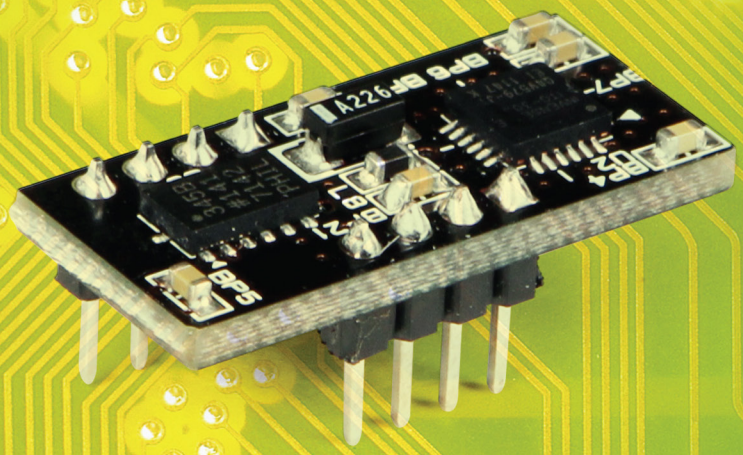
HOVIS Gyro/ Accelerometer Sensor DRI-0009

User Manual

- DRI-0009 가속도/자이로 센서 모듈
- 가속도/자이로 센서를 이용한 **'충격흡수'** 예제 따라하기
- 가속도/자이로 센서를 이용한 **'수평유지'** 예제 따라하기
- 가속도/자이로 센서를 이용한 **'제자리 보행 중 충격흡수'** 예제 따라하기



www.dongburobot.com

Dongbu Robot

CONTENTS

04p DRI-0009 가속도/자이로 센서 모듈
- 사양
- 특징
- 핀 배열정보
- 사용법
- Register 정보

07p 가속도/자이로 센서를 이용한 **'충격흡수'** 예제 따라하기

14p 가속도/자이로 센서를 이용한 **'수평유지'** 예제 따라하기

20p 가속도/자이로 센서를 이용한 **'제자리 보행 중 충격흡수'** 예제 따라하기

01 DRI-0009
가속도/자이로 센서 모듈

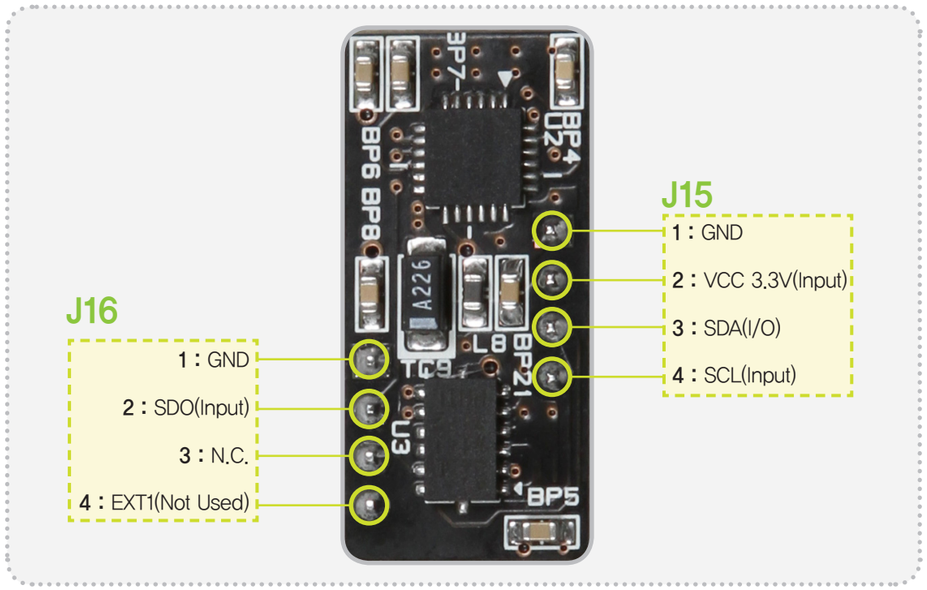
사양

무게 : 1.3g
크기 : 22.5 * 10.3x8.5mm
동작 온도 : -40~85℃
측정 범위 : -16g~+16g(가속도), -2000˚/s~+2000˚/s(자이로)
샘플링 주기 : 100Hz
공급 전압 : 2.1~3.6V
통신 방식 : Digital I^2C

특징

- HOVIS 제어기인 DRC-005T와 연결하여 가속도/자이로 센서의 값을 읽을 수 있습니다.
- X, Y, Z축 방향 가속도와 그 축을 중심으로 한 회전 각속도까지 읽을 수 있는 6축 센서입니다.
- 센서 데이터를 통해 DRC와 로봇이 어느 방향으로 얼마나 기울어져 있는지, 어느 방향으로 움직이고 있는지 측정할 수 있습니다.
- 3차원 상의 모든 축에 대해서 데이터를 얻을 수 있으므로, 로봇의 형태나 조립 상태에 상관 없이 측정이 가능합니다.
- 넘어짐 감지, 보행 보정, 중심 잡기, 충격 흡수 등의 알고리즘에 응용 가능합니다.

핀 배열 정보



사용법

DRC-005T의 윗면 커버를 벗기고 가속도/자이로 센서를 장착하면, 자동으로 DRC-005T가 센서를 인식하여 레지스터 맵의 Acc/Gyro Connected(RAM Register의 85번 Address) 항목이 0에서 1로 바뀌며, Acc X/Y/Z Value 및 Gyro X/Y/Z Value(RAM Register의 86번~97번 Address)가 100Hz의 샘플링 주기로 업데이트 됩니다.
사용자는 DR-Visual Logic 상의 Dynamics Sensor와 MPSU RAM Data의 ACCXVal~GYROZVal 항목을 활용해서 가속도/자이로 센서의 읽어 다양한 알고리즘에 응용할 수 있습니다.

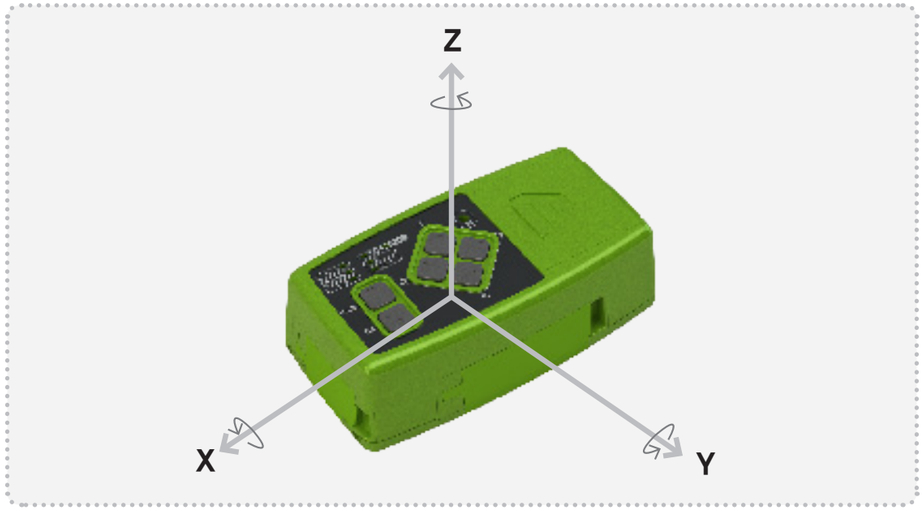
Register 정보

Addrress	구분	Bytes	유효범위	RW	비고
85	ACC/GYRO Connected	1	0~1	RO	가속도/자이로 센서 연결 여부
86	ACC X Value	2	-4096~4095	RO	가속도 센서 X축 Raw Data, 13bit
88	ACC Y Value	2	-4096~4095	RO	가속도 센서 Y축 Raw Data, 13bit
90	ACC Z Value	2	-4096~4095	RO	가속도 센서 Z축 Raw Data, 13bit
92	GYRO X Value	2	-32768~32767	RO	자이로 센서 X축 Raw Data, 16bit
94	GYRO Y Value	2	-32768~32767	RO	자이로 센서 Y축 Raw Data, 16bit
96	GYRO Z Value	2	-32768~32767	RO	자이로 센서 Z축 Raw Data, 16bit

Acc/Gyro Connected(RAM Register의 85번 Address)
가속도/자이로 센서 모듈이 장착되었는지를 나타내는 Flag입니다. 모듈이 장착되었다면 1이 되고, 장착되지 않은 상태라면 0의 값을 가집니다.

Acc X Value, Acc Y Value, Acc Z Value(RAM Register의 86번, 88번, 90번 Address)
가속도 센서의 X, Y, Z축 값의 Raw Data입니다. 가속도 센서는 현재 제어기에 작용하는 가속도 값을 측정합니다. 가속도 센서 각 축의 방향은 아래 그림과 같습니다. 가속도 센서의 각 축은 -4096~4095 의 값을 가질 수 있으며, 256이 1g(중력가속도, 9.8m/s2)에 해당합니다. 따라서 Raw Data를 g의 단위로 바꾸는 계산 식은 아래와 같습니다.

■ 가속도(g) = (Raw Data) / 256



Gyro X Value, Gyro Y Value, Gyro Z Value(RAM Register의 92, 94, 96번 Address)

자이로 센서의 X, Y, Z축 값의 Raw Data입니다. 자이로 센서는 현재 제어기가 돌아가는 회전 속도를 측정합니다. 오른손 엄지를 가속도 센서의 축의 방향으로 놓고, 오른손의 나머지 손가락을 접을 때의 방향이 해당 축의 (+) 회전 방향입니다. 다르게 말하면, 축 방향에서 원점을 내려다볼 때 반시계 방향이 (+) 회전 방향입니다. 자이로 센서의 각 축은 −32768~32767의 값을 가질 수 있으며, 약 16.38 이 1°/s(1초당 1° 회전하는 속도)에 해당합니다. 따라서 Raw Data를 °/s의 단위로 바꾸는 계산 식은 아래와 같습니다.

