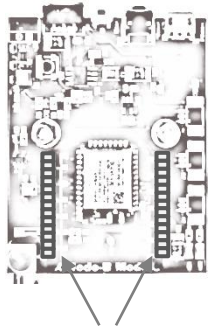
XNode로 배우는

저전력 무선 네트워크 프로그래밍

확장 모듈 제어

확장 모듈 제어 – B Type

- XNode는 26핀 확장 커넥터를 통해 센서 확장 지원
  - 전원, GPIO 및 특수 기능 인터페이스(ADC, I2C)
    - 리셋 버튼은 MCU 리셋 핀에 연결됨
    - XNode 할당
      - GPIO: 배터리 잔량 측정(D2(ADC2)), LED(D9)
      - I2C: Light Sensor(I2C1, 0x23), Tphg Sensor(I2C1, 0x77)
    - 확장 모듈 할당
      - GPIO: BASIC의 Buzzer(P0), PIR(P2)
      - I2C: BASIC의 Leds와 Buttons(I2C1, 0x24), IRTHERMO(I2C1, 0x5A), IMU(I2C1, 0x28), GPS(I2C1, 0x9A)

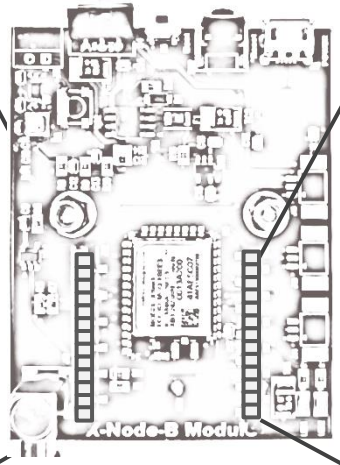


확장 커넥터

확장 모듈 제어 – B Type

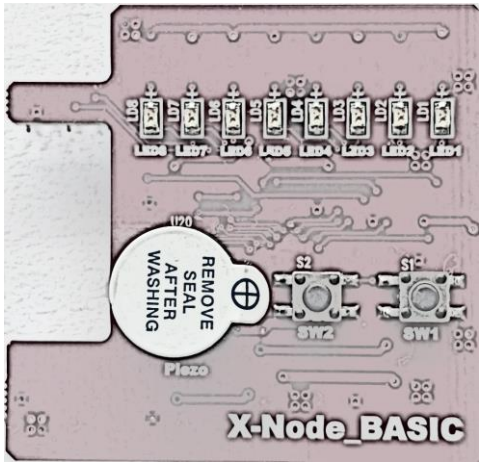
- 확장 모듈 포함 확장 커넥터 할당 정보

	5V	1	2	5V		
	3V3	3	4	3V3		
Reset Button	RESET	5	6	D4		
	D0/ADC0	7	8	D5		
Battery Remains	D2/**ADC2**	9	10	D6		
	D3/ADC3	11	12	D7		
	D8	13	14	**P2**	Ext PIR	
Led	**D9**	15	16	P9		
Ext BASIC(Buzzer)	**P0**/PWM0	17	18	P7/IN_ONLY		
Ext BASIC(Leds,buttons)	P1/PWM1/**I2C_SDA**	19	20	NOT USED		
Ext IRTHERMO, IMU, GPS	D1/ADC1/**I2C_SCL**	21	22	P5/SPI_MISO/OUT_ONLY		
	P8/SPI_CLK/IN_ONLY	23	24	P6/SPI_MOSI/IN_ONLY		
	GND	25	26	GND		



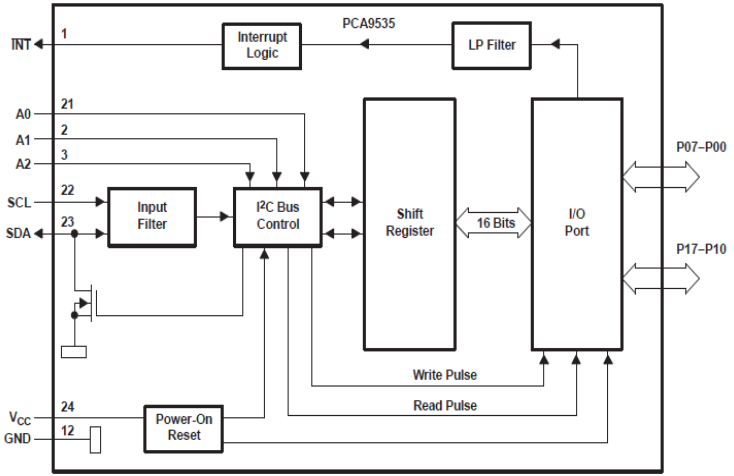
BASIC 확장 모듈

- X-Node BASIC
  - 8개의 Led와 2개의 버튼 및 부저로 구성
    - 다수의 Led와 버튼 제어를 위해 I2C 기반 GPIO 확장 칩(PCA9535) 내장
      - 버튼 입력 감지는 폴링으로 처리
    - 부저는 GPIO에 연결되며 간단한 비프 음 출력



BASIC 확장 모듈

- PCA9585는 MCU에서 제공하는 GPIO가 부족할 때 이를 확장하는 데 사용
  - I2C 컨트롤러와 시프트 레지스터, 입/출력 포트 및 인터럽트 컨트롤러로 구성
  - 2개(각 8bit)의 입/출력 포트는 레지스터 설정에 따라 각각 입력 또는 출력으로 사용
    - 포트 0: 입력으로 2개의 버튼에 연결됨
    - 포트 1: 출력으로 8개의 Led에 연결됨



BASIC 확장 모듈 제어

- 준비물

준비물	
1	PC
2	XNode 1ea
3	Micro type USB cable 1ea
4	XNode 제공 USB 메모리 (D: 로 가정)
5	BASIC 모듈

```
PS C:\XNode\BASIC> xnode -p com3 ls /flash/lib
/flash/lib/BASIC.py
/flash/lib/core_a.py
/flash/lib/core_b.py
/flash/lib/pop.py
```

- XNode와 PC를 USB 케이블로 연결
- D:₩Library₩EXT₩lib₩BASIC.py을 작업 폴더인 C:₩XNode₩BASIC₩lib로 복사
  - C:₩XNode₩BASIC₩lib
- VS Code를 실행한 후 C:₩XNode₩BASIC₩lib 폴더를 XNode에 복사
  - xnode -p <포트> put ₩BASIC₩lib

BASIC 확장 모듈 제어

- X-Node BASIC을 위한 Pop 라이브러리
  - Buzzer class
    - Buzzer() : Buzzer 객체 생성
    - Buzzer.on(): '삐' 소리 출력
    - Buzzer.off(): 소리 출력 멈춤
    - Buzzer.beep(delay, func=None, param=None, on=50, off=40): 일정 시간 비프 음 출력
      - delay: 출력 횟수. 시간은 on, off 인자에 의존
      - func: 한 번 울릴 때마다 호출될 사용자 함수. 기본값은 None
      - param: 사용자 함수를 호출할 때 전달할 인자. 기본값은 None
      - on: 밀리초 단위 on 유지 시간. 기본값은 50ms
      - off: 밀리초 단위 off 유지 시간. 기본값은 40ms

BASIC 확장 모듈 제어

- Leds class
  - Leds() : Leds 객체 생성
  - Leds(): 모든 Led 상태 반한
    - 반환 값은 튜플
  - Leds: 반복자 반환
    - *Led*.on(): 해당 Led 켜기
    - *Led*.off(): 해당 Led 끄기
  - Leds[n]: 인덱스 연산 호환
    - n: 0~7 범위 정수
  - Leds.write(n): 8bit 기준 저수준 쓰기
    - n: 8bit 범위 정수
  - Leds.clear(): 모든 Led 끄기

BASIC 확장 모듈 제어

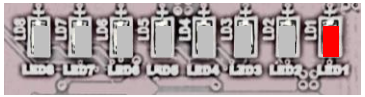
- Buttons class
  - Buttons() : Buttons 객체 생성
  - Buttons(): 모든 Button 상태 반환
    - 반환 값은 튜플
  - Buttons[n]: 인덱스 연산 호환. 읽기 전용
    - n: 0~1 범위 정수
  - Buttons.read(): 2bit 기준 저수준 읽기
    - Button 상태에 따라 0~3 사이 값 반환

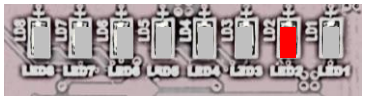
BASIC 확장 모듈 – LEDs

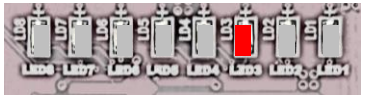
- Leds 제어 예제
  - 반복자를 이용해 Led 1에서 Led 8까지 차례로 켜고 끄기

```
01: from pop import Leds
02: from pop import time
03:
04: leds = Leds()
05:
06: for led in leds:
07:     led.on()
08:     time.sleep(.1)
09:     led.off()
10:     time.sleep(.1)
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_leds_iter.py
```







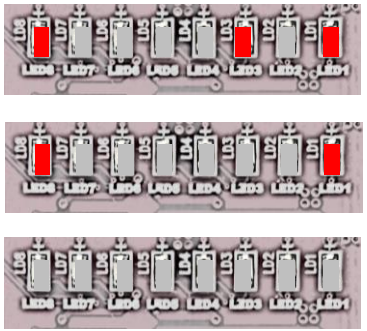
...

BASIC 확장 모듈 – LEDs

- 인덱스로 해당 Led 제어

```
01: from pop import Leds
02: from pop import time
03:
04: leds = Leds()
05:
06: print(leds()) #모든 Led 상태 출력
07: time.sleep(1)
08:
09: leds[0] = True; leds[2] = True; leds[7] = True
10: print(leds())
11: time.sleep(1)
12:
13: leds[2] = False
14: print(leds())
15: time.sleep(1)
16:
17: leds.clear()
18: print(leds())
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_leds_index.py
(False, False, False, False, False, False, False, False)
(True, False, True, False, False, False, False, True)
(True, False, False, False, False, False, False, True)
(False, False, False, False, False, False, False, False)
```

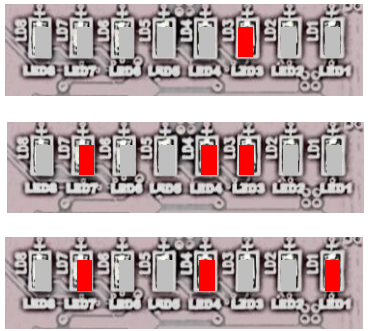


BASIC 확장 모듈 – LEDs

- write()에 난수를 전달해 해당 Led 켜고 끄기

```
01: from pop import Leds, rand
02: from pop import time
03:
04: leds = Leds()
05:
06: leds.write(128)
07: time.sleep(1)
08: leds.write(127)
09: time.sleep(1)
10:
11: for i in range(10):
12:     n = rand() % 255 + 1
13:     print(n)
14:     leds.write(n)
15:     time.sleep(.5)
16:
17: leds.clear()
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_leds_write.py
4
76
73
```

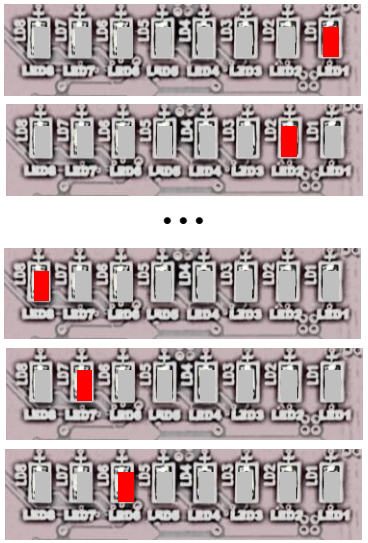


BASIC 확장 모듈 – LEDs

- 시프트 연산으로 Led를 차례로 왼쪽에서 오른쪽, 오른쪽에서 왼쪽으로 이동 반복하기

```
01: from pop import Leds
02: from pop import time
03:
04: leds = Leds()
05:
06: for _ in range(3):
07:     for i in range(8):
08:         leds.write(1<<i)
09:         time.sleep(.05)
10:
11:     for i in range(i, -1, -1):
12:         leds.write(1<<i)
13:         time.sleep(.05)
14:
15: leds.clear()
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_leds_shift.py
```



BASIC 확장 모듈 제어- Buttons

- Buttons 제어
  - read()로 버튼 상태 읽기. 버튼을 모두 누르면 종료

```
01: from pop import Buttons
02: from pop import time
03:
04: buttons = Buttons()
05:
06: while True:
07:     n = buttons.read()
08:     print(n)
09:     if n == 3:
10:         break
11:     time.sleep(.1)
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buttons_read.py
0
0
0
1
1
1
1
0
0
0
2
2
2
2
0
0
0
0
0
3
```

BASIC 확장 모듈 제어- Buttons

- 튜플로 버튼 상태 읽기. 버튼을 모두 누르면 종료

```
01: from pop import Buttons
02: from pop import time
03:
04: buttons = Buttons()
05:
06: while True:
07:     n = buttons()
08:     print(n)
09:     if n[0] == n[1] == 1:
10:         break
11:     time.sleep(.1)
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buttons_tuple.py
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(1, 0)
(0, 0)
(0, 0)
(0, 1)
(0, 1)
(0, 0)
(0, 0)
(0, 0)
(0, 0)
(1, 1)
```

BASIC 확장 모듈 제어- Buttons

- 약 10초간 인덱스로 해당 버튼을 누를 때만 반응하기

```
01: from pop import Buttons
02: from pop import time
03:
04: buttons = Buttons()
05:
06: for _ in range(100):
07:     for i in range(2):
08:         ret = buttons[i]
09:         if ret:
10:             print("Button[%d] Press"%(i+1))
11:         time.sleep(.1)
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buttons_index.py
Button[1] Press
Button[2] Press
Button[2] Press
Button[1] Press
Button[2] Press
Button[2] Press
Button[1] Press
Button[1] Press
```

BASIC 확장 모듈 제어- Buttons

- 해당 버튼의 눌렀다(Press) 놓은(Release) 상태 구분. <Ctrl> c를 눌러 강제 종료

```
01: from pop import Buttons
02: from pop import time
03:
04: buttons = Buttons()
05: bt0 = bt1 = False
06:
07: while True:
08:     if buttons[0] != bt0:
09:         bt0 = not bt0
10:         if bt0:
11:             print("Button0 Press")
12:         else:
13:             print("Button0 Release")
14:     if buttons[1] != bt1:
15:         bt1 = not bt1
16:         if bt1:
17:             print("Button1 Press")
18:         else:
19:             print("Button1 Release")
20:     time.sleep(.1)
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buttons_stat.py
Button0 Press
Button0 Release
Button1 Press
Button1 Release
Button0 Press
Button1 Press
Button0 Release
Button1 Release
```

