

깃허브 코파일럿을 이용한 바이브 코딩

김명균

(울산대학교 IT융합학부)

바이브코딩을 이용한 사물인터넷

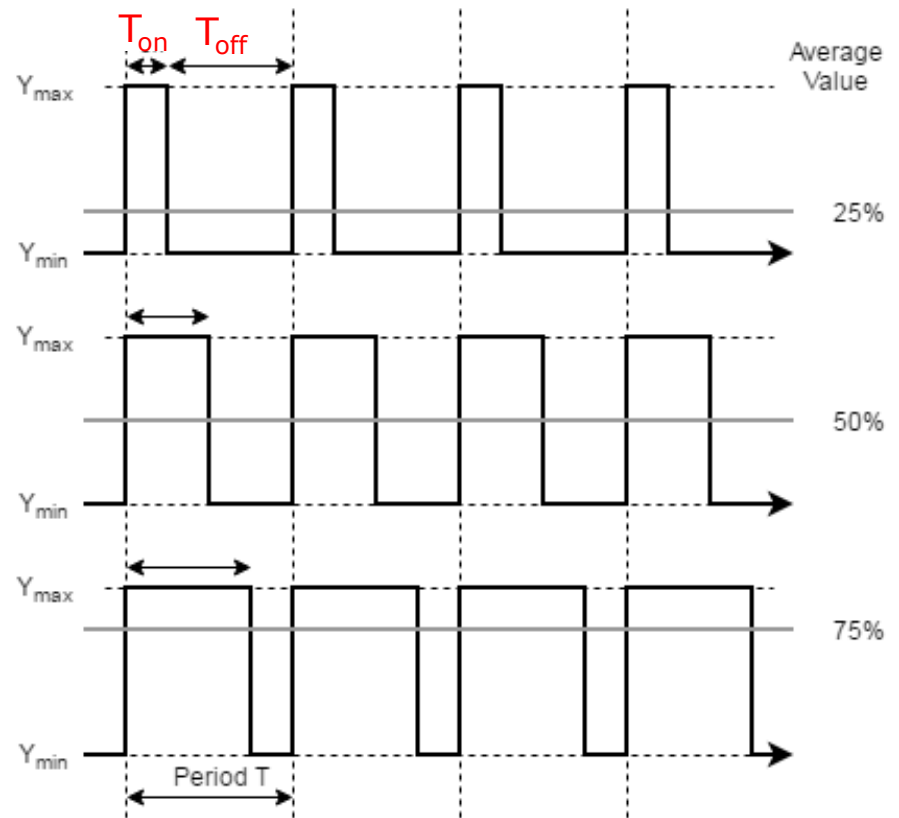
- 라즈베리파이 설치 및 환경 설정
- 깃허브 코파일럿 설치
- 바이브 코딩을 이용한 사물인터넷 – part2

사물인터넷 프로그래밍: PWM

■ PWM

- used to control the speed of motor or brightness of led by controlling the width of a pulse

- **period:** T
- **frequency:** $1 / T$ (default: 19.2 MHz)
- **duty_cycle:** T_{on} / T



사물인터넷 프로그래밍: PWM

■ PWM in Raspberry Pi

3.3V	1	2	5V	
GPIO2 (SDA1)	3	4	5V	
GPIO3 (SCL1)	5	6	GND	
GPIO4 (GPIO_GCLK)	7	8	GPIO14 (UART_TXD0)	
GND	9	10	GPIO15 (UART_RXD0)	
GPIO17 (GPIO_GEN0)	11	12	GPIO18 (GPIO_GEN1) PWM0	
GPIO27 (GPIO_GEN2)	13	14	GND	
GPIO22 (GPIO_GEN3)	15	16	GPIO23 (GPIO_GEN4)	
3.3V	17	18	GPIO24 (GPIO_GEN\$)	
GPIO10 (SPI0_MOSI)	19	20	GND	
GPIO9 (SPI0_MISO)	21	22	GPIO25 (GPIO_GEN6)	
GPIO11 (SPI0_CLK)	23	24	GPIO8 (SPI_CE0_N)	
GND	25	26	GPIO7 (SPI_CE1_N)	
ID_SD (I2C EEPROM)	27	28	ID_SC (I2C EEPROM)	
GPIO5	29	30	GND	
GPIO6	31	32	GPIO12 PWM0	
PWM1	GPIO13	33	34	GND
PWM1	GPIO19	35	36	GPIO16
GPIO26	37	38	GPIO20	
GND	39	40	GPIO21	