- 각속도(°/s) = (Raw Data) / 32768 X 2000

활용 하기 & 예제

가속도/자이로 센서를 사용해 넘어짐 감지, 보행 보정, 중심 잡기, 충격 흡수 등의 알고리즘을 구현 가능합니다. 가속도 센서 값을 사용한 넘어짐 감지는 DR-Visual Logic 예제의 acc.dts에 적용되어 있으며, 충격흡수, 수평 유지, 보행 관련한 예제는 다음 장을 참고하시면 됩니다.

02 가속도/자이로 센서를 이용한 '충격흡수' 예제 따라하기

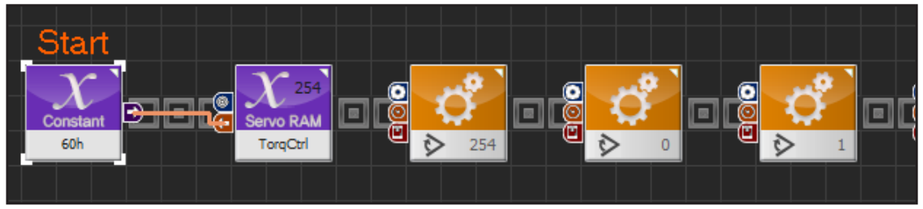
예제설명:

서 있는 상태의 로봇을 기울였다가 다시 원래대로 놓는 경우, 로봇이 관성에 의해서 반대쪽으로 기우뚱하면서 흔들리거나, 균형을 잡지 못하고 반대쪽으로 넘어지게 됩니다.
가속도/자이로 센서를 사용하여, 위와 같은 상황에서 로봇을 기울였다가 놓을 때 가속도 센서로 기울기를 측정한 후, 자이로 센서 값에 따라서 관절의 위치를 조정함으로써 충격을 흡수해 로봇이 넘어지지 않게 하는 예제입니다.
본 예제에서 각 Task파일은 다음과 같이 구성되어 있습니다. 프로그램 시작점은 main()함수입니다.

예제

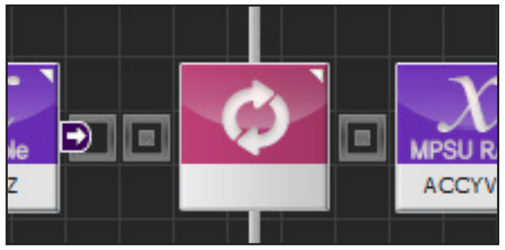
Task 파일	C-like 함수	설명
main.dts	main()	프로그램의 시작점이며, 다른 함수들을 포함합니다.

01



로봇을 초기화하고 차렷 자세로 보냅니다.
앞뒤로 기울어진 방향을 나타내는 TiltDirX 와, 좌우로 기울어진 방향을 나타내는 TiltDirZ 변수를 0으로 초기화합니다.
자이로 센서를 기준으로 하기 때문에 X를 앞뒤, Z를 좌우로 하였습니다.

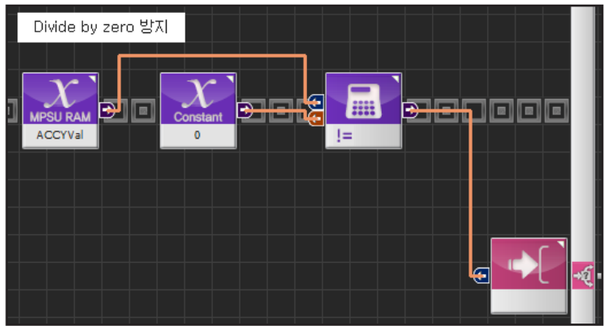
02



프로그램의 나머지 전체 부분을 감쌀 Loop 모듈로 들어갑니다. 이 Loop 문은 무한 반복되며 지속적으로 아래 과정을 반복합니다.

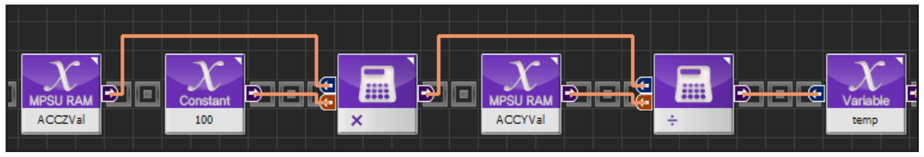
- 가속도 센서 값을 통해 앞뒤 기울기 추정(TiltDirX)
- 앞뒤 각속도(X축) 절대값 구하기(absGyroX)
- TiltDirX의 값과 absGyroX 값에 따라 모터 명령 내리기
- 가속도 센서 값을 통해 좌우 기울기 추정(TiltDirZ)
- 좌우 각속도(Z축) 절대값 구하기(absGyroZ)
- TiltDirZ의 값과 absGyroZ 값에 따라 모터 명령 내리기

03



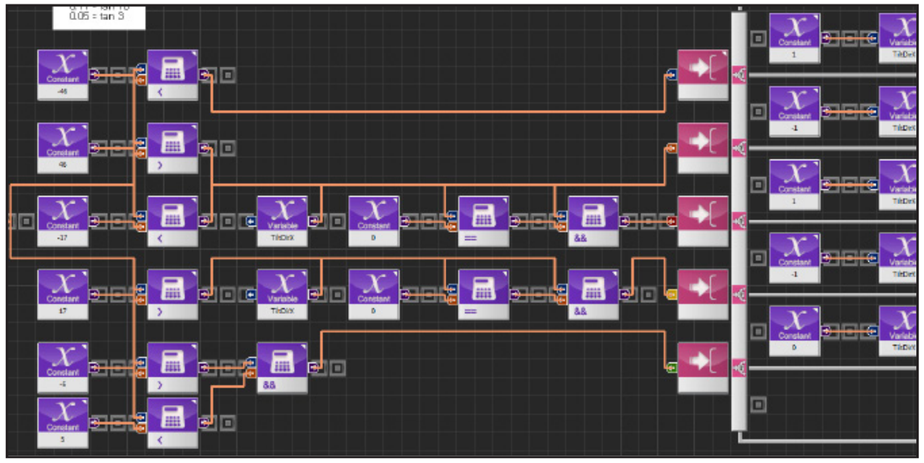
기울기를 추정할 때, Z축의 가속도 센서 값을 Y축의 가속도 센서 값으로 나눌 것입니다. 이때 0으로 나누는 오류를 방지하기 위해, Y축 가속도 센서 값을 읽어서 0이 아닌 지 검사 후 결과에 따라서 다른 루틴에 진입합니다.

04



Y축의 가속도 센서 값이 0이 아닌 경우입니다.
Z축의 가속도 센서 값을 Y축의 가속도 센서 값으로 나눕니다. 정수와 정수의 연산이므로 값이 1보다 작은 경우 0이 되기 때문에 이것을 방지하기 위해 미리 100으로 곱한 후 나눕니다. 나눈 값은 temp에 저장되어 다음 연산에 쓰입니다.
Y축 값은 서 있는 상태에서 −이며, Z값은 앞으로 기울어진 경우 +, 뒤로 기울어진 경우 −입니다.
따라서 temp 값은 앞으로 기울어진 경우 −, 뒤로 기울어진 경우 +가 됩니다.

※ 가속도/자이로 센서의 축 회전에 대한 자세한 설명은 part6의 "가속도/자이로 센서를 이용한 '보행' 예제 따라하기" 의 "Module 4-1"을 참조하시길 바랍니다.



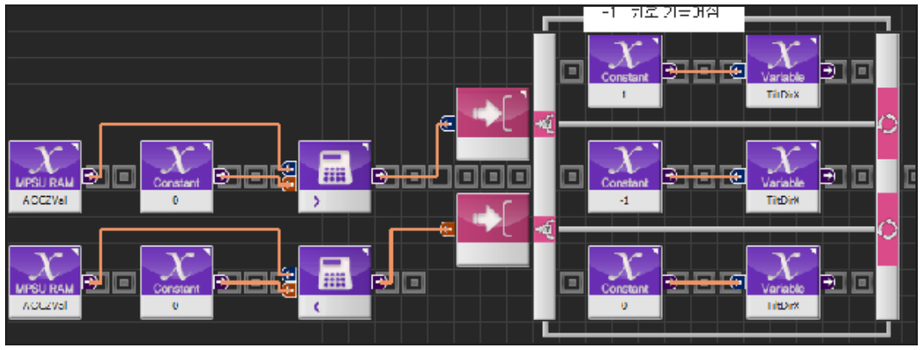
temp 값에 따라 기울기를 추정하여 TiltDirX 값을 바꿉니다.
: −46보다 작은 경우 약 25도 이상 앞으로 기울어져 있다는 뜻이므로 1로 바꿉니다.
: +46보다 큰 경우 약 25도 이상 뒤로 기울어져 있다는 뜻이므로 −1로 바꿉니다.
: −17보다 작고 TiltDirX 값이 0인 경우 약 10도 이상 앞으로 기울어져 있다는 뜻이므로 1로 바꿉니다.
: +17보다 크고 TiltDirX 값이 0인 경우 약 10도 이상 뒤로 기울어져 있다는 뜻이므로 −1로 바꿉니다.
: −5보다 크고 5보다 작은 경우 똑바로 서 있다는 뜻이므로 0으로 바꿉니다.

−17, +17일 때 TiltDirX 값을 검사하여 0일 때만 값을 바꾸는 이유는, 기존에 기울어져 있다가(TiltDirX가 1이나 −1) 놓았을 때 지면과의 반동으로 −17, +17 이상의 값이 들어올 수 있으므로 이때에는 값 변경을 하지 않기 위해서 입니다.

