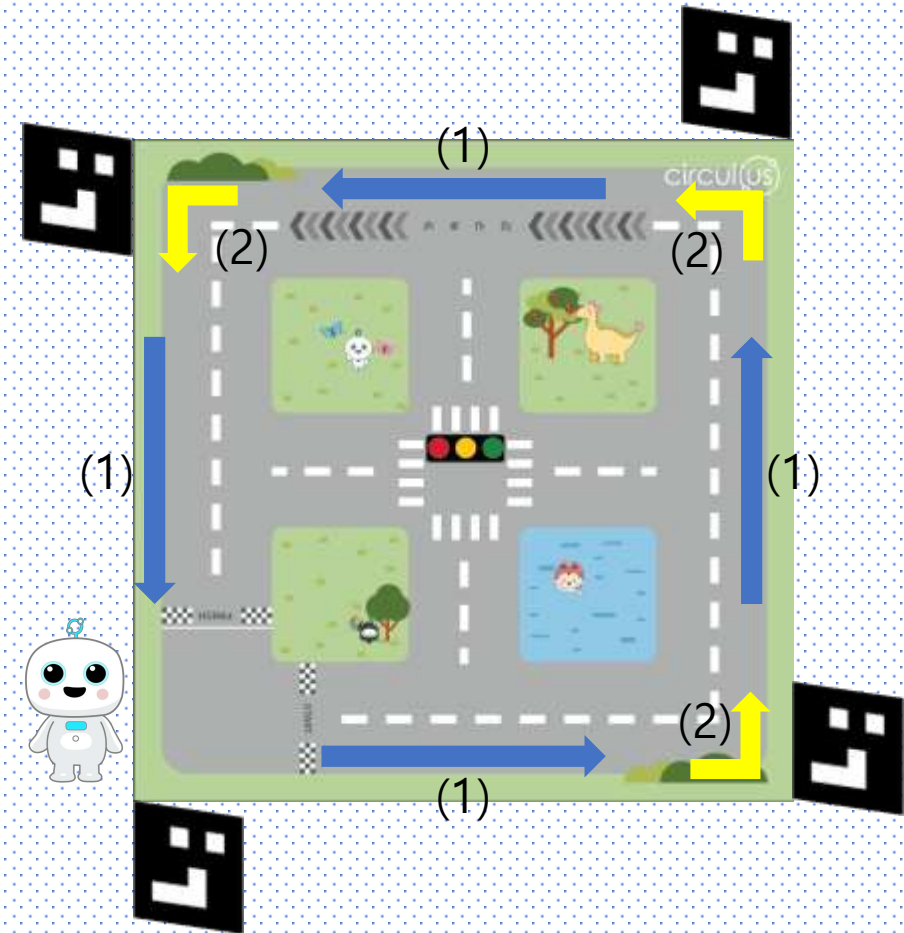
    if dX == None or distance == None:
        print(f'[인식불가]: 마커({MARKER_ID}) 인식 불가, 로봇 재배치 필요!')
    else:
        print(f'[인식성공]: {MARKER_ID} / {dX} / {distance}cm')

        if dX == '오른쪽':
            motion.set_motion('right_half')
        elif dX == '왼쪽':
            motion.set_motion('left_half')
        elif dX == '가운데':
            motion.set_motion('forward1')
```

sleep > 로봇이 완전히 정지상태로 이미지를 촬영하기 위해

- 이미지 촬영해서 마커를 인식
- 마커가 표시된 이미지를 IDE 뷰어에 표시
- 마커인식 결과 분석하여, 위치/거리 찾기
- 마커 인식 불가 상태
 - 마커 자체가 인식 안되었거나,
 - 인식된 마커가 지정한 마커가 아님
 - 경로 이탈되어 로봇을 다시 배치
- 지정한 마커가 인식된 상태
 - engine함수를 통해 얻은 위치/거리 출력
 - 위치에 따라 좌/우 회전 또는 직진
- 마커의 위치가 오른쪽일 경우, 로봇이 왼쪽에 있다는 의미이기 때문에, 오른쪽 회전
- 마커의 위치가 왼쪽일 경우, 로봇이 오른쪽에 있다는 의미이기 때문에, 왼쪽 회전
- right / right_half, left/left_half로 사전에 정의된 회전 동작 활용함. _half는 더 적은 범위의 회전

- 현재 가능한 것
 - 마커를 인식해서 일직선으로 직진하는 기능(1)
- 추가할 기능
 - 마커에 가까워졌을 때, 회전하는 기능(2)