BASIC 확장 모듈 제어- Buttons

- 첫 번째 버튼이 눌러져 있던 시간 계산. <Ctrl> c를 눌러 강제 종료

```
01: from pop import Buttons
02: from pop import time
03:
04: buttons = Buttons()
05:
06: bt0 = False
07:
08: while True:
09:     if buttons[0] != bt0:
10:         bt0 = not bt0
11:         if bt0:
12:             t0 = time.ticks_ms()
13:             print("Button1 Press")
14:         else:
15:             duration = time.ticks_ms() - t0
16:             print("Duration: %d ms"%(duration))
17:
18:     time.sleep(.1)
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buttons_duration.py
Button1 Press
Duration: 1550 ms
Button1 Press
Duration: 928 ms
Button1 Press
Duration: 3209 ms
```

BASIC 확장 모듈 제어- Buzzer

- Buzzer 제어
  - Buzzer로 2초간 '삐'소리 출력

```
01: from pop import Buzzer
02: from pop import time
03:
04: buzzer = Buzzer()
05:
06: buzzer.on()
07: print("Buzzer On")
08: time.sleep(2)
09:
10: buzzer.off()
11: print("Buzzer Off")
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buzzer.py
Buzzer On
Buzzer Off
```

BASIC 확장 모듈 제어- Buzzer

- beep()로 횟수만큼 소리 출력

```
01: from pop import Buzzer
02: from pop import time
03:
04: buzzer = Buzzer()
05:
06: print("beep On")
07: buzzer.beep(10)
08: print("beep Off")
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buzzer_beep.py
beep On
beep Off
```

BASIC 확장 모듈 제어- Buzzer

- beep()로 소리를 출력할 때마다 사용자 함수 실행

```
01: from pop import Buzzer
02: from pop import time
03:
04: buzzer = Buzzer()
05:
06: count = 0
07: def user():
08:     global count
09:     count += 1
10:     print("call user function: %d"%(count))
11:
12: buzzer.beep(10, user)
13: print("The End")
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_buzzer_beep_ex.py
call user function: 1
call user function: 2
call user function: 3
call user function: 4
call user function: 5
call user function: 6
call user function: 7
call user function: 8
call user function: 9
call user function: 10
The End
```

BASIC 확장 모듈 제어 – 종합 제어

- BASIC 종합
  - 첫 번째 버튼을 누르면 Buzzer로 비프 음을 출력하고 두 번째 버튼은 Leds에 난수 출력
    - 첫 번째 버튼을 2초 이상 눌렸다 떼면 프로그램 종료

```
01: from pop import Buzzer, Leds, Buttons, rand
02: from pop import time
03:
04: buzzer = Buzzer()
05: leds = Leds()
06: buttons = Buttons()
07:
08: bt0 = bt1 = False
09: print("Start")
10:
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_total.py
Start
19
40
88
The End
```

BASIC 확장 모듈 제어 – 종합 제어

- (계속)

```
11: while True:
12:     if buttons[0] != bt0:
13:         bt0 = not bt0
14:         if bt0:
15:             t0 = time.ticks_ms()
16:             buzzer.beep(5)
17:         else:
18:             if time.ticks_ms() - t0 > 2000:
19:                 break
20:
21:     if buttons[1] != bt1:
22:         bt1 = not bt1
23:         if bt1:
24:             n = rand() % 255 + 1
25:             leds.write(n)
25:             print(n)
27:         else:
28:             leds.clear()
29:
30:     time.sleep(.1)
31:
32: print("The End")
```

BASIC 확장 모듈 제어 – 종합 제어

- 앞 예제를 수정해 두 번째 버튼을 누르면 Leds에 난수 출력하고 떼면 다음 내용 처리
  - XNode의 Light와 Tphg로부터 조도 및 온도, 기압, 습도, 가스 측정값 출력

```
01: from pop import Battery, Light, Tphg
02: from pop import Buzzer, Leds, Buttons, rand
03: from pop import time
04:
05: battery = Battery()
06: light = Light()
07: tphg = Tphg()
08:
09: buzzer = Buzzer()
10: leds = Leds()
11: buttons = Buttons()
12:
13: bt0 = bt1 = False
14:
15: print("Start")
16:
```

```
PS C:\XNode\BASIC> xnode -p com3 run ext_basic_core.py
Start
222
3.80 130.00 28.22 1020.02 25.55 36964
128
3.80 127.00 28.24 1020.02 26.15 8485
225
3.80 130.00 28.30 1020.03 26.48 13111
The End
```

BASIC 확장 모듈 제어 – 종합 제어

- (계속)

```
17: while True:
18:     if buttons[0] != bt0:
19:         bt0 = not bt0
20:         if bt0:
21:             t0 = time.ticks_ms()
22:             buzzer.beep(5)
23:         else:
24:             if time.ticks_ms() - t0 > 2000:
25:                 break
26:
27:     if buttons[1] != bt1:
28:         bt1 = not bt1
29:         if bt1:
30:             n = rand() % 255 + 1
31:             leds.write(n)
32:             print(n)
33:         else:
34:             leds.clear()
```

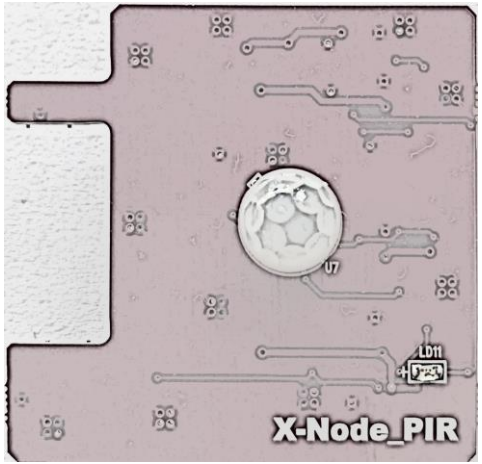
BASIC 확장 모듈 제어 – 종합 제어

- (계속)

```
35:
36:             v = battery.read()
37:             l = light.read()
38:             t, p, h, g = tphg.read()
39:             print("%.2f %.2f %.2f %.2f %.2f %d"%(v,l, t, p, h, g))
40:
41:     time.sleep(.1)
42:
43: print("The End")
```

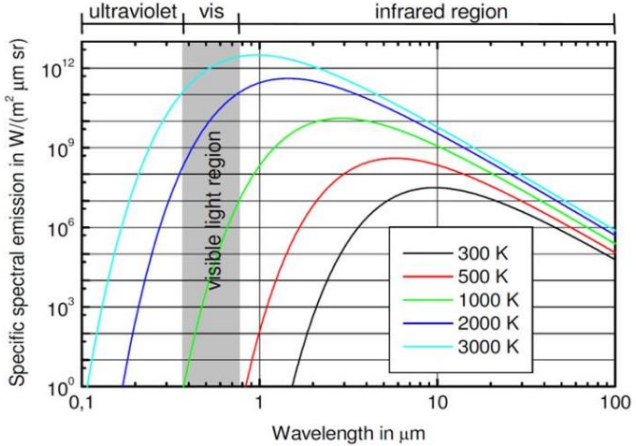
PIR 확장 모듈

- X-Node PIR (Passive infrared)
  - 사람이 방출하는 적외선량의 변화를 통해 움직임을 감지하는 센서 내장
    - 프레넬 렌즈를 통해 일정 구간의 적외선량 변화 감지
      - GPIO에 연결되며 움직임이 감지될 때마다 HIGH 출력
        - 동작에 따라 여러 번 출력될 수 있음
      - 사람과 유사한 양의 적외선을 방출하는 동물에도 적용 가능



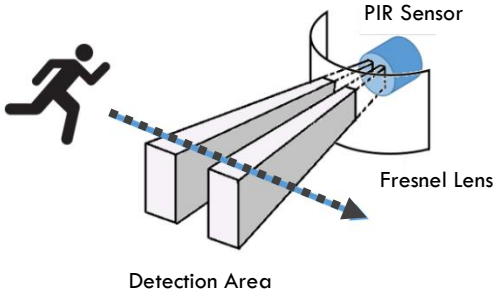
PIR 확장 모듈

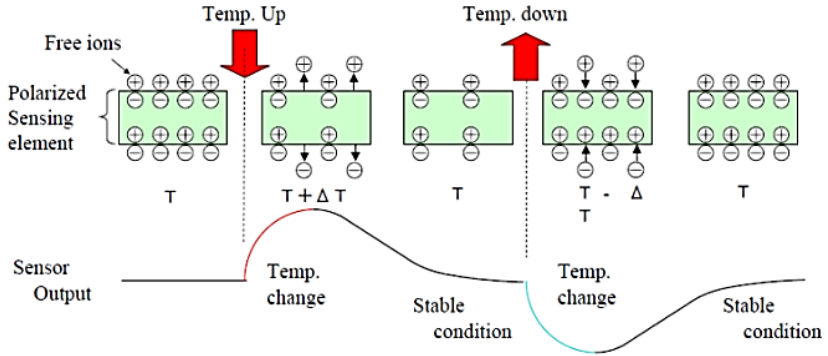
- PIR 센서 원리
  - 모든 물체는 절대온도(-273도)보다 높으면 그에 대응하는 복사선 방출
    - 온도가 높은 물체는 낮은 물체보다 더 많이 적외선을 방사하며, 온도가 높을수록 가시광 영역에 접근
    - 인체의 표면 온도는 옷이나 주위 온도에 따라 다르지만, 일반적으로 20~35도
      - 적외선 파장 분포는 대략 10um 근처에서 최대



PIR 확장 모듈

- 초전효과를 갖는 소자(강유전체)를 통해 적외선의 변화량을 미세한 전기 에너지로 변환
  - 강유전체는 외부의 전기장이 없이도 스스로 분극을 가지는 재료 (자발분극)
    - 소자에 온도 변화가 발생하면 표면에서 전자가 유기되는 현상을 초전효과라 함
    - 소자 표면에서 대전 중인 전하는 공기 중의 이온과 결합해 평형 상태 유지
  - 소자가 적외선을 감지하면 열적 변화에 의해 표면 온도가 상승하면서 평형이 깨짐
    - Pir 센서는 빼앗기거나 보충된 전하량의 변화를 전압 변화로 이용





PIR 확장 모듈 제어

- 준비물

준비물	
1	PC
2	XNode 1ea
3	Micro type USB cable 1ea
4	XNode 제공 USB 메모리 (D: 로 가정)
5	PIR 모듈

```
PS C:\XNode\PIR> xnode -p com3 ls /flash/lib
/flash/lib/PIR.py
/flash/lib/core_a.py
/flash/lib/core_b.py
/flash/lib/pop.py_
```

- XNode와 PC를 USB 케이블로 연결
- D:₩Library₩EXT₩lib₩PIR.py을 작업 폴더인 C:₩XNode₩PIR₩lib로 복사
  - C:₩XNode₩PIR₩lib
- VS Code를 실행한 후 C:₩XNode₩PIR 아래 lib 폴더를 XNode에 복사
  - xnode -p <포트> put lib

PIR 확장 모듈 제어

- X-Node PIR을 위한 Pop 라이브러리
  - Pir class
    - Pir() : Pir 객체 생성
    - Pir.read(): 저수준으로 Pir 상태 읽기
      - 반환 값은 움직임이 감지될 때 True. 아니면 False
    - Pir.check(mode=ENTER, delay=300): 움직임 감지
      - mode: 감지 시점 지정. Pir.ENTER(기본값), Pir.LEAVER, Pir.BOTH 중 하나
      - delay: 밀리초 단위 다음 감지 시간. 기본값은 300ms
      - 반환 값은 mode에 따라 움직임이 감지되면 True 또는 False . 움직임이 없으면 None

PIR 확장 모듈 제어

- PIR 제어
  - 사용자 움직임이 10회 감지되면 종료. (움직임에 따라 여러 번 인식될 수 있음)

```
01: from pop import Pir
02: from pop import time
03:
04: pir = Pir()
05:
06: count = 0
07:
08: while True:
09:     if pir.check():
10:         count += 1
11:         if count > 10:
12:             break
13:
14:         print("Detect!!!")
15:
16:     time.sleep(.1)
```

```
PS C:\XNode\PIR> xnode -p com3 run ext_pir_check.py
Detect!!!
Detect!!!
Detect!!!
Detect!!!
Detect!!!
```

PIR 확장 모듈 제어

- check()의 mode와 delay를 변경해 움직임을 세밀하게 감지

```
01: from pop import Pir
02: from pop import time
03:
04: pir = Pir()
05:
06: count = 0
07:
08: while True:
09:     n = pir.check(mode=Pir.BOTH, delay=50)
10:     if n ==  True:
11:         count += 1
12:         if count > 3:
13:             break
14:         print("Enter")
15:     elif n == False:
16:         print("Leave")
17:     else:
18:         print(n)
19:
20     time.sleep(.1)
```

```
PS C:\XNode\PIR> xnode -p com3 run ext_pir_check_both.py
None
None
None
None
None
Enter
None
None
None
None
Leave
Enter
Leave
None
None
None
None
Enter
None
Leave
```

PIR 확장 모듈 제어

- read()로 저수준 움직임 감지
  - 프로그램은 <Ctrl> c를 눌러 강제 종료

```
01: from pop import Pir
02: from pop import time
03:
04: pir = Pir()
05:
06: while True:
07:     n = pir.read()
08:     print(n)
09:     time.sleep(.1)
```

```
PS C:\XNode\PIR> xnode -p com3 run ext_pir_read.py
0
0
0
0
0
0
1
1
0
0
1
0
0
1
1
1
0
0
0
0
0
0
1
1
0
1
```

PIR 확장 모듈 제어

- read()를 이용해 10초간 움직인 횟수 카운트. (한 번 움직임이 여러 번 카운트 될 수 있음)

```
01: from pop import Pir
02: from pop import time
03:
04: pir = Pir()
05:
06: count = 0
07: t = time.ticks_ms()
08:
09: print("Start!!!")
10:
11: while True:
12:     if pir.read():
13:         count += 1
14:         if not count % 100:
15:             print("#", end='')
16:
17:     if time.ticks_ms() - t > 10 * 1000:
18:         break
19:
20: print("₩nYour are moving power: %d"%(count))
```

```
PS C:\XNode\PIR> xnode -p com3 run ext_pir_read_count.py
Start!!!
##############################################
Your are moving power: 4727
```

