
SNU Interceptor Controller (Ver.0)

April 2017
SNU MACRO

Getting Started with SNU Interceptor Controller

Copyright © 2017- Lab. MACRO at SNU

Comments or questions should be addressed to: its_me_chy@snu.ac.kr

Version 0.0.0	April 2017	Hwiyong Choi
---------------	------------	--------------

Table of Contents

1. Introduction to SNU Interceptor Controller	9
2. Hardware.....	9
2.1. SNU Interceptor Controller Cape.....	9
2.1.1. 1.8 VDC 전원공급부.....	11
2.1.2. Protection Diode	11
2.1.3. 1.8 VDC 신호발생기.....	11
2.1.4. Inertial Measurement Unit	12
2.1.5. Voltage Follower.....	12
2.1.6. Pin Header	13
2.2. SNU Interceptor Controller Cape PCB 제작.....	14
2.3. Beaglebone Black.....	15
3. Operating System.....	15
4. Interceptor Control.....	15
4.1. sensor.py & sensorTest.py	17
4.2. control.py & controlTest.py	19
4.3. motorTest.py	23
4.4. waveControl.py	24
4.5. staticTestSx.py.....	27
5. Getting Started.....	29
5.1. Installing Operating System.....	29
5.1.1. Installation of BBB Network-over-USB Driver.....	29
5.1.2. Booting Priority.....	30
5.1.3. Copying Debian on a Micro-SD Card.....	33
5.2. Cable 연결.....	34

5.3. Code 백업.....	35
5.4. Code 수정.....	37
5.5. Code 실행.....	38
5.6. 시험 후 MATLAB으로 시험 Data 불러오기.....	38

List of Figures

Figure 1 SNU Interceptor Controller Cape	9
Figure 2 1.8VDC Voltage Regulator	11
Figure 3 Protection Diode	11
Figure 4 1.8VDC Reference Signal Generator	12
Figure 5 Inertial Measurement Unit.....	12
Figure 6 Voltage Followers	13
Figure 7 Pin Headers	14
Figure 8 Pin Header Pin Map.....	14
Figure 9 Schematic of the SNU Interceptor Controller.....	16
Figure 10 Schematic of SNU Interceptor Controller (SIC)	17
Figure 11 Beaglebone Black Network-over-USB Driver Downloading	30
Figure 12 BeagleBone Getting Started Appears on the My-PC window.....	30
Figure 13 Putty Portable	31
Figure 14 Security Alert	31
Figure 15 Login	32
Figure 16 After Login as root	32
Figure 17 Disabling eMMC	32
Figure 18 eMMC Disabled	33
Figure 19 Win32 Disk Imager	33
Figure 20 Confirm Overwrite.....	34
Figure 21 Write Error.....	34
Figure 22 Win32 Disk Imager. Copying Operating System to a Micro SD.....	34
Figure 23 Cable Connection Diagram.....	35
Figure 24 FileZilla 1	36
Figure 25 FileZilla 2	37
Figure 26 List files	37
Figure 27 control.py Opened by nano Editor	38
Figure 28 MATLAB, Workspace	39

Figure 29 MATLAB, Import Data.....	39
Figure 30 MATLAB, Imported Data.....	40

List of Codes

Code 1. sensor.py	18
Code 2. sensorTest.py	19
Code 3. control.py	20
Code 4. controlTest.py	22
Code 5. motorTest.py	24
Code 6. waveControl.py	27
Code 7. staticTestS40.py	29

List of Tables

Table 1 Beaglebone Black Specifications	15
---	----

1. Introduction to SNU Interceptor Controller

SNU Interceptor Controller (이하 SIC)는 모형선 인터셉터 제어를 위해 제작되었으며 주요 특징은 다음과 같다.

- 두 개의 PWM (Pulse Width Modulation) 채널 제공 (최대 두 개 RC Servo Motor 독립 제어 가능)
- 한 개의 IMU (Inertial Measurement Unit) 내장 (Euler Angle 측정 가능)
- 두 개의 ADC (Analog to Digital Conversion) 채널 제공 (수조 예인 시험 시 Potentiometer를 사용한 모형선 자세 계측 가능)
- Beaglebone Black (Single Board Computer) 기반

Figure 1는 SNU Interceptor Controller Cape (이하 SIC Cape)이다. SIC Cape는 Beaglebone Black (이하 BBB)에 장착해 사용할 수 있도록 설계 및 제작되었다. Pin Header에는 RC Servo Motor와 Potentiometer를 연결할 수 있다.

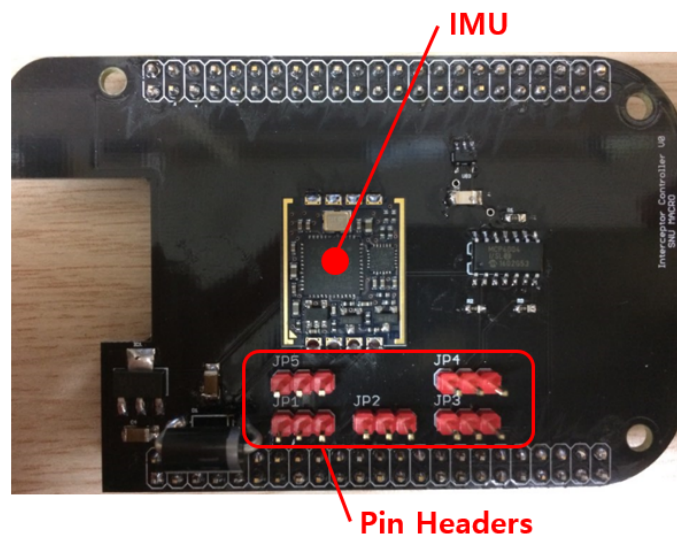


Figure 1 SNU Interceptor Controller Cape

2. Hardware

2.1. SNU Interceptor Controller Cape

SNU Interceptor Controller Cape (이하 SNU ICP)는 모형선 시험 시 BBB에 장착해 Interceptor 제어용 RC Servo Motor의 제어, IMU를 사용한 모형선의 자세 계측, Potentiometer를 사용한 모형선의 자세 (Pitch와 Heave) 계측이 가능하도록 구성되었다. SNU ICP는 아래와 같이 구성된다.

- 1.8 VDC 전원 공급부
- Protection Diode
- 1.8 VDC 신호 발생기
- IMU
- Voltage Follower
- Pin Header

이 것들의 구성도(회로도)는 “첨부1_Schematic.pdf”에 담았다. 위 구성들의 기능은 주요 노드들의 이름과 기능을 살펴본 뒤에 자세히 설명하겠다. 아래 나열한 주요 노드들의 이름들은 “첨부1_schematic.pdf”에도 적혀있다. “첨부1_schematic.pdf”의 좌측 상단부터 살펴보면

- GND: Ground
- 5VDC_EXT: 5 VDC 외부 전원의 (+)극
- REG1.8VDC: 강압된 1.8 VDC 전원
- REF1.8VDC: 1.8 VDC 기준 신호
- 3.3VDC_INTER: BBB에서 출력되는 3.3 VDC 전원
- PWM0: PWM 채널0 (PWM은 채널0부터 명명)
- PWM1: PWM 채널1
- UART1_BBB_TX: BBB의 UART 통신 TX
- UART1_BBB_RX: BBB의 UART 통신 RX
- ADC_VDD: ADC의 최대값 설정용 입력 신호
- ADC_IN_1: ADC 입력 채널1 (ADC는 채널1부터 명명)
- ADC_IN_2: ADC 입력 채널2
- SP1.8VDC_1: Separated REF1.8VDC 채널1
- SP1.8VDC_2: Separated REF1.8VDC 채널2

이제 SNU ICP의 주요 구성 요소들의 기능을 자세히 살펴보고겠다. 아래 Figure 2와 Figure 7는 모두 “첨부1_schematic.pdf”에서 가져온 것이다.

2.1.1. 1.8 VDC 전원공급부

Figure 2는 1.8 VDC 전원공급부이다. 주요 기능은 5 VDC 외부전원을 1.8 VDC로 강압시키는 것이다. 1.8 VDC는 Voltage Follower를 구현하기 위해 사용된 Operational Amplifier 전원 공급에 사용된다. LM1117S-1.8 (Texas Instruments)를 사용했으며 C1과 C4의 선정은 LM1117S-1.8 사양서의 권고 사항을 따랐다 (여기서 C1과 C4의 C는 Capacitor의 약자이다).

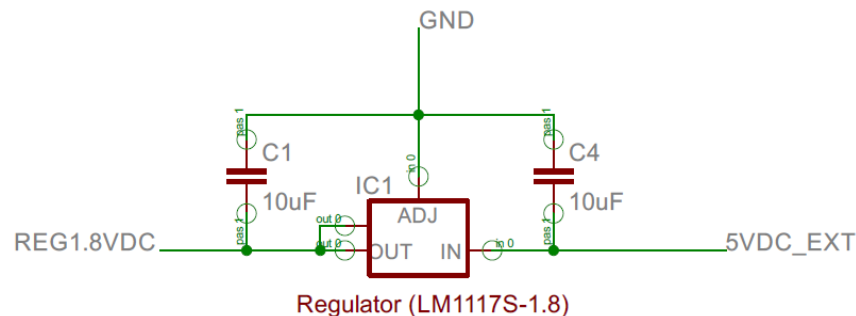


Figure 2 1.8VDC Voltage Regulator

2.1.2. Protection Diode

Figure 3는 Protection Diode이다. RC Servo Motor와 BBB가 전원을 서로 공유하기 때문에 RC Servo Motor 역회전 시 발생할 수 있는 역 기전력에 의한 회로 손상을 막고자 Protection Diode를 설치했다. 사용한 소자는 1N5408이다.

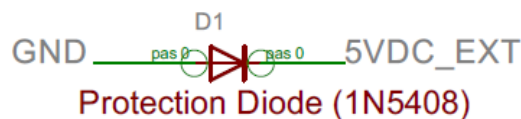


Figure 3 Protection Diode

2.1.3. 1.8 VDC 신호발생기

일반적으로 외부전원이나 Voltage Regulator을 거친 전압은 다양한 요인에 의해 시간에 따라 출력 값이 변할 수 있다. 만약 이들을 ADC의 기준 신호 (Reference Signal)로 사용한다면 ADC를 통해 측정한 전압 값의 오차가 커진다. 따라서 이들

에 비해 정확한 기준 신호가 필요하다. SNU ICP는 Figure 4에 나타낸 LM4120-1.8 (Texas Instruments)를 사용해 비교적 정확한 1.8VDC 기준 신호를 만든다. 여기서 C2의 C는 Capacitor이다. 이 또한 사양서의 권고 사항을 따라 부착한 것이다. 더 나아가 1.8VDC 기준 신호를 사용하는 이유는 BBB의 ADC에 입력될 수 있는 전압의 크기가 최대 1.8V 이기 때문에 이를 넘어서는 입력을 만들지 않기 위함이다.