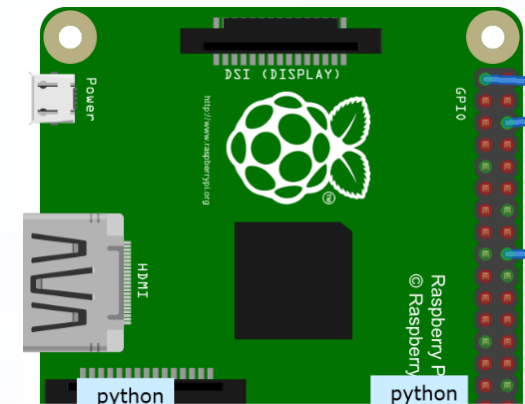
사물인터넷 프로그래밍: PWM

■ PWM programming using RPi.GPIO module

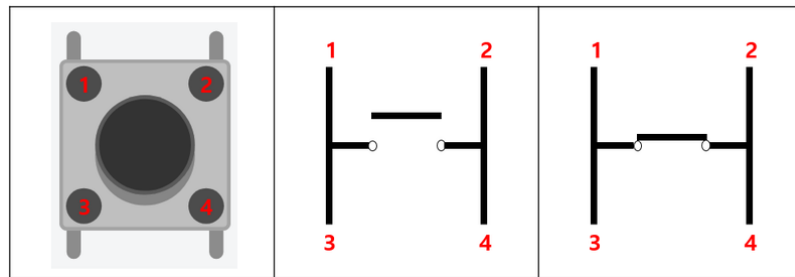
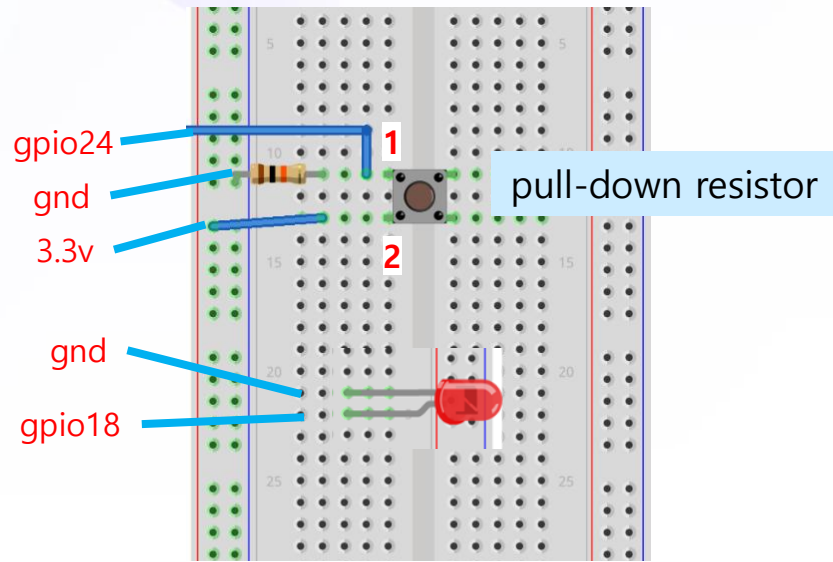
- `pwm = GPIO.PWM(pin, f)` // pwm 객체 생성; frequency f (Hz)
- `pwm.start(duty_cycle)` // pwm start with duty_cycle
- `pwm.ChangeFrequency(frequency)` // change PWM signal frequency
- `pwm.ChangeDutyCycle(duty cycle)` // change PWM signal duty_cycle (0 ~ 100)
- `pwm.stop()` // stop PWM signal