PIR 확장 모듈 제어

- PIR 종합
  - 움직임이 감지될 때마다 배터리 전압, 조도, 온도, 습도 측정값 출력

```
01: from pop import Battery, Light, Tphg
02: from pop import Pir
03: from pop import time
04:
05: battery = Battery()
06: light = Light()
07: tphg = Tphg()
08:
09: pir = Pir()
10:
```

```
PS C:\XNode\PIR> xnode -p com3 run ext_pir_core.py
BLTH: 3.80 96.00 30.15 28.04
BLTH: 3.80 103.00 30.17 27.99
BLTH: 3.80 103.00 30.19 27.87
BLTH: 3.80 102.00 30.19 27.81
```

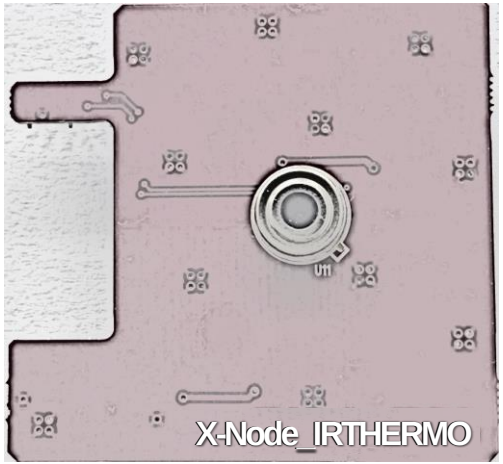
PIR 확장 모듈 제어

- PIR 종합
  - (계속)

```
11: while True:
12:     if pir.check():
13:         b = battery.read()
14:         l = light.read()
15:         t, _, h, _ = tphg.read()
16:
17:         print("BLTH: %.2f %.2f %.2f %.2f"%(b, l, t, h))
18:
19:     time.sleep(.1)
```

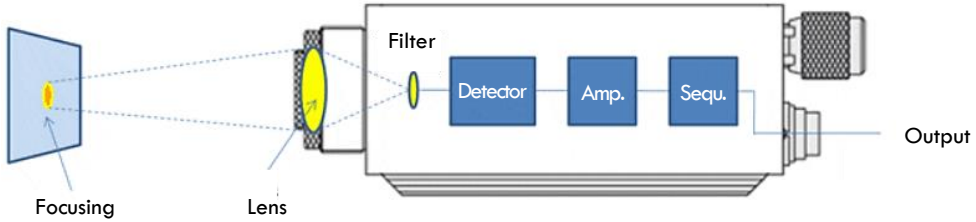
IRTHERMO 확장 모듈

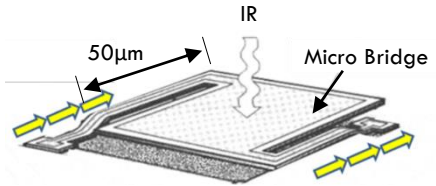
- IRTHERMO (IR Thermometer)
  - 적외선 기반 비접촉 온도 센서(MLX90614) 내장
    - I2C 버스에 연결되며 일정 거리에서 온도 측정
      - 적정 거리는 환경에 따라 다르며 약 40cm 권장
      - 센서 온도와 물체 온도 반환
        - 센서 온도: -40 ~ 125도
        - 물체 온도: -70 ~ 380도



IRTHERMO 확장 모듈

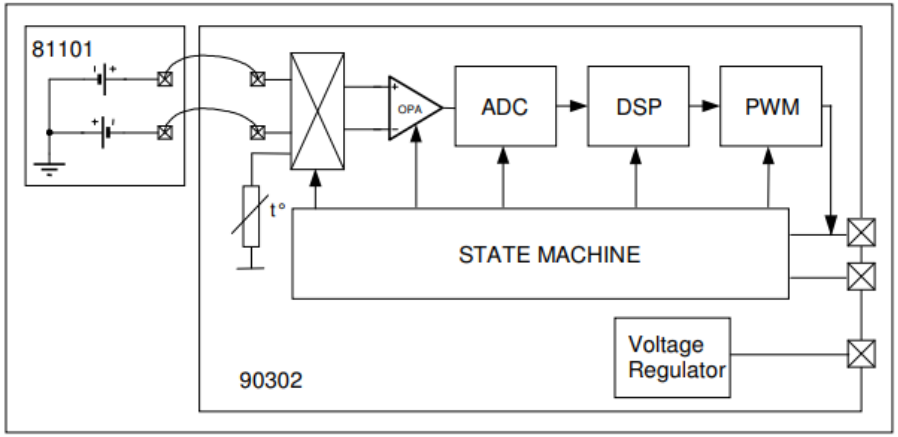
- 비접촉 온도 센서 원리
  - 렌즈를 통해 모인 빛을 필터로 걸러 적외선 범위만 사용
  - 필터를 통과한 적외선 광자량에 따라 발생하는 전기적 특정 이용
    - 디텍터에 흐르는 미세 전류는 필터를 통과한 적외선 광자량에 따라 달라짐
    - 증폭기를 통해 전류 변화를 증폭한 후 선형 회로에서 주변 온도와 방사율을 적용해 온도 계산





IRTHERMO 확장 모듈

- MLX90614는 물체가 방출하는 IR 파장을 흡수해 온도 감지
  - 저잡음 증폭기와 17bit ADC 및 강력한 DSP가 하나의 ASIC으로 통합
  - XNode는 I2C 버스를 통해 0.02도 분해능으로 보정된 온도를 읽음
    - PWM 출력은 0.14도 분해능으로 -20 ~ 120도 범위의 온도 출력



IRTHERMO 확장 모듈 제어

준비물

준비물	
1	PC
2	XNode 1ea
3	Micro type USB cable 1ea
4	XNode 제공 USB 메모리 (D: 로 가정)
5	IRTHERMO 모듈

```
PS C:\XNode\IRTHERMO> xnode -p com3 ls /flash/lib
/flash/lib/IRTHERMO.py
/flash/lib/core_a.py
/flash/lib/core_b.py
/flash/lib/pop.py
```

- XNode와 PC를 USB 케이블로 연결
- D:₩Library₩IRTHERMO을 작업 폴더인 C:₩XNode로 복사
  - C:₩XNode₩IRTHERMO
- VS Code를 실행한 후 C:₩XNode₩IRTHERMO 아래 lib 폴더를 XNode에 복사
  - xnode -p <포트> put lib

IRTHERMO 확장 모듈 제어

- X-Node IRTHEROM를 위한 Pop 라이브러리
  - IRThermo class
    - IRThermo() : IRThermo 객체 생성
    - IRThermo.ambient(): 주변 온도 측정
      - 반환 값은 주변 섭씨온도
    - IRThermo.object(): 대상 온도 측정
      - 반환 값은 대상 섭씨온도

IRTHERMO 확장 모듈 제어

- IRTHERMO 제어
  - 10회 대상 온도 측정

```
01: from pop import IRThermo
02: from pop import time
03:
04: thermo = IRThermo()
05:
06: for _ in range(10):
07:     print("Object: %.2f"%(thermo.object()))
08:     time.sleep(.5)
```

```
PS C:\XNode\IRTHERMO> xnode -p com3 run ext_irthermo_object.py
Object: 27.25
Object: 27.25
Object: 27.29
Object: 27.31
Object: 27.31
Object: 27.25
Object: 27.35
Object: 27.29
Object: 27.31
Object: 27.35
```

IRTHERMO 확장 모듈 제어

- 주변 온도와 대상 온도 함께 측정

```
01: from pop import IRThermo
02: from pop import time
03:
04: thermo = IRThermo()
05:
06: for _ in range(5):
07:     a = thermo.ambient()
08:     o = thermo.object()
09:     print("Ambient: %.2f, Object: %.2f"%(a, o))
10:     time.sleep(.5)
```

```
PS C:\XNode\IRTHERMO> xnode -p com3 run ext_irthermo_ambient.py
Ambient: 28.85, Object: 27.49
Ambient: 28.85, Object: 27.49
Ambient: 28.85, Object: 27.47
Ambient: 28.87, Object: 27.41
Ambient: 28.83, Object: 27.49
```

IRTHERMO 확장 모듈 제어

- IRTHERMO 종합
  - 0.5초마다 조도 값이 100lx보다 크면 Tphg의 온도와 IRThermo의 온도를 비교해 출력
    - Battery, Light 객체와 함께 사용할 때는 반드시 IRThermo 객체를 마지막에 만들 것

```
01: from pop import Battery, Light, Tphg
02: from pop import IRThermo
03: from pop import time
04:
05: battery = Battery()
06: light = Light()
07: tphg = Tphg()
08:
09: thermo = IRThermo() #Make sure to make it last
10:
11: t0 = t1 = time.ticks_ms()
12:
```

```
PS C:\XNode\IRTHERMO> xnode -p com3 run ext_irthermo_core.py
Object: 28.45, Ambient: 28.37, Temp: 30.04, Diff Temp: -1.67
Object: 28.45, Ambient: 27.35, Temp: 30.13, Diff Temp: -2.78
Object: 28.49, Ambient: 27.95, Temp: 30.30, Diff Temp: -2.35
Object: 28.45, Ambient: 27.79, Temp: 30.46, Diff Temp: -2.67
Battery: 3.80
Object: 28.51, Ambient: 28.05, Temp: 30.61, Diff Temp: -2.56
Object: 28.45, Ambient: 27.93, Temp: 30.74, Diff Temp: -2.81
Object: 28.43, Ambient: 27.93, Temp: 30.84, Diff Temp: -2.91
Object: 28.45, Ambient: 27.99, Temp: 30.94, Diff Temp: -2.95
Object: 28.49, Ambient: 28.11, Temp: 31.01, Diff Temp: -2.90
Object: 28.49, Ambient: 27.93, Temp: 31.08, Diff Temp: -3.15
Object: 28.45, Ambient: 27.99, Temp: 31.13, Diff Temp: -3.14
Object: 28.49, Ambient: 28.07, Temp: 31.17, Diff Temp: -3.10
Object: 28.51, Ambient: 28.05, Temp: 31.21, Diff Temp: -3.16
Object: 28.45, Ambient: 28.01, Temp: 31.24, Diff Temp: -3.23
Battery: 3.80
```

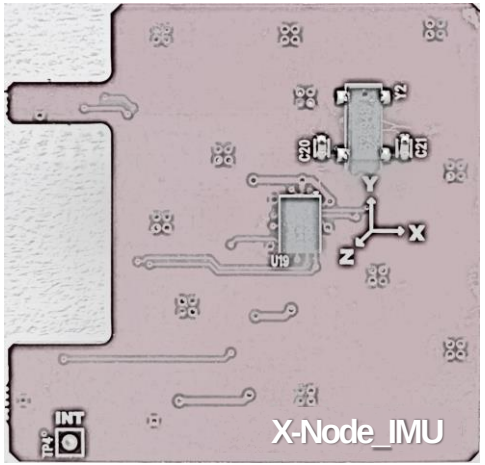
IRTHERMO 확장 모듈 제어

- (계속)

```
13: while True:
14:     if time.ticks_ms() - t1 >= 500:
15:         t1 = time.ticks_ms()
16:         l = light.read()
17:
18:         if l > 100:
19:             t,_,_,_ = tphg.read()
20:             a = thermo.ambient()
21:             o = thermo.object()
22:             print("Object: %.2f, Ambient: %.2f, Temp: %.2f, Diff Temp: %.2f"%(a, o, t, o-t))
23:
24:     if time.ticks_ms() - t0 >= 5000:
25:         t0 = time.ticks_ms()
26:         print("Battery: %.2f"%(battery.read()))
```

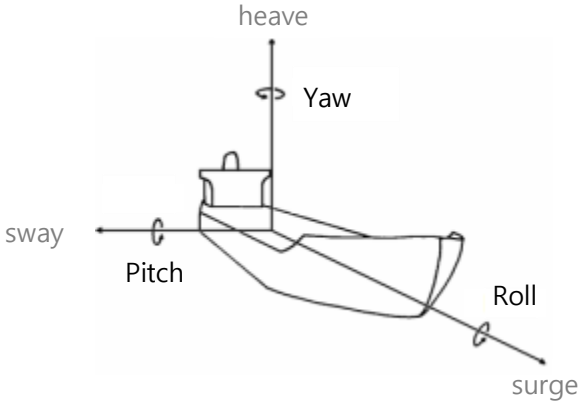
IMU 확장 모듈

- X-Node IMU (Inertial Measurement Unit)
  - I2C 버스에 연결되는 9축 관성 센서(BNO055) 내장
    - 사물에 가해진 특정한 힘, 각도 비율 및 사물을 둘러싼 자기장 측정
      - 충격 감지용 가속도계(Accelerometer) 3축
      - 회전 감지용 외전 속도계(Gyroscope) 3축
      - 자성 감지용 자력계(Magnetometer) 3축



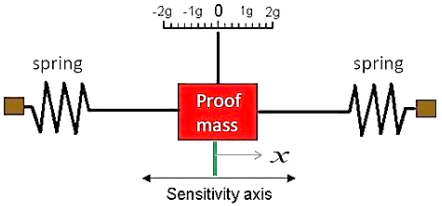
IMU 확장 모듈

- 관성 측정
  - 사물의 자세를 의미하는 각도는 롤(roll), 피치(pitch), 요(yaw)로 표현
    - 비행기나 배의 항법 장치, 로봇의 자세 제어 등에 적용
    - 중력 방향을 기준으로 롤과 피치 결정
      - 롤(좌,우): 배가 좌 또는 우로 기우는 방향
      - 피치(앞, 뒤): 배가 파도에 앞 또는 뒤로 치솟는 방향
      - 요: 배가 회전하는 방향
    - 가속도와 자이로 및 지자기 센서 이용
      - 가속도 : g (중력), m/s^2 (진동)
      - 자이로: degree/sec, rad/sec (라디안 각)
      - 지자기: t (테슬라)



