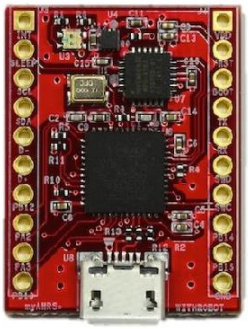
myAHRS+



9/25/2014

myAHRS+는 관성센서(가속도계, 자이로스코프)와 지자기 센서의 출력을 융합하여 얻은 3 차원 공간상의 방위각(Heading)과 자세(Attitude) 정보를 UART/I2C/USB 인터페이스로 출력하는 센서 모듈입니다. 다양한 프로젝트에 사용하실 수 있도록 가로 21.0mm x 세로 27.0mm 의 초소형으로 제작되었습니다.

또한, myAHRS+ 설정값 변경, 데이터 값의 그래프, 3D 자세 출력 등을 쉽게 사용할 수 있는 GUI(myAHRS+ Monitor) 프로그램을 제공합니다.

차 례

제 1 장

Revision History

날짜	내용
2014.09.25	최초 배포

제 2 장

myAHRS+ 개요

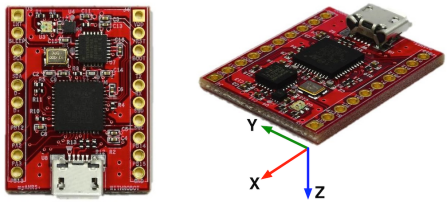


그림 2.1: myAHRS+ 와 좌표계 정의

myAHRS+ 는 관성센서(가속도계, 자이로스코프)와 지자기 센서의 출력을 융합하여 얻은 3 차원 공간상의 자세(Attitude)와 방위각(Heading)을 UART/I2C/USB 인터페이스로 출력하는 센서 모듈이다.

myAHRS+ 기능

- 센서
  - · 16 bit 3축 자이로스코프 : ± 2000 dps
  - · 16 bit 3축 가속도 센서 : ± 16 g
  - · 13 bit 3축 지자기 센서 : ± 1200 μT
- 소프트웨어
  - · 자세 및 센서값을 100 Hz까지 출력
    - · 자세 출력 : Euler angle, Quaternion
    - · 센서 출력 : 가속도, 각속도, 지자기
  - · 여러 센서간 구분이 용이하도록 센서별로 ID 부여
  - · 영점조정 파라미터 변경
  - · 모형 비행체등에서 설치 및 운용이 용이하도록 자세 오프셋 적용
- 연결
  - · USB : virtual comport로 PC와 연결

· OS 지원 : Windows, Linux, OSX

· UART : 보레이트 변경 (9600 ~460800 bps)

· I2C : up to 1kHz

- 기타

· Data ready 인터럽트 출력

· 저전력 Sleep mode 지원

제 3 장

Hardware

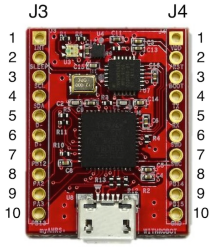


그림 3.1: 핀맵

핀 번호	이름	구분	설명
J3-1	INT	O	Data ready 인터럽트 출력
J3-2	SLEEP	I	슬립 모드 선택 (L-슬립 / H-일반 모드), 연결하지 않을 경우 일반 모드
J3-3	I2C_SCL	I	I2C 클록 입력
J3-4	I2C_SDA	I/O	I2C 데이터 입출력
J3-5	USB_DM	I/O	USB D-
J3-6	USB_DP	I/O	USB D+
J4-1	VDD	PWR	+5V 전원 입력
J4-2	nRST	I	Reset 입력 (L-Reset, H-정상)
J4-3	NCI		아무것도 연결하지 않음
J4-4	UART_TX	O	UART 송신
J4-5	UART_RX	I	UART 수신
J4-6	NC		아무것도 연결하지 않음
J4-7	NC		아무것도 연결하지 않음
J4-10	GND	PWR	파워 그라운드. 0V

표 3.1: 핀맵

Sleep mode

SLEEP 핀에 LOW 를 인가하면 sleep mode 로 전환한다. Sleep mode 에서는 모든 기능이 정지하며 10uA 이하의 소비전력만을 사용한다. SLEEP 핀에 HIGH 를 인가하면 Normal mode 로 전환한다. SLEEP 핀에 아무런 연결도 하지 않은 경우 Normal mode 로 동작한다.

Data Ready 인터럽트

myAHRS+ 는 내부 자세 및 센서 값을 갱신하면 INT 핀으로 펄스를 출력한다. I2C 인터페이스를 사용하는 경우 INT 핀의 rising edge 시점에 새로운 데이터를 읽는다.

USB Connector

PC와 USB A to micro B 케이블을 사용하여 연결한다.

LED 출력

myAHRS+ 의 LED로 동작상태를 확인 가능하다.

LED	상태
RED	”USB 연결상태를 표시 - ON : USB로 연결됨 - OFF : USB 연결되지 않음)”
GREEN	센서가 정상 동작중이면 1초 주기로 점멸함.

표 3.2: LED 출력 상태

전기적 특성

항목	최소	최대	단위
전원 전압 (VDD)	-0.3	6.0	V
UART 핀 전압 (UART_TX/UART_RX)	-0.3	5.3	V
USB 핀 전압 (USB_DM/USB_DP)	-0.3	5.3	V
I2C 핀 전압 (I2C_SCL/I2C_SDA)	-0.3	5.3	V
소비전류		52	mA
동작온도	-40	85	℃
NV 메모리 쓰기 횟수		100,000	

표 3.3: 전기적 특성

제 4 장

Revision History

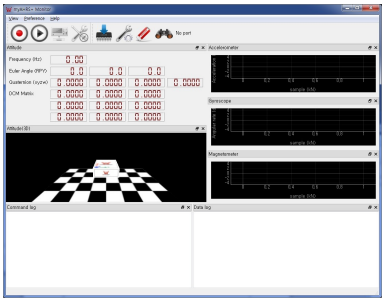
제 5 장

myAHRS+ Monitor

myAHRS+ Monitor는 PC에서 myAHRS+의 출력(자세 및 센서값)을 확인하고, myAHRS+의 여러 설정을 변경할 때 사용하는 사용자 어플리케이션이다.

제 1 절 개요

myAHRS+ Monitor의 모양새 및 툴바의 기능은 아래와 같다.



아이콘	기능
	데이터 로그 저장
	통신 시작 / 정지
	myAHRS+ 설정 확인/변경
	지자기 센서 영점조정
	펌웨어 업데이트
	myAHRS+ Monitor 설정
	모든 뷰어 초기화
	시리얼 포트 검색

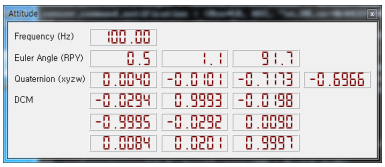
표 5.1: 아이콘 기능 개요

제 2 절 뷰어

myAHRS+ Monitor는 센서의 자세 및 센서값을 여러 뷰어를 통해 보여준다.

2.1 Attitude

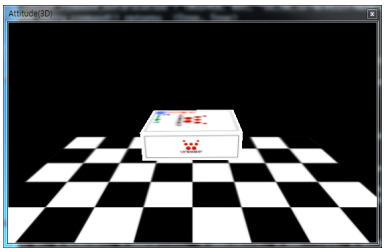
myAHRS+의 자세를 수치로 보여준다.



항목	내용
Frequency	자세 갱신 주기
Euler Angle	오일러각(degree), 순서대로 roll, pitch, yaw
Quaternion	쿼터니언, 순서대로 x, y, z, w (x,y,z : 벡터 성분, w : 스칼라 성분)
DCM	방향 코사인 행렬

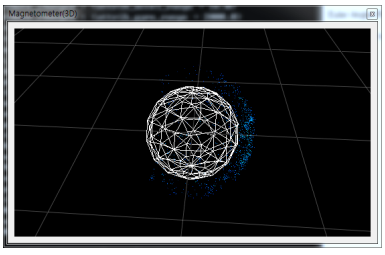
2.2 Attitude (3D)

시각적으로 이해하기 쉽도록 myAHRS+의 자세를 3차원 물체로 보여준다.



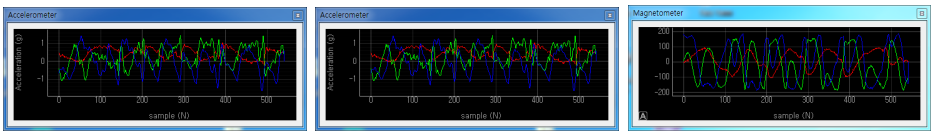
2.3 Magnetometer (3D)

myAHRS+의 지자기 센서 출력을 3D 산점도(scatter plot)으로 보여준다. 이를 통해 myAHRS+를 사용하는 환경의 자기장 왜곡 여부를 시각적으로 확인 가능하다.



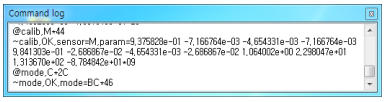
2.4 Graph

가속도, 자이로, 지자기값을 그래프로 보여준다. 적색은 x축, 녹색은 y축, 청색은 z축 데이터를 의미한다.



