임베디드응용및실습 과제

7주차. 과제 보고서

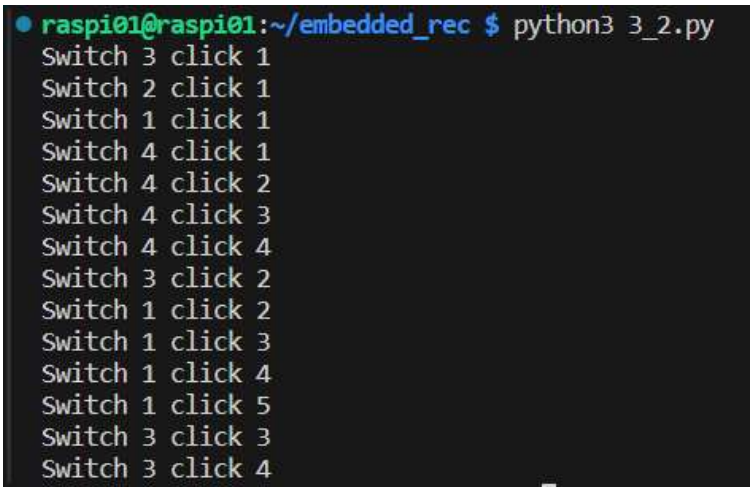
과 목 명	임베디드응용및실습
학 번	2020161082
이 름	이강산
제 출 일	2025년 10월 19일

1. 버튼 입력받기 구현

1) 코드를 추가하여 스위치를 눌렸을 때만 화면에 "click"이 표기되도록 변경
2) 몇 번 스위치가 눌렸는지 확인되도록 "click x" 등으로 화면 출력
3) 스위치를 눌렀을 때 0->1, 눌렀다 떼었을 때 1->0으로 값이 변경되므로 0->1인 경우만 동작하도록 변경
4) 4개의 스위치 입력받도록 해보자. 화면에 아래와 같이 출력되도록 한다. 단, 리스트를 최대한 활용하여 GPIO 전/후 값을 저장한다.

```
import RPi.GPIO as GPIO
import time

# 4) 4개의 스위치 핀 번호, 클릭 횟수, 이전 상태를 리스트로 관리
sw_pins = [5, 6, 13, 19]
click_counts = [0] * len(sw_pins)
prev_gpio_states = [0] * len(sw_pins)

GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)

for pin in sw_pins:
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

try:
    while True:
        for i, pin in enumerate(sw_pins):
            current_gpio_state = GPIO.input(pin)

            # 3) 스위치를 누르는 순간(0->1)만 감지
            if current_gpio_state == 1 and prev_gpio_states[i] == 0:
                click_counts[i] += 1
                # 1), 2) 스위치가 눌렸을 때만, 몇 번 스위치인지와 함께 "click" 출력
                print("Switch {0} click {1}".format(i+1, click_counts[i]))

            # 다음 감지를 위해 현재 상태를 이전 상태로 저장
            prev_gpio_states[i] = current_gpio_state

        time.sleep(0.1)

except KeyboardInterrupt:
    pass

finally:
    GPIO.cleanup()
```

- 실행 결과



- 고찰

해당 코드의 큰 흐름은 다음과 같습니다.

①코드 실행: 입력 대기

②버튼 입력: 버튼 종류, 카운트 출력

코드에서 스위치의 '이전 상태'와 현재 상태를 비교합니다. 버튼의 상태가 0->1이 되는 순간을 감지합니다. 상태 변화를 감지하는 방식으로 while 문을 통하여 상태를 계속해서 체크하는 방식으로 이것이 구현되어있습니다(폴링 방식). 때문에 지속적으로 cpu 할당을 요구하여 오랜 기간 신호가 들어오지 않는다면 비효율적인 방법입니다. 때문에 인터럽트 방식을 통해서 코드를 개선할 수 있습니다. 개선 시 cpu가 버튼 입력이 없는 경우에, 기존의 방식과 비교하여 동시에 다른 작업을 진행하고 있는 경우 다른 작업에 더 cpu를 할당할 수 있게 됩니다. 결과적으로 cpu의 효율이 증가할 수 있습니다.