IMU 확장 모듈

- 가속도 센서
  - 직선 방향으로 단위 시간에 대한 속도의 증감비를 나타내는 센서
    - 뉴턴의 제2 법칙(가속도 법칙): F = MA
    - 휴크의 법칙(용수철 법칙): F = kX
      - 변형 도가 작은 범위에서 탄성체의 응력과 변형도에 대한 공식
    - A = (kX) / M
      - k: 스프링 상수, M: 스프링에 달린 물체의 질량
      - 가속도는 스프링이 늘어난 길에 정비례
  - 3축 가속도 센서
    - 센서가 3차원에서 움직일 때 x, y, z축 방향의 가속도 측정

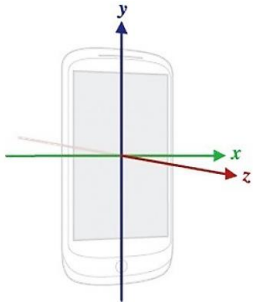


$$\vec{F} = m\vec{a} = k\vec{x}$$

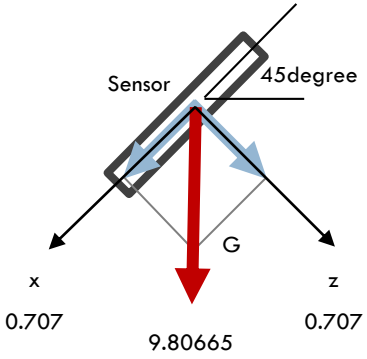
$$\vec{a} = \frac{k}{m}\vec{x}$$

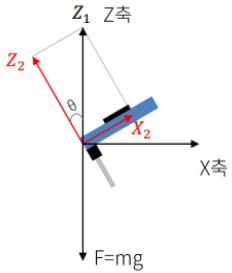
k : spring constant
m : mass of the proof mass
x : displacement

IMU 확장 모듈



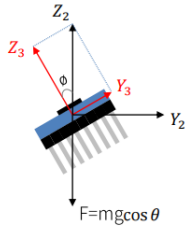
- 각도 계산 (또는 가속도 센서 보정)
  - 정지한 상태에서는 중력 가속도를 감지하므로 z축 방향으로 -g만큼의 값 출력
    - 중력 가속도는 9.80665 m/s^2
  - 센서의 초기 출력이 모두 0일 때, 물체를 y축 방향으로 45도 기울이면 z축 방향과 x축 방향으로 동일한 가속도가 측정됨
    - 중력 방향으로 g가 측정되어야 하므로 0.707g만큼 z축과 x축 방향으로 출력
  - 만약 정지된 상태에서 물체가 움직이기 시작하면 그때 측정되는 값은 기울기가 아님
    - 탄성체의 진동에 의해 출력값이 불규칙하게 변함
    - 충격 감지용으로 사용





$$Z_1 = -mg$$
$$\cos\theta = Z_2 / Z_1$$
$$Z_2 = Z_1 \times \cos\theta$$

$$\cos(90-\theta) = X_2 / Z_1$$
$$X_2 = Z_1 \times \cos(90-\theta)$$
$$X_2 = Z_1 \times \sin\theta$$



$$\cos\emptyset = Z_3 / Z_2$$
$$Z_3 = Z_2 \times \cos\emptyset$$
$$Z_3 = Z_1 \cos\theta \cos\emptyset$$

$$\cos(90-\emptyset) = Y_3 / Z_2$$
$$Y_3 = Z_2 \times \cos(90-\emptyset)$$
$$Y_3 = Z_2 \times \sin\emptyset$$
$$Y_3 = Z_1 \cos\theta \sin\emptyset$$

$$X_3 = X_2$$

$$Z_3^2 + Y_3^2 = m^2 g^2 \cos^2\theta \cos^2\emptyset + m^2 g^2 \cos^2\theta \sin^2\emptyset$$
$$Z_3^2 + Y_3^2 = m^2 g^2 \cos^2\theta (\cos^2\emptyset + \sin^2\emptyset)$$
$$Z_3^2 + Y_3^2 = m^2 g^2 \cos^2\theta$$

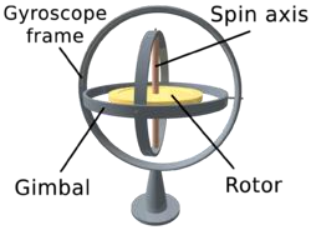
$$\sqrt{Z_3^2 + Y_3^2} = mg\cos\theta$$
$$X_3 = -mg\sin\theta$$

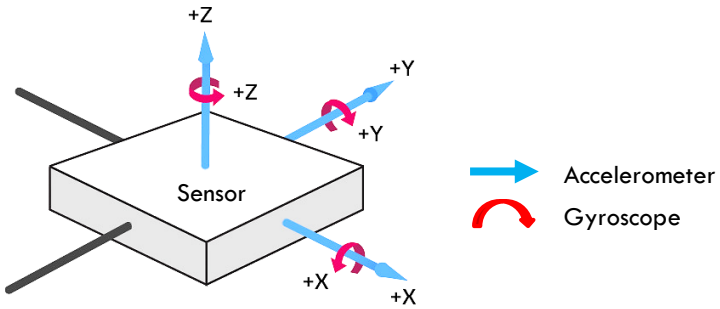
$$\frac{\sin\theta}{\cos\theta} = \tan\theta = \frac{-X_3}{\sqrt{Z_3^2 + Y_3^2}}$$

$$\therefore \ \theta = \tan^{-1}\frac{-X_3}{\sqrt{Z_3^2 + Y_3^2}}$$

IMU 확장 모듈

- 자이로 센서
  - 자이로 센서는 코리올리 힘(Coriolis Force)을 전기적 신호로 변환
    - 운동하는 물체의 속도에 비례하여 운동 방향으로 수직인 힘
    - 팽이가 회전할 때 회전축은 항상 지면과 수직 방향 유지
    - 자이로 센서는 이 축을 이용해 물체의 기울기 측정
  - 3축 자이로 센서
    - 센서가 3차원에서 움직일 때 x, y, z축 방향으로 시간당 회전하는 각속도를 이용해 각도 계산



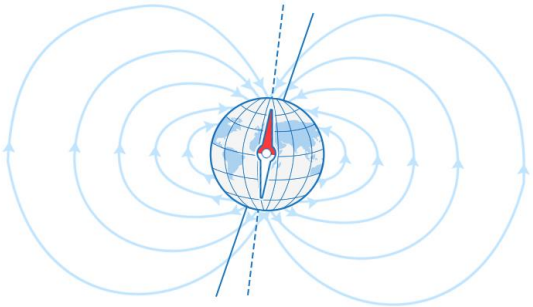


IMU 확장 모듈

- 각속도를 이용해 각도 계산
  - 정지한 상태에서 각속도는 0도/sec
  - 10초 동안 45도 기울어지면 평균 각속도는 4.5도/sec
  - 기울어진 후 멈춰서 45도를 유지하면 각속도는 0도/sec
  - 각속도에서 기울어진 각도를 계산하려면 전체 시간만큼 적분 필요
- 측정값은 잡음, 온도 변화 등으로 오류가 발생할 수 있음
  - 적분 과정에서 오류가 누적되면 시간이 지날수록 측정값의 오차도 커짐
  - 지자기 센서를 통해 보정 가능

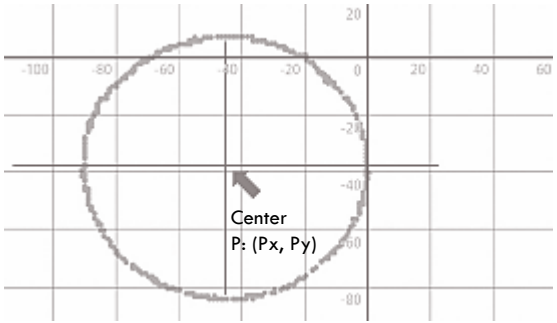
IMU 확장 모듈

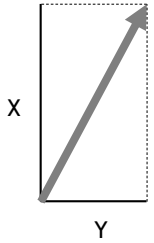
- 지자기 센서
  - 자기장은 방향과 크기를 갖는 벡터 물리량으로 움직이는 전하(전류) 또는 전자의 스핀 현상에 의해 발생
    - 자기 저항 효과 또는 홀 효과에 의해 변하는 물질의 전기 전도성 이용
      - 자기 저항 효과: 물체에 자기장이 인가되면 물체의 저항값이 변함
      - 홀 효과: 자기장 속에 도체를 놓고 자기장에 직각방향으로 전류를 흘리면 자기장과 전류에 수직 방향으로 전위차 발생
  - 3축 지자기 센서
    - 센서가 3차원에서 움직일 때 x, y, z축 방향으로 자기장의 크기 또는 자기력선의 크기와 방향 측정
    - 자기장 (Earth's magnetic field) 자체를 검출하는 용도보다는 자기장의 변화에 초점
      - 자이로 센서에게 자기장의 변화를 제공해 오차를 보정하도록 함



IMU 확장 모듈

- 자북에서의 각도 계산
  - 자북은 지자기 센서가 인식한 북쪽으로 북극점에서 조금 어긋나 있음
  - 기울기를 고려하지 않으면 X축, Y축만 사용 (수평 회전)
    - 주위 자장의 영향이 없는 이상적인 경우, 출력 분포도의 원 중심(Px, Py)은 0
    - 실제 주위 자장의 영향으로 중심이 어긋나기 때문에 원의 중심을 0으로 이동시켜야 함
    - angle = tan^-1 * (Y - Py) / (X - Px)
  - 지자기 센서를 기울이면 가속도 센서의 3축 값과 조합해 정확한 방위로 보정해야 함
  - 참조: https://thecavepearlproject.org/2015/05/22/calibrating-any-compass-or-accelerometer-for-Arduino

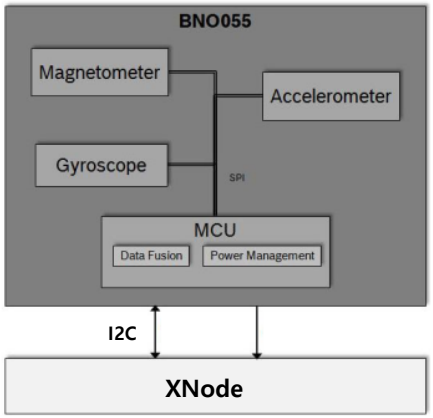


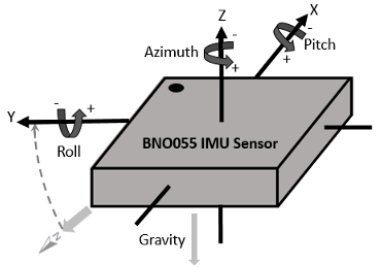


$$angle = \tan^{-1}\frac{Y - Py}{X - Px}$$

IMU 확장 모듈

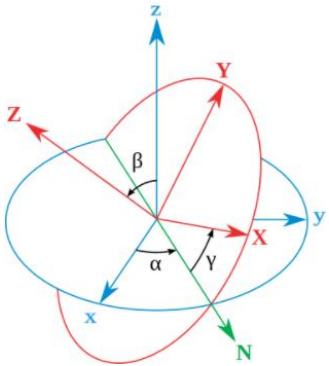
- BNO055
  - 3축 14bit 가속도와 3축 16bit 자이로 및 3축 지자기 센서로 구성
  - 3가지 센서를 묶어 퓨전(Fusion) 모드로 운영 가능
    - 선형 가속도, 중력(Gravity) 벡터, 오일러 각도, 쿼터니언 계산 결과 제공





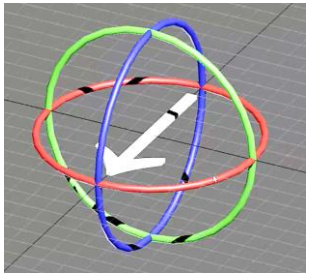
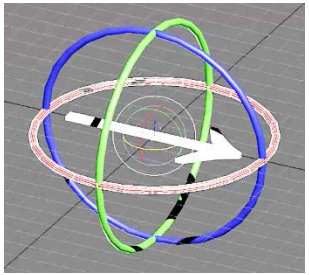
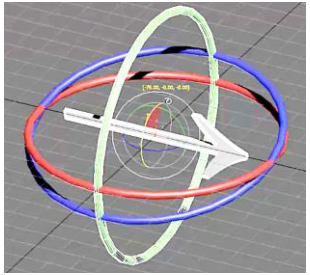
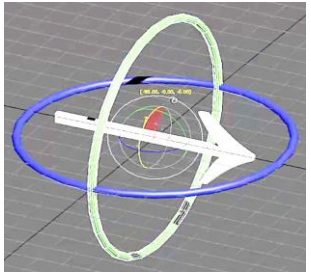
IMU 확장 모듈

- 오일러 각(Euler angle)
  - 형태가 고정되어 변하지 않는 물체(강체)의 방향을 3차원 공간 좌표계의 회전으로 변환
    - 세 축을 중심으로 하는 순차적 회전을 통한 매우 직관적인 표현법
    - 보간과 반복에 적합하지 않아 주로 결과의 신속한 시각화를 위해 사용
  - 주어진 3차원 공간 좌표계(x, y, z)와 이를 회전시킨 좌표계(X, Y, Z) 사이 강체 방향 (α, β, γ)
    - α (또는 ψ): z-축을 회전축으로 하여 회전된 x-y 좌표축의 각도
    - β (또는 θ): x-축을 회전축으로 하여 회전된 z-y 좌표축의 각도
    - γ (또는 φ): z-축을 회전축으로 하여 회전된 x-y 좌표축의 각도
  - 오일러 각의 범위
    - α, γ: 이상적인 상황에서 2π 라디안까지
    - β: -π/2 ~ π/2라디안으로 제한되며 이를 짐벌 록(gimbal lock)이라 함
      - 앞서 회전한 두 축의 영향으로 세 번째 회전의 가동 범위가 줄어들기 때문



IMU 확장 모듈

- 오일러 각의 짐벌 락 문제
  - x, y, z 축을 가진 객체가 존재할 때,
  - x축을 90도 회전
  - y축으로 90도 회전
  - x축과 z축이 겹쳐 한 축에 대해서는 계산 불가능 상태 발생
    - 3개의 축을 동시에 계산하지 않고 각 축을 독립적으로 판단하기 때문