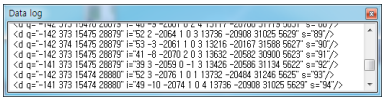
2.5 Command log

myAHRS+ Monitor가 myAHRS+에 전송한 명령과 그에 대한 응답을 출력한다.



2.6 Data log

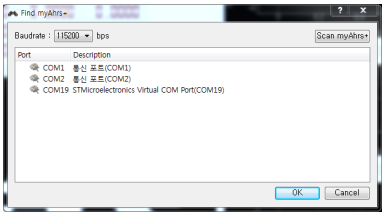
myAHRS+ 로부터 수신한 데이터 메시지를 출력한다. 데이터 전송형식(ASCII, BINARY) 및 종류에 따라 출력 형식이 바뀐다.



제 3 절 기능 상세

3.1 myAHRS+와 연결

(1) myAHRS+를 시스템의 USB포트에 연결하고 myAHRS+ Monitor를 실행한다.  버튼을 눌러 시스템에 설치된 시리얼 포트의 목록을 확인한다.

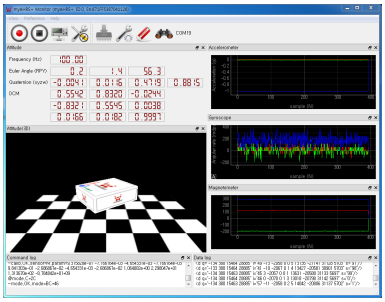


위 대화상자에서, myAHRS+의 COM Port는 “STMicroelectronics Virtual COM Port”로 나타난다.

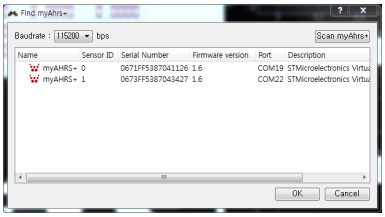
(2) OK 버튼을 누르면  버튼 우측에, 선택한 시리얼 포트가 보인다.

(3)  버튼을 누르면 선택한 시리얼 포트를 통해 myAHRS+와 통신을 시작한다.

아래와 같이 센서의 출력이 각 뷰어에 출력된다.



다수의 myAHRS+ 를 사용한다면, 시리얼 포트 이름만으론 센서를 구분하기 어렵다. 이러한 경우 시리얼 포트 목록 대화상자에서 scan myAHRS+ 버튼을 누르면 myAHRS+와 연결된 시리얼 포트의 목록이 나열된다.

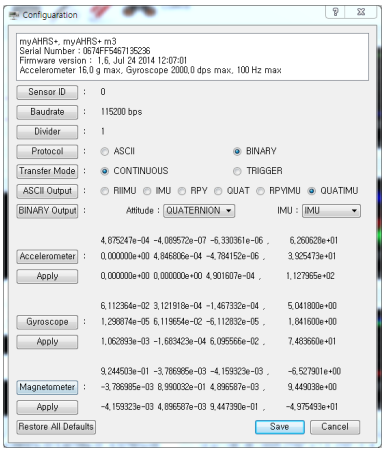


이 대화상자에서 각 센서의 센서 ID와 고유번호를 확인 가능하므로, 여러 센서를 사용하는 경우 보다 쉽게 센서 를 구분할 수 있다.

주의) 시스템에 연결되지 않은 블루투스 장치가 있거나 com0com 등의 가상 시리얼 포트등이 설치되어 있다면 위에 설명한 시리얼 포트 스캔이 정상동작하지 않을 수 있다.

3.2 설정 변경

myAHRS+를 연결한 상태에서  버튼을 누르면 아래와 같이 Configuration 대화상자가 나타난다.



대화상자 좌측에 자리한 각 버튼은 myAHRS+ 명령에 대응하므로 여기에서 대략적인 기능만 기술하고 자세한 내용은 5장에서 설명한다.

버튼	설명
Sensor ID	센서 ID를 변경한다.
Baudrate	UART의 Baudrate를 변경한다.
Divider	센서값 출력 주기를 변경한다. max_rate/divider 값이 출력 주파수가 된다.
Protocol	센서의 출력 프로토콜을 선택한다.
Transfer Mode	전송 모드를 선택한다. TRIGGER를 선택하면 트리거 입력 없이는 센서값을 출력하지 않음을 유의.
ASCII Output	출력 프로토콜이 ASCII일때 출력하는 데이터 메시지의 종류를 선택한다.
BINARY Output	출력 프로토콜이 BINARY일때 출력하는 데이터 메시지의 종류를 선택한다.
Accelerometer	가속도계 영점 조정 파라미터를 변경한다. 대화상자의 값을 변경한 후, 하단의 Apply 버튼을 눌러 센서에 전송한다.
Gyroscope	각속도계 영점 조정 파라미터를 변경한다. 대화상자의 값을 변경한 후, 하단의 Apply 버튼을 눌러 센서에 전송한다.
Magnetometer	지자기센서 영점 조정 파라미터를 변경한다. 대화상자의 값을 변경한 후, 하단의 Apply 버튼을 눌러 센서에 전송한다.
Restore All Defaults	모든 설정값을 공장 설정으로 되돌린다.
Save	현재 설정을 NVRAM에 저장한다. 저장하는 값은 다음과 같다. (Sensro ID, Baudrate, Orientation offset, Calibration parameters)

표 5.3: 버튼 기능 설명

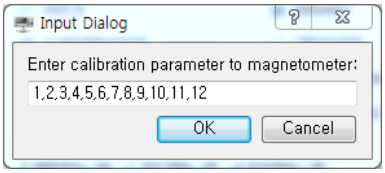
3.3 영점조정 파라미터

myAHRS+는 아래와 같은 방식으로 센서값을 보정한다.

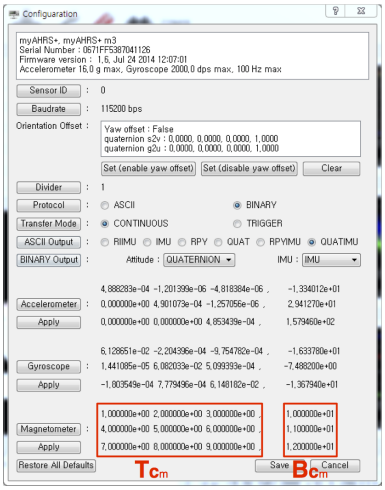
$$\mathbf{v}_c = \mathbf{T}_c(\mathbf{v}_r - \mathbf{b}_c)$$

- $\mathbf{v}_c$: 보정한 센서 출력
- $\mathbf{v}_r$: 보정하기전 센서 출력(Raw data)
- $\mathbf{T}_c$: 변환행렬(3x3 Matrix). scale factor, misalignment 등을 포함
- $\mathbf{b}_c$: 바이어스 벡터(Raw data에 대한 바이어스)

따라서 영점조정 파라미터는 각 센서별로 하나의 매트릭스(3x3)와 하나의 벡터(3x1)로 구성된다. 자신만의 알고리즘으로 영점조정한 결과를 적용하고 싶은 사용자는 다음과 같은 절차로 myAHRS+의 영점 조정 파라미터를 변경한다. 영점조정 파라미터를 변경하려는 센서(예를 들어 Magnetometer)의 버튼을 누르면 Input dialog가 실행된다.



Input dialog에 위와 같이 ‘,’로 구분한 파라미터값을 넣고 OK 버튼을 누르면 다음과 같이 변경한 값이 대화상자에 적용된다. 입력 순서는 $\mathbf{T}_{1,1}$, $\mathbf{T}_{1,2}$, $\mathbf{T}_{1,3}$, $\mathbf{T}_{2,1}$, $\mathbf{T}_{2,2}$, $\mathbf{T}_{2,3}$, $\mathbf{T}_{3,1}$, $\mathbf{T}_{3,2}$, $\mathbf{T}_{3,3}$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$ 순이다.



이 상태에서 Apply 버튼을 누르면 변경한 영점조정 파라미터를 센서에 전송하며, Save 버튼을 누르면 갱신된 영점조정 파라미터를 센서의 NVRAM 영역에 저장한다.

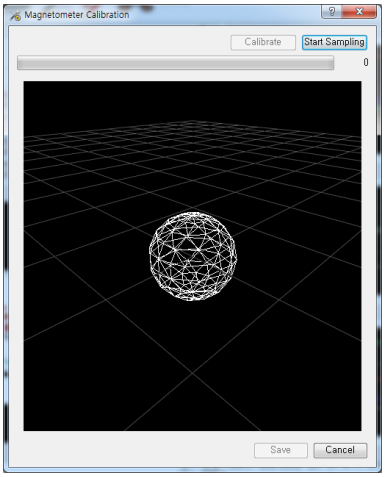
myAHRS+는 영점조정 과정을 거친 후 출고되므로 대부분의 경우 별도의 영점조정 없이 사용해도 무방하나, 지자기센서만은 사용자의 환경에 따라 영점조정 해야한다.