사물인터넷 프로그래밍: PWM

■ PWM 이용하기: 버튼, led 연결



WPI	BCM/Name	물리적 구성(핀)	BCM/Name	WPI
	3.3V	1	2	5V
8	GPIO 2/I2C SDA	3	4	5V
9	GPIO 3/I2C SCL	5	6	Ground
7	GPIO 4	7	8	GPIO 14/TXD
	Ground	9	10	GPIO 15/RXD
0	GPIO 17	11	12	GPIO 18
2	GPIO 27	13	14	Ground
3	GPIO 22	15	16	GPIO 23
	3.3 V	17	18	GPIO 24
12	GPIO 10/SPI MOSI	19	20	Ground
13	GPIO 9/SPI MISO	21	22	GPIO 25
14	GPIO 11/SPI CLK	23	24	GPIO 8/SPI_CE0
	Ground	25	26	GPIO 7/SPI_CE1
30	ID_SD	27	28	ID_SC
21	GPIO 5	29	30	Ground
22	GPIO 6	31	32	GPIO 12
23	GPIO 13	33	34	Ground
24	GPIO 19	35	36	GPIO 16
25	GPIO 26	37	38	GPIO 20
	Ground	39	40	GPIO 21
				29

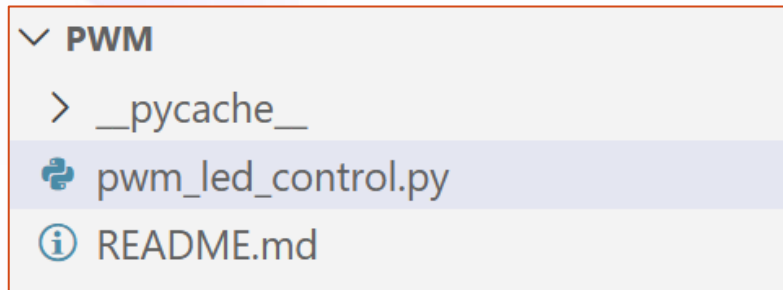


button connection

사물인터넷 프로그래밍: PWM

■ PWM 프로그래밍:

- 작업폴더: pwm



- 프롬프트:

현재 rp84041 이라는 이름의 라즈베리파이에 아래와 같은 센서들이 연결되어 있어.

[led] gpio18 핀에 연결됨, 초기상태는 off 상태

[버튼] 풀다운 저항으로 gpio24 핀에 연결됨

[동작모드] 자동모드, 정지모드 (초기상태는 정지모드), 버튼을 누르면 모드가 변함, 버튼입력은 인터럽트 핸들러를 사용함

[led 조절 모드] 증가모드, 감소모드 (초기 상태는 증가모드), 증가모드에서 led 밝기가 최대 밝기에 도달하면 감소모드로, 감소모드에서 led 밝기가 최소밝기가 되면 증가모드로 전환

[동작] 정지모드에서는 led의 밝기를 그대로 유지하고, 자동모드에서는 led 밝기를 10% 씩 증가 또는 감소 시킴

200ms 간격으로

RPi.GPIO 라이브러리를 이용한 pwm 제어를 통해 이와 같이 동작하는 프로그램을 만들어줘.

사물인터넷 프로그래밍: PWM

■ PWM 프로그래밍: 프로그램 다운로드 → 실행

- 프롬프트: 이 프로그램을 rp84041 호스트 pi 계정의 홈폴더 아래에 있는 IoT 폴더에 저장해줘.

- 실행:

```
pi@rp84041:~/IoT $ python pwm_led_control.py
PWM LED controller started
initial mode: stop, brightness: 0%, direction: up
mode: auto, brightness: 0%
brightness: 10%, direction: up
brightness: 20%, direction: up
brightness: 30%, direction: up
brightness: 40%, direction: up
brightness: 50%, direction: up
mode: stop, brightness: 50%
mode: auto, brightness: 50%
brightness: 60%, direction: up
brightness: 70%, direction: up
brightness: 80%, direction: up
brightness: 90%, direction: up
brightness: 100%, direction: down
brightness: 90%, direction: down
brightness: 80%, direction: down
```

