수직속도 결과 보고서

C코드 (fall.c로 첨부)

함수 정의

```
void LPF(float *Input, float *Output, float *PastInput, float *PastOutput)
{
  float CutOffFrequency   = 5 ;
  float SamplingFrequency = 50 ;

  float a1,b0,b1,w0;
  w0 = 2 * 3.14 * CutOffFrequency;
  a1 = (w0 - 2 * SamplingFrequency) / (2*SamplingFrequency + w0);
  b0 = w0 / ( 2 * SamplingFrequency + w0);
  b1 = b0;

  *Output = b0*(*Input) + b1*(*PastInput) - a1*(*PastOutput);
  *PastOutput = *Output;
  *PastInput = *Input;
}

void HPF(float *Input, float *Output, float *PastInput, float *PastOutput)
{
  // CutOffFrequency   를 10hz로 원할경우 10.0
  // SamplingFrequency 를 0.001ms로 원할경우 1/0.001 => 1000

  float CutOffFrequency   = 5 ;
  float SamplingFrequency = 50 ;

  float a1, b0, b1, w0;
  w0 = 2 * 3.14 * CutOffFrequency;
  a1 = (w0 - 2 * SamplingFrequency) / (2 * SamplingFrequency + w0);
  b0 = 2 * SamplingFrequency / (2 * SamplingFrequency + w0);
  b1 = b0;

  *Output = a1 * (*PastOutput) + b0 * (*Input) - b1 * (*PastInput);
  *PastOutput = *Output;
  *PastInput = *Input;
}

int sign(int input)
{
  int output=0;
  if(input>0)          output = 1;
  else if(input==0)   output = 0;
  else if(input<0)    output = -1;

  return output;
}
```