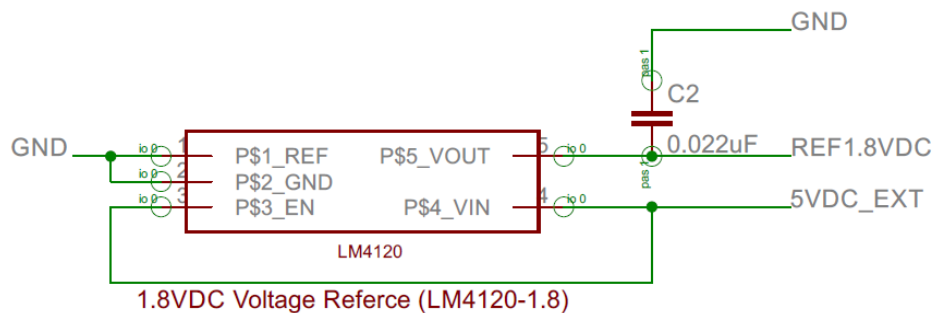


Figure 4 1.8VDC Reference Signal Generator

2.1.4. Inertial Measurement Unit

Figure 5는 SNU ICP에 장착한 IMU (E2BOX EBIIMU-9DOF_V3)이다. IMU의 사양은 “첨부2_EBIMU-9DOFV3”를 참고한다. IMU는 현재 Euler Angle을 측정한 후 UART 통신을 통해 측정 값을 BBB로 전송한다.

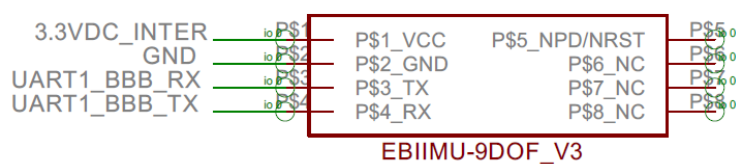


Figure 5 Inertial Measurement Unit

2.1.5. Voltage Follower

Voltage Follower는 입력된 전압을 그대로 출력에 전달하되 입력과 출력을 분리하는 기능을 한다. SNU ICP는 한 개의 1.8VDC 기준 신호를 세 곳에서 서로 영향을 주지 않으며 쓰고자 세 개의 Voltage Follower를 구성했다. Voltage Follower는 Operational Amplifier를 사용해 만들었다. 사용된 Operational Amplifier는

MCP6004 (Microchip)이다. 이 소자는 저 전압에서도 작동되며 GND보다 낮은 전원을 따로 필요로 하지 않는다 (일반적으로 Operational Amplifier는 음의 값을 갖는 전원이 별도로 필요하다). 또한 한 개 IC에 4개의 Operational Amplifier (Figure 6의 삼각형 모양)가 들어있다. 이중 3개에 LM4120-1.8가 만든 1.8 VDC 기준 신호를 입력한다. 최종적으로 3개의 출력 SP1.8VDC_1, SP1.8VDC_2, ADC_VDD가 만들어진다. 이들은 각각 Potentiometer 1, Potentiometer 2, BBB의 ADC의 기준 신호로 사용된다.

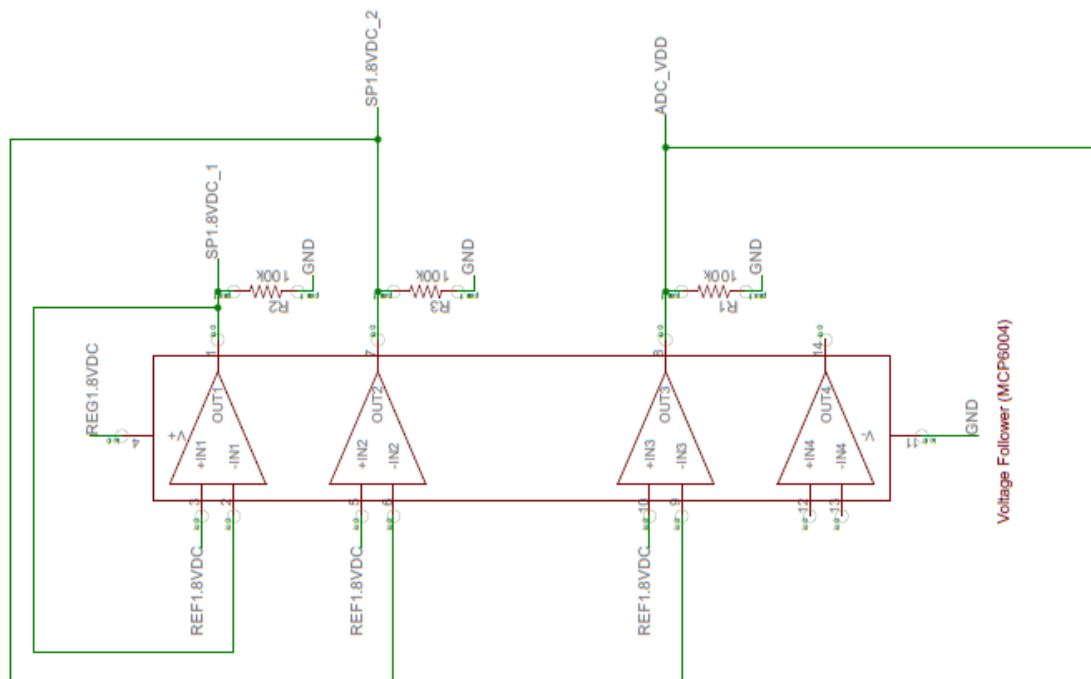


Figure 6 Voltage Followers

2.1.6. Pin Header

Figure 7는 SNU ICP의 Pin Header를 나타낸다. SNU ICP는 해당 Pin Header에 RC Servo Motor와 모형선 자세 측정용 Potentiometer를 연결한다.

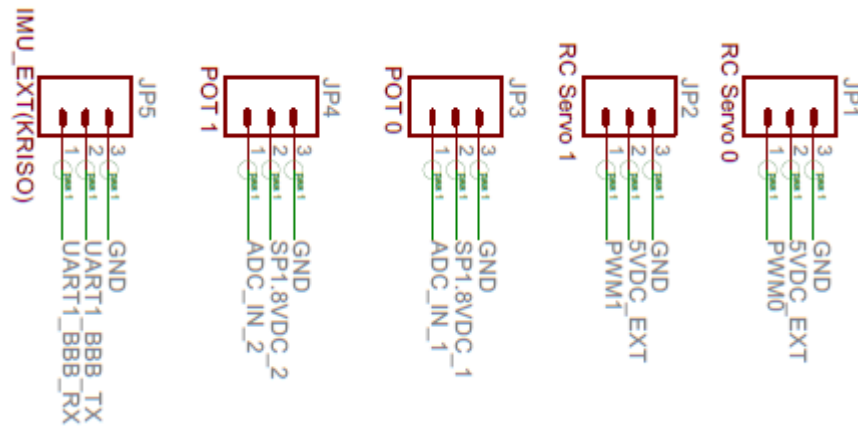


Figure 7 Pin Headers

JP1에는 Port Side의 Interceptor 구동용 RC Servo Motor (이하 MOT1)를 연결한다. JP2에는 Starboard의 Interceptor 구동용 RC Servo Motor (이하 MOT2)를 연결한다. JP3에는 Pitch 측정용 Potentiometer (이하 POT1, Figure 7과 숫자가 다름을 주의)를 연결한다. JP4에는 Heave 측정용 Potentiometer (이하 POT2, Figure 7과 숫자가 다름을 주의)를 연결한다. JP5에는 다른 IMU를 연결할 수 있다. 하지만 이 때 기존에 부착된 IMU는 반드시 제거해야 한다. Figure 8는 SNU ICP 실물 사진에 Pin Header 핀맵을 그린 것이다.



Figure 8 Pin Header Pin Map

2.2. SNU Interceptor Controller Cape PCB 제작

“첨부1_Schematic.pdf” 을 기반으로 만든 PCB도면 파일들은 “첨부 3_PCBFabrication” 에 들어있다. 해당 파일을 PCB 제작 업체에 맡기면 거의 동일한 SNU ICP를 얻을 수 있다 (단, 소자는 직접 연결해야 한다).

2.3. Beaglebone Black

BBB는 TI (Texas Instrument)사에서 개발한 SBC (Single Board Computer)이다. 다음 Table 1은 BBB의 사양을 표로 요약한 것이다.

Table 1 Beaglebone Black Specifications

항목	사양
Processor	AM335x 1GHz ARM Cortex-A8
RAM	512 MB DDR3
eMMC	4 GB
Connectivity	2x64 Pin Headers (UART, I2C, SPI, GPIO), 1xRJ45, HDMI, 1xUSB

SNU ICP는 BBB를 사용해 구동할 수 있다.

3. Operating System

SNU ICP는 BBB를 사용해 구동할 수 있다. BBB 운영체제로는 Debian 7.9를 사용했다. Debian image 파일은 <http://beagleboard.org/latest-images>에서 배포한 것을 사용했다. 이를 SNU ICP에 맞게 설정한 이미지는 “첨부4_debian_SNUICP”에 들어있다. 설정 내용은 다음과 같다.

- 고정 IP 주소 설정 (Over RJ45 Ethernet Port): 192.168.0.79
- HDMI 포트 사용 안 함으로 설정
- Wifi 사용 설정
- 2017년 4월 16일 기준 업데이트 적용

4. Interceptor Control

SIC가 어떤 방법을 이용해 모형선의 자세를 제어하는지 살펴보자. Figure 9은 SIC의 작동 원리를 간단히 그린 것이다. SIC는 D 제어기 (Derivative Controller)를 사용해 모형선의 자세를 제어한다. 모형선의 자세는 SNU ICP에 부착된 IMU를 사용해 측정한다. 시간에 따른 모형선 자세는 D 제어기에 입력되며, 모형선 자세의 변화율을 계산해 출력을 생성한다. D 제어기 출력은 Interceptor

(Interceptor 구동용 RC Servo Motor)에 입력된다. Interceptor의 스트로크 (Stroke) 변화는 모형선 자세를 변화시킨다. 이는 IMU가 다시 측정하며, 위 과정을 반복하며 모형선의 자세 변화를 줄이게 된다.

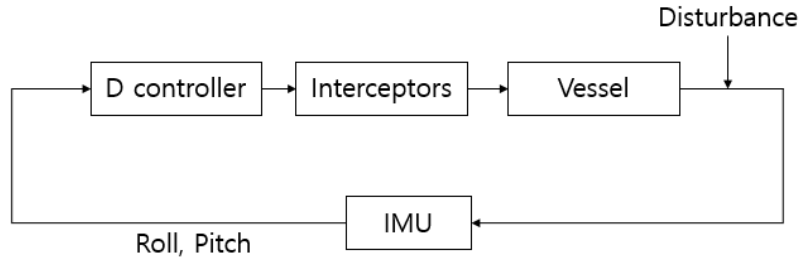


Figure 9 Schematic of the SNU Interceptor Controller

이제 Figure 9에 나타낸 방법을 SIC에 어떻게 구현했는지 설명하겠다.

Figure 10은 SNU ICP의 하드웨어 및 소프트웨어 구성도이다. Debian에 위치한 “waveControl.py”가 Main 코드에 해당하며 나머지 코드와 모듈들은 이를 보조하는 역할을 한다. 특별히 BBIO (Adafruit-BeagleBone-I0-Python)는 BBB에 내장된 통신과 Input/Output (이하 I/O)을 관리한다.

SNU ICP에 내장된 IMU를 사용해 측정한 모형선의 자세 정보는 BBB의 통신장치를 통해 전달되며 이 정보는 “sensor.py”가 해석해 “waveControl.py”가 D 제어 시 이용한다.

D 제어기는 “control.py”에 포함되어 있으며 이를 사용해 구한 D 제어기의 출력은 Pulse Width (PW) 형태로 표현되어 MOT1과 MOT2의 움직임을 조절해 Interceptor의 스트로크를 변화시킨다.

덧붙여, Figure 9에는 나타내지 않았지만 모형선의 자세를 추가적으로 계측하기 위해 사용되는 POT1과 POT2의 출력은 BBB의 ADC를 사용해 측정한다.