04번에 해당하는 C-like 소스 코드는 아래와 같습니다.

```
temp=( ( MPSU_ACCZVal * 100 ) / MPSU_ACCYVal )
if( ( temp < -46 ) )
{
        TiltDirX=1
}
else if( ( temp > 46 ) )
{
        TiltDirX=-1
}
else if( ( ( temp < -17 ) && ( TiltDirX == 0 ) ) )
{
        TiltDirX=1
}
else if( ( ( temp > 17 ) && ( TiltDirX == 0 ) ) )
{
        TiltDirX=-1
}
else if( ( ( temp > -5 ) && ( temp < 5 ) ) )
{
        TiltDirX=0
}
else
{
}
```

05



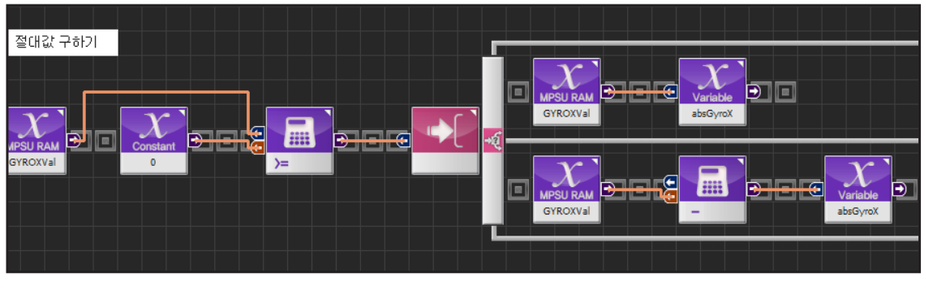
Y축 가속도 값이 0인 경우입니다. 이 경우 Z축 가속도 값만으로 기울기를 추정하여 TiltDirX 값을 바꿉니다.

: 0보다 큰 경우 앞으로 기울어져 있다는 뜻이므로 1로 바꿉니다.
: 0보다 작은 경우 뒤로 기울어져 있다는 뜻이므로 -1로 바꿉니다.
: 0인 경우 0으로 바꿉니다.

05번에 해당하는 C-like 소스 코드는 아래와 같습니다.

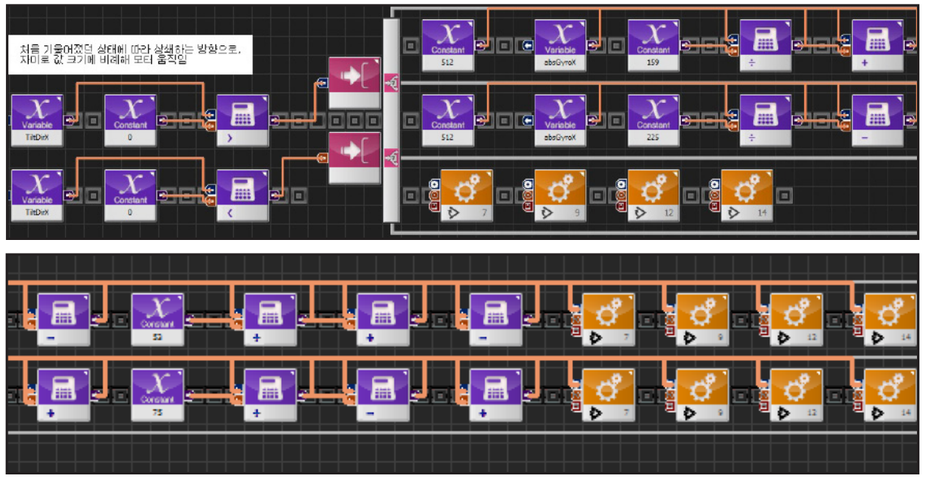
```
if( ( MPSU_ACCZVal > 0 ) )
{
        TiltDirX=1
}
else if( ( MPSU_ACCZVal < 0 ) )
{
        TiltDirX=-1
}
else
{
        TiltDirX=0
}
```

06



앞뒤 각속도(X축) 절대값을 구하여 absGyroX 에 저장합니다.

07



TiltDirX 의 값에 따라 방향을 결정짓고, absGyroX 의 값에 비례하여 각도를 결정지은 후 이에 따라 모터를 제어하여 충격을 흡수합니다.
앞으로 쏠렸다가 넘어지는 경우(TiltDirX 〉 0), 몸을 앞으로 기울여 뒤쪽으로 넘어가는 힘을 상쇄합니다.
뒤로 쏠렸다가 넘어지는 경우(TiltDirX 〈 0), 몸을 뒤로 기울여 앞쪽으로 넘어가는 힘을 상쇄합니다.

07번에 해당하는 C-like 소스 코드는 아래와 같습니다.

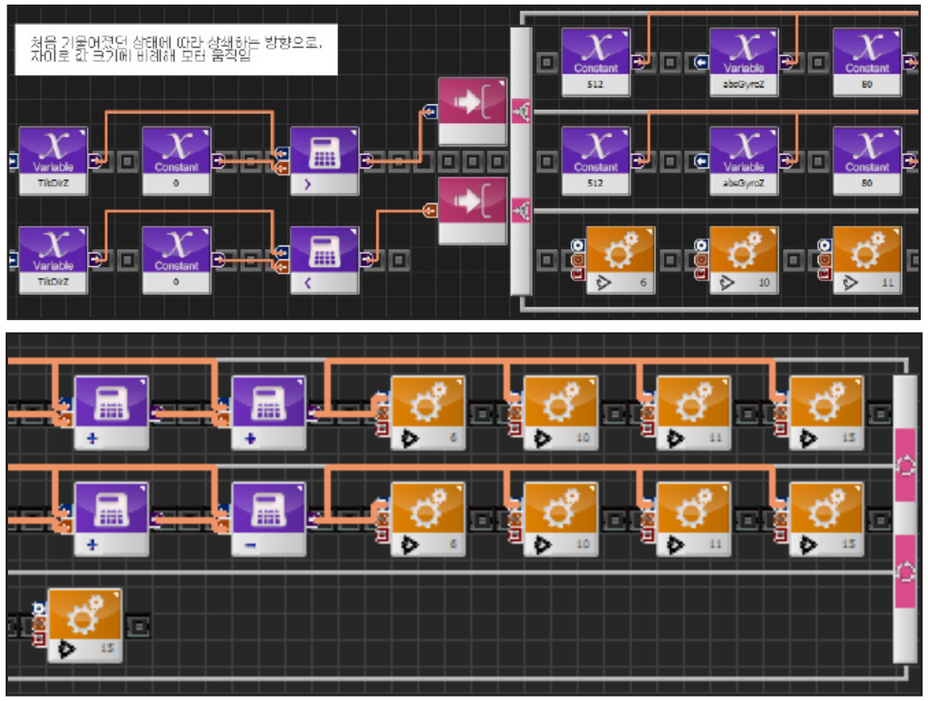
```
if( ( TiltDirX > 0 ) )
{
        jog( ( 512 + ( absGyroX / 159 ) ), 0, 7, 9 )
        jog( ( 512 + ( absGyroX / 53 ) ), 0, 9, 9 )
        jog( ( 512 - ( absGyroX / 159 ) ), 0, 12, 9 )
        jog( ( 512 - ( absGyroX / 53 ) ), 0, 14, 9 )
}
else if( ( TiltDirX < 0 ) )
{
        jog( ( 512 - ( absGyroX / 225 ) ), 0, 7, 9 )
        jog( ( 512 - ( absGyroX / 75 ) ), 0, 9, 9 )
        jog( ( 512 + ( absGyroX / 225 ) ), 0, 12, 9 )
        jog( ( 512 + ( absGyroX / 75 ) ), 0, 14, 9 )
}
else
{
        jog( 512, 0, 7, 9 )
        jog( 512, 0, 9, 9 )
        jog( 512, 0, 12, 9 )
        jog( 512, 0, 14, 9 )
}
```

08

지금까지는 앞뒤의 충격 흡수에 대해서 다루었습니다. 좌우의 충격 흡수도 마찬가지입니다.
03번~06번까지의 내용을 완전히 동일하게 적용하되, 가속도 센서의 축을 다르게 사용하면 됩니다.
좌우 충격 흡수에는 가속도 센서의 X축 값과 Y축 값이 사용됩니다.
Y축 값은 서 있는 상태에서 –이며, X축 값은 오른쪽으로 기울어진 경우 +, 왼쪽으로 기울어진 경우 –입니다.
따라서 04번에서 temp 값은 오른쪽으로 기울어진 경우 –, 왼쪽으로 기울어진 경우 +가 됩니다.

03번~06번을 동일하게 구성하여, 가속도 센서의 X, Y축 값을 사용해 좌우 기울기(TiltDirZ)를 결정하고 자이로 센서의 Z축 절대값(absGyroZ)를 구합니다.

09



TiltDirZ 의 값에 따라 방향을 결정짓고, absGyroZ 의 값에 비례하여 각도를 결정지은 후 이에 따라 모터를 제어하여 충격을 흡수합니다.
오른쪽으로 쏠렸다가 넘어지는 경우(TiltDirZ 〉 0), 몸을 오른쪽으로 기울여 왼쪽으로 넘어가는 힘을 상쇄합니다.
왼쪽으로 쏠렸다가 넘어지는 경우(TiltDirZ 〈 0), 몸을 왼쪽으로 기울여 오른쪽으로 넘어가는 힘을 상쇄합니다.