IMU 확장 모듈

- 쿼터니언(Quaternion) 회전
  - 확장된 복소수 체계를 이용해 3차원 회전 표현
    - $q = w + xi + yi + zk$ ($i: \sqrt{-1}$ $w: real\ part$ $i, j, k: imaginary\ units$ $x, y, z: imaginary\ components$)
    - $q = w + v$ ($w: scalar, v: vector$)
  - 오일러 각의 짐벌 락 문제를 보완하기 위해 각 축을 한꺼번에 계산
    - 행렬에 비해 연산 속도가 빠르고 메모리 소모도 적음
    - 최단 호(shortest arc) 보간으로 오류 발생이 적음
  - 쿼터니언 각(qw, qx, qy, qx)을 오일러 각(Φ,θ, Ψ)으로 변환하면 오일러 3차원 공간과 호환됨. 역도 성립

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2)) \\ \text{asin}(2(q_0 q_2 - q_3 q_1)) \\ \text{atan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2)) \end{bmatrix} \qquad q = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} cos(\phi/2)cos(\theta/2)cos(\psi/2) + sin(\phi/2)sin(\theta/2)sin(\psi/2) \\ sin(\phi/2)cos(\theta/2)cos(\psi/2) - cos(\phi/2)sin(\theta/2)sin(\psi/2) \\ cos(\phi/2)sin(\theta/2)cos(\psi/2) + sin(\phi/2)cos(\theta/2)sin(\psi/2) \\ cos(\phi/2)cos(\theta/2)sin(\psi/2) + sin(\phi/2)sin(\theta/2)cos(\psi/2) \end{bmatrix}$$

IMU 확장 모듈 제어

- 준비물

준비물	
1	PC
2	XNode 1ea
3	Micro type USB cable 1ea
4	XNode 제공 USB 메모리 (D: 로 가정)
5	IMU 모듈

```
PS C:\XNode\IMU> xnode -p com3 ls /flash/lib
/flash/lib/IMU.py
/flash/lib/core_a.py
/flash/lib/core_b.py
/flash/lib/pop.py
```

  - XNode와 PC를 USB 케이블로 연결
  - D:₩Library₩IMU를 작업 폴더인 C:₩XNode로 복사
    - C:₩XNode₩IMU
  - VS Code를 실행한 후 C:₩XNode₩IMU 아래 lib 폴더를 XNode에 복사
    - xnode -p <포트> put lib

IMU 확장 모듈 제어

- X-Node IMU를 위한 Pop 라이브러리
  - IMU class
    - IMU(): IMU 객체 생성
    - IMU.accel(): 가속도 센서값 반환
      - 반환 값은 x, y, z 순 튜플. 단위는 중력 가속도인 -9.8 ~ 9.8 m/s^2
    - IMU.lineraccel(): 축을 따라 중력을 제외한 (각 축 방향의 움직임 중심) 가속도 값 반환.
      - 반환 값은 x, y, z 순 튜플. 단위는 가속도와 같음
    - IMU.gravity(): 가속도 센서 값을 중력 중심으로 보정해 좀 더 완만한 가속도 값 반환
      - 반환 값은 x, y, z 순 튜플. 단위는 가속도와 같음
    - IMU.gyro(): 자이로 센서값 반환
      - 반환 값은 x(pitch), y(roll), z(yaw) 순 튜플. 단위는 rad/sec
    - IMU.magnetic(): 지자기 센서값(자기장) 반환
      - 반환 값은 x, y, z 순 튜플. 단위는 uT

IMU 확장 모듈 제어

- IMU.euler(): 오일러 각도로 센서 방향 반환
  - 반환 값은 azimuth, roll, pitch 순 튜플
    - azimuth: 0~360
    - roll: -90~90
    - pitch: -180~180
- IMU.quat(): 센서 방향에 대한 쿼터니언 배열 반환
  - 반환 값은 w, x, y, z 순 튜플로 -1.0 ~ 1.0 사잇값
- IMU.calibration(): 각 센서의 보정 상태 반환
  - 반환 값은 sys, gyro, accel, mag 순 튜플
  - 각 값의 범위는 0 ~ 3으로 3이면 보정 완료. 0이면 보정 전
    - sys >= 1이면 사용 가능

IMU 확장 모듈 제어

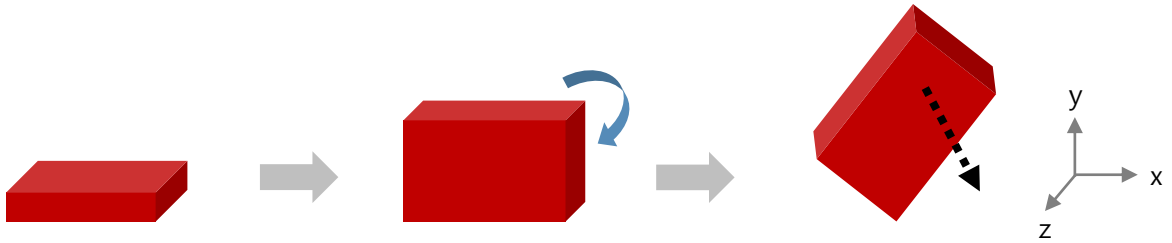
- IMU 제어
  - IMU 센서 보정 결과 확인

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     sys, gyro, accel, mag = imu.calibration()
08:     print("SGAM: %d %d %d %d"%(sys, gyro, accel, mag))
09:     if sys == gyro == accel == mag == 3:
10:         break
11:     time.sleep(1)
```

```
PS C:\XNode\IMU> xnode -p com3 run ext_imu_calibration.py
SGAM: 0 0 0 0
SGAM: 0 3 0 0
SGAM: 0 3 0 0
SGAM: 0 3 0 0
SGAM: 1 3 0 0
        :
        :
SGAM: 3 3 1 3
SGAM: 3 3 1 3
SGAM: 3 3 1 3
SGAM: 3 3 3 3
```

IMU 확장 모듈 제어

- 테스트 방법
  - 기본적으로 시스템과 지자기만 보정되면 모든 측정 결과는 유효함
  - XNode를 평지에 두면 즉시 자이로 센서가 보정됨
  - 이후 x축을 90도 돌려 XNode를 세운 후 천천히 z축을 회전시키면 시스템과 지자기 센서가 보정됨
    - 가볍게 흔들어도 됨
  - 이후 평지에서 x, y, z 축을 45도 기울여 두면 가속도 센서가 보정됨. (시간이 오래 걸리며, 생략 가능)
    - IMU 확장 모듈의 모서리를 평지에 세우고 천천히 앞으로 눕히면서 45도에 맞춤



IMU 확장 모듈 제어

- 센서 모니터(xmon)를 이용해 가속도, 선형 가속도, 그래비티 값 시각화
  - 센서 모니터에서 값을 읽을 수 있도록 쉼표(,) 또는 공백으로 항목을 구분하며 줄 단위로 출력

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     acc = imu.accel()
08:     lin = imu.lineraccel()
09:     gra = imu.gravity()
10:
11:     print("%.2f, %.2f, %.2f, "%(acc), end='')
12:     print("%.2f, %.2f, %.2f, "%(lin), end='')
13:     print("%.2f, %.2f, %.2f"%(gra))
14:
15:     time.sleep(.1)
```

IMU 확장 모듈 제어

- (계속)
  - 시리얼을 통해 가속도, 선형 가속도, 그래비티 값이 출력되는지 확인

```
PS C:\XNode\IMU> xnode -p com3 run ext_imu_accel_total.py
-0.09, 0.30, 9.27, 0.00, -0.01, -0.53, -0.08, 0.32, 9.80
-0.07, 0.30, 9.27, 0.01, -0.01, -0.54, -0.08, 0.32, 9.80
-0.06, 0.30, 9.28, 0.00, -0.02, -0.53, -0.08, 0.32, 9.80
-0.09, 0.27, 9.24, 0.00, 0.00, -0.53, -0.08, 0.32, 9.80
-0.06, 0.31, 9.24, 0.00, -0.01, -0.51, -0.08, 0.32, 9.80
-0.08, 0.31, 9.26, 0.00, -0.01, -0.53, -0.08, 0.32, 9.80
-0.08, 0.28, 9.26, 0.00, -0.02, -0.51, -0.08, 0.32, 9.80
```

  - 코드를 실행할 때 센서 모니터에서 시리얼을 읽을 수 있도록 -n 옵션을 추가해 시리얼 출력 제거

```
PS C:\XNode\IMU> xnode -p com3 run -n ext_imu_accel_total.py
PS C:\XNode\IMU>
```

  - 센서 모니터 실행

```
PS C:\XNode\IMU> xmon
PS C:\XNode\IMU>
```

IMU 확장 모듈 제어

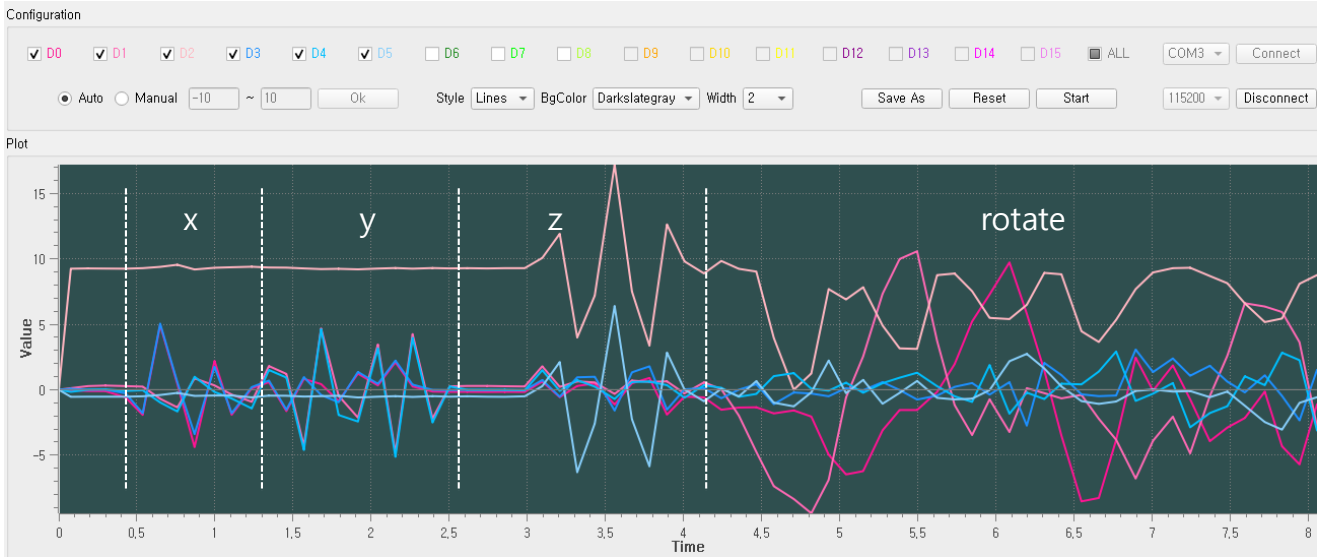
- (계속)
  - 포트(예: COM 3)를 선택한 후 'Connect' 버튼을 눌러 가속도 센서 데이터 시각화 시작
    - 가속도(D1(x), D2(y), D2(z)), 선형 가속도(D3(x), D4(y), D5(z)), 그래비티(D6(x), D7(y), D7(x)) 순

IMU 확장 모듈 제어

- (계속)
  - D3, D4, D5 항목 선택을 해제한 후 가속도와 그래비티 비교
    - 그래비티의 변화가 가속도에 비해 부드럽게 표현됨

IMU 확장 모듈 제어

- (계속)
  - D6, D7, D8 항목을 해제(D3, D4, D5 항목 선택)한 후 가속도와 선형 가속도 비교
    - 각 축 방향의 움직임은 유사한 결과를 갖지만, 회전할 땐 선형 가속도의 변화가 줄어듦



IMU 확장 모듈 제어

- 가속도 센서로 3축(x, y, z)의 움직임이 유효 범위를 벗어났는지 검사

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: x_gain = 0.6; y_gain = 0.9; z_gain = 0.6
07:
08: x_base, y_base, z_base = imu.accel()
09: print (x_base, y_base, z_base)
10:
11: while True:
12:     x, y, z = imu.accel()
13:
14:     if x < x_base - x_gain or x > x_base + x_gain:
15:         print("Detect X: %.2f"%(x))
16:
```

```
PS C:\XNode\IMU> xnode -p com3 run ext_imu_accel.py
-0.37 0.46 9.26
Detect X: 0.37
Detect X: 0.37
Detect Y: -0.64
Detect X: 0.27
Detect X: 0.35
Detect Z: 10.32
Detect Z: 8.64
```

IMU 확장 모듈 제어

- (계속)

```
17:     if y < y_base - y_gain or y > y_base + y_gain:
18:         print("Detect Y: %.2f"%(y))
19:
20:     if z < z_base - z_gain or z > z_base + z_gain:
21:         print("Detect Z: %.2f"%(z))
22:
23:     time.sleep(.1)
```

IMU 확장 모듈 제어

- RMS(Root Mean Square)를 적용해 가속도 센서를 충격 센서로 활용
  - RMS = $\sqrt{\left(\frac{1}{n}\right)\sum_{k=1}^{n} x_k^2}$ 일 때, 움직임이 없는 곳에서 일정 횟수 동안 RMS를 계산해 기준값 지정
  - 이후 기준값과 현재 x, y, z의 RMS를 비교해 충격 단계 판단

```
01: from pop import IMU
02: from pop import time
03: from pop import abs, sqrt
04:
05: imu = IMU()
06:
07: low_gain = 0.3
08: mid_gain = 1.5
09: high_gain = 7.0
10:
11: x = y = z = total = 0
12:
```

```
PS C:\XNode\IMU> xnode -p com3 run ext_imu_accel_shock.py
AVERAGE: 9.29
............
MID(5.26, 4.03)..............
LOW(9.99, 0.70)...................
MID(7.48, 1.81)..
LOW(8.63, 0.66)........
LOW(8.43, 0.86).
HIGH(21.69, 12.40)
MID(4.36, 4.93).................
```

IMU 확장 모듈 제어

- (계속)

```
13: for _ in range(10):
14:     x, y, z = imu.accel()
15:     total += x**2 + y**2 + z**2
16:     time.sleep(.1)
17:
18: base = sqrt(total / 10)
19: print("AVERAGE: %.2f"%(base))
20:
21: while True:
22:     x, y, z = imu.accel()
23:
24:     curr = sqrt(x**2 + y**2 + z**2)
25:     diff = abs(curr - base)
26:     level = None
27:
```

IMU 확장 모듈 제어

- (계속)