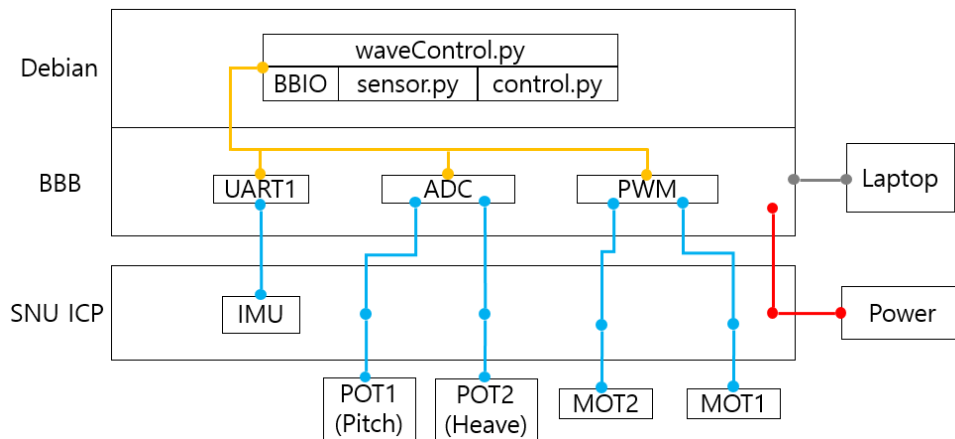


Figure 10 Schematic of SNU Interceptor Controller (SIC)

다음은 위 그림에 나타난 “sensor.py”, “control.py” 와 수조 시험에 사용된 코드, 그 밖의 코드들을 자세히 설명하겠다. SIC 개발에 사용된 프로그래밍 언어는 Python 2.7 (Python Software Foundation)이다. 4.1에서는 “sensor.py”와 이 것의 시험 코드인 “sensorTest.py”를 살펴본다. 4.2에서는 “control.py”와 이 것의 시험코드인 “controlTest.py”를 살펴본다. 4.3에서는 MOT1과 MOT2를 시험할 수 있는 코드인 “motor.py”를 살펴본다. 4.4에서는 Main 코드인 “waveControl.py”를 살펴본다. 마지막으로, 4.5에서는 Interceptor 정적시험에서 쓰인 “staticTestSx.py”를 살펴본다.

4.1. sensor.py & sensorTest.py

Code 1은 “첨부5_code”에 위치한 sensor.py를 그대로 옮긴 것이다. (a)는 사용법을 간단히 소개하고 있다. (b)는 코드의 본문에 해당된다. (c)는 아직 개발하지 않은 부분이다. SNU ICP에 장착된 IMU는 UART 통신을 통해 자세 정보를 문자열 *roll,pitch,yaw\r\n과 같이 계속 전송하는데, sensor.py은 BBB로 전송된 이 문자열 파싱(parsing)해 roll, pitch, yaw를 뽑아낸다.

코드를 살펴보자. 코드에는 EBIIMURead() 함수가 정의되어 있다. 이 함수는 먼저 빈 문자열(string) buffer buf0를 만든다. 그리고 flag를 참값으로 설정한다.

반복문 while은 flag가 참값일 동안은 IMU에서 보낸 문자를 한 개씩 buf1에 읽어 들이고 buf0에 차곡차곡 쌓는 것을 반복한다. flag는 buf0에 \r\n\이 있을 때

거짓으로 바뀐다. 만약 flag가 거짓으로 바뀌면 while은 종료되고 파싱을 시작한다.

파싱 방법은 다음과 같다. buf0에 저장된 *roll,pitch,yaw\r\n 중에 ,(coma)의 위치들을 찾아 순서대로 각각 idxSP1, idxSP2에 저장한다. 그리고 이 위치 정보들을 활용해 roll, pitch, yaw를 뽑아내고 이들을 반환(return)한다.

(a)	<pre> """ Data from the EBIIMU_V3: *roll,pitch,yaw\r\n (in string format) EBIIMURead(ser) parses the data and returns (roll, pitch, yaw) Usage: - Put an instance into 'ser' e.g. EulerAng = sensor.EBIIMURead(ser), then we can extract a roll using the following command roll = EulerAng[0] - We can extract a pitch using the following command pitch = EulerAng[1] - We can extract a yaw using the following command yaw = EulerAng[2] """ </pre>
(b)	<pre> def EBIIMURead(ser): buf0 = str() # An empty string buffer flag = True # A flag informs whether '\r\n' is included in buf0 while(flag): # repeat until '\r\n' is detected in buf0 buf1 = ser.read(1) buf0 = buf0 + buf1 if ('\r\n' in buf0): flag = False idxSP1 = buf0.find(',') # index of the first comma in buf0 idxSP2 = buf0.find(',', idxSP1 + 1) # index of the second comma in buf0 roll = float(buf0[buf0.find('*') + 1 : idxSP1]) # Coordinate def. of the KRISO ASV pitch = - float(buf0[idxSP1 + 1 : idxSP2]) # Coordinate def. of the KRISO ASV yaw = float(buf0[idxSP2 + 1 : buf0.find('\r\n')]) # Not considered return roll, pitch, yaw </pre>
(c)	<pre> """ #def LordSens_3DM_GX4_25 (ser): # to be updated """ </pre>

Code 1. sensor.py

Code 2 “sensorTest.py” 는 “sensor.py” 시험용 코드이다. 이는 “첨부 5_code” 에 위치한 “sensorTest.py” 를 그대로 옮긴 것이다. 이 코드는 “sensor.py” 에서 정의한 EBIIMURead()를 사용해 IMU가 측정한 모형선의 자세를 불러오고 화면에 출력한다.

코드를 살펴보면 (a)에서는 널리 알려진 python 모듈들을 불러오고 (b)에서는 “sensor.py” 를 불러온다. (c)에서는 UART 통신을 설정한다. UART.setup()는

사용할 채널을 설정한다. serial.Serial()은 포트와 Baud-rate를 설정한다. (d)에서는 설정한 UART 통신이 작동할 경우 while 문이 작동한다. while 문은 EBIIMURead()를 사용해 IMU가 측정한 모형선의 자세를 불러오고 화면에 출력하는 것을 반복한다.

(a)	<pre> """ Python Modules """ import serial import time """ BBIO """ import Adafruit_BBIO.UART as UART </pre>
(b)	<pre> """ Developed Module """ import sensor </pre>
(c)	<pre> """ Use UART1 """ UART.setup("UART1") """ Set Port and Baud-rate""" ser = serial.Serial(port = "/dev/ttyO1", baudrate = 115200) </pre>
(d)	<pre> ser.close() # Close the UART1 if it is opened ser.open() if ser.isOpen(): while(True): print(sensor.EBIIMURead(ser)) # Print roll,pitch and yaw ser.close() </pre>

Code 2. sensorTest.py

4.2. control.py & controlTest.py

Code 3는 “첨부5_code”에 위치한 “control.py”를 그대로 옮긴 것이다.

Control Class에는 세 개 Method timeSTMP, ctrlGain, dCtrl가 있다. timeSTMP는 CPU 시간을 읽어온다. ctrlGain는 제어기의 이득(Gain)을 설정한다. dCtrl은 이산시간 D 제어기이다. 세 Method의 사용법은 “controlTest.py”에서 설명하겠다.

	<pre> """ Public Python module """ import time class Control: """ return CPU time""" def timeSTMP(self): #self.STMP = int(round(1000 * time.time())) # [ms] setting self.STMP = time.time() # [s] setting """ We need to set Gains first before using the dctrl""" def ctrlGain(self, KP, KI, KD): self.KP = float(KP) self.KI = float(KI) self.KD = float(KD) """ Dertivative Controller """ def dCtrl(self, t0, t1, e0, e1): self.ud = self.KD * (e1 - e0) / float(t1 - t0) </pre>
--	--

```

""" Proportional Integral Derivative Controller (To be updated)
def pidCtrl(self, t0, t1, t2, e0, e1, e2):
    dt0 = t1 - t0
    dt1 = t2 - t1
    self.u = e0/dt0 * self.KD + e1 * (1 - self.KP - self.KD/dt0 + 0.5
* self.KI * dt1 - self.KD/dt0) + e2 * (self.KP + 0.5 * self.KI * dt0 +
self.KD/dt1)
"""

```

Code 3. control.py

다음 Code 4 “controlTest.py” 는 “control.py” 시험 코드이다. 이는 “첨부5_code” 에 위치한 “controlTest.py” 를 그대로 옮긴 것이다. 여기서는 앞에서 언급한 “sensor.py” 도 같이 사용했다.

코드를 살펴보면 (a)에서는 python 모듈들을 불러온다. 특별히, (b)에서는 “sensor.py” 와 “control.py” 를 모듈로서 불러온다.

(c)에서는 “control.py” 의 Control 클래스의 instance들을 정의한다. 각 instance를 살펴보면, timeSTMP0 와 timeSTMP1 은 각각 control 모듈 (“control.py”)의 Control 클래스에 정의된 timeSTMP Method를 쓰기 위해 정의되었다. timeSTMP0 에는 코드가 실행된 순간 CPU 시간(t_0)이 저장된다. timeSTMP1에는 t_0 이후에 반복해서 측정한 CPU의 시간 t_c 에서 t_0 를 뺀 값 $t = t_c - t_0$ 이 저장된다. ctrlRoll은 control모듈의 Control 클래스에 정의된 ctrlGain과 dCtrl Method를 쓰기 위해 정의되었다.

(d)는 “sensorTest.py” 를 설명하며 살펴보았기 때문에 설명을 생략한다. (e)에서는 D 제어기 이득(KD)을 설정한다. KD는 ctrlGain을 사용해 ctrlRoll에 적용된다. rollEr는 다음과 같이 정의된다. $rollEr = [k \quad e(k); k+1 \quad e(k+1)]$, $k = 0, 1, \dots$. 여기서 e 는 error를 의미한다. 앞에서 이미 설명한 것처럼 timeSTMP0.timeSTMP()는 t_0 를 설정한다.

(f)에서는 시험 시 저장될 측정 자료의 이름과 자료의 구조를 설정한다. “controlTest.dat” 은 측정 자료의 이름이며 f.write()의 괄호 안 str('')의 작은 따옴표 안에 오는 문자는 측정 항목의 이름이다.

(g)는 코드 종료를 하지 않으면 계속 반복된다. 먼저, t_c 를 측정한다. IMU를 사용한 모형선의 자세 측정은 “sensor.py” 의 EBIIMURead()를 사용하며 이를 통해 rollEr 가 갱신된 IMU의 측정 값과 t를 갖게된다. 그 후 D 제어기에 필요한

입력 값들을 입력한다. 특별히 Roll의 시간 변화율 q 도 구한다. D 제어기의 출력 `ctrlPitch.ud`은 `u`로 설정된다. 마지막 줄은 시험 자료를 저장하는 코드이다. `t`, `Roll`, `q` 순서로 저장된다.