09번에 해당하는 C-like 소스 코드는 다음과 같습니다.

```
if( ( TiltDirZ > 0 ) )
{
        jog( ( 512 + ( absGyroZ / 80 ) ), 0, 6, 9 )
        jog( ( 512 + ( absGyroZ / 80 ) ), 0, 10, 9 )
        jog( ( 512 + ( absGyroZ / 80 ) ), 0, 11, 9 )
        jog( ( 512 + ( absGyroZ / 80 ) ), 0, 15, 9 )
}
else if( ( TiltDirZ < 0 ) )
{
        jog( ( 512 - ( absGyroZ / 80 ) ), 0, 6, 9 )
        jog( ( 512 - ( absGyroZ / 80 ) ), 0, 10, 9 )
        jog( ( 512 - ( absGyroZ / 80 ) ), 0, 11, 9 )
        jog( ( 512 - ( absGyroZ / 80 ) ), 0, 15, 9 )
}
else
{
        jog( 512, 0, 6, 9 )
        jog( 512, 0, 10, 9 )
        jog( 512, 0, 11, 9 )
        jog( 512, 0, 15, 9 )
}
```

10

완성된 Dts 파일을 컴파일 한 후 다운로드하여 실행합니다.
로봇이 차렷 자세를 취합니다.
로봇을 전후 또는 좌우로 기울였다가 놓으면 반동을 흡수하며 다시 제자리에 서는 것을 관찰 할 수 있습니다.

03 가속도/자이로 센서를 이용한 '수평유지' 예제 따라하기

예제설명:
로봇균형잡기 예제는 로봇이 흔들리는 표면에서 스스로 균형을 유지하는 Task 입니다. 이 예제를 통해서 Lowpass 필터를 통한 가속도 센서 값의 획득과 로봇 균형 유지를 배울 수 있습니다.
본 예제는 다음과 같은 Task파일이 있습니다. 각 Task파일은 다음 함수와 1:1 매칭되며, 프로그램 시작점은 main()함수입니다.

예제

Task 파일	C-like 함수	설명
main.dts	main()	프로그램의 시작점이며, 다른 함수들을 포함합니다.
basic_pose.dts	basic_pose()	로봇이 균형을 잡기 위한 기본자세를 취합니다.
calibration.dts	calibration()	로봇이 기본자세를 취한 후, 100번에 걸쳐 가속도 센서 값을 읽고 이것의 평균값을 저장합니다. 이 값은 향후 가속도 값 보정에 사용됩니다.
filter.dts	filter()	x축 가속도 값의 두 번째 전 값, 이전 값, 그리고 현재 값을 1 : 3 : 6 으로 반영하여 값을 얻는 필터 함수입니다.
filter2.dts	filter2()	y축 가속도를 위와 같은 방법으로 얻는 필터 함수입니다.
move.dts	move()	가속도 값을 이용하여 오른쪽 다리의 균형을 유지하기 위해 실제 모터를 제어하는 함수입니다. (7, 8, 9번 모터)
move2.dts	move2()	위와 같으며, 왼쪽 다리에 해당합니다. (12, 13, 14번 모터)
a2e.dts	a2e()	모터를 제어하기 위하여 각도의 값(-166.7~166.7도)을 0~1023 값으로 변경합니다.

본 예제의 흐름은 다음과 같습니다.

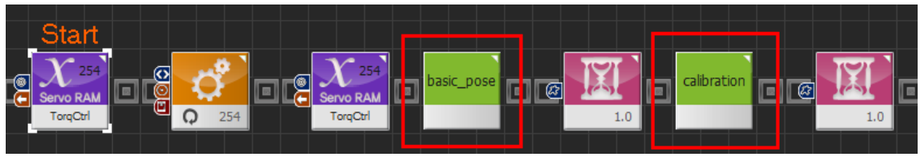
01. 수평 제어 준비
기본자세 및 Calibration – **basic_pose(), calibration()**

02. 가속도 센서 값 획득
Lowpass 필터를 통한 가속도 X축과 Z축 값 획득 – **filter(), filter2()**
* X축은 로봇의 좌우 기울임을 측정하며, Z축은 로봇의 전후 기울임을 측정합니다.

03. 로봇 수평자세 보정
획득한 가속도의 값을 따라 로봇 자세 제어 ※ 01번이 완료되면 02번~03번은 무한반복을 합니다.

04. 로봇 실행

01 수평 제어 준비



수평 제어 준비 단계에서는 로봇이 무릎을 45도 구부린 자세로 수평을 잡는 기본자세를 취합니다.
– basic_pose()

로봇이 기본자세를 잡고 나면 현재 자세에서의 가속도 센서 값을 100번 획득하여 평균값을 얻습니다. 이 값은 향후 가속도 센서 값을 보정하기 위하여 사용됩니다.
– calibration()

– basic_pose()
: 각 서보를 제어하여 정자세로 무릎을 굽히고 균형을 잡을 준비를 합니다.

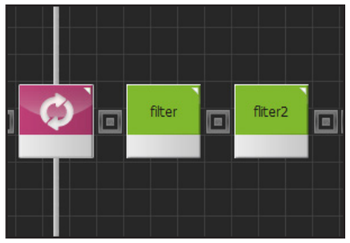
– calibration()
: 가속도 x와 z축에 대하여 각 100번씩 읽어 이에 대한 평균값을 cali_accX와 cali_accZ에 저장합니다.
: cali_accX와 cali_accZ는 향후 가속도 값 보정에 이용됩니다.

```
85  void basic_pose()
86  {
87          jog( 512, 0, 254, 100 )
88          jog( 235, 0, 0, 100 )
89          jog( 789, 0, 3, 100 )
90          jog( 235, 0, 1, 100 )
91          jog( 789, 0, 4, 100 )
92          jog( 378, 0, 7, 100 )
93          jog( 650, 0, 12, 100 )
94          jog( 235, 0, 8, 100 )
95          jog( 789, 0, 13, 100 )
96          jog( 650, 0, 9, 100 )
97          jog( 378, 0, 14, 100 )
98  }
```

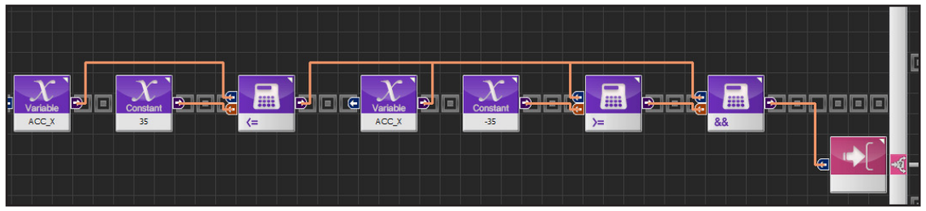
```
61  void calibration()
62  {
63          prev_x=0
64          prev_x2=0
65          prev_z=0
66          prev_z2=0
67          cali_accX=0
68          cali_accZ=0
69          for( i = 1 ~ 100 )
70          {
71                  prev_x2=prev_x
72                  prev_z2=prev_z
73                  cali_accX=( MPSU_ACCXVal + cali_accX )
74                  cali_accZ=( MPSU_ACCZVal + cali_accZ )
75                  prev_x=MPSU_ACCXVal
76                  prev_z=MPSU_ACCZVal
77                  for( j = 0 ~ 100 )
78                  {
79                  }
80          }
81          melody( 1 )
82          cali_accX=( cali_accX / 100 )
83          cali_accZ=( cali_accZ / 100 )
84  }
```

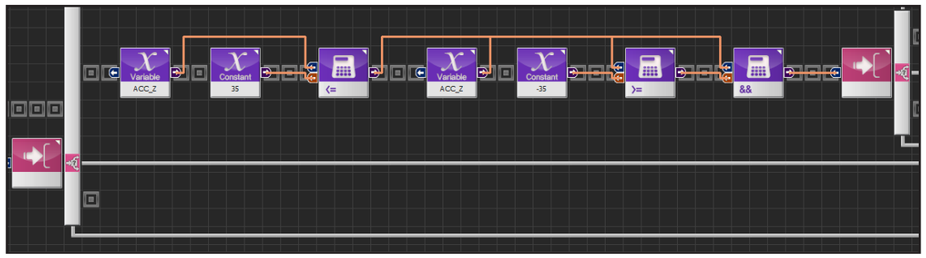
02 가속도 센서 값 획득

```
void fliter2()
{
        ACC_Z=( ( ( prev_z2 * 1 ) + ( ( ( MPSU_ACCZVal - cali_accZ ) * 6 ) + ( prev_z * 3 ) ) ) / 10 )
        prev_z2=prev_z
        prev_z=ACC_Z
}
void filter()
{
        ACC_X=( ( ( prev_x2 * 1 ) + ( ( ( MPSU_ACCXVal - cali_accX ) * 6 ) + ( prev_x * 3 ) ) ) / 10 )
        prev_x2=prev_x
        prev_x=ACC_X
}
```



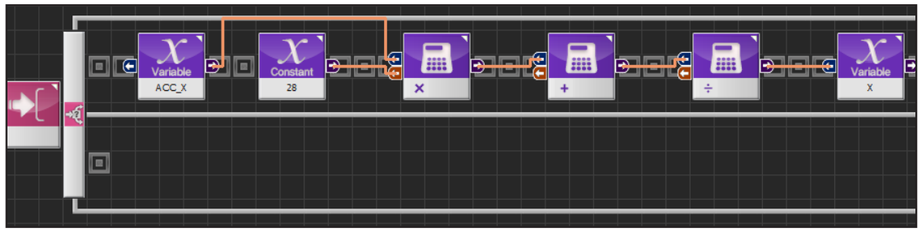
02번 가속도 센서 값 획득과 03번 로봇의 수평자세 보정은 무한반복 됩니다.
filter(), filter2()는 각각 가속도 x와 z축의 값을 획득합니다.
filter2()에서 prev_z2와 prev_z는 두번째 전과 바로 직전에 측정된 z축 가속도 값입니다. 그리고 MPSU_ACCZVal은 현재 z축 가속도 값이며 이 값은 앞의 calibration()에 의해 구해진 cali_accZ에 의해 값이 보정됩니다.
실제로 우리가 사용할 ACC_Z는 두번째 전 측정 값 부터 현재 측정 값까지 서로 다른 가중치로 적용하여 구합니다. 예제에서는 두번째 전 측정 값을 1/10, 이전 값을 3/10 그리고 현재 값을 6/10의 가중치로 반영합니다. 최근 값에 더 높은 신뢰를 주어 가속도 변화에 민감하게 반응하면서도 이전 값을 적절하게 반영하여 값을 안정적으로 관리하는 필터 기법을 큰 범위에서 Lowpass필터라고 할 수 있습니다.
실제로 자세 보정에 사용되는 값은 ACC_X와 ACC_Z입니다.