3.4 지자기 센서 영점 조정

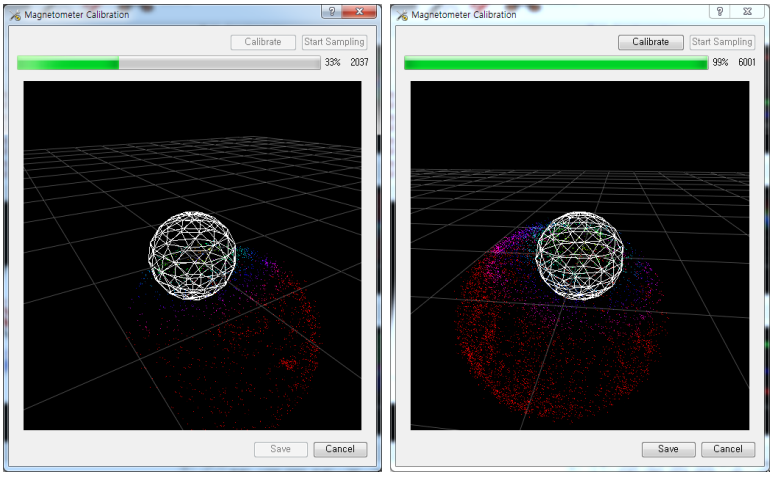
Magnetometer plot에서 지자기 센서 출력 분포가, 중심에 보이는 흰색 구를 중심으로 분포하지 않거나 구형 모양이 아닌 타원체를 형성한다면 지자기 센서 영점조정을 해야한다. 다음의 절차에 따라 지자기 센서를 영점조정한다.

(1) myAHRS+를 연결한 상태에서 버튼을 누른다.

(2) 대화상자가 출력되면 ‘Start Sampling’ 버튼을 누른다.

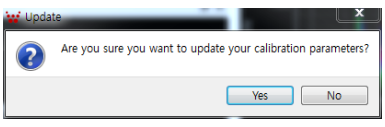


(3) 샘플링은 100Hz로 60초 동안 진행된다. 시간이 자날수록 scatter plot에 타원체가 서서히 모습을 드러내는데, 가능한 빈틈없이(타원체에 빈 곳이 없도록 골고루) 샘플링되도록 센서를 잘 돌려준다.

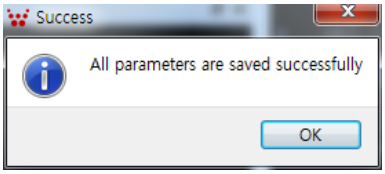


(4) 샘플링이 완료되면 'Calibrate' 버튼을 눌러 영점조정 파라미터를 계산한다. 파라미터 계산을 마치면 'Save' 버튼이 활성화된다. 만약 샘플링한 데이터가 부정확하다면 영점조정 파라미터를 계산할 수 없다는 메시지 박스가 출력된다.

(5) 영점조정 파라미터 계산을 끝내고 'Save' 버튼을 누르면 영점조정 파라미터 갱신여부를 재차 확인한다.

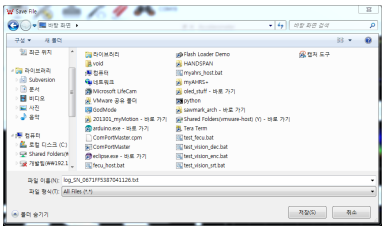


(6) 새로운 영점조정 파라미터를 전송하고, 센서의 NVRAM에 정상적으로 저장하면 다음과 같은 메시지 박스를 출력한다.



3.5 센서 출력 저장

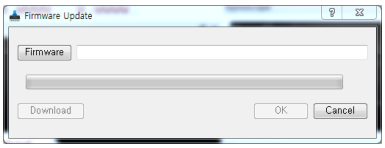
센서의 출력을 파일로 저장하려면, myAHRS+를 연결한 상태에서  을 누르고 저장할 파일 경로를 입력한다. 구분이 용이하도록 기본 파일명으로 센서의 일련번호를 사용하며, Data log 창에 출력하는 내용이 텍스트 파일로 저장된다.



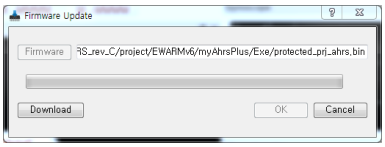
3.6 펌웨어 업그레이드

펌웨어 갱신 절차는 펌웨어 업데이트할 센서를 PC에 연결하지 않은 상태에서 시작한다.

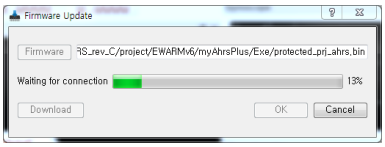
(1)  버튼을 눌러 펌웨어 업데이트 대화상자를 실행한다.



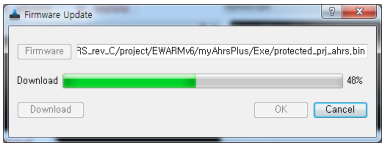
(2) 대화상자에서 ‘Firmware’ 버튼을 눌러 다운로드할 펌웨어 파일을 선택한다.



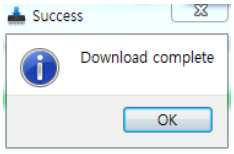
(3) ’Download’ 버튼을 눌러 다운로드를 시작하고, 대화상자가 센서 연결을 대기하는 동안, myAHRS+를 PC에 연결한다.



(4) myAHRS+ 인식에 성공하면 펌웨어 다운로드 절차를 진행한다. 펌웨어 다운로드중엔 myAHRS+의 적색 LED가 점멸한다.

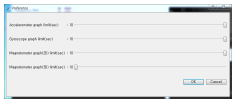


(5) 펌웨어 다운로드 절차를 정상적으로 완료하면 대화상자를 출력한다.



3.7 myAHRS+ Monitor 설정 변경

버튼을 눌러 설정 대화상자를 실행한다. 각 뷰어의 버퍼 크기를 조정할 수 있다.



제 6 장

myAHRS+ 명령

시리얼 포트(USB/UART)를 통해 myAHRS+와 통신에 사용하는 여러 메시지의 상세 내용을 기술한다. 이를 통해 myAHRS+의 센서 데이터를 수신하며 각종 설정을 변경한다.

제 1 절 메시지 프레임 정의

myAHRS+ 메시지의 형식은 다음과 같다.

헤더	기능
@, ~, $	통신 프레임 시작
\r\n	통신 프레임 끝
*	CRC 구분

요청 메시지

- Host(PC)가 센서로 전송하는 메시지다.
- [@][body(N byte)][*][CRC(2byte)][\r\n]

응답 메시지

- Host(PC)의 요청 메시지에 대하여 센서가 응답하는 메시지다.
- [~][body(N byte)][*][CRC(2byte)][\r\n]

데이터 메시지

- 문자열(ASCII)형식과 이진(BINARY)형식을 지원한다.
- 요청에 대한 응답과는 별도로 센서가 Host로 전송하는 메시지로서 센서의 각종 데이터를 담고 있다.
- [$][body(N byte)][*][CRC(2byte)][\r\n]

요청 메시지는 사용자가 센서로 전송하는 명령으로서 '@'로 시작하며 그에 대한 센서의 응답 메시지는 ' '로 시작한다.

- 요청 메시지의 첫 번째 필드는 명령이며 두 번째 필드부터 N(N 0)개의 파라미터가 이어진다.
- 응답 메시지의 첫 번째 필드는 요청 메시지의 명령과 동일하다.
- 응답 메시지의 두 번째 필드는 상태로서, 성공("OK")혹은 실패("ERROR")가 들어간다.
- 응답 메시지의 세 번째 필드부터 N(N 0)개의 파라미터가 이어진다.
    - 응답 메시지의 파라미터는 각종 속성(attribute)이며, 형식은 '속성 이름'='속성 값'이다.
- CRC 오류등으로 요청한 명령을 실행할 수 없는 경우 에러 메시지를 출력한다.

데이터 메시지는 센서의 자세를 포함한 각종 데이터를 담은 메시지로서 명령/응답과는 무관하게 송신한다.

- 연속 모드일때는 사용자가 설정한 주기로 출력한다.
- 트리거 모드일 때는 사용자의 요청이 있을 때 출력한다.

데이터 메시지의 상세 내용은 "데이터 메시지"절에서 기술한다.

위 메시지(요청, 응답, 데이터)에서 body는 구분 문자(,)로 구분한 M개의 필드로 구성된다.

CRC는 프레임 시작부터 body의 끝까지 바이트 단위로 exclusive or 한 값을 ASCII로 표현한 값이다.

- 예를들어 CRC값이 0x1A라면 문자열 "1A"으로 표현한다.
- CRC 값과 '*' 사이에 공백은 없어야 한다

명령 및 응답 예

버전 요청 : "@version*3A"
버전 응답 : "˜version,OK,product=myAHRS+,platform=myAHRS+ m3,
sn=464432970808430886,ver=1.6,build=Jul 24 2014 12:07:01*6F"

제 2 절 명령 및 응답 메시지 상세

설명의 편의를 위해 명령 예에서 CRC 필드("*XX")와 프레임 끝(\r\n)은 생략한다.

2.1 펌웨어 버전

제품명과 펌웨어 버전 그리고 센서 일련번호등의 일반 정보를 요청한다.