(a)	<pre> <i>""" Public python modules """</i> import serial import numpy as np import time <i>""" pyBBIO """</i> import Adafruit_BBIO.UART as UART </pre>
(b)	<pre> <i>""" Developed Modules (placed in the same dir.) """</i> import sensor import control </pre>
(c)	<pre> <i>""" Instances """</i> timeSTMP0 = control.Control() timeSTMP1 = control.Control() ctrlRoll = control.Control() </pre>
(d)	<pre> <i>""" Overlay DTO using pyBBIO """</i> UART.setup("UART1") # IMU <i>""" Set port and baud rate """</i> ser = serial.Serial(port = "/dev/ttyO1", baudrate = 115200) ser.close() ser.open() </pre>
(e)	<pre> if ser.isOpen(): <i>""" Set gain of a PID controllers """</i> KD = 0.03 ctrlRoll.ctrlGain(0,0,KD) # pidGain(KP, KI, KD) <i>""" Error matrices initialization """</i> rollEr = np.matrix('0.0 0.0;1 1') # [t0 e0; t1 e1] <i>""" initial time stamp """</i> timeSTMP0.timeSTMP() </pre>
(f)	<pre> <i>""" Open .dat file to record values """</i> f = open('controlTest.dat', 'w') f.write(str('t') + '\t' + str('rollEr') + '\t' + str('q') + '\t' + str('u') + '\n') #Indices </pre>

(g)

```
""" The Main Loop """
while(True):
    """ Measure current CPU time and update 't' """
    timeSTMP1.timeSTMP() # Current CPU time
    t = timeSTMP1.STMP - timeSTMP0.STMP # Current CPU time - Initial
time stamp
    """ Measure Euler angles """
    EulerAng = sensor.EBIIMURead(ser)
    """ Update the error matrix (2 x 2) """
    rollEr[0] = rollEr[1]
    rollEr[1] = [t, EulerAng[0]]
    """ Derivative control """
    ctrlRoll.dCtrl(rollEr[0,0],rollEr[1,0],rollEr[0,1],rollEr[1,1])
    """ time derivative of roll """
    q = ctrlRoll.ud / KD
    """ Set Motor Positions (refer the calibration chart)"""
    u = ctrlRoll.ud
    """ record values """
    f.write(str(t) + '\t' + str(EulerAng[0]) + '\t' + str(q) + '\t'
+ str(u) + '\n')
```

Code 4. controlTest.py

Figure 11은 위 코드를 시험한 결과이다. 시험은 Roll변화를 준 뒤에 D 제어기의 출력(u)을 살펴보는 절차로 구성되었다. 보다시피 q값에 맞춰 u가 동일한 양상으로 변하는 것을 살펴볼 수 있다. 코드 실행과 시험 자료 받는 법은 5.5와 5.6에 설명했다.

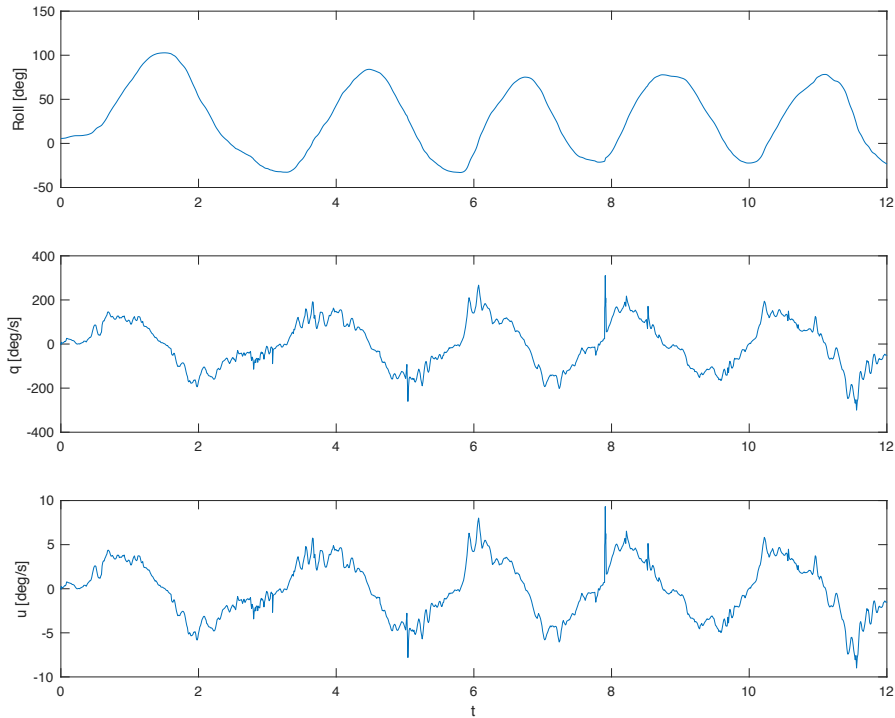


Figure 11 Test Results, controlTest.py

4.3. motorTest.py

Code 5 “motorTest.py” 는 “첨부5_code” 에 위치한 “motorTest.py” 를 그대로 옮긴 것이다. “motorTest.py” 는 Interceptor 구동에 사용되는 RC Servo Motor를 시험할 수 있다.

코드를 살펴보면 (a)에서는 python 모듈들을 불러온다. (b)에서는 “control.py” 의 Control 클래스의 instance를 정의한다. (c)에서는 두 RC Servo Motor를 초기 설정한다. P9_14와 P9_16은 각각 PWM0와 PWM1에 해당한다. 10.9와 9.4 값은 Interceptor의 스트로크가 각각 0%로 가도록 하는 MOT1과 MOT2 입력 값이다 (d)에서는 초기 CPU 시간 t_0 를 측정한다. (e)는 MOT1과 MOT2가 sinusoidal 움직임을 만들도록 구성되었다. 우선 현재 CPU 시간 t_c 를 측정하고 $t_c - t_0$ 를 계산해 t를 생성한다. ome는 다음 줄과 다다음 줄에 있는 sin함수의 주기를 정의한다. mot_pos1과 mot_pos2는 t의 함수이다. t가 증가함에 따라 각각 8.6과 7.2를 기점으로 $2\pi/ome$ sec 주기로 진동한다. 여기서 8.6과 7.2는 각각 좌

우 Interceptor의 스트로크가 50%로 가도록 하는 모터 입력 값이다. (f)는 실제로 실행되지 않으므로 설명을 생략한다.

(a)	<pre> """ Public Python Modules""" import time import math """ BBIO """ import Adafruit_BBIO.PWM as PWM """ Developed Modules """ import control </pre>
(b)	<pre> """ Instances """ timeSTMP0 = control.Control() timeSTMP1 = control.Control() </pre>
(c)	<pre> """ Start and set duty as '0' """ PWM.start("P9_14", 10.9, 50, 0) # PWM.start(pin, duty, freq, polarity) PWM.start("P9_16", 9.4, 50, 0) # PWM.start(pin, duty, freq, polarity) </pre>
(d)	<pre> """ initial time (t_0) """ timeSTMP0.timeSTMP() # t_0 </pre>
(e)	<pre> """ Sinusoidal Motion""" while(True): timeSTMP1.timeSTMP() # Current CPU time (t_c) t = timeSTMP1.STMP - timeSTMP0.STMP # t = t_c - t_0 ome = 1 # Period setting mot_pos1 = 2.0*math.sin(ome*t) + 8.6 # Motor Position Update (Mot1) mot_pos2 = 2.0*math.sin(ome*t) + 7.2 # Motor Position Update (Mot2) PWM.set_duty_cycle("P9_14",mot_pos1) # Set Motor Position (Mot1) PWM.set_duty_cycle("P9_16",mot_pos2) # Set Motor Position (Mot2) </pre>
(f)	<pre> PWM.stop("P9_14") PWM.stop("P9_16") PWM.cleanup() </pre>

Code 5. motorTest.py

4.4. waveControl.py

Code 6는 “waveControl.py”는 “첨부5_code”에 위치한 “waveControl.py”를 그대로 옮긴 것이다. 이 코드는 모형선의 자세를 측정한 뒤 D 제어를 사용해 Interceptor를 제어한다. 즉, 모형선의 자세 변화율을 측정한 뒤 이를 상쇄하려 한다.

코드를 살펴보면, (a)에서는 필요한 모듈들을 불러온다. (b)에서는 사용할 instance들을 정의한다. (c)에서는 “motorTest.py”에서 쓰인 RC Servo Motor 초기 설정 방법을 그대로 따른다. (d)에서는 두 Interceptor의 스트로크가 40%가 되도록 설정한다. 여기서 Interceptor의 스트로크를 0%로 먼저 설정한 뒤 40%로 설정하는 이유는 Interceptor의 기어의 Back-lash 현상에 의한 Interceptor

스트로크 오차를 줄이기 위함이다. (e)에서는 D 제어기의 이득(KD)을 설정한 뒤 ctrlPitch.ctrlGain()과 ctrlRoll.ctrlGain()를 사용해 제어기에 적용한다. rollEr와 pitchEr는 다음과 같이 정의된다. $\text{rollEr} = [k \quad e(k); k+1 \quad e(k+1)]$, $\text{pitchEr} = [k \quad e(k); k+1 \quad e(k+1)]$, $k = 0, 1, \dots$. 여기서 e 는 error를 의미한다. 앞에서 이미 설명한 것처럼 timeSTMP0.timeSTMP()는 t_0 를 설정한다. (f)에서는 시험 시 저장될 측정 자료의 이름과 자료의 구조를 설정한다. “waveTest.dat”은 측정 자료의 이름이며 f.write()의 괄호 안 str('')의 작은 따옴표 안에 오는 문자는 측정 항목의 이름이다. (g)에서는 t_c 를 측정한다. (h)에서는 IMU와 Potentiometer를 사용해 모형선의 자세를 측정한다. IMU를 사용한 모형선의 자세 측정은 “sensor.py”의 EBIIMURead()를 사용한다. Potentiometer를 사용한 모형선의 자세 측정에는 pyBBIO의 ADC.read()를 사용한다. ADC_Ch1과 ADC_Ch2에는 Pitch와 Heave 측정용 potentiometer의 저항 값 변화에 의한 전압 값 변화를 DC (Digital Conversion)한 값이 저장된다. ADC_Ch1과 ADC_Ch2에 저장된 값은 단순히 DC한 값이므로 Pitch와 Heave로 변환되어야 한다. 이를 위해 pitchPOT과 heavePOT에는 Pitch-ADC_Ch1관계와 Heave-ADC_Ch2관계를 시험을 통해 파악해 입력했다. (i)에서는 rollEr와 pitchEr가 갱신된 IMU의 측정 값과 t를 갖도록 갱신한다. (j)에서는 D 제어기에 필요한 입력 값들을 입력한다. 특별히 Pitch의 변화율 q 도 구한다. (k)에서는 D 제어기의 출력 ctrlPitch.ud에 따른 MOT1과 MOT2의 위치를 설정한다. 여기서 9.128과 7.628은 Interceptor의 스트로크가 각각 40%로 가도록 하는 모터 입력 값이다 (모형선의 저항이 가장 작은 스트로크). 덧붙여, Roll 제어는 하지 않아 ctrlRoll.ud를 사용하지 않았음을 명시한다. 마지막으로 (l)은 시험 자료를 저장하는 코드이다. t, Roll, Pitch, ADC_Ch1, ADC_Ch2, pitchPOT, heavePOT, q, mot1_pos, mot2_pos 순서로 저장한다.

```

(a) """ Public python modules """
import serial
import numpy as np
import time
""" BBIO """
import Adafruit_BBIO.UART as UART
import Adafruit_BBIO.PWM as PWM
import Adafruit_BBIO.ADC as ADC
""" Developed Modules (placed in the same dir.) """
import sensor
import control