추락 판단 알고리즘

```
//축변환
float Acc_Y = ((float)(sData.accX))/8192 ;
float Acc_Z = ((float)(sData.accY))/8192 ;
float Acc_X = ((float)(sData.accZ))/8192 ;

float Gyr_Y = ((float)(sData.gyroX))/65.536f ;
float Gyr_Z = ((float)(sData.gyroY))/65.536f ;
float Gyr_X = ((float)(sData.gyroZ))/65.536f ;

//LPF
float lpf_X =  (float)Acc_X;
float lpf_Y =  (float)Acc_Y;
float lpf_Z =  (float)Acc_Z;

LPF( &lpf_X, &ax_1, &ax_2, &ax_3  ) ;
LPF( &lpf_Y, &ay_1, &ay_2, &ay_3  ) ;
LPF( &lpf_Z, &az_1, &az_2, &az_3  ) ;

float lpf_gyX = Gyr_X ;
float lpf_gyY = Gyr_Y ;
float lpf_gyZ = Gyr_Z ;

LPF( &lpf_gyX, &gx_1, &gx_2, &gx_3  ) ;
LPF( &lpf_gyY, &gy_1, &gy_2, &gy_3  ) ;
LPF( &lpf_gyZ, &gz_1, &gz_2, &gz_3  ) ;
```

```
//전체속도
float a_norm = sqrt(pow(Acc_X,2) + pow(Acc_Y,2) + pow(Acc_Z,2));
float ax_deb = Acc_X - 9.8*Acc_X/a_norm;       //default 속도를 뺀거(움직인 속도를 나타냄)
float ay_deb = Acc_Y - 9.8*Acc_Y/a_norm;
float az_deb = Acc_Z - 9.8*Acc_Z/a_norm;

//HPF(ax_deb, ay_deb, az_deb)
float hpf_X = ax_deb;
float hpf_Y = ay_deb;
float hpf_Z = az_deb;

HPF( &hpf_X, &ax_deb_1, &ax_deb_2, &ax_deb_3  ) ;
HPF( &hpf_Y, &ay_deb_1, &ay_deb_2, &ay_deb_3  ) ;
HPF( &hpf_Z, &az_deb_1, &az_deb_2, &az_deb_3  ) ;

ax_deb_h[thirdcount%3] = ax_deb_1;
ay_deb_h[thirdcount%3] = ay_deb_1;
az_deb_h[thirdcount%3] = az_deb_1;

//Simpson's Rule
if(thirdcount == 2)       thirdfull = true;
else if(thirdcount == 1)  twicefull = true;

if(thirdfull == true)
{
    if(thirdcount == 0)
    {
        vel_xdeb[thirdcount] = vel_xdeb[1] + ((ax_1 + 4 * ax_deb_h[2] + ax_deb_h[0]) * 0.02 / 3);
        vel_ydeb[thirdcount] = vel_ydeb[1] + ((ay_1 + 4 * ay_deb_h[2] + ay_deb_h[0]) * 0.02 / 3);
        vel_zdeb[thirdcount] = vel_zdeb[1] + ((az_1 + 4 * az_deb_h[2] + az_deb_h[0]) * 0.02 / 3);
        //  Signum Function
        if(sign(vel_xdeb[1]) == sign(ax_1/a_norm))
        {
          vel_xdeb[1] = 0;
        }
        if(sign(vel_ydeb[1]) == sign(ay_1/a_norm))
        {
          vel_ydeb[1] = 0;
        }
        if(sign(vel_zdeb[1]) == sign(az_1/a_norm))
        {
          vel_zdeb[1] = 0;
        }
    }
    else if(thirdcount == 1)
    {
        vel_xdeb[thirdcount] = vel_xdeb[2] + ((ax_1 + 4 * ax_deb_h[0] + ax_deb_h[1]) * 0.02 / 3);
        vel_ydeb[thirdcount] = vel_ydeb[2] + ((ay_1 + 4 * ay_deb_h[0] + ay_deb_h[1]) * 0.02 / 3);
        vel_zdeb[thirdcount] = vel_zdeb[2] + ((az_1 + 4 * az_deb_h[0] + az_deb_h[1]) * 0.02 / 3);
        //  Signum Function
        if(sign(vel_xdeb[2]) == sign(ax_1/a_norm))
        {
          vel_xdeb[2] = 0;
        }
        if(sign(vel_ydeb[2]) == sign(ay_1/a_norm))
        {
          vel_ydeb[2] = 0;
        }
        if(sign(vel_zdeb[2]) == sign(az_1/a_norm))
        {
          vel_zdeb[2] = 0;
        }
    }
    else if(thirdcount == 2)
    {
        vel_xdeb[thirdcount] = vel_xdeb[0] + ((ax_1 + 4 * ax_deb_h[1] + ax_deb_h[2]) * 0.02 / 3);
        vel_ydeb[thirdcount] = vel_ydeb[0] + ((ay_1 + 4 * ay_deb_h[1] + ay_deb_h[2]) * 0.02 / 3);
        vel_zdeb[thirdcount] = vel_zdeb[0] + ((az_1 + 4 * az_deb_h[1] + az_deb_h[2]) * 0.02 / 3);
        //  Signum Function
        if(sign(vel_xdeb[0]) == sign(ax_1/a_norm))
        {
          vel_xdeb[0] = 0;
        }
        if(sign(vel_ydeb[0]) == sign(ay_1/a_norm))
        {
          vel_ydeb[0] = 0;
        }
        if(sign(vel_zdeb[0]) == sign(az_1/a_norm))
        {
          vel_zdeb[0] = 0;
        }
    }
    //전체속도RMS
    norm_deb = sqrt((pow(vel_xdeb[thirdcount],2) + pow(vel_ydeb[thirdcount],2) + pow(vel_zdeb[thirdcount],2)));
}

//Roll, Pitch
```

```
roll = -atan2(ay_1, sqrt(pow(ax_1, 2) + pow(az_1, 2)));
if (az_1 <= 0.2)
{
    pitch = atan2(ax_1, sqrt(pow(ay_1, 2) + pow(az_1, 2)));
}
else
{
    pitch = atan2(ax_1, -sqrt(pow(ay_1, 2) + pow(az_1, 2)));
}
PhiSaved_acc = roll * 180 / 3.14;
ThetaSaved_acc = pitch * 180 / 3.14;

//과거 값 저장
if(twicefull == true)
{
    if(thirdcount==0) Acc_VP[2] = Acc_V[2];
    else Acc_VP[thirdcount-1] = Acc_V[thirdcount-1];
}

//수직가속도
if (az_1 > 0 && (abs(ThetaSaved_acc) < 90))   //보통의 서있음
{
  Acc_V[thirdcount] = -sin(ThetaSaved_acc/180*3.14)*ax_1 - cos(ThetaSaved_acc/180*3.14)*sin(PhiSaved_acc/180*3.14)*ay_1 - cos(PhiSaved
}
else      //거꾸로 상태
{
  Acc_V[thirdcount] = -sin(ThetaSaved_acc/180*3.14)*ax_1 + cos(ThetaSaved_acc/180*3.14)*sin(PhiSaved_acc/180*3.14)*ay_1 + cos(PhiSaved
}

int_flag=0;
static_flag=0;

if(thirdfull == true)
{
    //jerk
    if(thirdcount == 0)        jerk[2] = (Acc_V[0]-Acc_V[1])/0.04;
    else if(thirdcount == 1)   jerk[0] = (Acc_V[1]-Acc_V[2])/0.04;
    else if(thirdcount == 2)   jerk[1] = (Acc_V[2]-Acc_V[0])/0.04;

    jerkcount++;

    if(jerkcount >= 2)
    {
        for(int i=0; i<=2; i++)
        {
            if((fabsf(Acc_VP[i])<0.6) && (fabsf(jerk[i])<6))  int_flag +=1;
            if(fabsf(Acc_VP[i]) < 0.01)      static_flag +=1;
        }
    }

    if(int_flag == 3)
    {
        if(thirdcount == 0)  u0[thirdcount] = u0[2] * 0.9;
        else  u0[thirdcount] = u0[thirdcount-1] * 0.9;
    }
    else      //(수직가속도 --적분--> 수직속도)
    {
        if(thirdcount == 0)  u0[0] = u0[1] + 9.8*(Acc_V[1]+4*Acc_V[2]+Acc_V[0]) * 0.02 / 3;
        else if(thirdcount == 1)  u0[1] = u0[2] + 9.8*(Acc_V[2]+4*Acc_V[0]+Acc_V[1]) * 0.02 / 3;
        else if(thirdcount == 2)  u0[2] = u0[0] + 9.8*(Acc_V[0]+4*Acc_V[1]+Acc_V[2]) * 0.02 / 3;
    }

    if(static_flag == 2)
    {
        if(thirdcount == 2)   Vel_V[0] = 0;
        else Vel_V[thirdcount+1] = 0;
    }
}

//lag filter
if(thirdcount==0)
{
    u1[thirdcount] = 0.985 * u1[2] + 0.015 * u0[thirdcount];
    u2[thirdcount] = 0.985 * u2[2] + 0.015 * u1[thirdcount];
}
else
{
    u1[thirdcount] = 0.985 * u1[thirdcount-1] + 0.015 * u0[thirdcount];
    u2[thirdcount] = 0.985 * u2[thirdcount-1] + 0.015 * u1[thirdcount];
}
Vel_V[thirdcount] = u0[thirdcount] - u2[thirdcount];

if ((Vel_V[thirdcount] > 1.5) & ((fabsf(PhiSaved_acc) > 20) | (fabsf(ThetaSaved_acc) > 20)) & (norm_deb > 0.8))
{
    trigger_process_start(1);
    UART5_Printf("fall detected!!!!!!!!!!!!!!!!!!!!!!!!\r\n");
}
```