- 요청
  @version

- 응답
  ˜version,OK,product=myAHRS+,platform=myAHRS+
  m3,sn=464432970808430886,ver=1.5,build=Jul 13 2014 22:50:17

- 속성

속성 이름	의미
product	제품명
platform	보드 정보
sn	센서 일련번호
ver	펌웨어 버전
build	빌드 일자

2.2 센서 일련번호

- 요청
  @sn

- 응답
  ˜sn,OK,sn=센서 일련번호

- 속성

속성 이름	의미
sn	센서 일련번호

2.3 센서 ID

사용자가 지정 가능한 센서 ID다. 여러 개의 센서를 사용하는 경우, 센서와 PC가 통신하는 시리얼포트 이름(예를들어 COM3)으로 센서를 구분하기에는 어려움이 있으므로, 센서 구분을 위해 사용한다.

사용 가능한 ID의 범위는 0 ˜65535이며 파라미터(ID) 없이 명령을 실행하면 현재 설정된 ID를 출력한다.

- 요청
  @id,ID

- 응답
  ~id,OK,id=ID

- 속성

속성 이름	의미
id	센서 ID

2.4 센서 감도

가속도/자이로 센서의 측정 범위와 감도(sensitivity)를 출력한다.

- 요청
  @sensitivity

- 응답
  ~sensitivity,OK,acc_range=16,gyro_range=2000,acc_sensitivity=4.882813e-04,
  gyro_sensitivity=6.097561e-02

- 속성

속성 이름	의미
acc_range	가속도계 측정 범위 (g)
gyro_range	자이로 스코프 측정 범위 (dps)
acc_sensitivity	가속도계 sensitivity (g/LSB)
gyro_sensitivity	자이로스코프 sensitivity (dps/LSB)

2.5 데이터 출력 모드

센서가 출력하는 데이터 메시지 형식과 출력 방식을 선택한다.

- 'A' : 데이터 메시지를 문자열(ASCII) 형식으로 지정
- 'B' : 데이터 메시지를 이진(BINARY) 형식으로 지정
- 'C' : 연속 모드. 사용자가 정한 주기로 연속하여 출력
- 'T' : 트리거 모드. 사용자의 요청이 있을 때만 출력

출력 모드를 변경하려면 위의 문자를 조합(MODE_STRING)하여 사용한다. 예를들어 문자열 형식으로 연속모드로 설정하는 경우 "AC"를 사용한다. 파라미터 없이 명령을 실행하면 현재 설정을 출력한다.

데이터 메시지를 이진 형식으로 지정하면 구현하기 까다로운 프로토콜 처리를 감수해야 하므로, myAHRS+ SDK를 사용하지 않고 데이터 메시지를 직접 해석해야만 하는 경우 문자열 형식으로 지정하여 사용하기를 권장한다.

- 요청
  @mode,MODE_STRING

- 응답
  ~mode,OK,mode=MODE_STRING

- 속성

속성 이름	의미
mode	출력 모드

- 예
  문자열 형식으로 연속 출력하려면 @mode,AC

2.6 데이터 출력 포맷(ASCII) 설정

데이터 메시지를 문자열(ASCII) 형식으로 지정하였을 때 출력할 데이터 메시지의 종류(ASCII_FORMAT)를 선택한다. 지원하는 메시지 형식은 아래와 같다.

RIIMU	IMU 센서의 raw data(정수) 출력
IMU	IMU 센서의 출력을 calibration parameter로 보정한 값. 가속도 단위(g), 각속도 단위(dps), 온도 단위('C) 지자기 센서 출력은 물리적 단위 없음.
RPY	센서의 자세를 오일러각(degree)으로 출력
QUAT	센서의 자세를 쿼터니언으로 출력
RPYIMU	센서의 자세(오일러각)와 imu 센서 값을 출력, 센서값 출력 순서와 단위는 IMU 메시지와 동일
QUATIMU	센서의 자세(쿼터니언)와 imu 센서 값을 출력, 센서값 출력 순서와 단위는 IMU 메시지와 동일

파라미터 없이 명령을 실행하면 현재 설정을 출력한다. 각 포맷별 상세 설명은 '데이터 메시지' 절에서 기술한다.

- 요청
  @asc_out,ASCII_FORMAT

- 응답
  ~asc_out,OK,fmt=ASCII_FORMAT

- 속성

속성 이름	의미
fmt	설정된 메시지 형식명

- 예
  쿼터니언과 IMU 데이터를 출력하려면 @asc_out,QUATIMU

2.7 데이터 출력 포맷(BINARY) 설정

데이터 출력 모드가 이진 형식인 경우 출력 데이터를 선택한다.

자세	EULER	오일러각 출력
	QUATERNION	쿼터니언 출력
IMU	RIIMU	보정하지 않은 센서값(가속도,각속도,지자기,온도)
	IMU	보정한 센서값(가속도,각속도,지자기,온도)

BINARY_FORMAT 으로 위에 열거한 항목의 조합으로 출력 항목을 지정한다. BINARY_FORMAT 안에 ‘,’가 들어가지 않음을 주의.

데이터 메시지를 이진 형식으로 지정하면 구현하기 까다로운 프로토콜 처리를 감수해야 하므로, myAHRS+ SDK를 사용하지 않고 데이터 메시지를 직접 해석해야만 하는 경우 문자열 형식으로 지정하여 사용하기를 권장한다.

- 요청
  @bin_out,BINARY_FORMAT

- 응답
  ~bin_out,OK,fmt=BINARY_FORMAT

- 속성

속성 이름	의미
fmt	설정된 메시지 형식명

- 예
  @bin_out,EULER RIIMU : 오일러각과 보정하지 않은 센서값을 출력한다.
  @bin_out,IMU : 보정한 센서값만 출력한다.

2.8 데이터 요청

데이터 출력 모드가 트리거 모드일 경우 센서에 데이터를 요청하는 명령이다. 명령에 대한 응답은 없으며 대신 데이터 메시지를 출력한다. 연속 모드로 설정되어 있는경우 에러 응답을 출력한다.

- 요청
  @trig

- 응답
  없음

2.9 데이터 출력 주기 설정

데이터 출력 모드를 연속 모드로 설정하였을 때 데이터 메시지 출력 주기를 설정한다. 센서의 최대 출력 주파수를 사용자가 입력한 분주값(divider)로 나눈 값으로 출력 주기를 결정한다. 가령 최대 출력 주파수가 100Hz고 분주값이 5 라면 데이터 출력 주파수는 20Hz가 된다. 이 명령에 대한 응답으로 센서는, 사용자가 설정한 분주값과 최대 출력 주파수를 출력한다. 파라미터 없이 명령을 실행하면 현재 설정을 출력한다.

- 요청
  @divider,DIVIDER

- 응답
  ˜divider,OK,divider=DIVIDER,max_rate=100

- 속성

속성 이름	의미
divider	분주값
max_rate	최대 출력 주파수

- 예
  @divider,1 : divider를 1로 설정

2.10 센서 영점조정 파라미터 변경

각 센서(가속도, 자이로, 지자기)의 영점조정 파라미터값을 변경한다. CALIBRATION_PARAMETERS 없이 명령을 실행하면 현재 설정을 출력한다.

- 요청
  @calib, SENSOR_TYPE, CALIBRATION_PARAMETERS

  SENSOR_TYPE

  - 'A' : 가속도계
  - 'G' : 자이로스코프
  - 'M' : 지자기센서

  CALIBRATION_PARAMETERS

  - $\mathbf{T}_{1,1}$, $\mathbf{T}_{1,2}$, $\mathbf{T}_{1,3}$, $\mathbf{T}_{2,1}$, $\mathbf{T}_{2,2}$, $\mathbf{T}_{2,3}$, $\mathbf{T}_{3,1}$, $\mathbf{T}_{3,2}$, $\mathbf{T}_{3,3}$, $\mathbf{b}_1$, $\mathbf{b}_2$, $\mathbf{b}_3$

- 응답
  ˜calib,OK,sensor=SENSOR_TYPE,param= CALIBRATION_PARAMETERS

- 속성

속성 이름	의미
sensor	센서 종류
param	영점조정 파라미터

- 예
  @calib,A,1.000000e+00,2.000000e+00,3.000000e+00,4.000000e+00,5.000000e+00,
  6.000000e+00,7.000000e+00,8.000000e+00,9.000000e+00,1.000000e+01,1.100000e+01,
  1.200000e+01

2.11 보레이트 변경

UART 인터페이스의 보레이트를 변경한다. myAHRS+가 지원하는 보레이트 목록은 아래와 같다.

- 9600, 14400, 19200, 38400, 57600, 115200, 230400, 460800

보레이트 변경 메시지를 수신하면, 응답 메시지를 전송한 후 사용자가 지정한 보레이트로 변경된다. 파라미터 없이 명령을 실행하면 현재 설정을 출력한다.

- 요청
  @baudrate,BAUDRATE

- 응답
  ~baudrate,OK,baudrate=BAUDRATE

- 속성

속성 이름	의미
baudrate	보레이트

- 예
  @baudrate,115200 : 보레이트를 115200bps로 설정

2.12 사용자 설정 저장

NVRAM에 저장하는 사용자 설정의 종류는 다음과 같다.

- 센서 ID
- UART 보레이트
- 영점조정 파라미터
- 사용자 좌표계

참고록 위에 열거한 항목 이외의 설정값들은 NVRAM에 저장하지 않으므로 매 번 센서 초기화시 원하는 설정값으로 변경해야 한다.

- 요청
  @save

- 응답
  ~save,OK

2.13 공장 설정

모든 설정을 기본값(출하시 설정)으로 되돌린다.