사물인터넷 프로그래밍: PWM 코드

```
import signal
import time
import RPi.GPIO as GPIO

LED_PIN = 18
BUTTON_PIN = 24

PWM_FREQUENCY_HZ = 1000
STEP_PERCENT = 10
STEP_INTERVAL_SECONDS = 0.1
DEBOUNCE_MS = 200

MODE_STOP = "stop"
MODE_AUTO = "auto"

DIRECTION_UP = "up"
DIRECTION_DOWN = "down"

mode = MODE_STOP
direction = DIRECTION_UP
brightness = 0
running = True
pwm = None
```

```
def clamp_brightness(value):
    return max(0, min(100, value))

def toggle_mode(channel):
    global mode

    mode = MODE_AUTO if mode == MODE_STOP else MODE_STOP
    print(f"mode: {mode}, brightness: {brightness}%")

def update_brightness():
    global brightness, direction

    if direction == DIRECTION_UP:
        brightness = clamp_brightness(brightness + STEP_PERCENT)
        if brightness >= 100:
            direction = DIRECTION_DOWN
    else:
        brightness = clamp_brightness(brightness - STEP_PERCENT)
        if brightness <= 0:
            direction = DIRECTION_UP

    pwm.ChangeDutyCycle(brightness)
    print(f"brightness: {brightness}%, direction: {direction}")
```

사물인터넷 프로그래밍: PWM 코드

```
def stop_program(signum, frame):
    global running
    running = False

def setup():
    global pwm

    GPIO.setmode(GPIO.BCM)
    GPIO.setup(LED_PIN, GPIO.OUT)
    GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

    pwm = GPIO.PWM(LED_PIN, PWM_FREQUENCY_HZ)
    pwm.start(brightness)

    GPIO.add_event_detect(BUTTON_PIN, GPIO.RISING,
        callback=toggle_mode, bouncetime=DEBOUNCE_MS)

def cleanup():
    if pwm is not None:
        pwm.ChangeDutyCycle(0)
        pwm.stop()
    GPIO.cleanup()
```