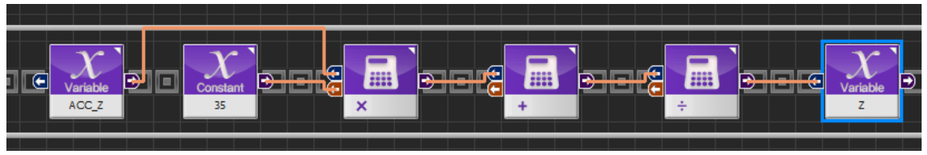
획득한 가속도 센서 값이 너무 작거나 커서 로봇이 제어불능 상태에 빠지지 않도록 합니다.
(-35 ~ +35 이상의 값은 에러로 취급하여 무시하도록 코딩)

03 로봇 수평 자세 보정



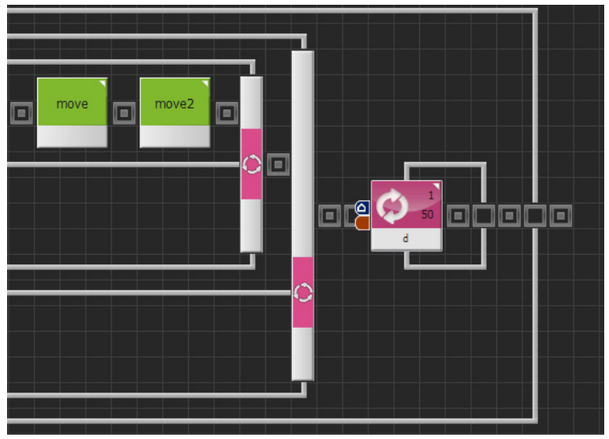

```
void main()
{
        SERVO_TorqCtrl[254]=0x0
        jog( 0, 1, 254, 60 )
        SERVO_TorqCtrl[254]=0x60
        basic_pose()
        delay( 1000 )
        calibration()
        delay( 1000 )
        while( true )
        {
                filter()
                fliter2()
                if( ( ( ACC_X <= 35 ) && ( ACC_X >= -35 ) ) )
                {
                        if( ( ( ACC_Z <= 35 ) && ( ACC_Z >= -35 ) ) )
                        {
                                X=( ( ( ACC_X * 28 ) + 50 ) / 100 )
                                Z=( ( ( ACC_Z * 35 ) + 50 ) / 100 )
                                move()
                                move2()
                        }
                        else
                        {
                        }
                }
                else
                {
                }
                for( d = 1 ~ 50 )
                {
                }
        }
}
```

ACC_X의 값을 이용하여 자세 보정에 이용하도록 적정한 값을 실험적인 방법으로 적용합니다.
실험적인 방법이란 여러 번의 실험을 통해 최적화된 결과를 획득하는 방법입니다.
X는 실제 자세 보정에 사용될 값이며, ACC_X의 28/100에 해당합니다. VL에서는 실수 연산이 되지 않아 28을 곱하고 반올림을 위하여 50을 더한 후 100으로 나누어 줍니다.
위쪽은 X를 계산하기 위한 부분이며, C-like로 보면 아래쪽과 같습니다.



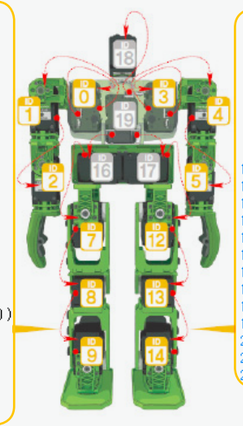

```
99  void main()
100 {
101         SERVO_TorqCtrl[254]=0x0
102         jog( 0, 1, 254, 60 )
103         SERVO_TorqCtrl[254]=0x60
104         basic_pose()
105         delay( 1000 )
106         calibration()
107         delay( 1000 )
108         while( true )
109         {
110                 filter()
111                 fliter2()
112                 if( ( ( ACC_X <= 35 ) && ( ACC_X >= -35 ) ) )
113                 {
114                         if( ( ( ACC_Z <= 35 ) && ( ACC_Z >= -35 ) ) )
115                         {
116                                 X=( ( ( ACC_X * 28 ) + 50 ) / 100 )
117                                 Z=( ( ( ACC_Z * 35 ) + 50 ) / 100 )
118                                 move()
119                                 move2()
120                         }
121                         else
122                         {
123                         }
124                 }
125                 else
126                 {
127                 }
128                 for( d = 1 ~ 50 )
129                 {
130                 }
131         }
132 }
```

이전과 마찬가지로 ACC_Z의 값을 이용하여 자세 보정에 이용하도록 적정한 값을 실험적인 방법으로 적용합니다.
Z는 실제 자세 보정에 사용될 값이며, ACC_Z의 35/100에 해당합니다. VL에서는 실수 연산이 되지 않아 28을 곱하고 반올림을 위하여 50을 더한 후 100으로 나누어 줍니다.
위쪽은 Z를 계산하기 위한 부분이며, C-like로 보면 아래쪽과 같습니다.



가속도 ACC_X와 ACC_Z으로 부터 구해진 X, Z을 이용하여 로봇의 모터를 실제로 동작시켜 자세를 보정합니다.

```
27  void move()
28  {
29          nValue=( ( -45 + X ) + Z )
30          a2e()
31          jog( nValue, 0, 7, 150 )
32          nValue=( ( 45 - X ) - Z )
33          a2e()
34          jog( nValue, 0, 9, 150 )
35          nValue=( ( -45 + X ) * 2 )
36          a2e()
37          if( ( nValue > 280 ) )
38          {
39                  nValue=280
40          }
41          else
42          {
43                  jog( nValue, 0, 8, 150 )
44          }
45          nValue=( ( X * 3 ) * -1 )
46          a2e()
47          jog( nValue, 0, 10, 150 )
48  }
```




```
1   void move2()
2   {
3           nValue=( ( 45 + X ) - Z )
4           a2e()
5           jog( nValue, 0, 12, 150 )
6           nValue=( ( -45 - X ) + Z )
7           a2e()
8           jog( nValue, 0, 14, 150 )
9           nValue=( ( 45 + X ) * 2 )
10          a2e()
11          if( ( nValue < 750 ) )
12          {
13                  nValue=750
14          }
15          else
16          {
17                  jog( nValue, 0, 13, 150 )
18          }
19          nValue=( X * -3 )
20          a2e()
21          jog( nValue, 0, 15, 150 )
22  }
```

각각 move()와 move2()는 7,8,9번 모터와 12,13,14번 모터를 움직입니다.
다음 소스에서 a2e() 는 모터의 angle(-166.7도 ~ 166.7도)를 0~1023의 pos값으로 바꾸는 역할을 합니다.

04 로봇 실행

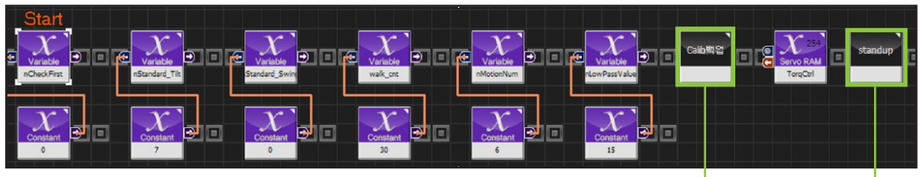
완성된 dts 파일을 컴파일 한 후 다운로드합니다.
로봇을 판자 위에 올리고 DRC의 Mode 버튼을 눌러 Task 를 실행합니다.
로봇이 무릎을 구부린 기본자세를 취합니다. 이후 약 5초간 Calibration이 진행되므로 로봇을 움직이지 않습니다.
비프음이 들리면, 판자를 전후좌우로 비틀며 로봇이 균형을 잡는 것을 확인합니다.

04 가속도/자이로 센서를 이용한 '제자리 보행 중 충격흡수' 예제 따라하기

예제

Task 파일	C-like 함수	설명
main.dts	main()	프로그램의 메인 코드로 전체적인 동작 흐름제어가 프로그래밍 되어 있습니다.
Calib백업.dts	Calib백업()	전체 모터의 캘리브레이션 값을 읽은 후 백업해 두는 소스입니다.
Standup.dts	Standup()	차렷자세로 로봇을 움직이는 소스로 로봇 동작이 멈출 때까지 이 모듈에서 멈춰있게 됩니다.
WT_AccGyro.dts /WM_AccGyro.dts	WT_AccGyro() /WM_AccGyro()	시간(WT)대기 혹은 동작(WM)대기 함수로 동작을 대기 하면서 가속도 및 자이로 센서값을 읽어 캘리브레이션 값을 보정해 주는 함수입니다.
drAccGyro.dts	drAccGyro()	가속도 및 자이로 센서를 읽어서 필터링된 값을 반환하는 모듈입니다.
drAcc.dts	drAcc()	가속도 센서의 값을 읽어서 각도값으로 환산하여 주는 함수입니다.
drGyro.dts	drGyro()	자이로 센서의 값을 읽어서 회전 가속력으로 환산하여 주는 함수입니다.
drLowPassFilter.dts	drLowPassFilter()	로패스 필터를 구현한 모듈로 값의 급격한 변화를 막아주는 함수입니다.
draTan.dts	draTan()	삼각함수 중 탄젠트의 역함수로 각도값을 반환합니다.