- 요청
  @factory

- 응답
  ~factory,OK

제 3 절 데이터 메시지

3.1 문자열(ASCII) 형식 메시지

RIIMU	형식	$RIIMU, sequence number, accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z, magnet_x, magnet_y, magnet_z, temperature
	예	$RIIMU,27,25,26,-1901,-16,-7,-11,259,-147,55,158
IMU	형식	$IMU, sequence number, accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z, magnet_x, magnet_y, magnet_z, temperature
	예	$IMU,60,0.0297,0.0019,-1.0056,0.0153,-0.0282,0.3487,129.5813,-110.5982,142.4527,35.5
RPY	형식	$RPY, sequence number, roll, pitch, yaw
	예	$RPY,68,0.04,1.56,34.22
QUAT	형식	$QUAT, sequence number, x, y, z, w
	예	$QUAT,55,-0.0037,0.0134,0.2932,0.9560
RPYIMU	형식	$RPYIMU, sequence number, roll, pitch, yaw, accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z, magnet_x, magnet_y, magnet_z, temperature
	예	$RPYIMU,82,0.04,1.67,34.07,0.0307,0.0014,-1.0095,-0.0435,0.0919,0.1660,137.2258,-90.1564,134.8918,35.6
QUATIMU	형식	$QUATIMU, sequence number, x, y, z, w, accel_x, accel_y, accel_z, gyro_x, gyro_y, gyro_z, magnet_x, magnet_y, magnet_z, temperature
	예	$QUATIMU,02,-0.0039,0.0135,0.2940,0.9557,0.0238,0.0034,-0.9978,-0.0448,-0.0896,0.2866,136.5006,-86.5058,134.0961,35.8

3.2 이진(Binary) 형식 메시지

데이터 메시지를 이진 형식으로 지정하면 구현하기 까다로운 프로토콜 처리를 감수해야 하므로, myAHRS+ SDK를 사용하지 않고 데이터 메시지를 직접 해석해야만 하는 경우 문자열 형식으로 지정하여 사용하기를 권장한다. myAHRS+ SDK를 사용하는 경우 사용자가 센서 데이터 메시지 해석을 구현할 필요는 없으므로 사용자 가이드에선 이에 관한 상세한 내용을 생략한다.

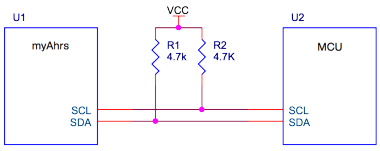
제 7 장

I2C 인터페이스

제 1 절 개요

myAHRS+는 UART/USB 이외에 I2C 인터페이스를 제공한다. 이를 통해 사용자의 임베디드 시스템에 myAHRS+를 쉽게 인터페이스 할 수 있다.

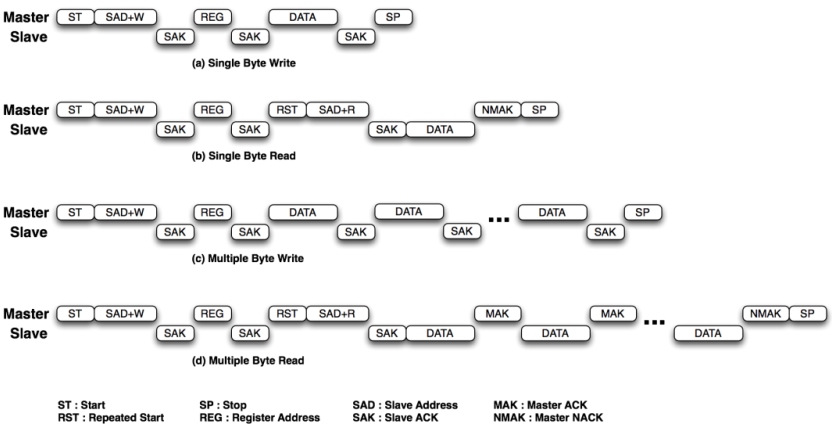
myAHRS+는 I2C slave 로 동작하며, I2C 버스에는 pull-up저항이 필요하다. 일반적인 경우 4.7kΩ 을 사용하며, 상황에 따라 1kΩ 10kΩ 의 저항을 사용 가능하다.



I2C 포트는3.3V 전압 레벨에서 동작하며, 5V IO를 사용하는 MCU와 별도의 회로 없이 바로 연결해서 사용 가능. 기타 사양은 다음과 같다.

- I2C Slave address : 7bit, 0x20
- Data bit : 8bit
- I2C clock speed : Normal mode(100KHz), Fast mode(400KHz)

myAHRS+의 I2C 인터페이스는 4 가지 시퀀스를 지원하며, 각 시퀀스의 구성은 다음과 같다.



제 2 절 I2C 레지스터

myAHRS+의 레지스터들은 다음과 같다.

레지스터 이름	속성	주소	기본값	설명
WHO_AM_I	Read only	0x01	0xB1	
REV_ID_MAJOR	Read only	0x02	-	
REV_ID_MINOR	Read only	0x03	-	
STATUS	Read only	0x04	0x80	센서 상태에 따라 변경됨
I_ACC_X_LOW	Read only	0x10	data	가속도 raw data
I_ACC_X_HIGH	Read only	0x11	data	가속도 raw data
I_ACC_Y_LOW	Read only	0x12	data	가속도 raw data
I_ACC_Y_HIGH	Read only	0x13	data	가속도 raw data
I_ACC_Z_LOW	Read only	0x14	data	가속도 raw data
I_ACC_Z_HIGH	Read only	0x15	data	가속도 raw data
I_GYRO_X_LOW	Read only	0x16	data	각속도 raw data
I_GYRO_X_HIGH	Read only	0x17	data	각속도 raw data
I_GYRO_Y_LOW	Read only	0x18	data	각속도 raw data
I_GYRO_Y_HIGH	Read only	0x19	data	각속도 raw data
I_GYRO_Z_LOW	Read only	0x1A	data	각속도 raw data
I_GYRO_Z_HIGH	Read only	0x1B	data	각속도 raw data
I_MAGNET_X_LOW	Read only	0x1C	data	지자기 raw data
I_MAGNET_X_HIGH	Read only	0x1D	data	지자기 raw data
I_MAGNET_Y_LOW	Read only	0x1E	data	지자기 raw data
I_MAGNET_Y_HIGH	Read only	0x1F	data	지자기 raw data
I_MAGNET_Z_LOW	Read only	0x20	data	지자기 raw data
I_MAGNET_Z_HIGH	Read only	0x21	data	지자기 raw data
C_ACC_X_LOW	Read only	0x22	data	보정한 가속도
C_ACC_X_HIGH	Read only	0x23	data	보정한 가속도
C_ACC_Y_LOW	Read only	0x24	data	보정한 가속도
C_ACC_Y_HIGH	Read only	0x25	data	보정한 가속도
C_ACC_Z_LOW	Read only	0x26	data	보정한 가속도
C_ACC_Z_HIGH	Read only	0x27	data	보정한 가속도
C_GYRO_X_LOW	Read only	0x28	data	보정한 각속도
C_GYRO_X_HIGH	Read only	0x29	data	보정한 각속도
C_GYRO_Y_LOW	Read only	0x2A	data	보정한 각속도
C_GYRO_Y_HIGH	Read only	0x2B	data	보정한 각속도
C_GYRO_Z_LOW	Read only	0x2C	data	보정한 각속도
C_GYRO_Z_HIGH	Read only	0x2D	data	보정한 각속도

C_MAGNET_X_LOW	Read only	0x2E	data	보정한 지자기
C_MAGNET_X_HIGH	Read only	0x2F	data	보정한 지자기
C_MAGNET_Y_LOW	Read only	0x30	data	보정한 지자기
C_MAGNET_Y_HIGH	Read only	0x31	data	보정한 지자기
C_MAGNET_Z_LOW	Read only	0x32	data	보정한 지자기
C_MAGNET_Z_HIGH	Read only	0x33	data	보정한 지자기
TEMPERATURE_LOW	Read only	0x34	data	온도
TEMPERATURE_HIGH	Read only	0x35	data	온도
ROLL_LOW	Read only	0x36	data	오일러각
ROLL_HIGH	Read only	0x37	data	오일러각
PITCH_LOW	Read only	0x38	data	오일러각
PITCH_HIGH	Read only	0x39	data	오일러각
YAW_LOW	Read only	0x3A	data	오일러각
YAW_HIGH	Read only	0x3B	data	오일러각
QUATERNIAN_X_LOW	Read only	0x3C	data	쿼터니언
QUATERNIAN_X_HIGH	Read only	0x3D	data	쿼터니언
QUATERNIAN_Y_LOW	Read only	0x3E	data	쿼터니언
QUATERNIAN_Y_HIGH	Read only	0x3F	data	쿼터니언
QUATERNIAN_Z_LOW	Read only	0x40	data	쿼터니언
QUATERNIAN_Z_HIGH	Read only	0x41	data	쿼터니언
QUATERNIAN_W_LOW	Read only	0x42	data	쿼터니언
QUATERNIAN_W_HIGH	Read only	0x43	data	쿼터니언

2.1 WHO_AM_I

myAHRS+임을 의미하는 상수다.

2.2 REV_ID_MAJOR, REV_ID_MINOR

펌웨어 리비전 번호다.