사물인터넷 프로그래밍: PWM 코드

```
def main():
    signal.signal(signal.SIGINT, stop_program)
    signal.signal(signal.SIGTERM, stop_program)

    setup()
    print("PWM LED controller started")
    print(f"initial mode: {mode}, brightness: {brightness}%, direction: {direction}")

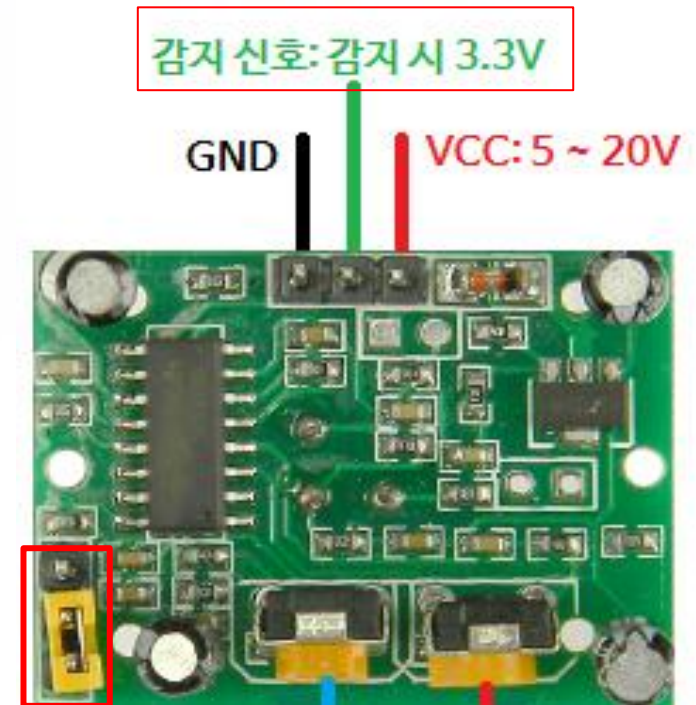
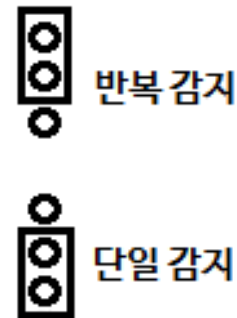
    try:
        while running:
            if mode == MODE_AUTO:
                update_brightness()
                time.sleep(STEP_INTERVAL_SECONDS)
    finally:
        cleanup()
        print("PWM LED controller stopped")

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

사물인터넷 프로그래밍: PIR 센서

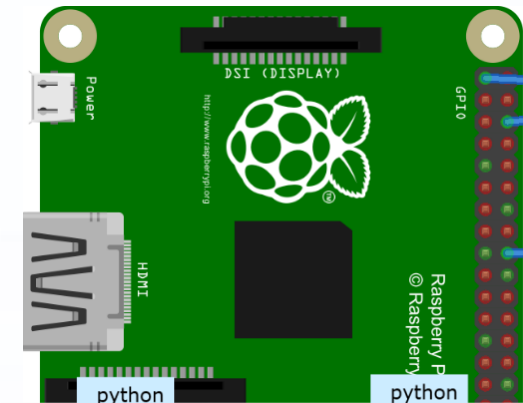
■ PIR 센서: HC-SR501

- 움직임 감지 센서

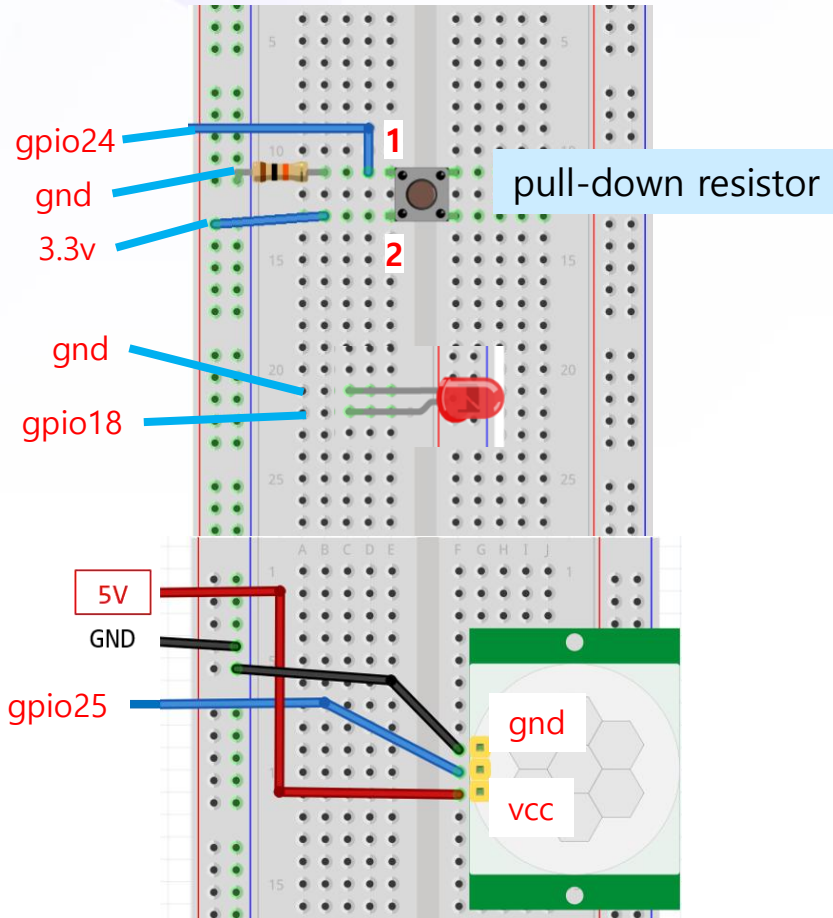


사물인터넷 프로그래밍: PIR 센서

PIR 센서 프로그래밍 : 버튼, led, PIR 센서 연결



WPI	BCM/Name	물리적 구성(핀)	BCM/Name	WPI
	3.3V	1	5V	2
8	GPIO 2/I2C SDA	3	5V	4
9	GPIO 3/I2C SCL	5	Ground	
7	GPIO 4	7	GPIO 14/TXD	15
	Ground	9	GPIO 15/RXD	16
0	GPIO 17	11	GPIO 18	1
2	GPIO 27	13	Ground	
3	GPIO 22	15	GPIO 23	4
	3.3 V	17	GPIO 24	5
12	GPIO 10/SPI MOSI	19	Ground	
13	GPIO 9/SPI MISO	21	GPIO 25	6
14	GPIO 11/SPI CLK	23	GPIO 8/SPI_CE0	10
	Ground	25	GPIO 7/SPI_CE1	11
30	ID_SD	27	ID_SC	31
21	GPIO 5	29	Ground	
22	GPIO 6	31	GPIO 12	26
23	GPIO 13	33	Ground	
24	GPIO 19	35	GPIO 16	27
25	GPIO 26	37	GPIO 20	28
	Ground	39	GPIO 21	29



사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프롬프트 v1

■ 프롬프트:

현재 이 PC에 호스트 이름이 rp84041인 라즈베리파이가 연결되어 있고, 이 라즈베리파이에 아래와 같은 센서들이 연결되어 있어.

[led] gpio18 번에 연결되어 있음,

[버튼] 풀다운 저항으로 gpio24번에 연결되어 있음,

[HC-SR501 센서] gpio25번에 연결되어 있음,

이와 같은 센서가 연결된 라즈베리파이에서 아래와 같이 동작하는 프로그램을 만들어줘.

[모드] 감지모드, 중지모드 (초기상태는 중지모드), 버튼을 누르면 모드 상태가 바뀜,

[동작] 감지상태에서는 3초 간격으로 PIR 센서로부터 움직임을 감지하고, 움직임을 감지되면 led를 켜고 움직임을 없으면 led를 off 상태로 하고, 중지상태에서는 감지동작을 중지하고, led도 off 상태로 함.

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 실행 v1