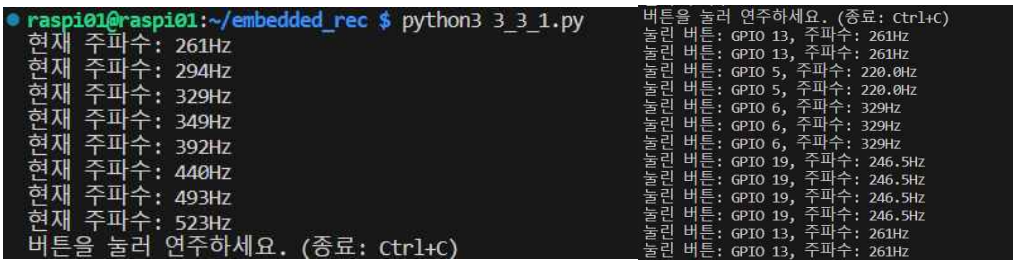
2. 부저 음계 출력 구현

1) "도레미파솔라시도" 음계를 출력
2) 나만의 경적 소리 구현
3) 스위치를 한번 누르면 경적 소리가 나도록 구현
4) 스위치 4개를 사용하여 나만의 음악을 연주

```
import RPi.GPIO as GPIO
import time

# 사용할 핀 번호 설정
BUZZER = 12
sw_pins = [5, 6, 13, 19]

# '도레미파솔라시도' 음계 주파수 리스트
scale = [261, 294, 329, 349, 392, 440, 493, 523]

# 각 스위치 핀에 '도, 레, 미, 파' 음을 할당 (Dictionary 자료형 사용)
note_map = {
    sw_pins[0]: scale[5]/2,   # 5번핀: 라
    sw_pins[1]: scale[2],   # 6번핀: 미
    sw_pins[2]: scale[0],   # 13번핀: 도
    sw_pins[3]: scale[6]/2    # 19번핀: 시
}

# GPIO 설정
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUZZER, GPIO.OUT)
for pin in sw_pins:
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# PWM 객체 생성 (초기 주파수 100Hz)
p = GPIO.PWM(BUZZER, 100)

try:
    # 1. 프로그램 시작 시 '도레미파솔라시도' 연주
    p.start(50)  # Duty Cycle 50%로 PWM 시작
    for freq in scale:
        p.ChangeFrequency(freq)
        print("현재 주파수: {0}Hz".format(freq))
        time.sleep(0.5)
    p.stop()  # 연주가 끝나면 PWM 정지
    print("버튼을 눌러 연주하세요. (종료: Ctrl+C)")
```

```
    # 2. 버튼 입력을 기다리는 무한 루프
    while True:
        pressed_pin = None
        # 모든 스위치 핀을 확인하여 눌린 버튼이 있는지 찾음
        for pin in sw_pins:
            if GPIO.input(pin) == 1:
                pressed_pin = pin
                break # 버튼 하나가 눌리면 더 이상 찾지 않음

        if pressed_pin is not None:
            # 3. 버튼이 눌렸으면 해당 음으로 주파수 변경 후 소리 재생
            freq = note_map[pressed_pin]
            p.ChangeFrequency(freq)
            p.start(50)
            print("눌린 버튼: GPIO {0}, 주파수: {1}Hz".format(pressed_pin, freq))
        else:
            # 4. 아무 버튼도 눌리지 않았으면 소리 정지
            p.stop()

        time.sleep(0.1) # CPU 부하를 줄이기 위한 짧은 대기


except KeyboardInterrupt:
    pass

finally:
    p.stop()
    GPIO.cleanup()
```

- 실행 결과



- 고찰

① 코드 실행: 자동 음계 출력 (도레미파솔라시도)

② 입력 대기

③ 버튼 입력: 사용자가 누른 버튼에 해당하는 음계가 버튼이 눌려있는 동안 출력됩니다.

④ 입력 대기: 버튼에서 손을 떼는 경우 다시 입력 대기 상태로 돌아갑니다.