```

(b)	<pre> """ Instances """ timeSTMP0 = control.Control() timeSTMP1 = control.Control() ctrlRoll = control.Control() ctrlPitch = control.Control() </pre>
(c)	<pre> """ Overlay DTO using pyBBIO """ PWM.start("P9_14", 10.9, 50, 0) # PWM.start(pin, duty, freq, polarity) PWM.start("P9_16", 9.4, 50, 0) # PWM.start(pin, duty, freq, polarity) UART.setup("UART1") # IMU ADC.setup() # ADC """ Set port and baud rate """ ser = serial.Serial(port = "/dev/ttyO1", baudrate = 115200) ser.close() ser.open() </pre>
(d)	<pre> """ Position initialization """ time.sleep(0.5) mot1_pos = 10.9 # PWM_mot1 with interceptor Stroke 0% mot2_pos = 9.4 # PWM_mot2 with interceptor Stroke 0% PWM.set_duty_cycle("P9_14", mot1_pos) PWM.set_duty_cycle("P9_16", mot2_pos) time.sleep(0.5) """ Position initialization for the theoretical 40% stroke""" mot1_pos = 9.128 # PWM_mot1 with interceptor Stroke 40% (ideal value) mot2_pos = 7.628 # PWM_mot2 with interceptor Stroke 40% (ideal value) PWM.set_duty_cycle("P9_14", mot1_pos) PWM.set_duty_cycle("P9_16", mot2_pos) time.sleep(0.5) </pre>
(e)	<pre> """ Set gain of a PID controllers """ KD = 0.02923 ctrlRoll.ctrlGain(0,0,KD) # pidGain(KP, KI, KD) ctrlPitch.ctrlGain(0,0,KD) # pidGain(KP, KI, KD) """ Error matrices initialization """ rollEr = np.matrix('0.0 0.0;1 1') # [t0 e0; t1 e1] pitchEr = np.matrix('0.0 0.0;1 1') # [t0 e0; t1 e1] """ initial time stamp """ timeSTMP0.timeSTMP() </pre>
(f)	<pre> """ Open .dat file to record values """ f = open('waveTest.dat','w') f.write(str('t') + '\t' + str('rollEr') + '\t' + str('pitchEr') + '\t' + str('ADC_Ch1') + '\t' + str('ADC_Ch2') + '\t' + str('pitchPOT') + '\t' + str('heavePOT') + '\t' + str('q') + '\t' + str('mot1_pos') + '\t' + str('mot2_pos') + '\n') #Indices </pre>
(g)	<pre> """ Measure current CPU time and update 't' """ timeSTMP1.timeSTMP() # Current CPU time t = timeSTMP1.STMP - timeSTMP0.STMP # Current CPU time - Initial time stamp </pre>
(h)	<pre> """ Measure Euler angles """ EulerAng = sensor.EBIIMURead(ser) """ Read ADC values from the potentiometers """ ADC_Ch1 = ADC.read("AIN1") # [ADC Value] ADC_Ch2 = ADC.read("AIN2") # [ADC Value] """ convert ADC value into physical size """ pitchPOT = -307 * ADC_Ch1 + 142.2 # [deg] heavePOT = -720.53 * ADC_Ch2 + 969.47 # [mm] </pre>
(i)	<pre> """ Update the error matrices (2 x 2) """ rollEr[0] = rollEr[1] rollEr[1] = [t, EulerAng[0]] </pre>

	<pre>pitchEr[0] = pitchEr[1] pitchEr[1] = [t, EulerAng[1]]</pre>
(j)	<pre>""" Derivative control """ ctrlRoll.dCtrl(rollEr[0,0],rollEr[1,0],rollEr[0,1],rollEr[1,1]) ctrlPitch.dCtrl(pitchEr[0,0],pitchEr[1,0],pitchEr[0,1],pitchEr[1,1]) """ time derivative of pitch """ q = ctrlPitch.ud / KD</pre>
(k)	<pre>""" Set Motor Positions (refer the calibration chart) """ mot1_pos = 9.128 - ctrlPitch.ud mot2_pos = 7.628 - ctrlPitch.ud PWM.set_duty_cycle("P9_14", mot1_pos) PWM.set_duty_cycle("P9_16", mot2_pos)</pre>
(l)	<pre>""" record values """ f.write(str(t) + '\t' + str(EulerAng[0]) + '\t' + str(EulerAng[1]) + '\t' + str(ADC_Ch1) + '\t' + str(ADC_Ch2) + '\t' + str(pitchPOT) + '\t' + str(heavePOT) + '\t' + str(q) + '\t' + str(mot1_pos) + '\t' + str(mot2_pos) + '\n') ser.close()</pre>

Code 6. waveControl.py

4.5. staticTestSx.py

Code 7 “staticTestS40.py” 는 “첨부5_code” 에 위치한 “staticTestS40.py” 를 그대로 옮긴 것이다. 이 코드는 정적 시험을 위해 준비된 코드로 Interceptor의 스트로크를 일정한 상태로 유지한다.

정적 시험은 총 6가지 (0%, 20%, 40%, 60%, 80%, 100%)의 스트로크에 대해 진행되었으며, 각 스트로크에 알맞은 코드는 각각 “첨부5_code” 에 위치한 “staticTestS0.py” , “staticTestS20.py” , “staticTestS40.py” , “staticTestS60.py” , “staticTestS80.py” , “staticTestS100.py” 이다. 이 코드들은 (b)를 제외하고 서로 같다. 따라서 이들 중에 “staticTestS40.py” 즉, Code 7을 뽑아 설명하겠다.

Code 7은 Code 6과 거의 동일하다. 다른 부분은 (b)인데, Code 6의 (k)에서는 D 제어기의 출력을 사용하지만 Code 7의 (b)에서는 고정된 상수 값을 사용해 두 Interceptor의 스트로크가 일정하게 유지되도록 한다. 나머지 정적 시험 코드 즉, “staticTestS0.py” , “staticTestS20.py” , “staticTestS60.py” , “staticTestS80.py” , “staticTestS100.py” 은 (b)의 상수 값이 스트로크에 알맞은 상수 값으로 설정되어있다.

(a)

```
""" BBIO """
import Adafruit_BBIO.UART as UART
import Adafruit_BBIO.PWM as PWM
import Adafruit_BBIO.ADC as ADC
""" Public python modules """
import serial
import numpy as np
import time
""" DIY Modules (placed in the same dir.) """
import sensor
import control
""" Instances """
timeSTMP0 = control.Control()
timeSTMP1 = control.Control()
ctrlRoll = control.Control()
ctrlPitch = control.Control()
""" Overlay DTO using pyBBIO """
PWM.start("P9_14", 10.9, 50, 0) # PWM.start(pin, duty, freq,
polarity)
PWM.start("P9_16", 9.4, 50, 0) # PWM.start(pin, duty, freq,
polarity)
UART.setup("UART1") # IMU
ADC.setup() # ADC
""" Set port and baud rate """
ser = serial.Serial(port = "/dev/ttyO1", baudrate = 115200)
ser.close()
ser.open()
if ser.isOpen():
    """ Position initialization for the static test """
    time.sleep(0.5)
    mot1_pos = 10.9 # PWM_mot1 with interceptor Stroke 0%
    mot2_pos = 9.4 # PWM_mot2 with interceptor Stroke 0%
    PWM.set_duty_cycle("P9_14", mot1_pos)
    PWM.set_duty_cycle("P9_16", mot2_pos)
    time.sleep(0.5)
    """ Set gain of a PID controllers """
    KD = 50
    ctrlRoll.ctrlGain(0,0,KD) # pidGain(KP, KI, KD)
    ctrlPitch.ctrlGain(0,0,KD) # pidGain(KP, KI, KD)
    """ Error matrices initialization """
    rollEr = np.matrix('0.0 0.0;1 1') # [t0 e0; t1 e1]
    pitchEr = np.matrix('0.0 0.0;1 1') # [t0 e0; t1 e1]
    """ initial time stamp """
    timeSTMP0.timeSTMP()
    """ Open .dat file to record values """
    f = open('writeTest.dat', 'w')
    f.write(str('t') + '\t' + str('rollEr') + '\t' + str('pitchEr') +
'\t' + str('ADC_Ch1') + '\t' + str('ADC_Ch2') + '\t' + str('pitchPOT')
+ '\t' + str('heavePOT') + '\t' + str('q') + '\t' + str('mot1_pos') +
'\t' + str('mot2_pos') + '\n') #Indices

    """ The Main Loop """
    while(True):
        """ Measure current time and update 't' """
        timeSTMP1.timeSTMP() # Current CPU time
        t = timeSTMP1.STMP - timeSTMP0.STMP # Current CPU time - Initial
time stamp
        """ Measure an Euler angles """
        EulerAng = sensor.EBIIMURead(ser)
        """ Read ADC values from the potentiometers """
```

	<pre> ADC_Ch1 = ADC.read("AIN1") # [ADC Value] ADC_Ch2 = ADC.read("AIN2") # [ADC Value] """ convert ADC value into physical size """ pitchPOT = -307 * ADC_Ch1 + 142.2 # [deg] heavePOT = -720.53 * ADC_Ch2 + 969.47 # [mm] """ Update the error matrices (2 x 2) """ rollEr[0] = rollEr[1] rollEr[1] = [t, EulerAng[0]] pitchEr[0] = pitchEr[1] pitchEr[1] = [t, EulerAng[1]] """ Derivative control """ ctrlRoll.dCtrl(rollEr[0,0],rollEr[1,0],rollEr[0,1],rollEr[1,1]) ctrlPitch.dCtrl(pitchEr[0,0],pitchEr[1,0],pitchEr[0,1],pitchEr[1,1]) """ time derivative of pitch """ q = ctrlPitch.ud / KD </pre>
(b)	<pre> """ Set Motor Positions (refer the calibration chart) """ mot1_pos = 8.855 mot2_pos = 7.4 PWM.set_duty_cycle("P9_14", mot1_pos) PWM.set_duty_cycle("P9_16", mot2_pos) </pre>
(c)	<pre> """ record values """ f.write(str(t) + '\t' + str(EulerAng[0]) + '\t' + str(EulerAng[1]) + '\t' + str(ADC_Ch1) + '\t' + str(ADC_Ch2) + '\t' + str(pitchPOT) + '\t' + str(heavePOT) + '\t' + str(q) + '\t' + str(mot1_pos) + '\t' + str(mot2_pos) + '\n') #print(EulerAng[1],q) ser.close() </pre>

Code 7. staticTestS40.py

5. Getting Started

5.1. Installing Operating System

SNU ICP 사용에 쓰인 Debian OS를 설치하는 방법을 소개하겠다. Windows x64 (Microsoft) 기준으로 설명하겠다.

5.1.1. Installation of BBB Network-over-USB Driver

<http://beagleboard.org/getting-started>에 접속하면 Figure 12과 같은 화면에서 BBB Network-over-USB Driver를 다운로드 할 수 있다. 64-bit installer를 클릭해 Driver를 다운로드 한 뒤에 이를 설치한다.

Step 2 Install drivers		
Install the drivers for your operating system to give you network-over-USB access to your Beagle. Additional drivers give you serial access to your board.		
Operating System	USB Drivers	Comments
Windows (64-bit)	64-bit installer	If in doubt, try the 64-bit installer first. <ul style="list-style-type: none"> Note #1: Windows Driver Certification warning may pop up two or three times. Click "Ignore", "Install" or "Run". Note #2: To check if you're running 32 or 64-bit Windows see this: https://support.microsoft.com/kb/827218 Note #3: On systems without the latest service release, you may get an error (0xc000007b). In that case, please install the following and retry: https://www.microsoft.com/en-us/download/confirmation.aspx?id=13523 Note #4: You may need to reboot Windows. Note #5: These drivers have been tested to work up to Windows 10
Windows (32-bit)	32-bit installer	
Mac OS X	Network Serial	Install both sets of drivers.
Linux	mkudevrule.sh	Driver installation isn't required, but you might find a few udev rules helpful.