2.3 STATUS

센서 상태이며 모든 초기화과정을 정상적으로 마치면 최상위 비트가 1이 된다.

2.4 I_ACC_X_LOW ~I_MAGNET_Z_HIGH

영점조정 파라미터로 보정하지 않은 센서 출력 값(signed 16bit)을 저장한 레지스터. LOW 레지스터에 하위 8bit를, HIGH 레지스터에 상위 8bit를 저장한다. 원래 정수이므로 실수로 변환할

필요없다.

2.5 C_ACC_X_LOW ~C_ACC_Z_HIGH

보정한 가속도 센서의 출력 값(signed 16bit)을 저장한 레지스터. LOW 레지스터에 하위 8bit를, HIGH 레지스터에 상위 8bit를 저장한다. scale factor(16/32767.0)를 곱하여 g 단위의 가속도로 변환한다.

$$\text{가속도(g)} = \text{C\_ACC} \times 16.0 \,/\, 32767$$

2.6 C_GYRO_X_LOW ~C_ GYRO _Z_HIGH

보정한 각속도 센서의 출력 값(signed 16bit)을 저장한 레지스터. LOW 레지스터에 하위 8bit를, HIGH 레지스터에 상위 8bit를 저장한다. scale factor(2000/32767.0)를 곱하여 dps(°/s) 단위의 가속도로 변환한다.

$$\text{각속도(dps)} = \text{C\_GYRO} \times 2000 \,/\, 32767$$

2.7 C_MAGNET_X_LOW ~C_ MAGNET _Z_HIGH

보정한 지자기 센서의 출력 값(signed 16bit)을 저장한 레지스터. LOW 레지스터에 하위 8bit를, HIGH 레지스터에 상위 8bit를 저장한다. 보정한 지자기 센서의 출력값은 물리적인 단위를 상실하므로 굳이 실수로 변환할 필요없다.

만약 자기장 측정값이 필요하다면 보정하지 않은 지자기 센서 출력(I_MAGNET_X_LOW ~I_ MAGNET _Z_HIGH)에 scale factor(0.3)을 곱하면 μT 단위의 자기장 측정 값을 얻을 수 있다.

$$\text{지자기}(\mu\text{T}) = \text{I\_MAGNET} \times 0.3$$

2.8 TEMPERATURE_LOW ~TEMPERATURE _HIGH

온도 센서의 출력 값(signed 16bit)을 저장한 레지스터. LOW 레지스터에 하위 8bit를, HIGH 레지스터에 상위 8bit를 저장한다. scale factor(200/32767.0)를 곱하여 ℃ 단위의 온도로 변환한다.

$$\text{온도}(℃) = \text{TEMPERATURE\_N} \times 200 \,/\, 32767$$

2.9 ROLL_LOW ~YAW_HIGH

오일러각(signed 16bit)을 저장한 레지스터. LOW 레지스터에 하위 8bit를, HIGH 레지스터에 상위 8bit를 저장한다. scale factor(180/32767.0)를 곱하여 degree(°) 단위의 각도로 변환한다.

$$\text{오일러각}(°) = \text{ROLL(PITCH, YAW)} \times 180 \,/\, 32767$$

2.10 QUATERNIAN_X_LOW ~QUATERNIAN_W_HIGH

쿼터니언(signed 16bit)을 저장한 레지스터. LOW 레지스터에 하위 8bit를, HIGH 레지스터에 상위 8bit를 저장한다. scale factor(1/32767.0)를 곱하여 실수로 변환한다.

쿼터니언 = X(Y,Z,W) / 32767(scale factor)

제 8 장

myAHRS+ SDK

제 1 절 개요

사용자의 어플리케이션에 myAHRS+ 를 쉽게 적용할 수 있도록 SDK를 제공한다. myAHRS+ SDK는 Windows, Linux, OS X에서 사용 가능하다.

myAHRS+ SDK는 하나의 C++ 헤더파일(myahrs_plus.hpp)로 구현하였으므로, 사용자는 자신의 어플리케이션에 myahrs_plus.hpp를 추가하여 myAHRS+를 간단히 인터페이스 할 수 있다.

OS	컴파일러
Windows	visual studio 2010
Linux, OS X	g++ 4.2.1 이상, "-std=c++11" 옵션

표 8.1: 사용환경과 컴파일 환경

제 2 절 예제 프로그램 컴파일 및 실행

myAHRS+ SDK와 함께 제공하는 예제코드는 다음과 같이 빌드하고 실행한다.

2.1 Windows

2.1.1 컴파일

windows_example\windows_example.sln을 visual studio 2010으로 실행하여 컴파일한다. 컴파일을 마치면 windows_example\Release(Debug)\test_myahrs_plus.exe 가 생긴다.

2.1.2 시리얼 포트 확인

시스템의 USB포트에 myAHRS+를 연결한 후, 제어판의 장치관리자 혹은 myAHRS+ monitor를 사용하여 myAHRS+가 사용하는 COM포트를 확인한다.

2.1.3 예제 코드 실행

```
F> test_myahrs_plus.exe 시리얼 포트 목록
```

예를들어 시스템에 myAHRS+ 두 개를 연결하고 각각의 시리얼 포트가 COM3, COM4 라면 cmd 창에 다음과 같이 입력한다.

```
F> test_myahrs_plus.exe COM3 COM4
```

2.2 Linux, OSX

리눅스를 설치한 PC혹은 라즈베리파이 같은 임베디드 보드에서는 다음과 같은 과정으로 예제코드를 빌드하고 실행한다.

2.2.1 컴파일

터미널에 다음과 같이 입력하여 예제 프로그램을 컴파일한다.

```
$ cd 예제파일 경로
$ make clean; make all
```

2.2.2 시리얼 포트 확인

시스템의 USB포트에 myAHRS+를 연결한 후, 다음의 명령으로 시리얼 장치의 경로를 확인한다.

```
$ ls -l /dev/tty*
$
```

리눅스는 /dev/ttyACM 으로 인식하며 OSX은 /dev/tty.usbmodem으로 인식한다.

2.2.3 예제 코드 실행

터미널에 다음과 같이 입력하여 실행한다.

```
$ ./test_myahrs_plus 시리얼 포트 목록
$
```

예를들어 시스템에 myAHRS+ 두 개를 연결하고 각각의 시리얼 장치가 /dev/ttyACM0, /dev/ttyACM1 이라면 터미널에 다음과 같이 입력한다.

```
$ ./test_myahrs_plus /dev/ttyACM0 /dev/ttyACM1
$
```

NOTE) 사용자의 시스템에 따라 시리얼 장치의 사용권한이 제한되는 경우가 있다. 이 때는 해당 장치의 접근권한을 수정하거나 아래와 같이 예제 프로그램을 실행한다.

```
$ sudo ./test_myahrs_plus /dev/ttyACM0 /dev/ttyACM1
$
```

예제 프로그램이 정상적으로 실행되면, 6개의 예제 프로그램이 순서대로 실행된다.

제 3 절 예제 프로그램 설명

사용자가 myAHRS+ SDK의 사용법을 쉽게 익힐 수 있도록 몇 가지 예제 프로그램을 제공한다.

3.1 예제 공통 함수

아래의 함수는 이어서 설명할 예제 프로그램에서 공통적으로 사용하는 함수이다.

```
1  /*
2   * myAHRS+ SDK
3   */
4  #include "myahrs_plus.hpp"
5  using namespace WithRobot;
6
7  /*
8   * 에러가 발생하면 에러 메시지를 출력하고 프로그램을 종료한다.
9   */
10 void handle_error(const char* error_msg)
11 {
12     fprintf(stderr, "ERROR: %s\n", error_msg);
13     exit(1);
14 }
15
16 /*
17  * 사용자가 'q'를 입력하면 프로그램을 종료한다.
18  */
19 void wait_for_user_input()
20 {
21     printf("\npress enter key to continue or \'q\' to quit.\n");
22     char c = getchar();
23     if(c == 'q') {
24         exit(1);
25     }
26 }
```

3.2 Hello myAHRS+

myAHRS+ SKD를 통해 myAHRS+를 사용하는 가장 기본적인 예제다.

```
1  void ex1_synchronous_read_ascii(const char* serial_device, int baudrate)
2  {
3      printf("\n### %s() ###\n", __FUNCTION__);
4
5      // myAHRS+ 변수를 선언한다.
6      MyAhrsPlus sensor;
7
8      // 센서 데이터 변수 선언
9      SensorData sensor_data;
10     uint32_t sample_count = 0;
11
12     // myAHRS+와 통신을 시작한다.
13     //  - serial_device : 프로그램 실행시 argument로 입력한 시리얼 장치 이름
14     //  - baudrate : 보드레이트
15     if(sensor.start(serial_device, baudrate) == false) {
16         handle_error("start() returns false");
17     }
```

```
18
19      // 데이터 메시지(ASCII)를 오일러각으로 설정한다.
20      if(sensor.cmd_ascii_data_format("RPY") == false) {
21          handle_error("cmd_asc_out() returns false");
22      }
23
24      // 데이터 출력 주기를 설정한다.
25      //  - divider 값이 1 이므로 데이터 메시지를 100 Hz로 출력한다.
26      if(sensor.cmd_divider("1") ==false) {
27          handle_error("cmd_divider() returns false");
28      }
29
30      // 전송 모드를 ASCII, 연속 모드로 설정한다.
31      if(sensor.cmd_mode("AC") ==false) {
32          handle_error("cmd_mode() returns false");
33      }
34
35      // 데이터 메시지 300 개만 받고 프로그램 종료
36      while(sample_count < 300) {
37          // 데이터 메시지를 수신할때까지 대기한다.
38          if(sensor.wait_data() == true) {
39              sample_count = sensor.get_sample_count();
40
41              // sensor로부터 갱신한 센서 데이터를 읽는다.
42              sensor.get_data(sensor_data);
43
44              // 센서 데이터 구조체에서 오일러각을 읽어 출력한다.
45              // - 데이터 메시지를 오일러각으로 선택했으므로 sensor_data.euler_angle 만 갱신된다.
46              EulerAngle& e = sensor_data.euler_angle;
47              printf("%04d) EulerAngle (roll = %.2f, pitch = %.2f, yaw = %.2f)\n",
48                     sample_count, e.roll, e.pitch, e.yaw);
49          }
50      }
51
52      // myAHRS+ 와 통신을 종료한다.
53      sensor.stop();
54
55      printf("END OF TEST(%s)\n\n", __FUNCTION__);
56  }
```