```
28:     if diff > high_gain:
29:         level = "HIGH"
30:     elif diff > mid_gain:
31:         level = "MID"
32:     elif diff > low_gain:
33:         level = "LOW"
34:
35:     if level:
36:         print("₩n%s(%.2f, %.2f)"%(level, curr, diff), end='')
37:         time.sleep(.5)
38:     else:
39:         print(".", end='')
40:         time.sleep(.1)
```

IMU 확장 모듈 제어

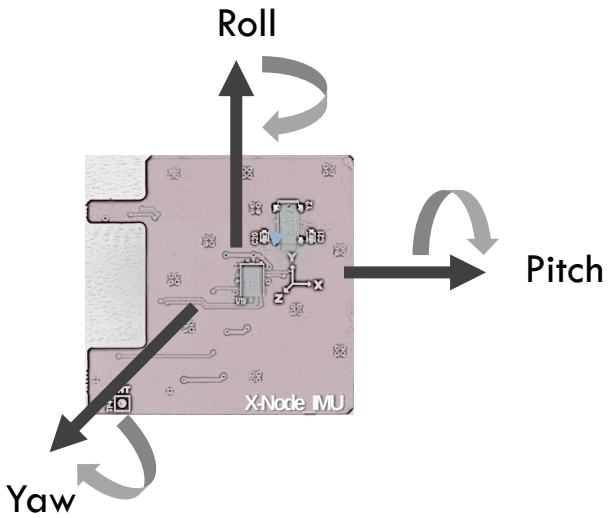
- 자이로 센서의 움직임으로 XNode의 롤, 피치, 요 판별

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     x, y, z = imu.gyro()
08:
09:     if x*x > 1:
10:         print("Pitch: %.2f"%(x))
11:
12:     if y*y > 1:
13:         print("Roll: %.2f"%(y))
14:
15:     if z*z > 1:
16:         print("Yaw: %.2f"%(z))
17:
18:     time.sleep(.1)
```

```
PS C:\XNode\IMU> xnode -p com3 run ext_imu_gyro.py
Pitch   (1.01, 1.03)
Pitch   (1.48, 2.18)
Pitch   (1.09, 1.19)
Pitch   (-1.53, 2.34)
Pitch   (-1.20, 1.44)
Roll    (1.01, 1.02)
Roll    (-2.14, 4.56)
Roll    (-1.52, 2.30)
Roll    (1.30, 1.69)
Roll    (1.85, 3.44)
Yaw     (2.15, 4.60)
Yaw     (1.65, 2.72)
Yaw     (2.03, 4.12)
```

IMU 확장 모듈 제어

- (계속)
  - 이해를 돕기 위해 처음은 평평한 바닥에 놓고 테스트할 것
  - XNode를 바닥에 놓을 때 Y축이 앞쪽. 따라서 Y축 회전을 롤, X 축 회전은 피치, Y축 회전은 요



IMU 확장 모듈 제어

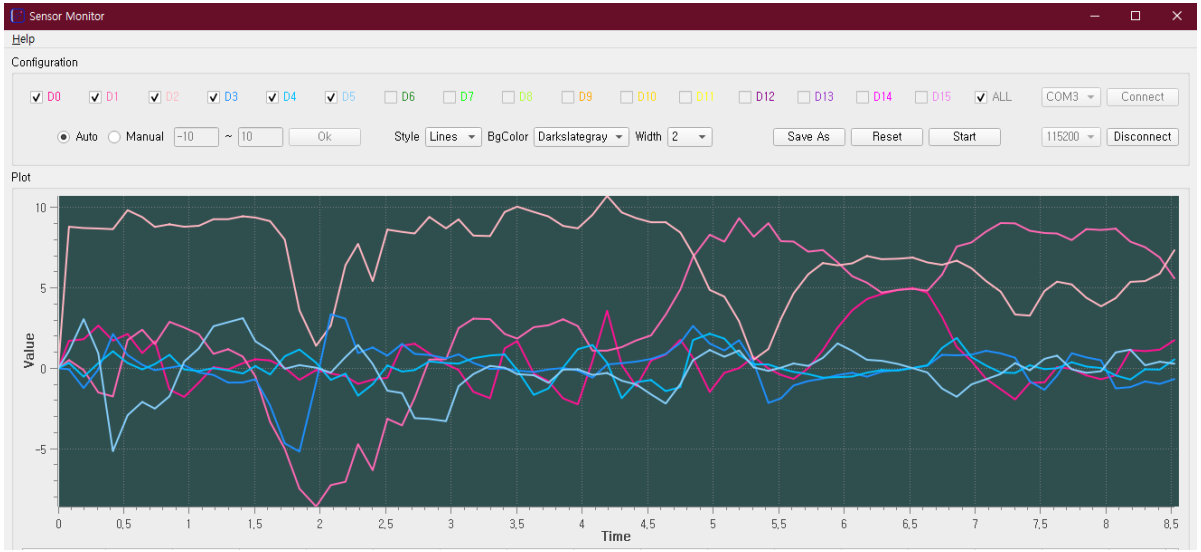
- 센서 모니터(xmon)를 이용해 가속도와 자이로 값 시각화
  - 가속도는 해당 축 방향의 움직임(또는 충격) 파악에 적합하고 자이로는 해당 축 기준 회전 파악에 적합함

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     acc = imu.accel()
08:     gyr = imu.gyro()
09:
10:     print("%.2f, %.2f, %.2f, "%(acc), end='')
11:     print("%.2f, %.2f, %.2f"%(gyr))
12:
13:     time.sleep(.1)
```

IMU 확장 모듈 제어

- (계속)
  - -n 옵션을 추가해 프로그램을 실행한 후 센서 모니터 실행

```
PS C:\XNode\IMU> xnode -p com3 run -n ext_imu_accel_gyro.py
PS C:\XNode\IMU> xmon
```



IMU 확장 모듈 제어

- (계속)
  - 롤, 피치, 요 동작에 따른 가속도, 자이로 값 비교. 요 동작은 가속도로는 파악하기 어려움

IMU 확장 모듈 제어

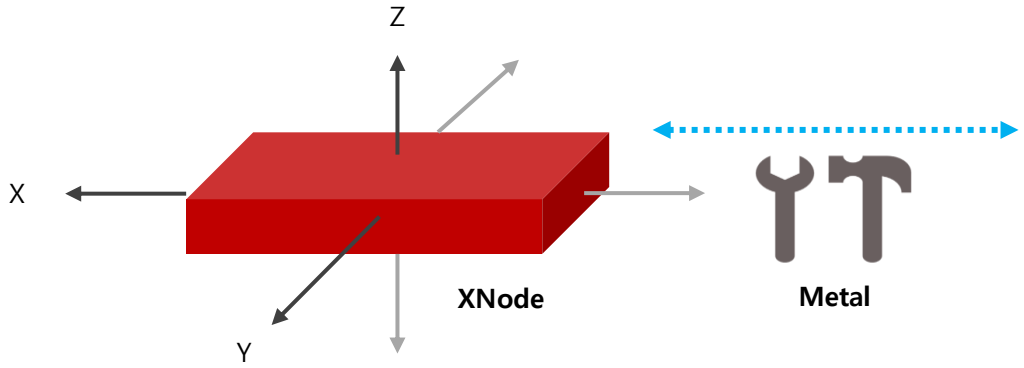
- 지자기 센서로 주변 금속(자성체) 탐지

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     while True:
08:         _, _, _, mag = imu.calibration()
09:         if mag == 3:
10:             break
11:         else:
12:             print("*", end='')
13:         time.sleep(.1)
14:
15:     x, y, z = imu.magnetic()
16:     print("₩nMeasure: %03.2f, %03.2f, %03.2f"%(x, y, z), end='')
17:     time.sleep(.1)
```

```
PS C:\XNode\IMU> xnode -p com3 run ext_imu_magnetic.py
**********************************************************
Measure: 13.50, -16.69, -12.00
Measure: 20.06, -11.25, -9.75
Measure: -7.75, -24.00, -4.56
Measure: 3.25, -21.00, -16.69
Measure: 1.25, -15.75, -28.75
Measure: -6.50, -6.75, -35.00
```

IMU 확장 모듈 제어

- (계속)
  - XNode를 흔들면 지자기 센서가 보정됨
  - XNode를 평평한 바닥에 놓고 자성체를 띄는 금속을 x, y, z 방향으로 가깝게 근접시킴
  - 자성체가 지자기 센서에 너무 가깝게 접근하면 센서 보정 시간이 필요함



IMU 확장 모듈 제어

- 오일러 각으로 롤, 피치, 요(아지무스) 표시
  - 가속도, 자이로, 지자기 센서 결과를 융합해 좀 더 신뢰성 있는 결과물 도출.
    - 요(아지무스) 값은 360도 나침반으로 활용 가능

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: old_azimuth = old_roll = old_pitch = 0
07:
08: while True:
09:     azimuth, roll, pitch = imu.euler()
10:
11:     azimuth = int(azimuth)
12:     roll = int(roll)
13:     pitch = int(pitch)
14:     update = 0
```

```
PS C:\XNode\IMU> xnode -p com3 run ext_imu_euler.py
000 -2 00
000 -3 -1
359 -3 -3
357 -3 -7
359 -3 -2
358 -2 02
358 -2 04
358 -2 01
358 -2 00
354 09 00
354 18 -1
353 16 00
```

IMU 확장 모듈 제어

- (계속)

```
15:
16:     if old_azimuth != azimuth:
17:         old_azimuth = azimuth
18:         update += 1
19:
20:     if old_roll != roll:
21:         old_roll = roll
22:         update += 1
23:
24:     if old_pitch != pitch:
25:         old_pitch = pitch
26:         update += 1
27:
28:     if update:
29:         print("%02d, %02d, %02d"%(azimuth, roll, pitch))
30:
31:     time.sleep(.1)
```

IMU 확장 모듈 제어

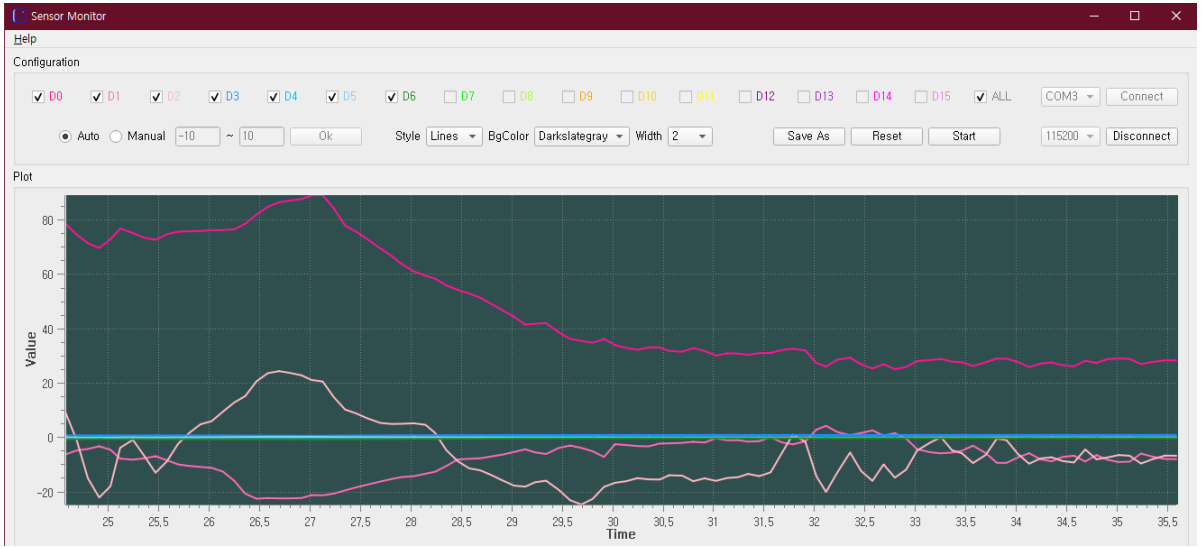
- 센서 모니터(xmon)를 이용해 오일러 각과 쿼터니언 값 시각화

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     euler = imu.euler()
08:     quat = imu.quat()
09:
10:     print("%.2f, %.2f, %.2f, "%(euler), end='')
11:     print("%.2f, %.2f, %.2f %.2f"%(quat))
12:
13:     time.sleep(.1)
```

IMU 확장 모듈 제어

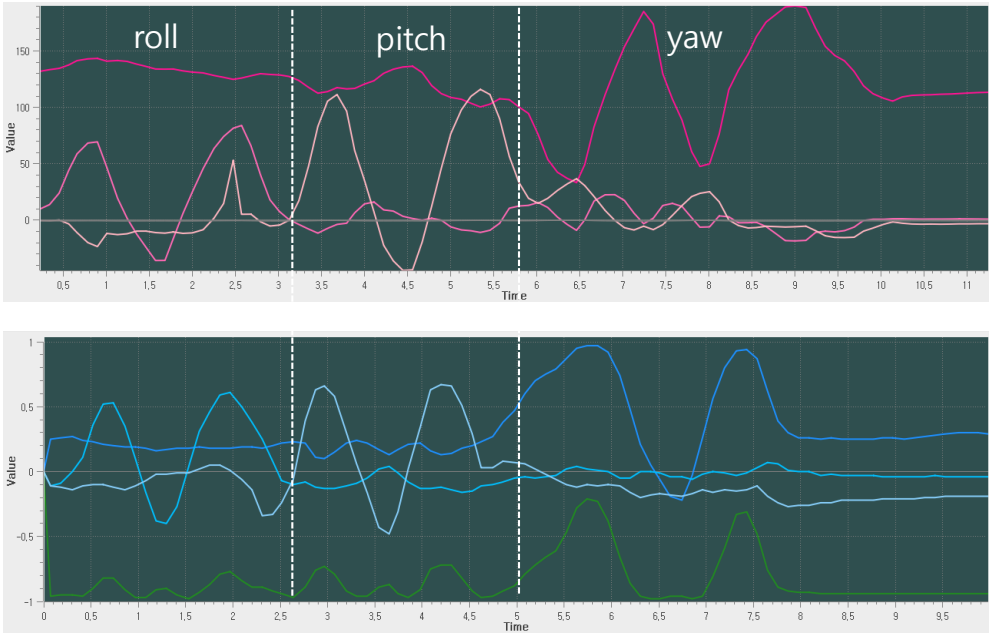
- (계속)
  - -n 옵션을 추가해 프로그램을 실행한 후 센서 모니터 실행

```
PS C:\XNode\IMU> xnode -p com3 run -n ext_imu_euler_quat.py
PS C:\XNode\IMU> xmon
```