Figure 12 Beaglebone Black Network-over-USB Driver Downloading

설치가 완료되면 제공된 USB Cable을 사용해 BBB와 PC를 연결한다 (이 때는 전원을 따로 연결할 필요가 없다). 그러면 대략 30초 정도 후에 내PC에 Figure 13과 같이 “BeagleBone Getting...” 아이콘이 나타나는 것을 확인 할 수 있다. 만약 나타나지 않는다면 Driver 설치를 올바르게 진행하지 못 한 것이다.

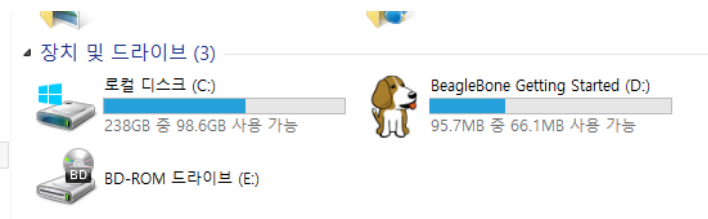


Figure 13 BeagleBone Getting Started Appears on the My-PC window

5.1.2. Booting Priority

새로 구매한 BBB는 eMMC에 이미 OS가 설치되어 있다. 그리고 부팅 우선 순위가 eMMC로 설정되어 있다. 하지만 우리는 u-SD에 OS를 설치해 사용할 것이다. 따라서 BBB가 booting할 때 eMMC를 읽지 않도록 설정해야 한다.

이를 위해 BBB에 접속할 수 있는 Tool인 Putty Portable (이하 Putty)을 실행시킨다. Putty는 검색을 통해 쉽게 다운로드 할 수 있고 설치 없이 바로 사용할 수 있다. Putty를 실행시키면 Figure 14과 같은 창이 뜨는데 Host Name에 192.168.7.2를 입력하고 Open 버튼을 누른다. 여기서 192.168.7.2는 USB 연결에 사용되는 BBB의 IP주소이다.

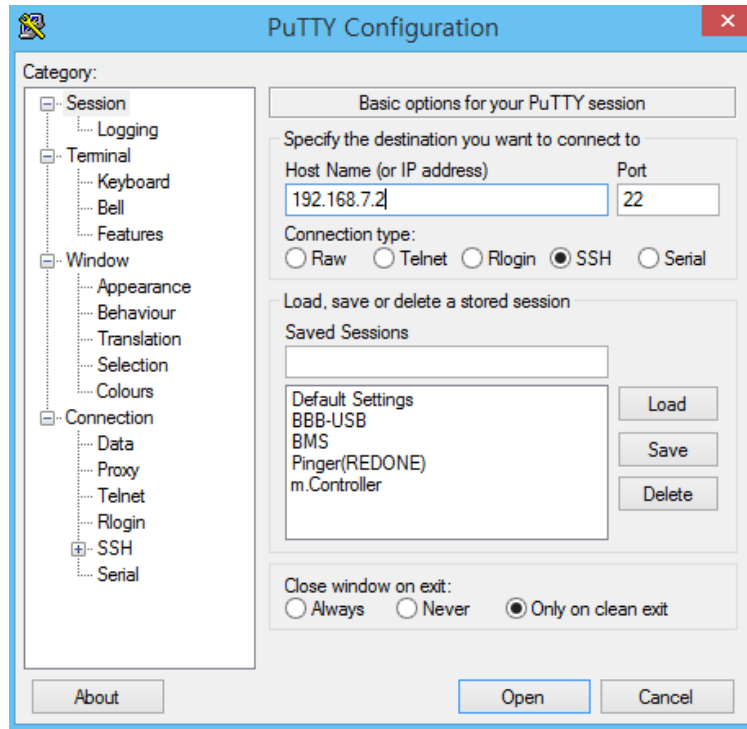


Figure 14 Putty Portable

간혹 Figure 15과 같은 보안 경고 창이 뜨기도 하는데 이 때는 무시하고 “예 (Y)” 버튼을 누른다.

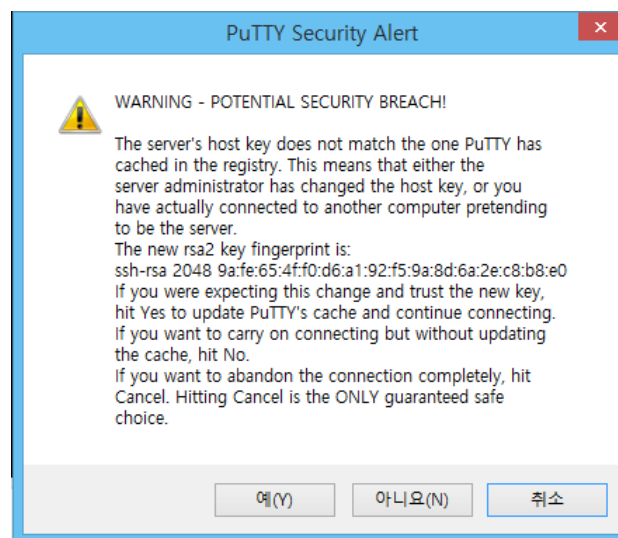


Figure 15 Security Alert

그러면 Figure 16과 같이 로그인 창이 뜨는데 root라고 입력한 뒤 Enter 버튼을 누르면 (로그인 시간이 대략 20초 정도 소요될 수 있다.)

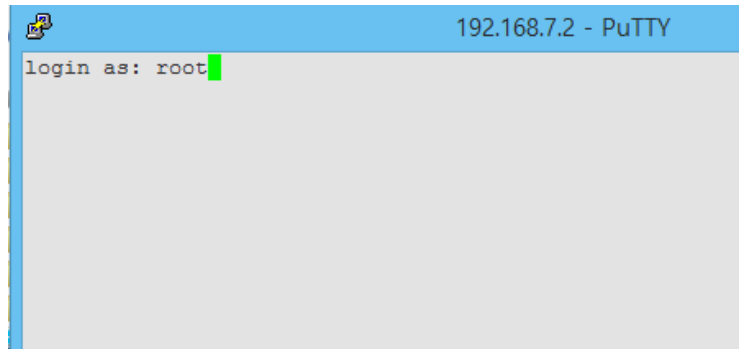


Figure 16 Login

Figure 17와 같이 로그인이 된다.

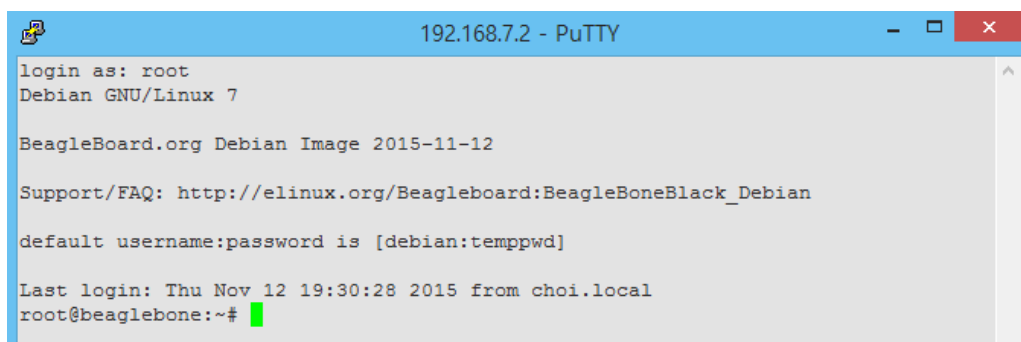


Figure 17 After Login as root

여기에 바로 Figure 18과 같이 `dd if=/dev/zero of=/dev/mmcblk1 bs=1M count=1`라고 입력한다.

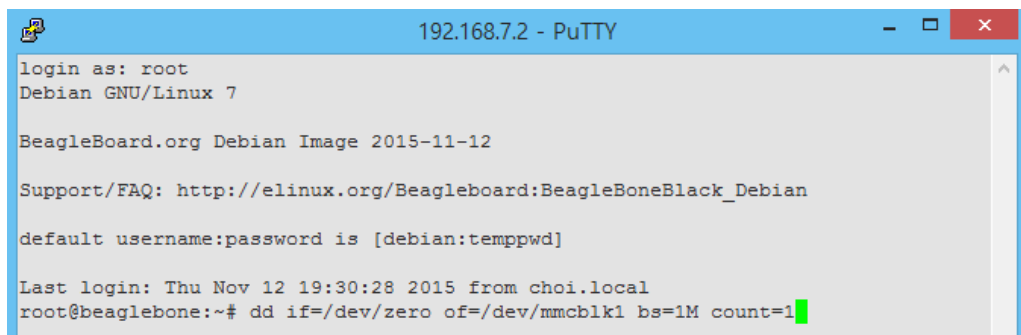
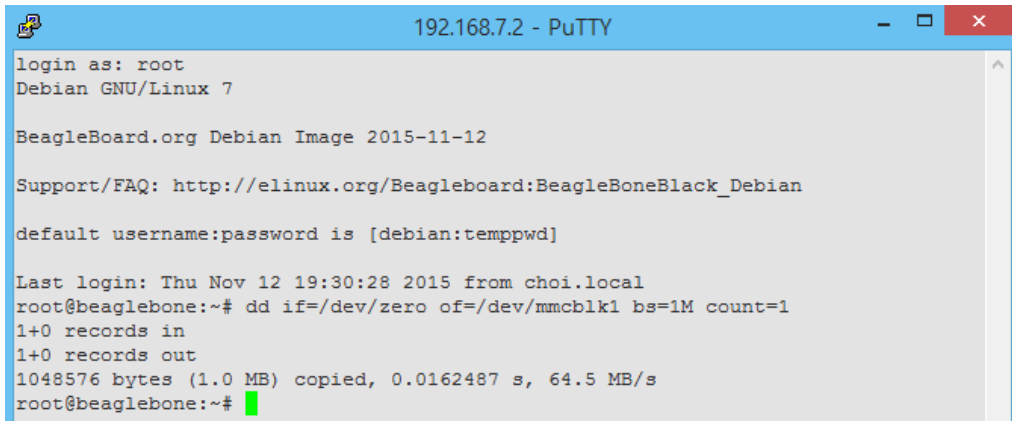


Figure 18 Disabling eMMC

그러면 Figure 19와 같이 작업이 완료된 것을 확인할 수 있다.



```
192.168.7.2 - PuTTY
login as: root
Debian GNU/Linux 7

BeagleBoard.org Debian Image 2015-11-12

Support/FAQ: http://elinux.org/Beagleboard:BeagleBoneBlack_Debian

default username:password is [debian:temppwd]


Last login: Thu Nov 12 19:30:28 2015 from choi.local
root@beaglebone:~# dd if=/dev/zero of=/dev/mmcblk1 bs=1M count=1
1+0 records in
1+0 records out
1048576 bytes (1.0 MB) copied, 0.0162487 s, 64.5 MB/s
root@beaglebone:~#
```

Figure 19 eMMC Disabled

이제 BBB의 전원을 끈다 (poweroff라고 입력하면 꺼진다).

5.1.3. Copying Debian on a Micro-SD Card

복사(설치)할 Debian OS 파일은 “첨부4_debian_SNUICP” 폴더 안에 위치한 “debian_kriso_20170406.7z” 이다. 이 파일은 압축되어 있으므로 압축을 풀어야 한다. 압축은 7zip을 이용해 풀 수 있으며 이는 <http://www.7-zip.org/>에서 다운받을 수 있다. 압축을 풀면 “debian_kriso_20170406.img” 이 생성된다. 이를 u-SD에 복사해야 하는데 이를 위해 Win32 Disk Imager (이하 Win32DI)를 사용한다. 이는 <https://sourceforge.net/projects/win32diskimager>에서 다운받을 수 있다.