'도레미파솔라시도'의 음계를 scale 리스트에서 저장하여 필요할 때 호출하여 사용하는 방식으로 처리했습니다. 배열의 형식으로 호출할 수 있는 방식이 개인적으로 적절하다고 생각하여 사용했습니다. 다른 옥타브의 음의 경우 *2를 하면 한 옥타브 위의 음을 쓸 수 있습니다. 아래 옥타브의 경우 /2를 통해 구현할 수 있기에 scale[5]/2의 형식으로 아래 옥타브를 사용했습니다.

구현 과정에서 이 코드 또한 while을 통한 폴링 방식을 사용하여 cpu 효율에도 문제가 있을 수 있으나, 두 개 이상의 음을 한 번에 출력하는 것이 불가능하다는 한계가 존재합니다. 화음 자체가 1개의 스피커로 출력하는 것이 가능한지 자체를 잘 모르겠으나 인터럽트 방식으로 처리할 경우 2개 이상의 입력이 있을 경우에도 받을 수 있습니다.

3. 자동차 움직이기 구현

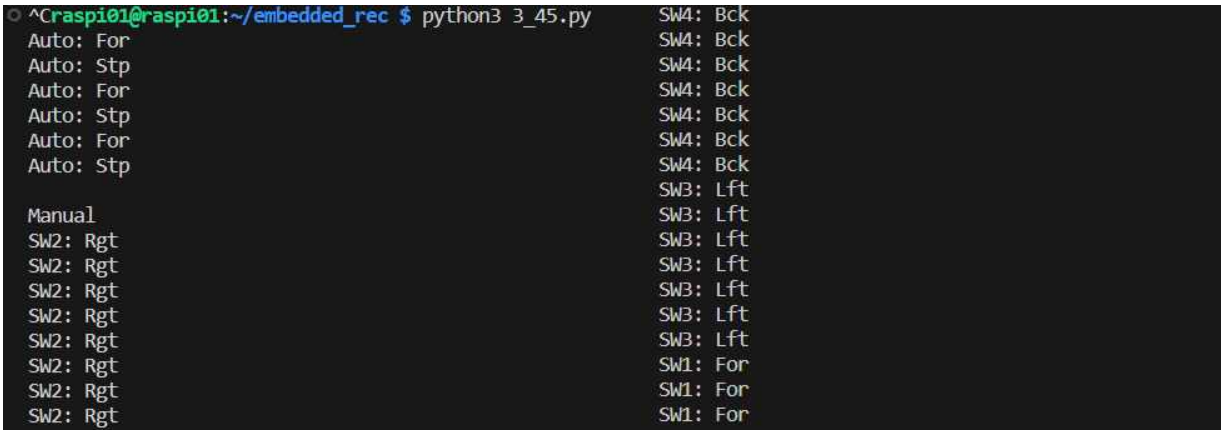
1) 오른쪽 모터부분의 코드를 추가하여 정방향으로 50%로 동작->정지->동작->정지...
2) 스위치를 입력 받아 자동차 조종하기
SW1 : 앞
SW2 : 오른쪽
SW3 : 왼쪽
SW4 : 뒤
print문을 사용하여 어느 스위치가 눌렸는지 출력

```
import RPi.GPIO as GPIO
import time

# --- 핀 번호 설정 ---
# 모터 A (왼쪽)
PWMA = 18; AIN1 = 22; AIN2 = 27
# 모터 B (오른쪽)
PWMB = 23; BIN1 = 25; BIN2 = 24
# 스위치
SW1, SW2, SW3, SW4 = 5, 6, 13, 19

# --- GPIO 설정 ---
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
motor_pins = [PWMA, AIN1, AIN2, PWMB, BIN1, BIN2]
for pin in motor_pins:
    GPIO.setup(pin, GPIO.OUT)
switch_pins = [SW1, SW2, SW3, SW4]
for pin in switch_pins:
    GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)

# --- PWM 객체 생성 ---
L_Motor = GPIO.PWM(PWMA, 500)
L_Motor.start(0)
R_Motor = GPIO.PWM(PWMB, 500)
R_Motor.start(0)

# --- 모터 제어 함수 ---
def move_forward(speed=50):
    GPIO.output(AIN1, 0); GPIO.output(AIN2, 1)
    GPIO.output(BIN1, 0); GPIO.output(BIN2, 1)
    L_Motor.ChangeDutyCycle(speed)
    R_Motor.ChangeDutyCycle(speed)
```

```
def move_backward(speed=50):
    GPIO.output(AIN1, 1); GPIO.output(AIN2, 0)
    GPIO.output(BIN1, 1); GPIO.output(BIN2, 0)
    L_Motor.ChangeDutyCycle(speed)
    R_Motor.ChangeDutyCycle(speed)

def turn_left(speed=50):
    GPIO.output(AIN1, 1); GPIO.output(AIN2, 0)
    GPIO.output(BIN1, 0); GPIO.output(BIN2, 1)
    L_Motor.ChangeDutyCycle(speed)
    R_Motor.ChangeDutyCycle(speed)

def turn_right(speed=50):
    GPIO.output(AIN1, 0); GPIO.output(AIN2, 1)
    GPIO.output(BIN1, 1); GPIO.output(BIN2, 0)
    L_Motor.ChangeDutyCycle(speed)
    R_Motor.ChangeDutyCycle(speed)

def stop():
    L_Motor.ChangeDutyCycle(0)
    R_Motor.ChangeDutyCycle(0)

# --- 메인 코드 ---
try:
    # --- 상태 관리 변수 ---