```
pi@rp84041:~/IoT $ python pir_motion_controller.py
Initial mode: STOPPED
Press the button to switch between DETECTING and STOPPED.
Mode changed: DETECTING
No motion: LED OFF
Motion detected: LED ON
No motion: LED OFF
No motion: LED OFF
No motion: LED OFF
Motion detected: LED ON
No motion: LED OFF
No motion: LED OFF
No motion: LED OFF
No motion: LED OFF
No motion: LED OFF
```

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프롬프트 v2

쓰레드를 사용하지 말고 하나의 쓰레드만 사용하도록 프로그램을 수정하고, 버튼 동작은 인터럽트를 이용하여 입력받도록 수정해줘.

라즈베리파이 호스트 rp84041 의 pi 계정 홈폴더에 IoT 폴더가 있어. 만든 프로그램을 IoT폴더에 복사해줘

프로그램을 중지 모드일 때는 현재 중지 모드라는 출력만 하고, 감지동작을 하지 않고 어떠한 출력도 하지 않도록 수정해줘. 감지동작과 출력은 감지상태일 때만 해줘.

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프롬프트 v2

이 프로그램을 pi 계정의 ~/IoT 폴더에 새로운 이름으로 전송해줘.

■ 실행:

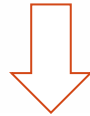
버튼을 한번 눌렀는데 2번 눌러진 것처럼 동작하네. 문제점을 찾아줘.

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프롬프트 v2

수정한 프로그램을 pi 계정의 ~/IoT 폴더에 새로운 이름으로 전송해줘.

프로그램 실행시 감지모드에서 센서가 움직임 감지를 잘 하지 못하는 경우가 많은데 센서 문제인가?



제 판단으로는 센서 고장보다는 3초마다 한 번만 읽는 현재 방식 + HC-SR501 설정 문제일 가능성이 큼니다.

감지를 더 잘 하려면 감지모드에서는 PIR을 더 짧은 간격, 예를 들어 0.1초 나 0.2초 마다 확인하고, 출력만 필요할 때 하도록 바꾸는 게 좋습니다. 버튼은 지금처럼 인터럽트 대기 방식을 유지하면서도 감지 신뢰도를 높일 수 있습니다.

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 실행 v2