01 선언부 및 초기화




```
448 void standup()
449 {
450         SERVO_TorqCtrl[254]=96
451         jog( 512, 0, 254, 69 )
452         jog( 232, 0, 0, 69 )
453         jog( 232, 0, 1, 69 )
454         jog( 792, 0, 3, 69 )
455         jog( 792, 0, 4, 69 )
456         delay( 1000 )
457 }
458 void <<Calib백업>>()
459 {
460         Calib_RA_W=SERVO_CaliDiff[0]
461         Calib_LA_W=SERVO_CaliDiff[3]
462         Calib_RL_W=SERVO_CaliDiff[6]
463         Calib_RL_UP=SERVO_CaliDiff[7]
464         Calib_RL_KNEE=SERVO_CaliDiff[8]
465         Calib_RF_T=SERVO_CaliDiff[9]
466         Calib_RF_W=SERVO_CaliDiff[10]
467         Calib_LL_W=SERVO_CaliDiff[11]
468         Calib_LL_UP=SERVO_CaliDiff[12]
469         Calib_LL_KNEE=SERVO_CaliDiff[13]
470         Calib_LF_T=SERVO_CaliDiff[14]
471         Calib_LF_W=SERVO_CaliDiff[15]
472 }
```

1. nCheckFirst

nCheckFirst(최초실행 판단) 변수는 외부에서 0으로 리셋해 주어야 합니다. 한 번만 리셋하면 되며 이후에는 필요 없습니다.

2. nStandard_Tilt(앞뒤)/Swing(좌우)

실제 안정화를 위해 로봇 상체가 어느 정도는 기울어져야 하는데 이 변수에 로봇 보행 시 상체의 기울어진 값을 넣어주면 됩니다.

3. walk_cnt

초기보행 및 종료보행을 제외하고 반복적으로 진행할 보행 횟수를 여기에 입력합니다.
ex) 30 – 30걸음 반복

4. Calib백업

Calibration 조정을 위해 각 모터의 Calibration 상태를 백업 받아 놓습니다.

5. nMotionNum

모션을 시작, 반복, 종료 3구간으로 나누어서 실행하고 있기 때문에 시작 모션에 해당하는 번호를 넣어둡니다.

6. nLowPassValue

내부적으로 LowPass 필터를 사용하고 있으므로 해당 필터의 파라미터로 사용하고 있습니다. 0~100까지의 값으로 이 값이 낮을수록 응답속도가 느려지게 됩니다.

02

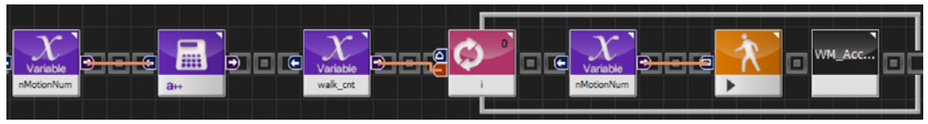
```
484    motionready( nMotionNum )
485    nValue_Time=20
486    WT_AccGyro()
487    motion( nMotionNum )
488    WM_AccGyro()
```

– 모션 준비 자세를 취하고 최초 한걸음을 걷는 모션입니다.

– nMotionNum

: 수행할 모션의 번호를 지정(6–시작모션, 7–반복보행, 8–종료모션)하여 시작모션만 지정하면 연속된 모션을 수행합니다.

단, DR–Sim에서 모션번호를 동일하게 맞추거나 Task 를 개인 취향에 맞게 수정합니다.




```
484    motionready( nMotionNum )
485    nValue_Time=20
486    WT_AccGyro()
487    motion( nMotionNum )
488    WM_AccGyro()
489    nMotionNum++
```

– Walk_cnt 에 입력된 보행 수만큼 반복적으로 보행을 하는 부분입니다.

– WM_AccGyro/WT_AccGyro

: 모션을 수행하면서 Acc/Gyro 센서를 이용해 보정을 하는 함수입니다.

WM_AccGyro : 모션 종료 시까지 대기합니다.

WT_AccGyro : 100 ms 단위의 시간값(예:20 – 2초)을 nValue_Time 에 입력한 후 동작하면 해당 시간동안 대기하며 보정합니다.

03

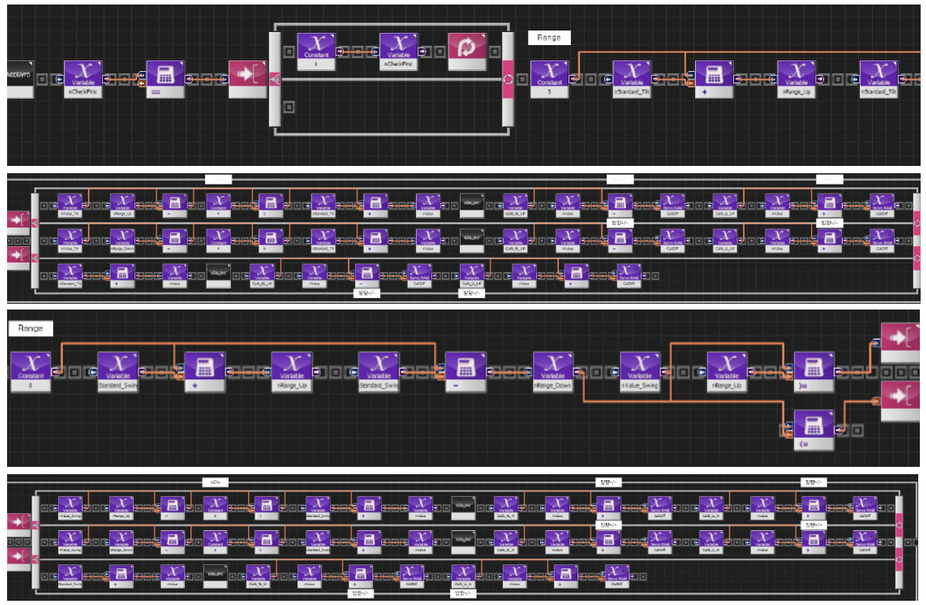
```
510    nMotionNum++
511    motion( nMotionNum )
512    WM_AccGyro()
513    delay( 1000 )
```

– 반복 보행 후 마지막 종료 모션을 수행하되, WM_AccGyro 함수를 이용해 보정은 계속적으로 수행합니다.

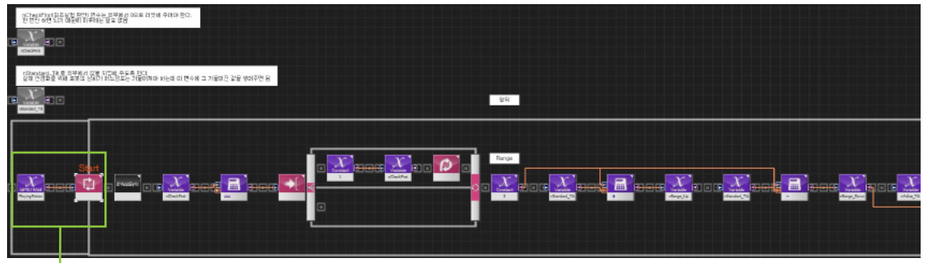
```
514    standup()
515    <<Calib복구>>()
```

– "Calib 복구" 모듈에서 수정한 보정위치를 Task 를 종료하기 전 원래의 값으로 되돌립니다.

Module 1 WM_AccGyro - 전체 프로그래밍 흐름



Module 1–1

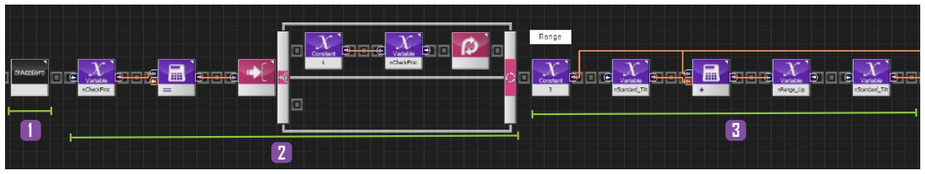


모션이 수행되고 있는 동안 반복

– WM_AccGyro

: 모션이 플레이하고 있는 동안 사용하는 대기문 대신 사용하기 위해 만든 모듈로 외부에서 변수의 초기값을 한 번만 선언해 두고 이 모듈을 사용하면 됩니다. 이 모듈에서 관절의 Calibration 위치값을 조정하여 실제 모터의 움직임을 제어하게 됩니다.

● Module 1-2



1 "AccGyro"모듈에서 가속도 센서와 자이로 센서의 결과값을 피드백 받습니다.

2 단, 최초 실행 시의 값은 필터링이 되지 않은 값이므로 외부적으로 최초 실행 코드를 nCheckFirst 변수를 이용하여 구분해 주어야 합니다.

3 얻어진 값을 주어진 기준값(nStandard_Tilt)과 비교하여 ±nRange(±3) 이상의 변화량이 있는 경우 관절을 조정하도록 조건을 정의하였습니다.

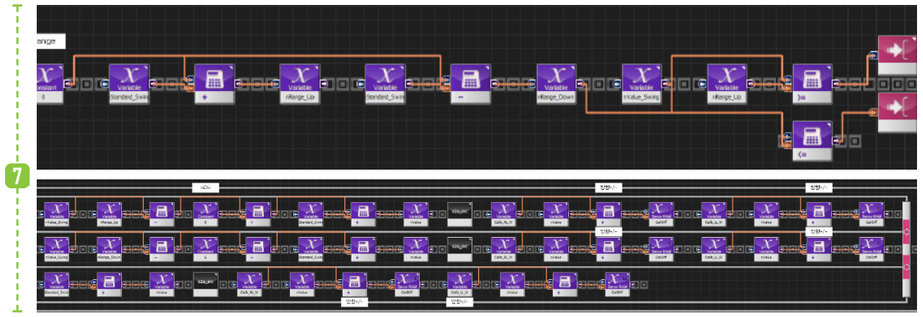
● Module 1-3

4 5 발생한 오차값을 7, 12번 모터에 적용합니다.