```
from openpibo.vision import Camera
from openpibo.vision import Detect
from openpibo.motion import Motion
import time
```

```
MARKER_LENGTH = 8.5
```

```
# 인식할 마커 번호 목록
```

```
MARKER_LIST = [1, 2, 3, 4]
```

```
index = 0
```

```
# 상태> '직진' or '회전'
```

```
STATE = '직진'
```

```
camera = Camera()
```

```
detect = Detect()
```

```
motion = Motion()
```

```
def get_dirX(x):
```

```
    if x > 420:
```

```
        dirX = "오른쪽" # right
```

```
    elif x < 220:
```

```
        dirX = "왼쪽" # left
```

```
    else:
```

```
        dirX = "가운데" # center
```

```
    return dirX
```

```
def engine(data, _id):
```

```
    distance, dX = None, None
```

```
    for item in data:
```

```
        if item['id'] == _id:
```

```
            distance = item['distance']
```

```
            dX = get_dirX(item['center'][0])
```

```
            break
```

```
    return dX, distance
```

• 추가된 부분

1. MARKER_ID를 MARKER_LIST로 변경하여, 마커를 4개 선택하고 좌측과 같이 배치
2. STATE 추가하여, '직진', '회전' 상태 추가



```
while True:
    print(f'[진행상태]: {STATE}')

    # '회전' 상태: 다음 마커를 찾기 위해 회전하는 상태
    if STATE == '회전':
        motion.set_motion('left')

    motion.set_motion('stop')
    time.sleep(2)
    image = camera.read()
    items = detect.detect_marker(image, MARKER_LENGTH)
    camera.imshow_to_ide(items['img'])
    dX, distance = engine(items['data'], MARKER_LIST[index])
```

• 추가된 부분

- 1) STATE가 '회전'이면, 다음 순서의 마커를 찾는 상태
- 2) 마커를 인식하지 못했을 때, 상태 체크를 추가하여, '회전' 이면 재배치 하지 않음 -> 회전하는 중이니, 마커가 없는 것이 정상
- 3) MARKER_LIST를 모두 실행했으면, "완주"
- 4) 마커를 인식했을 때, 마커와의 거리가 30cm 이내이면, index를 1 증가시켜 다음 마커를 인식하도록 함.
- 5) 30cm 이상이면, '직진' 상태 유지

```
if dX == None or distance == None:
    if STATE != '회전':
        # '회전' 상태가 아닌데, 마커를 찾지 못했으면, 경로 이탈
        print(f'[인식불가]: 마커({MARKER_LIST[index]}) 인식 불가, 로봇 재배치 필요!')
    else:
        print(f'[마커인식]: {MARKER_LIST[index]} / {dX} / {distance}cm')