포맷된 u-SD (용량 8 GB 이상, Class 10)를 컴퓨터에 연결하고 Win32DI를 실행시킨 뒤에  버튼을 눌러 “debian_kriso_20170406.img” 를 선택한다. 그리고 Device를 u-SD의 경로로 설정한다 (Figure 20 참고).

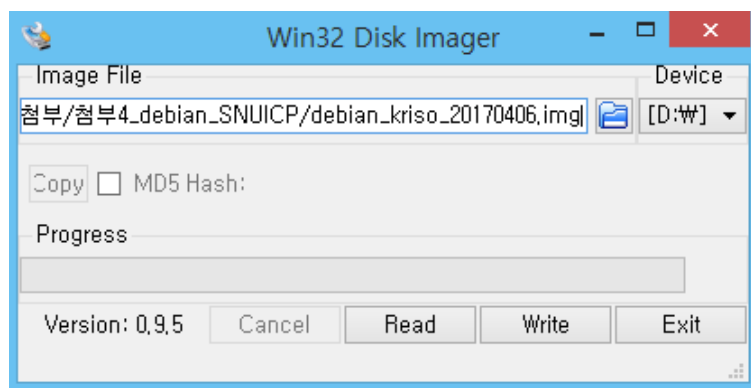


Figure 20 Win32 Disk Imager

Figure 21과 같은 창이 뜨면 Yes를 클릭한다.

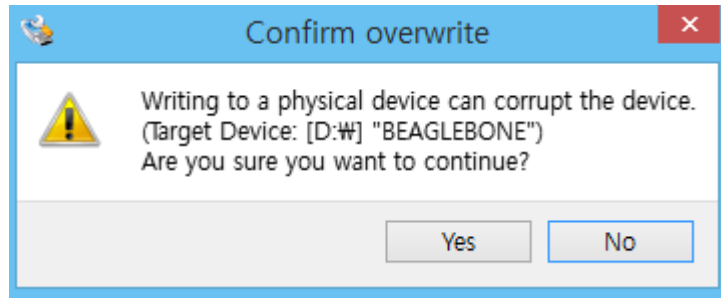


Figure 21 Confirm Overwrite

만약 u-SD포맷이 되지 않았거나 용량이 8 GB보다 작은 u-SD를 사용하는 경우 Figure 22과 같은 에러가 발생할 수 있다.

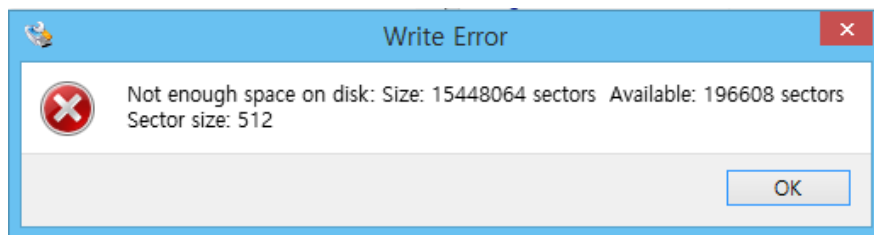


Figure 22 Write Error

별 문제가 없다면 Figure 23과 같이 복사가 진행된다. Progress가 모두 완료되면 Exit 버튼을 누르고 u-SD를 BBB의 u-SD Slot에 설치한다.

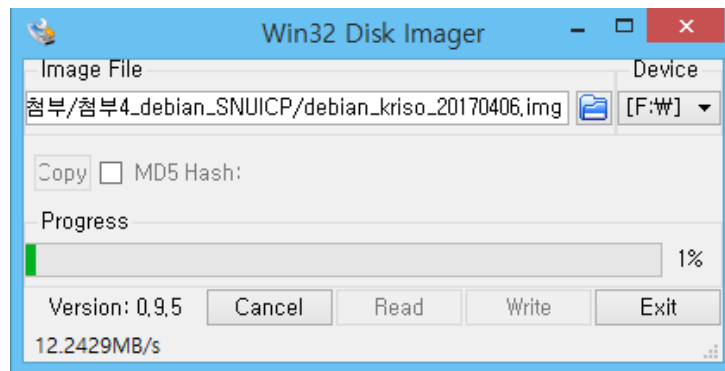


Figure 23 Win32 Disk Imager. Copying Operating System to a Micro SD

5.2. Cable 연결

케이블 연결은 Figure 24를 참고해 연결한다. 이 때 Laptop과 SNU ICP 사이의 USB 케이블 연결은 하지 않으며 Barrel Connector를 통해 전원을 공급한다. JP1~JP4 연결 시 반드시 핀맵(Figure 7)을 꼭 참고한다. 그리고 Barrel Connector는 항상 다른 케이블들이 모두 연결된 상태에서 연결한다.

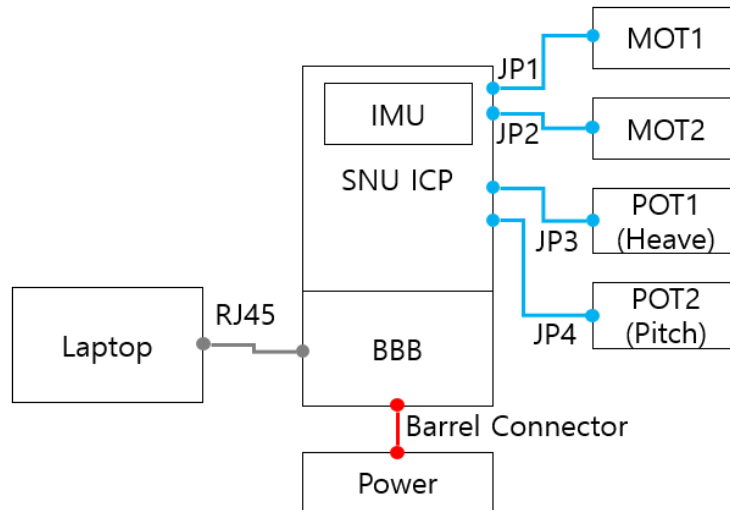


Figure 24 Cable Connection Diagram

5.3. Code 백업

Section 4에서 살펴본 코드들은 BBB내부에 저장되어 있다. 이 코드들을 백업하는 방법을 소개하겠다. 코드 백업을 위해 FileZilla를 이용한다. 이는 <https://filezilla-project.org/> 에서 받을 수 있다. FileZilla는 다른 PC에 접속하여 파일 주고받기를 할 수 있게 만든 툴이다. FileZilla를 실행시키면 Figure 25와 같은 화면이 활성화되는데, “호스트” 란에는 BBB의 IP주소를 입력한다. 이 때, BBB는 부팅이 완료된 상태여야 한다. USB케이블을 사용해 BBB와 Laptop을 연결할 경우 “호스트” 란에 192.168.7.2를 입력하며 RJ45 Ethernet Port를 통해 BBB와 Laptop을 연결할 경우 192.168.0.79를 입력한다. “사용자명” 란에는 root를 입력한다. “포트” 란에는 22를 적는다. 그 후에 빠른 연결을 누른다.

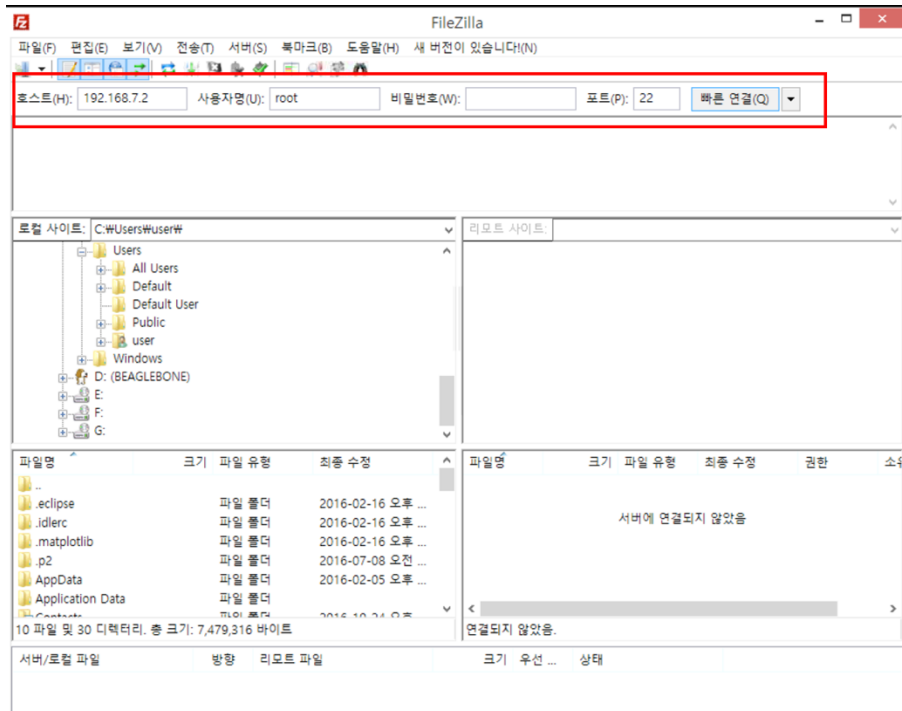


Figure 25 FileZilla 1

그러면 Figure 26와 같이 BBB와 연결이 이루어지고 실선 네모 상자 안처럼 BBB의 파일 디렉토리가 표시된다. “리모트 사이트”란에 `/home/debian/Desktop/IntercepCtrl`를 입력하면 (혹은 마우스를 이용해 해당 폴더를 선택하면) 점선 네모 상자 안처럼 section 4에서 살펴본 코드들이 들어있는 것을 확인할 수 있다. 이 중에 원하는 파일을 선택한 뒤 드래그 앤 드롭(실선 네모 안에 원하는 폴더로 드롭)을 통해 파일을 Laptop으로 복사할 수 있다. 같은 원리로 laptop에 있는 파일을 BBB로 전송할 수 있다.

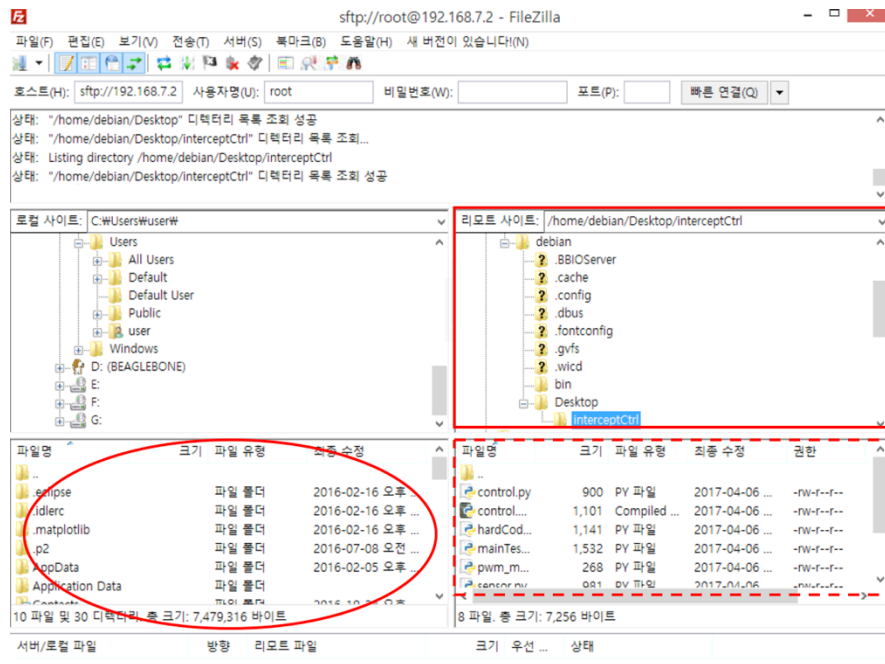


Figure 26 FileZilla 2

5.4. Code 수정

BBB에 들어있는 코드를 수정하는 방법을 살펴보겠다. 코드를 수정할 수 있는 방법은 크게 세 가지가 있다.