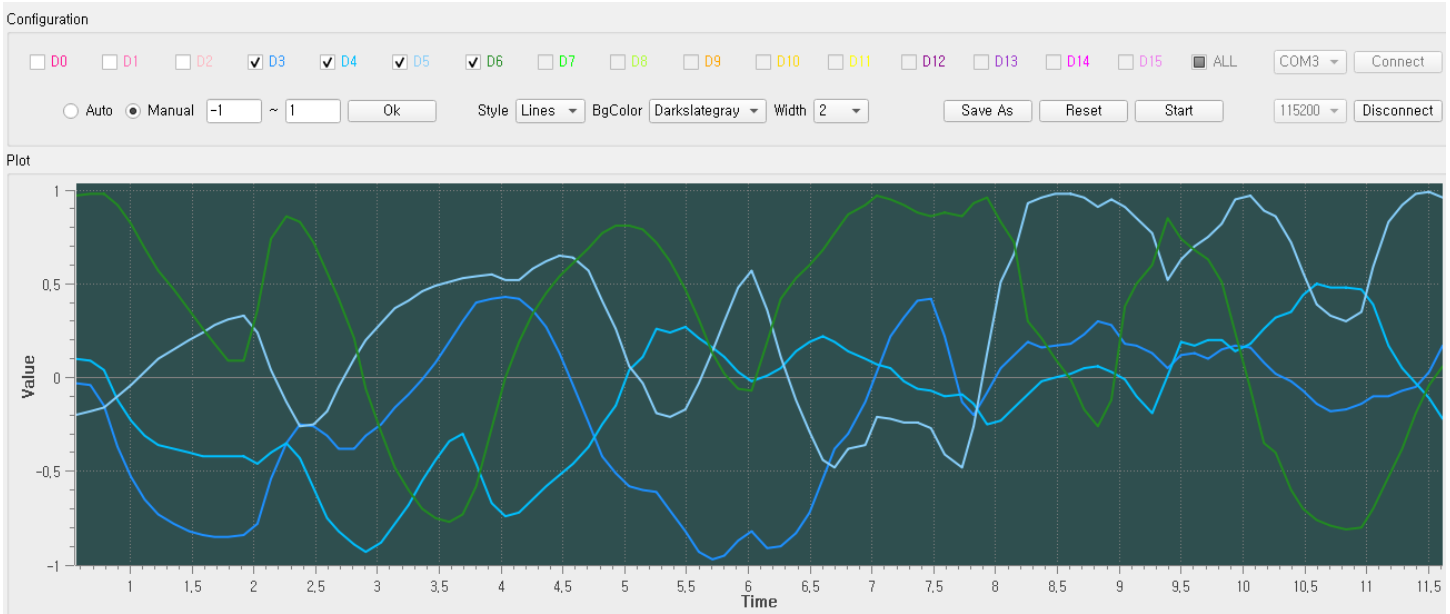
IMU 확장 모듈 제어

- (계속)
  - 오일러 값은 자동, 쿼터니언 값은 수동으로 -1.0 ~ 1.0 사이 범위를 지정해 파악



IMU 확장 모듈 제어

- (계속)
  - 회전 상태 표현은 짐벌 락 문제가 없는 쿼터니언이 유리함



IMU 확장 모듈 제어

- 3차원 그래픽 시뮬레이터(xquat)로 쿼터니언 값 시각화
  - XNode에 코드를 옮긴 후 테스트 권장

```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     w, x, y, z = imu.quat()
08:
09:     print("%f, %f, %f, %f"%(w, x, y, z))
10:     time.sleep(.1)
```

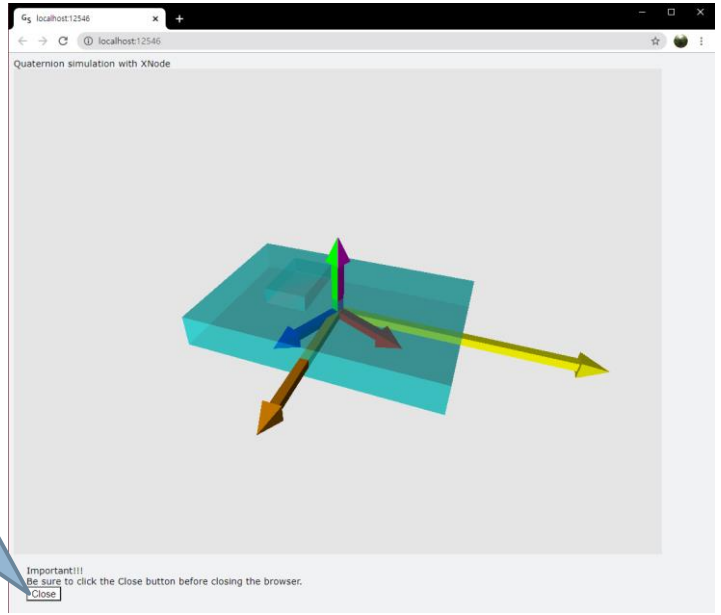
IMU 확장 모듈 제어

- (계속)
  - 3차원 그래픽 시뮬레이터 실행하면 크롬 또는 엣지 브라우저(WebGL 사용)가 실행됨

```
PS C:\XNode\IMU> xnode -p com3 put ext_imu_quat.py main.py
```

XNode로 옮긴 코드를 실행하기 위해
약 2초간 XNode의 리셋 버튼을 누를 것

```
PS C:\XNode\IMU> xquat com3
```

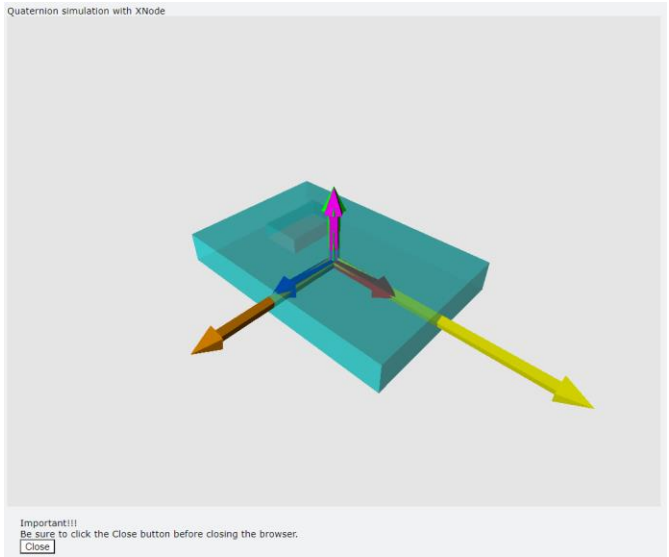


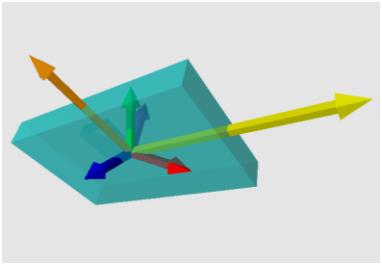
웹브라우저를 닫기 전에 Close
버튼을 눌러 시리얼 연결을 끊을 것!

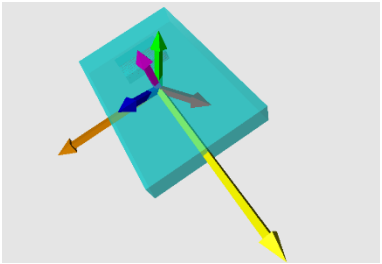
만약 그냥 웹브라우저를 닫았으면
XNode의 USB 케이블을 분리했다
다시 연결 할 것.

IMU 확장 모듈 제어

- (계속)
  - XNode를 움직이면 쿼터니언 값을 기반으로 시뮬레이터에 상태가 표시됨
    - 필요에 따라 오른쪽 마우스 버튼을 누른 채 마우스를 움직이면 기준 위치를 변경할 수 있음







IMU 확장 모듈 제어

- IMU 전체 데이터 시각화를 통해 응용에 필요한 각 값의 의미 파악

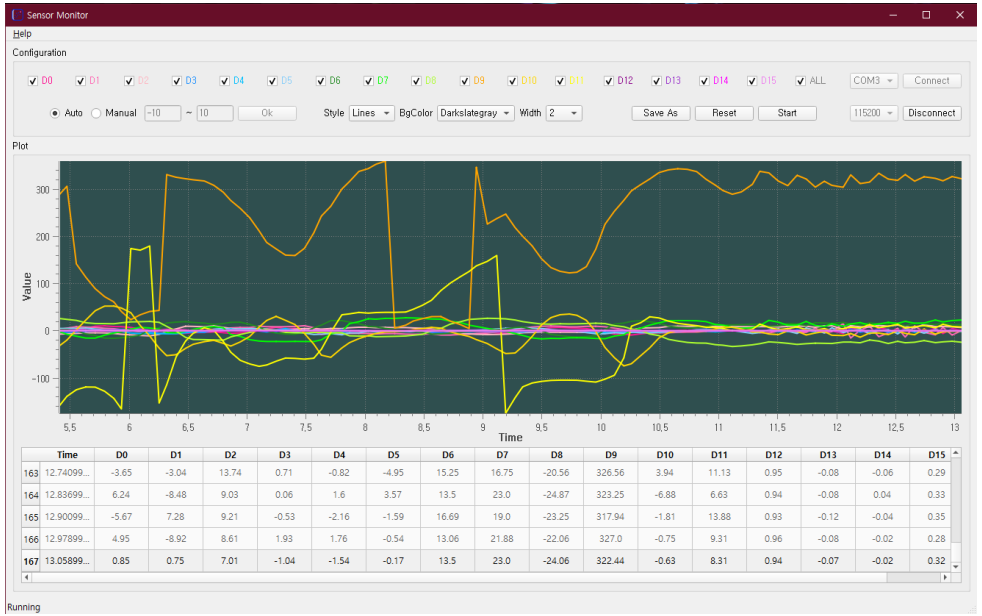
```
01: from pop import IMU
02: from pop import time
03:
04: imu = IMU()
05:
06: while True:
07:     ac = imu.accel()
08:     gy = imu.gyro()
09:     ma = imu.magnetic()
10:     eu = imu.euler()
11:     qu = imu.quat()
12:
13:     for data in ac, gy, ma, eu:
14:         print(("%.2f," * 3)%(data), end='')
15:     print(("%.2f," * 4)%(qu))
16:
17:     time.sleep(0.05)
```

IMU 확장 모듈 제어

- (계속)
  - 코드를 XNode로 옮긴 후 XNode 리셋 및 센서 모니터 실행

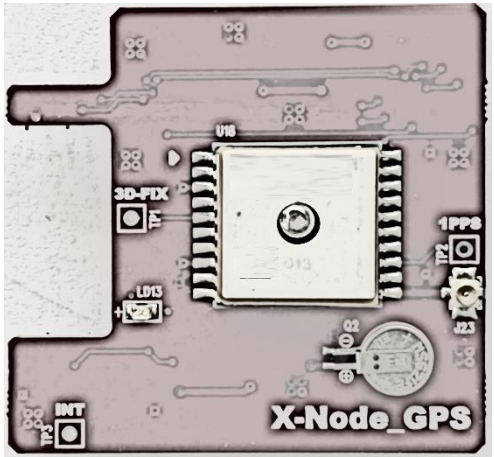
```
PS C:\XNode\IMU> xnode -p com3 put ext_imu_total.py main.py
PS C:\XNode\IMU> xmon
```

  - 특정 항목만 선택한 후 변화 파악



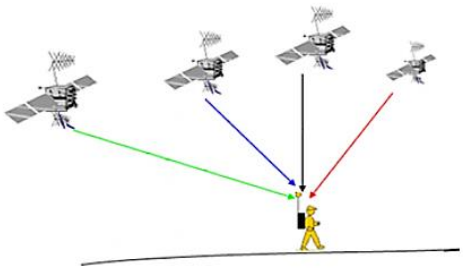
GPS 확장 모듈

- X-Node GPS
  - I2C 버스에 연결되는 GPS 센서(MT3339) 내장
    - 안테나 내장형으로 GPS 위성으로부터 수신한 신호 출력
      - 충전 가능한 배터리 내장으로 최대 30일 동안 GPS에 전원 공급
        - 위성 궤도예측 알고리즘 내장으로 빠른 위치 수신 가능
        - 외부에서 수 십 초에서 수 분 이내 위성 신호 수신 가능
      - XNode에 전원이 공급되면 내장 배터리도 충전됨
        - 내장 배터리가 방전되면 위성 신호를 수신하는데 30분 이상 걸릴 수 있음



GPS 확장 모듈

- GNSS (Global Navigation Satellite System)
  - 지구 항해 목적으로 신호를 전송하는 위성의 글로벌 시스템
    - 전 세계 어느 곳에서든지 인공위성을 이용해 자신의 위치(시간, 위도, 경도, 해발 고도) 파악 가능
      - 오차는 3 ~ 10m
    - GPS (미국), GLONASS (러시아), Galileo (EU), Beidou (중국)가 있음
      - QZSS (일본), IRNSS (인도)는 자국 근처 지역에 한정



GPS 확장 모듈

- NMEA (National Marine Electronics Association) 0183
  - 항법 수신기로부터 수신한 데이터를 응용에 전달하기 위한 표준 프로토콜
    - 물리 계층: RS232, RS-422 등 전기적인 전송 규격 정의
    - 데이터링크 계층: 전송 속도, data bit, parity bit, stop bit 등 정의
    - 응용 계층: GNSS(GPS 포함), Loran, Omega, Transit 등을 위한 문장 형식 정의
      - 8bit 아스키 문자 사용
      - '$'로 시작하며 첫 2개 문자는 제품 종류로 GPS는 'GP', 수심 측정 장비인 Depth Sounder인 경우 'SD'
      - 다음 3개 문자는 데이터 종류로 GPS는 'GGA', 'GSA', 'GSV', 'RMC', 'VTG' 중 하나
      - ','로 항목을 구분하고 '*' 문자 다음에 체크썸을 포함해 문장이 끝남
      - 체크섬은 '$'와 '*' 사이 모든 항목을 XOR 하여 계산
      - 문장 마지막에 응용프로그램이 줄 바꿈을 인식하도록 <CR><LF>가 추가됨