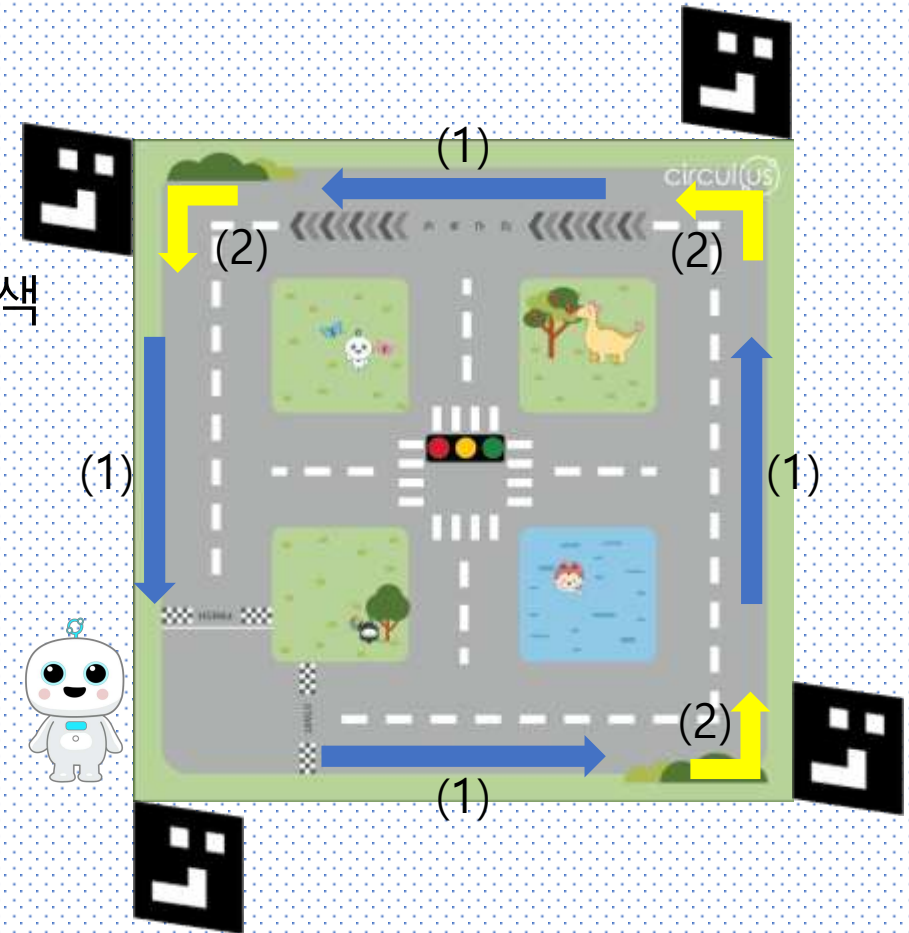
    if distance < 30:
        if len(MARKER_LIST) == index+1:
            print("[완료] 완주했습니다.")
            break
        # 마커와의 거리가 30cm 이내이면, 다음 마커로 변경하고, 상태를 '회전'으로 변경
        # 다음 마커를 찾을 때까지 회전
        print(f'[마커번호 변경]: ({MARKER_LIST[index]} > {MARKER_LIST[index+1]})')
        STATE = '회전'
        index += 1
    else:
        # 마커 위치에 따라 우측/좌측/전진하여 마커가 중앙에 위치하도록 조정 (직진)
        # 마커와의 거리가 30cm 이상이기 때문에, 직진해야 함
        STATE = '직진'
        # right, left 동작의 _half는 좀 더 적은 회전
        if dX == '오른쪽':
            motion.set_motion('right_half')
        elif dX == '왼쪽':
            motion.set_motion('left_half')
        elif dX == '가운데':
            motion.set_motion('forward1')
```

- 현재 가능한 것

- 마커를 인식해서 일직선으로 직진하는 기능(1)
- 마커에 가까워졌을 때, 회전(2)

- 추가할 기능

- 표지판 이미지 인식
- 경로를 이탈했을 때, 머리를 좌우측 이동하여 마커 검색



```
from openpibo.vision import Camera
from openpibo.vision import Detect
from openpibo.vision import TeachableMachine
from openpibo.motion import Motion
import time
```

```
MARKER_LENGTH = 8.5
# 인식할 마커 번호 목록
MARKER_LIST = [1,2,3,4]
index = 0
```

```
STATE = '직진'
```

```
camera = Camera()
detect = Detect()
motion = Motion()
tm = TeachableMachine()
```

```
# 티쳐블머신 모델 불러오기
tm.load('/home/pi/mymodel/model_unquant.tflite', '/home/pi/mymodel/labels.txt')
```

```
def get_dirX(x):
    if x > 420:
        dirX = "오른쪽" # right
    elif x < 220:
        dirX = "왼쪽" # left
    else:
        dirX = "가운데" # center
    return dirX

def engine(data, _id):
    distance, dX = None, None

    for item in data:
        if item['id'] == _id:
            distance = item['distance']
            dX = get_dirX(item['center'][0])
            break
    return dX, distance
```

- 추가된 부분

- 1) 표지판 이미지 분류를 위해 모델 초기화 및 불러오기