3.3 동기식으로 실행

MyAhrsPlus::wait_data() 함수는 myAHRS+의 데이터 메시지를 수신할 때까지 프로그램을 잠시 정지시킨다. 한편 본 예제는 myAHRS+로부터 받은 쿼터니언을 방향코사인 행렬과 오일러각으로 변환하는 방법을 보여준다.

```
1   void ex2_synchronous_read_binary(const char* serial_device, int baudrate)
2   {
3       printf("\n### %s() ###\n", __FUNCTION__);
4
5       MyAhrsPlus sensor;
6       SensorData sensor_data;
7       uint32_t sample_count = 0;
8
9       if(sensor.start(serial_device, baudrate) == false) {
10          handle_error("start() returns false");
11      }
12
```

```
13      /*
14       *  데이터 메시지(BINARY)를 쿼터니언과 IMU로 설정한다.
15       *  - 즉 자세와 모든 센서 데이터를 출력하도록 한다.
16       */
17      if(sensor.cmd_binary_data_format("QUATERNION, IMU") == false) {
18          handle_error("cmd_binary_data_format() returns false");
19      }
20
21      // 100 Hz
22      if(sensor.cmd_divider("1") ==false) {
23          handle_error("cmd_divider() returns false");
24      }
25
26      // 전송 모드를 BINARY, 연속 모드로 설정한다.
27      if(sensor.cmd_mode("BC") ==false) {
28          handle_error("cmd_mode() returns false");
29      }
30
31      while(sample_count < 300) {
32          if(sensor.wait_data() == true) {
33              sample_count = sensor.get_sample_count();
34              sensor.get_data(sensor_data);
35
36              // 센서 데이터로부터 쿼터니언과 imu 구조체를 읽어온다.
37              Quaternion& q = sensor_data.quaternion;
38              ImuData<float>& imu = sensor_data.imu;
39              printf("%04d) Quaternion(xyzw)=%.4f,%.4f,%.4f,%.4f, "
40                     "Accel(xyz)=%.4f,%.4f,%.4f, "
41                     "Gyro(xyz)=%.4f,%.4f,%.4f, "
42                     "Magnet(xyz)=%.2f,%.2f,%.2f\n",
43                      sample_count,
44                      q.x, q.y, q.z, q.w,
45                      imu.ax, imu.ay, imu.az,
46                      imu.gx, imu.gy, imu.gz,
47                      imu.mx, imu.my, imu.mz);
48
49              // 쿼터니언을 방향코사인행렬로 변환한다.
50              DirectionCosineMatrix dcm;
51              dcm.set(q);
52              printf("%04d) Direction cosine matrix %s\n",
53                     sample_count, dcm.to_string().c_str());
54
55              // 쿼터니언을 오일러각으로 변환한다.
56              EulerAngle euler_angle_from_quaternion = q.to_euler_angle();
57              printf("%04d) euler angle from quaternion %s\n",
58                     sample_count, euler_angle_from_quaternion.to_string().c_str());
59
60              // 방향코사인행렬을 오일러각으로 변환한다.
61              EulerAngle euler_angle_from_dcm = dcm.to_euler_angle();
62              printf("%04d) euler angle from dcm %s\n",
63                     sample_count, euler_angle_from_dcm.to_string().c_str());
64          }
65      }
66
67      sensor.stop();
68
69      printf("END OF TEST(%s)\n\n", __FUNCTION__);
70  }
71
```

3.4 비동기식으로 실행

MyAhrsPlus::wait_data()를 사용하여 프로그램 흐름을 잠시 정지하는 방식은, 간단한 프로그램을 만들기엔 상당히 편리하다. 그렇지만, 좀 더 유연한 응용프로그램을 구성하기엔 이러한 방식은 오히려 불편하다.

이러한 경우 센서의 속성과 센서값이 갱신될 때 호출되는 콜백 함수를 등록하여 자세 갱신을 비동기적으로 처리하면 편리하다.

```
1  // myAHRS+의 속성이 갱신될 때 호출되는 콜백함수
2  void ex3_callback_attribute(void* context, int sensor_id,
3                              const char* attribute_name, const char* value)
4  {
5      printf("## sensor_id %d, Attribute has been changed(%s, %s)\n",
6             sensor_id, attribute_name, value);
7  }
8
9  // myAHRS+의 데이터가 갱신될 때 호출되는 콜백함수
10 void ex3_callback_data(void* context, int sensor_id, SensorData* sensor_data)
11 {
12     int* counter = (int*)context;
13     (*counter)++;
14
15     Quaternion& q = sensor_data->quaternion;
16     ImuData<float>& imu = sensor_data->imu;
17
18     printf("%04d) sensor_id %d, "
19            "Quaternion(xyzw)=%.4f,%.4f,%.4f,%.4f, Accel(xyz)=%.4f,%.4f,%.4f, "
20            "Gyro(xyz)=%.4f,%.4f,%.4f, Magnet(xyz)=%.2f,%.2f,%.2f\n",
21             *counter,
22             sensor_id,
23             q.x, q.y, q.z, q.w,
24             imu.ax, imu.ay, imu.az,
25             imu.gx, imu.gy, imu.gz,
26             imu.mx, imu.my, imu.mz);
27 }
28
29 void ex3_asynchronous_read_binary(const char* serial_device, int baudrate)
30 {
31     printf("\n### %s() ###\n", __FUNCTION__);
32
33     MyAhrsPlus sensor;
34
35     int sample_counter = 0;
36
37     // 사용자가 정의한 콜백 함수를 등록한다.
38     sensor.register_attribute_callback(ex3_callback_attribute, 0);
39     sensor.register_data_callback(ex3_callback_data, &sample_counter);
40
41     if(sensor.start(serial_device, baudrate) == false) {
42         handle_error("start() returns false");
43     }
44
45     if(sensor.cmd_binary_data_format("QUATERNION, IMU") == false) {
46         handle_error("cmd_binary_data_format() returns false");
47     }
48
49     if(sensor.cmd_divider("1") ==false) {
50         handle_error("cmd_divider() returns false");
51     }
```

```
52
53      if(sensor.cmd_mode("BC") ==false) {
54          handle_error("cmd_mode() returns false");
55      }
56
57      while(sample_counter < 100) {
58          Platform::msleep(100); // 100 msec 동안 sleep
59      }
60
61      sensor.stop();
62
63      printf("END OF TEST(%s)\n\n", __FUNCTION__);
64  }
```

3.5 모든 속성 확인

myAHRS+의 모든 속성값은 다음과 같이 확인할 수 있다.

```
1   void ex4_read_all_properties(const char* serial_device, int baudrate)
2   {
3       printf("\n### %s() ###\n", __FUNCTION__);
4
5       std::string line(120, '#');
6
7       MyAhrsPlus sensor;
8
9       if(sensor.start(serial_device, baudrate) == false) {
10          handle_error("start() returns false");
11      }
12
13      // 모든 속성의 리스트를 받는다.
14      std::vector<std::string> attribute_list = sensor.get_attribute_list();
15
16      printf("%s\n", line.c_str());
17      for(size_t i=0; i<attribute_list.size(); i++) {
18          std::string value;
19          // 각 속성의 값을 읽어 화면에 출력한다.
20          if(sensor.get_attribute(attribute_list[i].c_str(), value) == true) {
21              printf("#  %s : %s\n", attribute_list[i].c_str(), value.c_str());
22          }
23      }
24      printf("%s\n", line.c_str());
25
26      sensor.stop();
27
28      printf("END OF TEST(%s)\n\n", __FUNCTION__);
29  }
```