```
pi@rp84041:~/IoT $ python pir_motion_controller.py
Initial mode: STOPPED
Press the button to switch between DETECTING and STOPPED.
Mode changed: DETECTING
No motion: LED OFF
Mode changed: STOPPED
Mode changed: DETECTING
No motion: LED OFF
No motion: LED OFF
No motion: LED OFF
No motion: LED OFF
Motion detected: LED ON
No motion: LED OFF
^C
Program stopped.
```

모션감지가 잘 동작하지 않음

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프롬프트 v2

그럼 감지모드에서는 감지를 0.5초 간격으로 하고 led 상태와 화면 출력은 3초 간격으로 하도록 프로그램을 수정하고, 수정한 프로그램을 새로운 파일에 저장해줘.

■ 실행:

```
pi@rp84041:~/IoT $ python pir_motion_fast_sample_controller.py
현재 중지 모드
현재 감지 모드
피직임 감지 : LED ON
피직임 감지 : LED ON
피직임 감지 : LED ON
현재 중지 모드
현재 감지 모드
피직임 없음 : LED OFF
피직임 감지 : LED ON
피직임 없음 : LED OFF
피직임 감지 : LED ON
피직임 감지 : LED ON
피직임 감지 : LED ON
피직임 감지 : LED ON
```

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프로그램 (part1)

```
from enum import Enum
import time
from typing import Optional
import RPi.GPIO as GPIO

LED_PIN = 18
BUTTON_PIN = 24
PIR_PIN = 25
DETECTION_INTERVAL_SECONDS = 3
BUTTON_BOUNCE_MS = 200
BUTTON_RELEASE_SETTLE_SECONDS = 0.05

class Mode(Enum):
    STOPPED = "중지 모드"
    DETECTING = "감지 모드"

def setup_gpio() -> None:
    GPIO.setmode(GPIO.BCM)
    GPIO.setwarnings(False)
    GPIO.setup(LED_PIN, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
    GPIO.setup(PIR_PIN, GPIO.IN)

def set_led(is_on: bool) -> None:
    GPIO.output(LED_PIN, GPIO.HIGH if is_on else GPIO.LOW)
```

사물인터

■ PIR 센서 프로그래밍 : 프로그램 (part2)

```
def wait_for_button(timeout_seconds: Optional[float] = None) -> bool:
    if timeout_seconds is None:
        channel = GPIO.wait_for_edge(
            BUTTON_PIN,
            GPIO.RISING,
            bouncetime=BUTTON_BOUNCE_MS,
        )
    else:
        timeout_ms = max(1, int(timeout_seconds * 1000))
        channel = GPIO.wait_for_edge(
            BUTTON_PIN,
            GPIO.RISING,
            timeout=timeout_ms,
            bouncetime=BUTTON_BOUNCE_MS,
        )

    if channel != BUTTON_PIN:
        return False

    wait_for_button_release()
    return True

def wait_for_button_release() -> None:
    while GPIO.input(BUTTON_PIN) == GPIO.HIGH:
        GPIO.wait_for_edge(
            BUTTON_PIN,
            GPIO.FALLING,
            timeout=100,
            bouncetime=BUTTON_BOUNCE_MS,
        )

    time.sleep(BUTTON_RELEASE_SETTLE_SECONDS)
```

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프로그램 (part3)

```
def toggle_mode(current_mode: Mode) -> Mode:
    if current_mode == Mode.STOPPED:
        next_mode = Mode.DETECTING
    else:
        next_mode = Mode.STOPPED
        set_led(False)

    print(f"현재 {next_mode.value}", flush=True)
    return next_mode

def read_motion_and_update_led() -> None:
    motion_detected = GPIO.input(PIR_PIN) == GPIO.HIGH
    set_led(motion_detected)

    if motion_detected:
        print("움직임 감지: LED ON", flush=True)
    else:
        print("움직임 없음: LED OFF", flush=True)
```

사물인터넷 프로그래밍: PIR 센서

■ PIR 센서 프로그래밍 : 프로그램 (part5)

```
def main() -> None:
    try:
        run()
    except KeyboardInterrupt:
        print("\n프로그램을 종료합니다.", flush=True)
    finally:
        try:
            set_led(False)
        except RuntimeError:
            pass
        GPIO.cleanup()

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