- Debian에 내장된 Editor 이용
- Laptop에 내장된 Editor에서 코드를 수정한 뒤 FileZiller를 이용해 BBB로 코드 전송
- Laptop에 내장된 Editor에서 BBB에 접속한 뒤 코드 수정

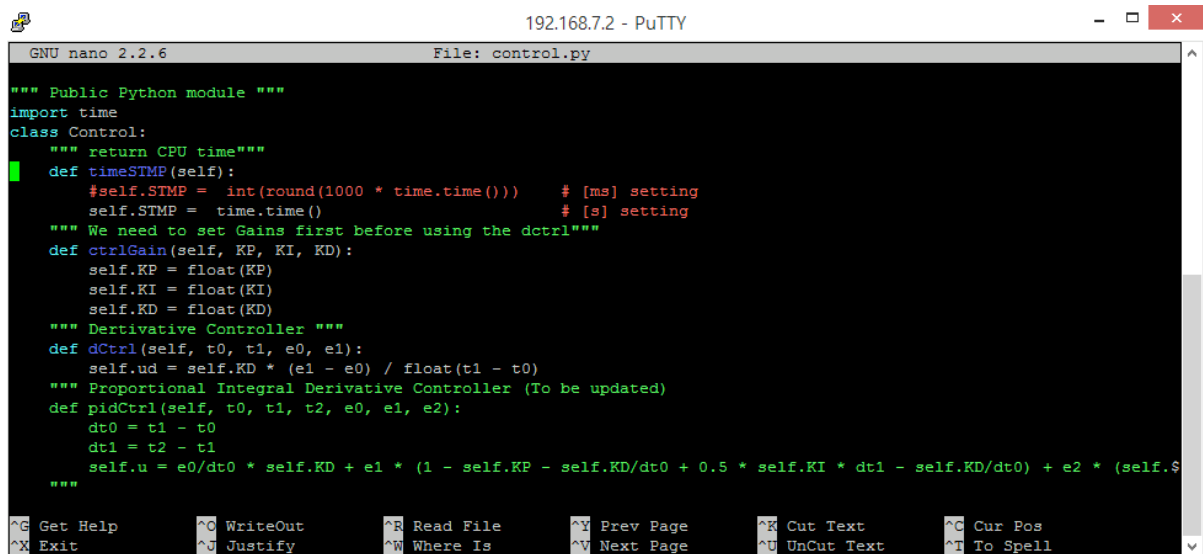
여기서는 첫 번째 방법을 살펴보겠다. 첫 번째 방법은 Debian에 접속만 할 수 있으면 laptop에 관계없이 코드를 수정할 수 있는 장점이 있다.

Putty를 사용해 BBB에 접속한 뒤 (5.1.2 참고) 코드가 있는 곳으로 이동하자. 이동에 사용된 명령어는 `cd /home/debian/Desktop/interceptCtrl`이다. 그리고 해당 디렉토리의 파일들을 조회하는 `ls` 명령어를 입력하면 Figure 1과 같이 파일들이 조회되는 것을 확인할 수 있다.

```
root@beaglebone:/home/debian/Desktop/interceptCtrl# ls
ADCTest.py      sensor.py      staticTestS20.py  waveControl.py
control.py      sensorTest.py  staticTestS40.py
controlTest.py  staticTestS0.py  staticTestS60.py
motorTest.py    staticTestS100.py  staticTestS80.py
```

Figure 27 List files

이들 중에 “control.py” 를 수정해 보자. nano control.py를 입력하면 Figure 28과 같이 “control.py” 가 nano Editor로 열린 것을 확인할 수 있다.



```
192.168.7.2 - PuTTY
GNU nano 2.2.6 File: control.py

""" Public Python module """
import time
class Control:
    """ return CPU time"""
    def timeSTMP(self):
        #self.STMP = int(round(1000 * time.time())) # [ms] setting
        self.STMP = time.time() # [s] setting
    """ We need to set Gains first before using the dctrl"""
    def ctrlGain(self, KP, KI, KD):
        self.KP = float(KP)
        self.KI = float(KI)
        self.KD = float(KD)
    """ Dertivative Controller """
    def dCtrl(self, t0, t1, e0, e1):
        self.ud = self.KD * (e1 - e0) / float(t1 - t0)
    """ Proportional Integral Derivative Controller (To be updated) """
    def pidCtrl(self, t0, t1, t2, e0, e1, e2):
        dt0 = t1 - t0
        dt1 = t2 - t1
        self.u = e0/dt0 * self.KD + e1 * (1 - self.KP - self.KD/dt0 + 0.5 * self.KI * dt1 - self.KD/dt0) + e2 * (self.KI * dt1)
    """ """

^G Get Help      ^O WriteOut      ^R Read File      ^Y Prev Page      ^K Cut Text      ^C Cur Pos
^X Exit          ^J Justify       ^W Where Is       ^V Next Page      ^U UnCut Text    ^T To Spell
```

Figure 28 control.py Opened by nano Editor

코드를 수정하는 방법은 메모장 사용법과 유사하다. 코드를 수정한 뒤 ctrl + O를 입력하면 저장된다. nano Editor를 종료하고 싶으면 ctrl + x를 입력한다.

5.5. Code 실행

코드의 실행 명령어는 python filename이다. 여기서 filename은 확장자를 포함한 코드 이름이다. 실행된 코드는 ctrl + c를 입력하면 중단된다.

5.6. 시험 후 MATLAB으로 시험 Data 불러오기

“waveControl.py” 와 “staticTestSx.py” 는 지속적으로 자료를 획득해 확장자 .dat 형식으로 저장한다 (4.4와 4.5참고). 시험 자료 분석을 위해 여러 툴을 사용할 수 있겠지만, 여기에서는 주로 쓰이는 MATLAB R2017a (MathWorks)에 시험 자료를 불러오는 방법을 살펴보겠다.

FileZillar를 이용해 .dat확장자로 저장된 시험 자료를 Laptop으로 옮긴다. 그리고 MATLAB을 실행시킨다. 옮긴 시험 자료를 드래그 앤 드롭으로 MATLAB의 Workspace (Figure 29 MATLAB, WorkspaceFigure 29의 점선 안)에 놓으면

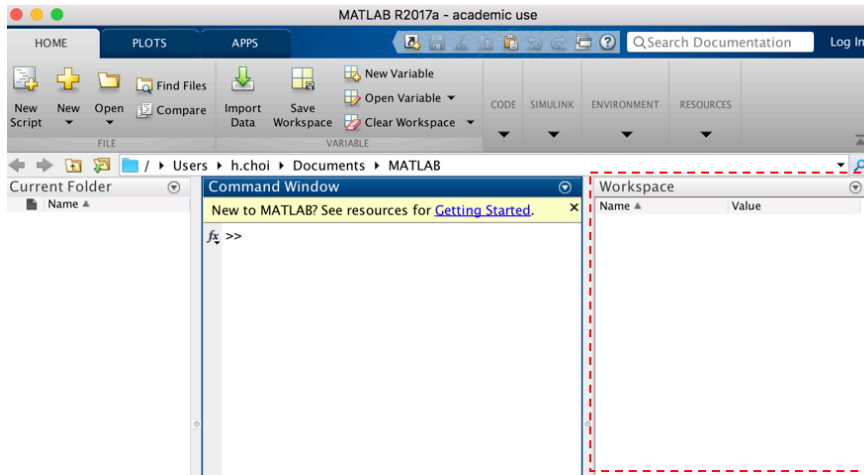


Figure 29 MATLAB, Workspace

Figure 30와 같은 창이 활성화된다. 이는 자료의 구조와 값을 보여준다. 이 때 import Selection(점선 안)을 클릭하면

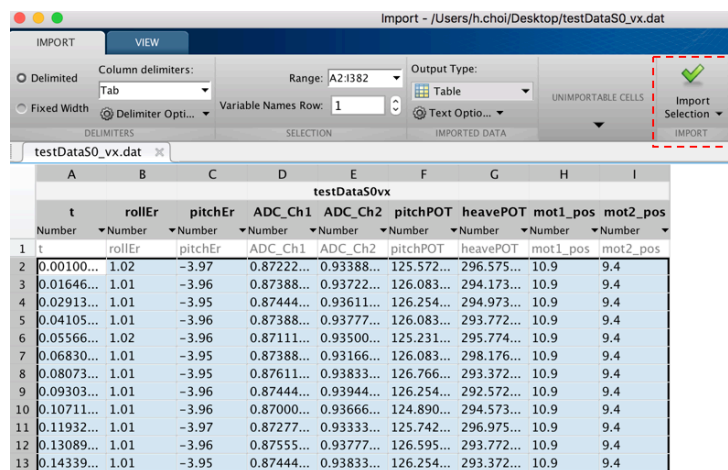


Figure 30 MATLAB, Import Data

Figure 31처럼 Workspace(점선 안)에 자료가 불러진 것을 확인할 수 있다.

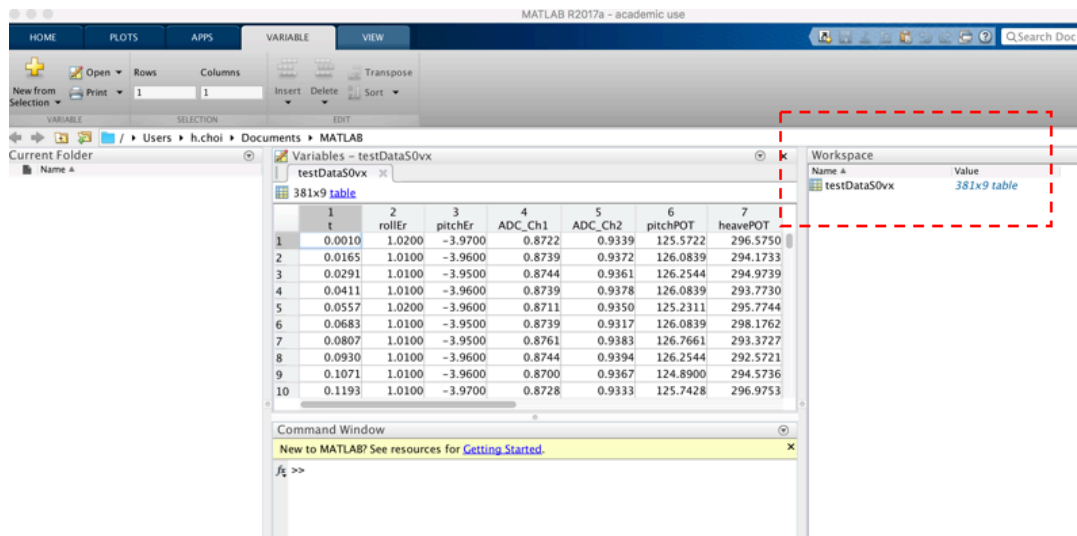


Figure 31 MATLAB, Imported Data