3.6 iMyAhrsPlus 상속

myAHRS+ SDK의 MyAhrsPlus 클래스로 좀 더 복잡한 응용프로그램을 만들기엔 한계가 있다. 이러한 경우 사용자는 iMyAhrsPlus 클래스를 직접 상속하여 자신만의 클래스를 구현할 수 있다.

```
1   // iMyAhrsPlus 를 상속한 클래스
2   class UserDefinedAhrs : public iMyAhrsPlus
```

```
3  {
4      Platform::Mutex lock;
5      SensorData sensor_data;
6
7  public:
8      int sample_count;
9
10     UserDefinedAhrs(std::string port="", unsigned int baudrate=115200)
11     : iMyAhrsPlus(port, baudrate), sample_count(0) {}
12     ~UserDefinedAhrs() {}
13
14     // 용도가 정의되어 있으므로 센서 초기화 과정을 method로 만든다.
15     bool initialize() {
16         bool ok = false;
17         do {
18             if(start() == false) break;
19             if(cmd_binary_data_format("QUATERNION, IMU") == false) break;
20             if(cmd_divider("1") == false) break;
21             if(cmd_mode("BC") == false) break;
22             ok = true;
23         } while(0);
24
25         return ok;
26     }
27
28     inline void get_data(SensorData& data) {
29         LockGuard _l(lock);
30         data = sensor_data;
31     }
32
33     inline SensorData get_data() {
34         LockGuard _l(lock);
35         return sensor_data;
36     }
37
38 protected:
39     // iMyAhrsPlus::OnSensorData()를 재정의한다.
40     void OnSensorData(int sensor_id, SensorData data) {
41         sample_count++;
42         {
43             LockGuard _l(lock);
44             sensor_data = data;
45
46             // 데이터 출력의 자세를 QUATERNION으로 지정하였으므로 오일러각으로 변환한다.
47             sensor_data.euler_angle = sensor_data.quaternion.to_euler_angle();
48         }
49         do_something(sensor_id);
50     }
51
52     // iMyAhrsPlus::OnAttributeChange()를 재정의한다.
53     void OnAttributeChange(int sensor_id, std::string attribute_name, std::string value) {
54         printf("OnAttributeChange(id %d, %s, %s)\n",
55                 sensor_id, attribute_name.c_str(), value.c_str());
56     }
57
58 private:
59     void do_something(int sensor_id) {
60         std::string line(50, '-');
61         printf("%s\n", line.c_str());
62
63         Quaternion& q = sensor_data.quaternion;
```

```
        EulerAngle& e = sensor_data.euler_angle;
        ImuData<float>& imu = sensor_data.imu;

        printf("%04d) sensor_id %d, "
               "Quaternion(xyzw)=%.4f,%.4f,%.4f,%.4f, Angle(rpy)=%.1f, %.1f, %.1f, "
               "Accel(xyz)=%.4f,%.4f,%.4f, Gyro(xyz)=%.4f,%.4f,%.4f, "
               "Magnet(xyz)=%.2f,%.2f,%.2f\n",
               sample_count,
               sensor_id,
               q.x, q.y, q.z, q.w,
               e.roll, e.pitch, e.yaw,
               imu.ax, imu.ay, imu.az,
               imu.gx, imu.gy, imu.gz,
               imu.mx, imu.my, imu.mz);
    }
};
```

```
void ex5_user_defined_class(const char* serial_device, int baudrate)
{
    printf("\n### %s() ###\n", __FUNCTION__);

    UserDefinedAhrs sensor(serial_device, baudrate);

    if(sensor.initialize() == false) {
        handle_error("initialize() returns false");
    }

    while(sensor.sample_count < 300) {
        Platform::msleep(100);
    }

    printf("END OF TEST(%s)\n\n", __FUNCTION__);
}
```

이러한 방식으로 사용자는 iMyAhrsPlus 클래스를 원하는대로 확장하여 사용할 수 있다. 가령 위 예제의 UserDefinedAhrs::do_something() 함수 안에 iMyAhrsPlus 가 지원하지 않는 어떤 기능 혹은 알고리즘등을 넣을 수 있다.

참고로 iMyAhrsPlus::OnAttributeChange()과 iMyAhrsPlus::OnSensorData() 함수는 메인 쓰레드와는 별개의 쓰레드에서 호출된다. 따라서 UserDefinedAhrs의 어떤 속성은 전형적인 쓰레드간의 공유자원이므로 클래스 속성의 접근 제어를 위해 mutex를 사용한다.

3.7 myAHRS Array

모션 캡쳐등의 응용에서 myAHRS+를 사용하려면, 하나의 응용 프로그램이 다수의 myAHRS+ 센서를 제어해야 한다. 이러한 경우 시스템에 연결한 다수의 myAHRS+ 센서를 하나의 객체(class)로 정의하여 사용하면 편리하다.

본 예제는 이러한 경우에 응용할 수 있는 예제이다.

```
class MyAhrsArray;
class AhrsElement : public iMyAhrsPlus
{
    MyAhrsArray* owner;
    Platform::Mutex lock;
    SensorData sensor_data;
```

```

public:
    int sample_count;

    AhrsElement(MyAhrsArray* _owner, std::string port, unsigned int baudrate)
    : iMyAhrsPlus(port, baudrate), owner(_owner), sample_count(0){}
    ~AhrsElement() {}

    bool initialize() {
        bool ok = false;
        do {
            if(start() == false) break;
            if(cmd_binary_data_format("QUATERNION, IMU") == false) break;
            if(cmd_ascii_data_format("QUATIMU") == false) break;
            if(cmd_divider("1") == false) break;
            if(cmd_mode("BC") == false) break;
            ok = true;
        } while(0);
        return ok;
    }

    SensorData get_data() {
        LockGuard _l(lock);
        return sensor_data;
    }

protected:
    void OnSensorData(int sensor_id, SensorData d) {
        {
            LockGuard _l(lock);
            sample_count++;
            sensor_data = d;
            sensor_data.euler_angle = sensor_data.quaternion.to_euler_angle();
        }
        // 데이터 처리를 소유자에게 위임.
        owner->OnSensorData(this, sensor_id, sensor_data);
    }

    void OnAttributeChange(int sensor_id, std::string attribute_name, std::string value) {
        // 속성 처리를 소유자에게 위임.
        owner->OnAttributeChange(this, sensor_id, attribute_name, value);
    }
};
```

```
class MyAhrsArray
{   // AhrsElement의 배열
    std::vector<AhrsElement*> sensor_array;

    // 센서 ID와 AhrsElement 객체를 대응시킨 map
    std::map<int, AhrsElement*> sensor_map;
    bool activated;
    bool debug;

public:
    MyAhrsArray(std::vector<std::string> port_list, int baudrate, bool dbg)
    : activated(false), debug(dbg) {
        for(size_t i=0; i<port_list.size(); i++) {
            sensor_array.push_back(new AhrsElement(this, port_list[i], baudrate));
        }
    }

```

```
18      ~MyAhrsArray() {
19          stop();
20          for(size_t i=0; i<sensor_array.size(); i++) {
21              delete sensor_array[i];
22          }
23      }
24
25      // 모든 센서 초기화
26      bool initialize() {
27          if(activated) {
28              return true;
29          }
30
31          activated = false;
32          char str_id[16];
33          for(size_t i=0; i<sensor_array.size(); i++) {
34              if(sensor_array[i]->initialize() == true) {
35                  sprintf(str_id, "%d", i);
36                  sensor_array[i]->cmd_id(str_id); // 센서 구분을 위해 ID를 부여한다
37                  sensor_map[i] = sensor_array[i];
38                  Platform::msleep(10);
39              }
40              else {
41                  stop();
42                  return false;
43              }
44          }
45
46          for(size_t i=0; i<sensor_array.size(); i++) {
47              sensor_array[i]->sample_count = 0;
48          }
49
50          activated = true;
51          return true;
52      }
53
54      // 모든 센서 정지
55      void stop() {
56          for(size_t i=0; i<sensor_array.size(); i++) {
57              sensor_array[i]->stop();
58          }
59          sensor_map.clear();
60          activated = false;
61      }
62
63      bool get_data(int sensor_id, SensorData& data) { // 센서 ID에 대응하는 센서 값을 읽어온다
64          std::map<int, AhrsElement*>::iterator it = sensor_map.find(sensor_id);
65          if(it == sensor_map.end()) {
66              return false;
67          }
68          else {
69              data = it->second->get_data();
70              return true;
71          }
72      }
73
74      void show_all(int seq)  { // 모든 센서의 자세를 출력한다.
75          for(size_t i=0; i<sensor_array.size(); i++) {
76              printf("### DATA(seq %d, sensor_id=%d, sample count=%d): Q(%s)\n",
77                     seq, sensor_array[i]->get_sensor_id(),
78                     sensor_array[i]->sample_count,
```

```
                    sensor_array[i]->get_data().quaternion.to_string().c_str());
        }
        printf("\n");
    }

protected:
    void OnSensorData(AhrsElement* sensor, int sensor_id, SensorData& data) {
        // do something...
    }

    void OnAttributeChange(AhrsElement* sensor, int sensor_id,
                           std::string& attribute_name, std::string& value) {
        // do something...
    }
};
```

```
void ex6_multiple_sensors(std::vector<std::string>& serial_device_list, int baudrate)
{
    printf("\n### %s() ###\n", __FUNCTION__);

    static const bool debug = false;
    MyAhrsArray sensor_array(serial_device_list, baudrate, debug);

    if(sensor_array.initialize() == false) {
        handle_error("activate() returns false");
    }

    int sequence = 0;
    for(int i=0; i<5000; i++) {
        Platform::msleep(20);
        sensor_array.show_all(sequence++);
    }

    sensor_array.stop();

    printf("END OF TEST(%s)\n\n", __FUNCTION__);
}
```