6 단, 오차값을 적용하지 않는 경우 원래의 값을 적용하도록 합니다.

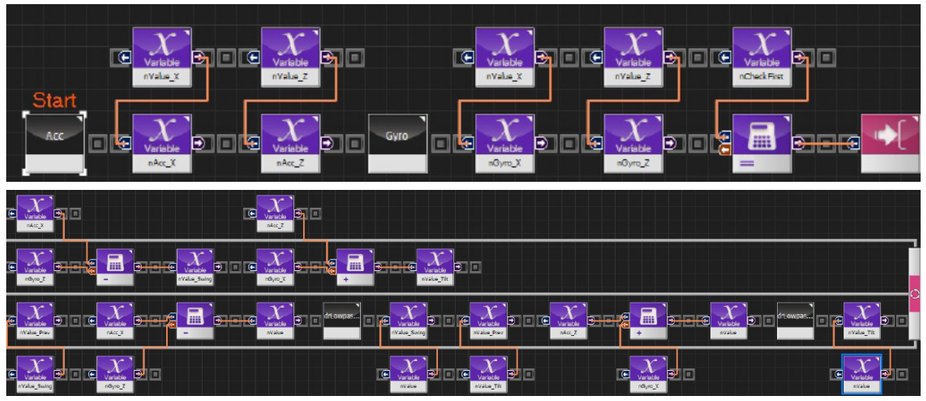
```
if( ( nValue_Tilt >= nRange_Up ) )
{
        nValue=( ( ( nValue_Tilt - nRange_Up ) / 4 ) + nStandard_Tilt )
        a2e_inc()
        SERVO_CaliDiff[7]=( Calib_RL_UP - nValue )
        SERVO_CaliDiff[12]=( Calib_LL_UP + nValue )
}
else if( ( nValue_Tilt <= nRange_Down ) )
{
        nValue=( ( ( nValue_Tilt - nRange_Down ) / 4 ) + nStandard_Tilt )
        a2e_inc()
        SERVO_CaliDiff[7]=( Calib_RL_UP - nValue )
        SERVO_CaliDiff[12]=( Calib_LL_UP + nValue )
}
else
{
        nValue=( 0 + nStandard_Tilt )
        a2e_inc()
        SERVO_CaliDiff[7]=( Calib_RL_UP - nValue )
        SERVO_CaliDiff[12]=( Calib_LL_UP + nValue )
}
```

● Module 1-4

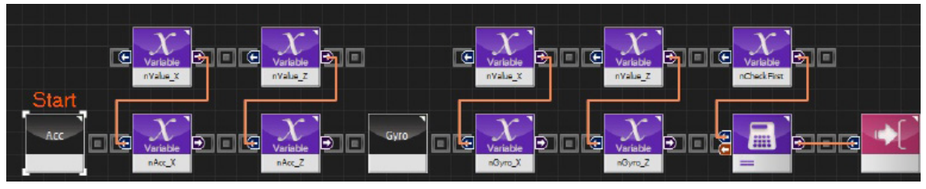


7 같은 방식으로 좌우 모터[6,11번]도 동일하게 적용하도록 합니다.

Module 2 drAccGyro - 전체 프로그래밍 흐름



● Module 2-1

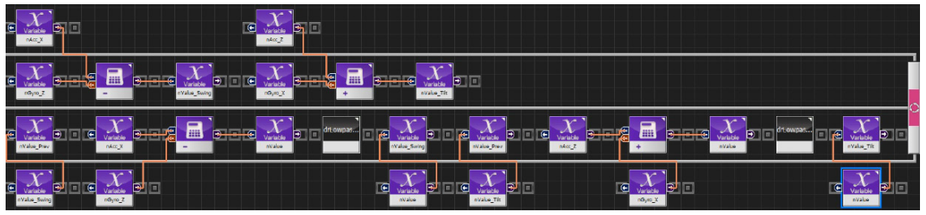


- drAccGyro.dts

: 얻어진 가속도/자이로 센서값을 이용하여 현재의 자세를 검출하는 모듈로 내부에 필터링 연산을 포함한 2차 연산 함수모듈 입니다.

- 가속도(Acc)센서 모듈에서 축에 대한 해석을 취하고 자이로(Gyro)센서 모듈에서도 값을 얻어옵니다.

● **Module 2-2**

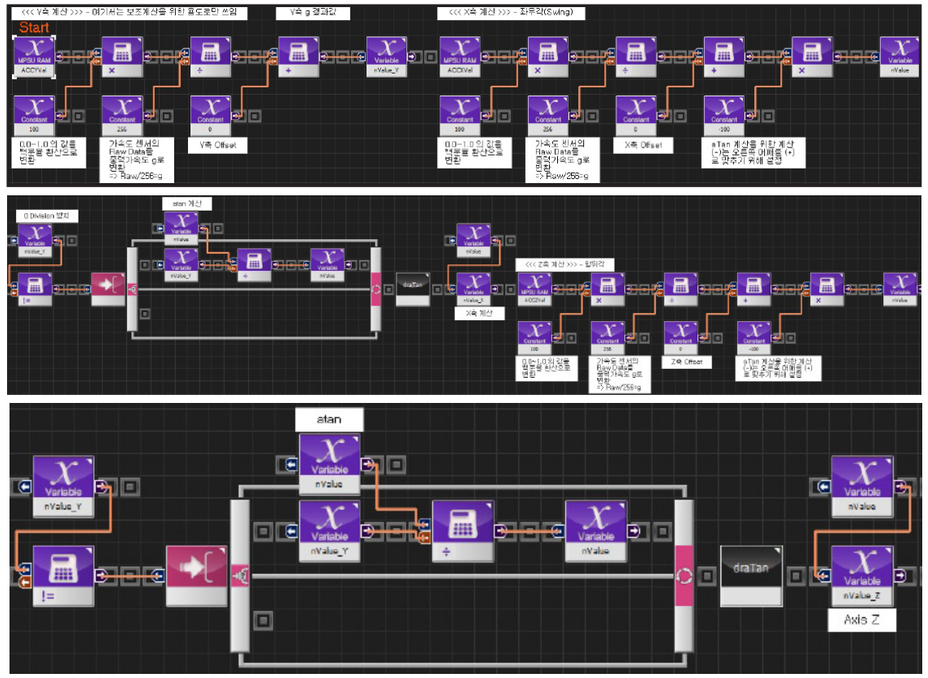


8 최초 연산 시엔 얻어진 초기값을 그대로 넣도록 하고

9 이후부터는 필터링된 결과값을 리턴하도록 합니다.

```
if( ( nCheckFirst == 0 ) )
{
        nValue_Swing=( nAcc_X - nGyro_Z )
        nValue_Tilt=( nAcc_Z + nGyro_X )
}
else
{
        nValue_Prev=nValue_Swing
        nValue=( nAcc_X - nGyro_Z )
        drLowpassfilter()
        nValue_Swing=nValue
        nValue_Prev=nValue_Tilt
        nValue=( nAcc_Z + nGyro_X )
        drLowpassfilter()
        nValue_Tilt=nValue
}
```

Module 3 Acc - 전체 프로그래밍 흐름



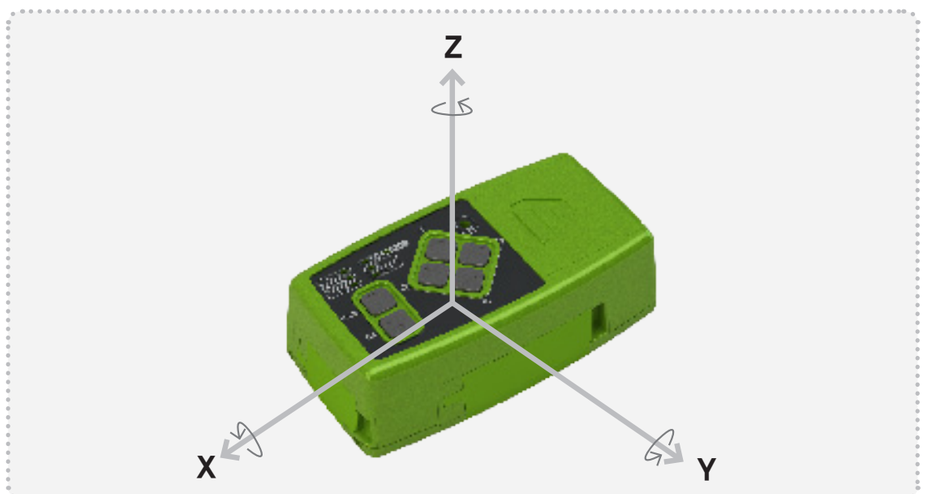
● **Module 3-1** – Acc.dts

: 가속도 센서의 Raw 데이터를 각도값으로 바꾸어 주는 모듈입니다.

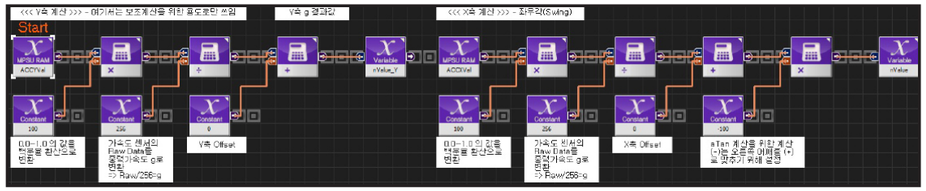
: 3축 가속도 센서이므로 360도에 대해 전부 대응가능합니다. 하지만 연산처리부하를 줄이기 위해서 현재 모듈에서는 180도
정도(-90~+90도)로만 연산처리하고 있습니다.

: 가속도 센서의 각 축의 정의는 아래와 같습니다. (256 = 1g(중력가속도 9.8m/s2))
※ 가속도(g) = RawData / 256

: 축의 센서상의 오차 Offset이 존재할 수 있으므로 계산과정 중 Offset 이라는 값이 들어가게 됩니다.