    patrol_active = True  # True일 때만 자율 주행 활성화
    last_patrol_time = time.time()
    patrol_is_moving = False

    while True:
        # 4개의 스위치 상태를 한 번에 읽기
        sw1_state = GPIO.input(SW1)
        sw2_state = GPIO.input(SW2)
        sw3_state = GPIO.input(SW3)
        sw4_state = GPIO.input(SW4)

        # 1. 버튼이 하나라도 눌렸는지 확인
        if sw1_state or sw2_state or sw3_state or sw4_state:
            # 버튼이 눌리는 순간, 자율 주행 모드를 영구적으로 비활성화
            if patrol_active:
                print("₩nManual")
                patrol_active = False
```

```
            # 눌린 버튼에 따라 수동 조작 실행
            if sw1_state:
                print("SW1: For")
                move_forward()
            elif sw2_state:
                print("SW2: Rgt")
                turn_right()
            elif sw3_state:
                print("SW3: Lft")
                turn_left()
            elif sw4_state:
                print("SW4: Bck")
                move_backward()

        # 2. 아무 버튼도 눌리지 않았을 경우
        else:
            # (1) 아직 자율 주행 모드일 때 -> 전진/정지 반복
            if patrol_active:
                if time.time() - last_patrol_time > 1.0:
                    patrol_is_moving = not patrol_is_moving
                    last_patrol_time = time.time()
                    if patrol_is_moving:
                        print("Auto: For")
                        move_forward(100)
                    else:
                        print("Auto: Stp")
                        stop()
            # (2) 수동 조작 후 버튼에서 손을 뗀 상태 -> 그냥 정지
            else:
                stop()

        time.sleep(0.02)

except KeyboardInterrupt:
    pass

finally:
    GPIO.cleanup()
```

- 실행 결과



- 고찰

① 코드 실행: 자동차가 전진(1s) -> 정지(1s) -> 전진(1s) -> 정지(1s)... 를 자동으로 반복합니다.

② 버튼 입력: 자동차가 자동조작을 영구적으로 멈추고 스위치에 해당하는 움직임으로 주행합니다.(전후좌우)

③ 입력 대기: 자동차가 수동으로 조작하는 것(버튼 입력)을 대기합니다.

위 코드에선 time.sleep()으로 프로그램 멈추는 대신, 시간을 지속적으로 확인하는 방식으로 전진, 정지를 반복합니다. 이를 비동기 방식이라고 한다고 합니다. 때문에 자동으로 주행하다가도 버튼 입력을 인식하는 경우 자동으로 주행하는 기증이 바로 꺼지고 수동제어로 바뀌게 됩니다. 이를 통해 버튼으로만 작동하게 바뀝니다. 이 코드 또한 폴링 방식을 사용하기 때문에 동시에 2개의 버튼을 누르는 작업이 불가능하여 좌회전+직진과 같은 두 개가 결합된 신호를 받지 못하며, 대각선 움직임 또한 버튼으로 구현하지 않았기 때문에 2개 이상의 행동이 결합된 내용을 받지 못합니다.