GPS 확장 모듈

- 응용 계층의 GPS 주요 문장 형식
  - GGA (Global Positioning System Fix Data)
    - $GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65

Name	Example	Units	Description
Message ID	$GPGGA		GGA protocol header
UTC Time	064951.000		UTC 시간 (hhmmss.sss) 9시간 더하면 한국 시간
Latitude	2307.1256		위도 (ddmm.mmmm). 도(dd), 분(mm.mmmm)
N/S Indicator	N		N=북(적도 북쪽), S=남(적도 남쪽)
Longitude	12016.4438		경로 (dddmm.mmmm). 도(ddd), 분(mm.mmmm)
E/W Indicator	E		E=동경, W=서경
Position Fix Indicator	1		0=위성 인식 안됨, 1=위성 데이터, 2=DGPS이용 보정 데이터
Satellites Used	8		위성 수. 0 ~ 14
HDOP	0.95		2차원적 오차 결정(수평 방향)
MSL Altitude	39.9	meters	해수면 기준 고도
Units	M		단위. meters
Geoidal Separation	17.8	meters	타원체로 모델링한 지구와 구체로 모델링된 지구의 고도 차이
Units	M		단위. meters
Age of Diff. Corr		second	DGPS 사용시 마지막 업데이트 시간과 기지국 ID
Checksum	65		

GPS 확장 모듈

- RMC (Recommended Minimum Navigation Information)
  - 권장되는 최소한의 GNSS 데이터
  - $GPRMC,064951.000,A,2307.1256,N,12016.4438,E,0.03,165.48,260406,3.05,W,A*2C

Name	Example	Units	Description
Message ID	$GPRMC		RMC 헤더
UTC Time	064951.000		UTC 시간 (hhmmss.sss)
Status	A		상태. A=신뢰 함, V=신뢰할 수 없음
Latitude	2307.1256		위도 (ddmm.mmmm)
N/S Indicator	N		N=북, S=남
Longitude	12016.4438		경도 (dddmm.mmmm)
E/W Indicator	E		E=동, W=서
Speed over Ground	0.03		노트(knote) 단위 속도 (km/h 변환시 약 1.852을 곱함)
Course over Ground	165.48		진북 기준 수평경로 (시계방향 360도 방위각)
Date	260406		날짜 (ddmmyy)
Magnetic Variantion	3.05		자기 값 (제조업체 서비스로 생략될 수 있음)
E/W Indicator	W		자기 방향. E=east, W=west
Mode	A		A=자동, D=차동, E=예상
Checksum	2C		

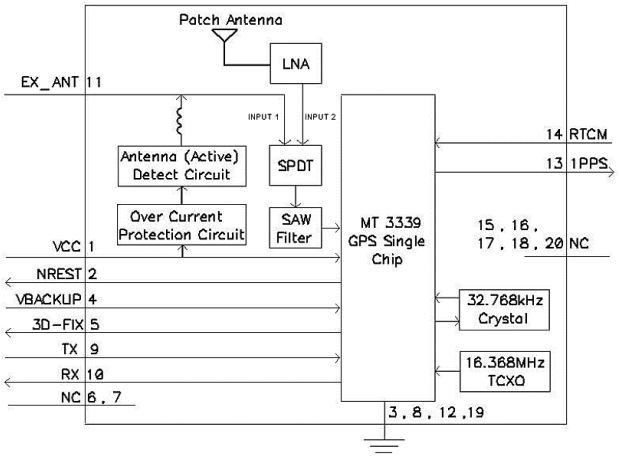
GPS 확장 모듈

- VTG (Course and speed information relative to the ground)
  - 진행 방향 및 속도
  - $GPVTG,165.48,T,,M,0.03,N,0.06,K,A*37

Name	Example	Units	Description
Message ID	$GPVTG		VTG 헤더
Course	165.48	degrees	지도에 대응하는 수평경로
Reference	T		True (진북 기준)
Course		degrees	자기장에 대응하는 수평경로
Reference	M		Magnetic (자북 기준)
Speed	0.03		노트 단위 속도
Units	N	knots	노트 단위로 측정
Speed	0.06		Km/h 단위 속도
Units	K	km/hr	Km/h 단위로 측정
Mode	A		A=자동, D=차동, E=예상
Checksum	37		

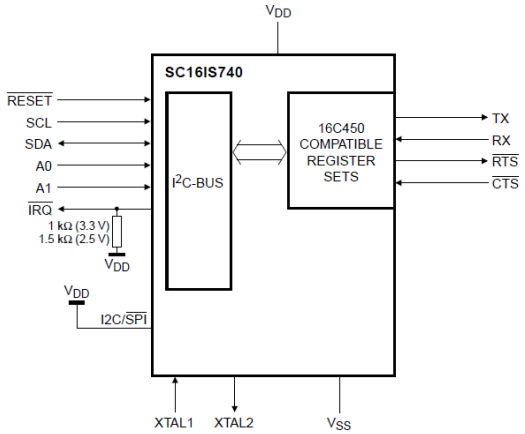
GPS 확장 모듈

- MT3339
  - 안테나 내장형으로 위성으로부터 GPS 신호를 수신해 UART로 출력
    - 안테나 전환 기능 탑재로 외부 안테나 연결 가능
    - 66개의 검색 채널과 22개의 동시 추적 채널로 최대 210개의 PRN 채널 지원



GPS 확장 모듈

- SC16IS750 칩을 통해 MT3339의 UART를 I2C 버스에 연결
  - I2C 버스에 UART 장치 연결 지원
    - 전송 속도는 크리스탈 주파수에 의존적이며 일반적으로 9600(디폴트) ~ 19200bps 범위로 사용
      - 내부 레지스터에 분주 값을 설정해 속도 결정
      - div = crystal_freq / (board * 16)
    - parity는 사용하지 않으며 stop bit는 0, data bit는 8로 설정
    - 수신 오류
      - MCU 응용프로그램이 인터럽트를 사용할 수 없을 때
        - 폴링 과정에서 수신 데이터를 놓칠 수 있음
      - 계산한 분주 값이 실수일 때
        - 실수만큼의 오류율발생



GPS 확장 모듈 제어

- 준비물

준비물	
1	PC
2	XNode 1ea
3	Micro type USB cable 1ea
4	XNode 제공 USB 메모리 (D: 로 가정)
5	GPS 모듈

```
PS C:\XNode\GPS> xnode -p com3 ls /flash/lib
/flash/lib/GPS.py
/flash/lib/core_a.py
/flash/lib/core_b.py
/flash/lib/pop.py
```

  - XNode와 PC를 USB 케이블로 연결
  - D:₩Library₩GPS.py를 작업 폴더인 C:₩XNode로₩GPS₩lib 복사
    - C:₩XNode₩GPS₩lib
  - VS Code를 실행한 후 C:₩XNode₩GPS 아래 lib 폴더를 XNode에 복사
    - xnode -p <포트> put lib

GPS 확장 모듈 제어

- X-Node GPS를 위한 Pop 라이브러리
  - GPS class
    - GPS(): GPS 객체 생성
      - 초기 출력 모드는 GPS.OUTPUT_GGA, 업데이트 속도는 일반 (1초)
    - GPS.setFastUpdate(fast=True): 빠른 업데이트 속도 설정
      - fast: 기본값인 True는 빠른 (0.5초) 업데이트 적용. False는 일반 업데이트
    - GPS.setOutputMode(out): 출력 모드(MNEA) 설정
      - out: GPS.OUTPUT_GGA, GPS.OUTPUT_RMC, GPS.OUTPUT_VTG 중 하나
    - GPS.read(): GPS 데이터 읽기
      - GPS 데이터를 바이트 문자열로 반환. 내부 읽기 오류면 빈 바이트 반환
    - GPS.parse(): GPS 데이터 읽기
      - GPS 데이터를 딕셔너리로 반환. 내부 읽기 오류면 빈 딕셔너리 반환

GPS 확장 모듈 제어

- GPS 수신기로부터 GPGGA 데이터 읽기
  - 위성 신호를 수신할 수 없는 곳(실내 등)에서는 빈 데이터가 출력됨

```
01: from pop import GPS
02: from pop import time
03:
04: gps = GPS()
05:
06: while True:
07:     data = gps.read()
08:     print(data)
```

```
PS C:\XNode\GPS> xnode -p com3 run ext_gps_read.py
$GPGGA,054635.000,3620.5309,N,12719.2698,E,1,07,1.04,55.0,M,21.2,M,,*5F
$GPGGA,054636.000,3620.5310,N,12719.2697,E,1,07,1.04,55.0,M,21.2,M,,*5B
$GPGGA,054637.000,3620.5311,N,12719.2697,E,1,06,1.30,55.0,M,21.2,M,,*5D
$GPGGA,054638.000,3620.5312,N,12719.2696,E,1,06,1.30,55.0,M,21.2,M,,*50
$GPGGA,054639.000,3620.5313,N,12719.2696,E,1,06,1.30,55.0,M,21.2,M,,*50
```

```
PS C:\XNode\GPS> xnode -p com3 run ext_gps_read.py
b'$GPGGA,054137.001,,,,,0,00,,,M,,M,,*7D'
b'$GPGGA,054139.001,,,,,0,00,,,M,,M,,*73'
b'$GPGGA,054140.001,,,,,0,00,,,M,,M,,*7D'
b'$GPGGA,054141.001,,,,,0,00,,,M,,M,,*7C'
b'$GPGGA,054142.001,,,,,0,00,,,M,,M,,*7F'
```

GPS 확장 모듈 제어

- 5초마다 출력 모드를 바꿔가며 출력
  - parse()는 GPS 데이터를 딕셔너리로 읽음

```
01: from pop import GPS
02: from pop import time
03:
04: gps = GPS()
05:
06: t0 = time.ticks_ms()
07: out = (o for o in [GPS.OUTPUT_RMC, GPS.OUTPUT_VTG, GPS.OUTPUT_GGA])
08:
09: while True:
10:     if time.ticks_ms() - t0 > 5000:
11:         t0 = time.ticks_ms()
12:
```

GPS 확장 모듈 제어

- (계속)

```
13:         try:
14:             gps.setOutputMode(next(out))
15:         except StopIteration:
16:             out = (o for o in [GPS.OUTPUT_RMC, GPS.OUTPUT_VTG, GPS.OUTPUT_GGA])
17:             gps.setOutputMode(next(out))
18:
19:     data = gps.parse()
20:     print(data)
```

```
PS C:\XNode\GPS> xnode -p com3 run ext_gps_parse.py
{'reference_a': 'T', 'checksum': '*31', 'mode': 'A', 'speed_n': '0.37', 'id': '$GPVTG', 'course_b': '', 'course_a': '253.31', 'units_n':
'N', 'reference_b': 'M', 'speed_k': '0.68', 'units_k': 'K'}
{'ns_indicator': 'N', 'utctime': '042741.000', 'units_g': 'M', 'hdop': '0.85', 'position_fix_indicator': '1', 'units_a': 'M',
'geoidal_separation': '21.2', 'msl_altitude': '46.9', 'checksum': '*5E', 'age_diff_corr': '', 'longitude': '12719.2463', 'latitude':
'3620.5369', 'ew_indicator': 'E', 'satellites_used': '11', 'id': '$GPGGA'}
{'date': '060121', 'utctime': '042747.000', 'speed_over_ground': '0.19', 'magnetic_variation_indicator': '', 'status': 'A',
'ew_indicator': 'E', 'checksum': '*65', 'magnetic_variation': '', 'mode': 'A', 'longitude': '12719.2458', 'ns_indicator': 'N',
'latitude': '3620.5372', 'id': '$GPRMC', 'course_over_ground': '253.31'}
```

GPS 확장 모듈 제어

- 이동 경로를 파일에 저장
  - 파일 이름은 'gps_x.dat'로 x는 실행할 때마다 1씩 증가

```
01: from pop import GPS
02: from pop import time
03: import os
04:
05: FILE_NAME = "gps_1.dat"
06:
07: lst = os.listdir()
08: if FILE_NAME in lst:
09:     lst.remove('lib')
10:     lst.remove('main.py')
11:     lst.sort()
12:     FILE_NAME = "gps_%d.dat"%(int(int((lst[-1].split('_'))[-1].split('.')[0]))+1)
13:
14: gps = GPS()
15:
16: with open(FILE_NAME, "w") as f:
```

GPS 확장 모듈 제어

- (계속)

```
17:     while True:
18:         gps.setOutputMode(GPS.OUTPUT_GGA)
19:         gga = gps.parse()
20:         if not gga: continue
21:
22:         gps.setOutputMode(GPS.OUTPUT_RMC)
23:         rmc = gps.parse()
24:         if not rmc: continue
25:
26:         date = rmc['date']; utc = gga['utctime']
27:
28:         f.write("=" * 25); f.write("₩n")
29:         f.write("%s-%s-%s %d:%s:%s₩n"%(date[4:], date[2:4], date[:2], (int(utc[:2]) + 9) % 24, utc[2:4], utc[4:6]))
30:         f.write("latitude: %s₩n"%(gga['latitude']))
31:         f.write("longitude: %s₩n"%(gga['longitude']))
32:         f.write("altitude: %s meter₩n"%(gga['msl_altitude']))
33:         f.write("speed: %d km/h₩n"%(float(rmc['speed_over_ground']) * 1.852))
34:         f.write("course: %s degree₩n"%(rmc['course_over_ground']))
```

GPS 확장 모듈 제어

- (계속)
  - XNode를 이동시키면서 데이터를 축적해야 하기에 ext_gps_total.py 파일 XNode로 옮겨서 예제 실행

```
xnode -p com3 put ext_gps_total.py main.py
```

  - 실외에서 이동하면서 GPS 데이터 수집 후 결과 확인
    - 속도는 GPS 오차에 의해 보정이 필요할 수도 있음.

```
PS C:\XNode\GPS> xnode -p com3 ls /flash
/flash/gps_1.dat
/flash/gps_2.dat
/flash/gps_3.dat
/flash/gps_4.dat
/flash/gps_5.dat
/flash/lib
/flash/main.py
```

```
PS C:\XNode\GPS> xnode -p com3 get /flash/gps_1.dat
========================
21-01-06 17:54:07
latitude: 3620.5163
longitude: 12719.2840
altitude: 97.6 meter
speed: 2 km/h
course: 300.19 degree
========================
21-01-06 17:54:11
latitude: 3620.5170
longitude: 12719.2857
altitude: 98.7 meter
speed: 1 km/h
course: 333.58 degree
```