● Module 3-2

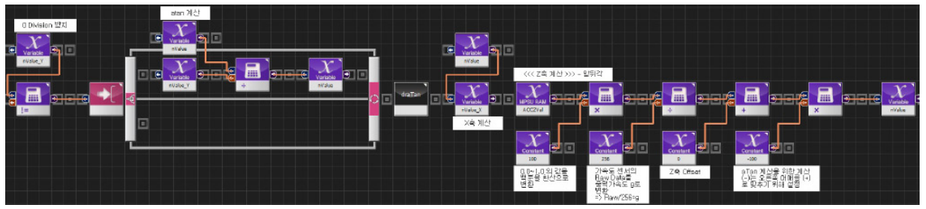


– Y축 중력가속도 g 의 계산

: Y = RawData(Y) * 100 / 256 : −1.0 ~ 1.0 사이의 값이 정규화된 값이지만 이 100을 곱해 실수연산을 정수연산으로 바꾸도록 합니다.

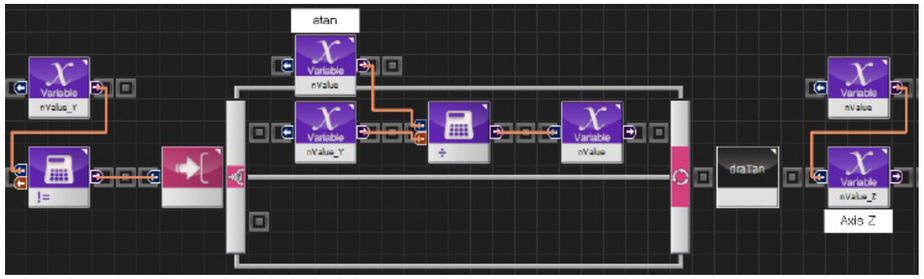
: 나머지 X와 Z에 관한 연산도 동일한 연산과정을 취합니다. 단, 개인의 취향에 의해 축의 방향은 바꾸기도 합니다. 해당 모듈의 X, Z 축에서는 방향을 (−)를 곱하여 반대 방향으로 전환하였습니다.

● Module 3-3



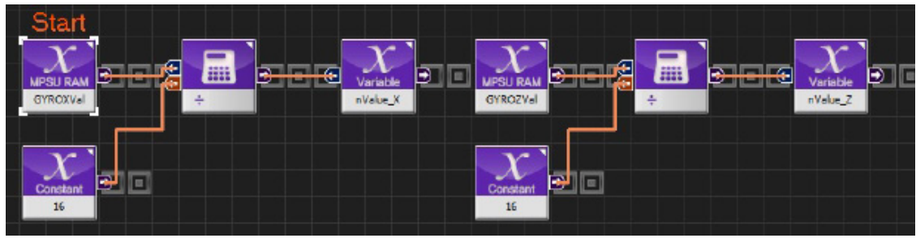
– X축도 동일한 과정으로 g값을 구한 후 이 값을 하나의 평면상 선분으로 보고 Y축과 이루는 각을 tan()의 역함수를 이용하여 값을 구하여 보면 현재 로봇이 기울어진 정도를 알 수 있습니다. 단, Module 3-2는 Y축과 X축 간 연관 각도이므로 로봇이 옆으로 누웠을 때 각도(이후 Swing으로 정의)를 알 수 있습니다.

● Module 3-4



– Z축도 동일한 과정으로 g값을 구한 후 이 값을 하나의 평면상 선분으로 보고 Y축과 이루는 각을 tan()의 역함수를 이용하여 값을 구하여 보면 현재 로봇이 기울어진 정도를 알 수 있습니다. 단, Module 3-2는 Y축과 Z축 간 연관 각도이므로 로봇이 앞뒤로 기울이는 정도의 각도(이후 Tilting으로 정의)를 알 수 있습니다.

Module 4 Gyro - 전체 프로그래밍 흐름



– Gyro.dts

: 축의 회전 속도를 측정하기 위한 센서로 속도 계산법은 아래와 같습니다.

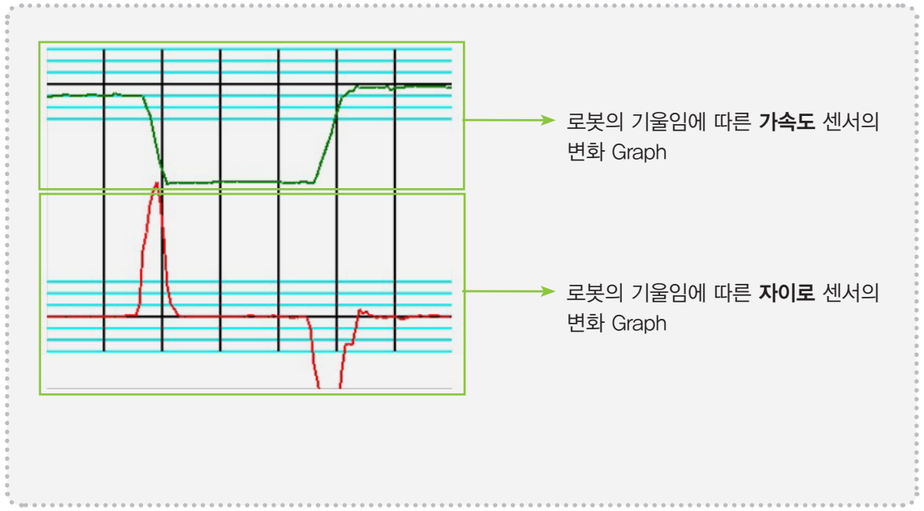
각속도 = RawData / 32768 * 2000

: 속도처리량 증대와 연산의 Overflow를 막기 위해 위 식을 아래와 같이 수정할 수 있습니다.

각속도 = RawData / 16

: 구하고자 하는 축은 Swing과 Tilting이므로 2개의 축만 구합니다.

● Module 4-1



– 가속도 그래프와 자이로 그래프

: 가속도 그래프는 현재 로봇의 g값을 구한 것이고 자이로는 로봇이 회전하는 동안의 회전 속도를 구한 것입니다. 그래서

두 값의 차이는 위 그래프와 같습니다.(똑바로 서있는 로봇을 앞쪽으로 기울인다고 가정할 경우 위쪽의 녹색 그래프가

가속도 센서(Acc) Z축, 아래쪽의 적색 그래프가 자이로 센서(Gyro) X축의 반응입니다.)

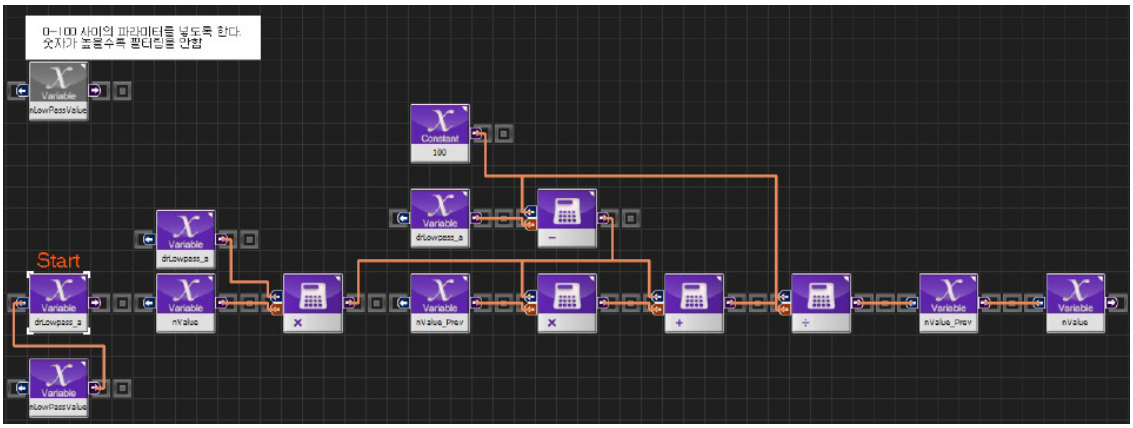
: 로봇을 앞으로 기울이는 것이므로 수학적으로 얘기하면 X축을 기준으로 회전을 하는 것입니다. 때문에 X축은 기울기의

변화량은 없고 축의 회전이 발생하므로 X축은 회전속도를 측정할 수가 있습니다. 이 경우 Y축과 Z축은 축이 X축을

기준으로 돌아가기 때문의 현재 상태의 중력가속도가 변화하게 됩니다.

: 결론적으로 로봇이 앞으로 기울여지는 경우 Acc의 Z,Y(Y는 현재 보조연산으로만 사용하므로 제외)가 변화하고, Gyro의 X 값이 변화하게 됩니다.

Module 5 drLowPassFilter - 전체 프로그래밍 흐름



– 급격한 값의 변화(High Pass)를 노이즈로 보고 값의 변화를 점진적으로 변화하게 해주는 필터로 수식은 아래처럼 정의됩니다.

값 = alpha * 현재값 + (1.0 – alpha) * 이전값

– Alpha(가중치)의 값이 높을수록 응답이 빨라지며 정수연산을 위하여 실제로는 100을 곱하여 사용하였습니다.

Module 6 a2e_inc - 전체 프로그래밍 흐름

– 각도값을 모터에서 실제 사용하는 RawData(Engineering Value of Degree: 이후 Evd 로 통칭)로 변환하여 주는 함수입니다.
512 기준으로 연산된 함수는 아니고 주어진 각도의 상대각도를 계산하여 주는 함수입니다.

기타

– Atan.dts

: 제곱근, 삼각함수 등은 테일러 급수를 가지고 구할 수 있으나 값의 Overflow 발생확률이 높고 연산처리속도가 저조하여 Case by Case 형태의 연산형태를 취합니다.