```
while True:
    print(f'[진행상태]: {STATE}')

    if STATE == '회전':
        motion.set_motion('left')

        motion.set_motion('stop')
        time.sleep(2)
        image = camera.read()

        # 티쳐블머신 활용 카드 분류
        cardname, scores = tm.predict(image)
        score = int(max(scores)*100)
        if score > 90:
            # 카드 인식이 높은 확률(90%)로 되었을 때,
            print(f'[카드인식]: {cardname} {score}%')

        items = detect.detect_marker(image, MARKER_LENGTH)
        camera.imshow_to_ide(items['img'], 1)

        dX, distance = engine(items['data'], MARKER_LIST[index])
```

• 추가된 부분

- 1) 티쳐블머신 모델로 이미지 분류 (90% 이상 확률로 분류될 때, 표시)
 - 표지판 이미지를 인식하여, 동작을 추가할 경우,
 - if score > 90: 안에 cardname에 따른 동작을 추가할 수 있음

예시)

```
if score > 90:
    print(f'[카드인식]: {cardname} {score}%')
    if cardname == "정지":
        time.sleep(3)
    elif ...
```

```

if dX == None or distance == None:
    if STATE != '회전':
        # 마커가 없을 때, 좌우측 머리를 돌려서 마커가 있는지 2차 확인 기능 추가
        motion.set_motor(4, -20) # 목 오른쪽 이동
        time.sleep(2)
        image = camera.read()
        items = detect.detect_marker(image, MARKER_LENGTH)
        camera.imshow_to_ide(items['img'], 1)
        dX, distance = engine(items['data'], MARKER_LIST[index])

    if dX != None and distance != None:
        motion.set_motion('right') # 머리를 우측으로 돌린 상황에서 마커가 인식되었으므로 오른쪽 회전
    else:
        motion.set_motor(4, 20) # 목 왼쪽 이동
        time.sleep(2)
        image = camera.read()
        items = detect.detect_marker(image, MARKER_LENGTH)
        camera.imshow_to_ide(items['img'], 1)
        dX, distance = engine(items['data'], MARKER_LIST[index])

    if dX != None and distance != None:
        motion.set_motion('left') # 머리를 좌측으로 돌린 상황에서 마커가 인식되었으므로 왼쪽 회전
    else:
        # 좌우측 머리를 이동해도 마커를 못 찾을 때, 경로 이탈
        print(f'[마커인식불가]: {MARKER_LIST[index]} - 로봇 재배치 필요')
    
```

• 추가된 부분

- 마커가 인식되지 않았고, '직진'상태 (STATE != '회전')
 - ✓ 8_automove_ext.py 는 여기서 경로 이탈
 - ✓ 머리를 오른쪽으로 이동하여 마커 찾기
 - ① 마커 찾음 - 오른쪽 회전
 - ② 마커 못찾음
 - 머리를 왼쪽으로 이동하여 마커 찾기
 - ① 마커 찾음 - 왼쪽 회전
 - ② 마커 못찾음
 - 경로 이탈
- 2차 마커 확인 기능
 - ✓ motion.set_motor(4, -20)
 - ✓ motion.set_motor(4, 20)
 - ✓ '20'을 수정하면, 얼굴이 좌우측으로 더 많이 이동
 - ✓ 더 많이 이동할 경우, sleep도 적절하게 수정 필요

```
else: /* if dX == None or distance == None: */
    print(f'[마커인식]: {MARKER_LIST[index]} / {dX} / {distance}cm')

    if distance < 30:
        if len(MARKER_LIST) == index+1:
            print("[완료] 완주했습니다.")
            break

    print(f'[마커번호 변경]: ({MARKER_LIST[index]} > {MARKER_LIST[index+1]})')
    STATE = '회전'
    index += 1
else:
    STATE = '직진'

if dX == '오른쪽':
    motion.set_motion('right_half')
elif dX == '왼쪽':
    motion.set_motion('left_half')
elif dX == '가운데':
    motion.set_motion('forward1')
```

8_automove_ext.py와 동일

감사합니다