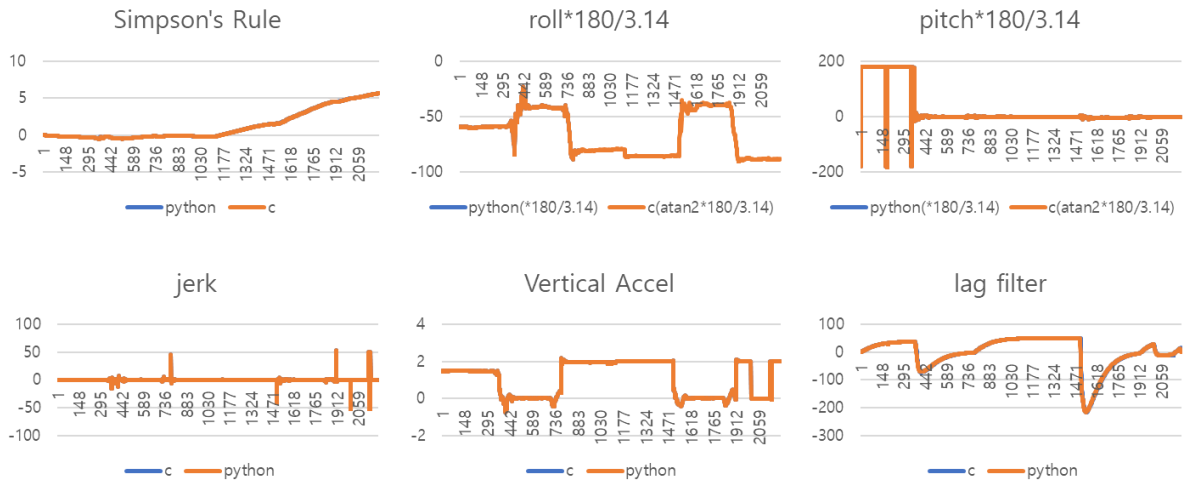
```
thirdcount++;
if(thirdcount == 3)       thirdcount = 0;
```

수직속도의 용도

강한 외력이 동반되어 추락 할 경우 외력에 의해 수직속도의 양상이 자유낙하와 다를것이라 예상

외력 혹은 원심력이 동반된 추락의 상황을 파악하는데 활용하려 함

변환 및 테스트 과정



1. visual studio를 이용해 .csv파일 결과 출력 후 비교

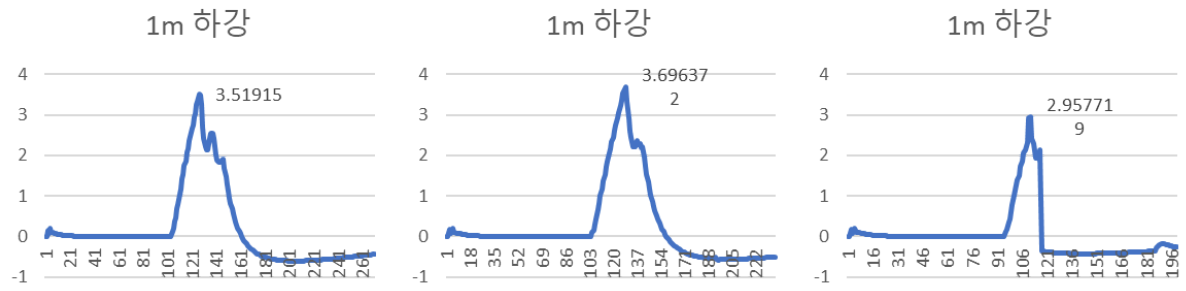
filter결과 제외 동일(python코드가 함수화로 공개가 안되어있어 변환 불가)

2. 결과 확인 후 iar에 코드 적용 후 테스트

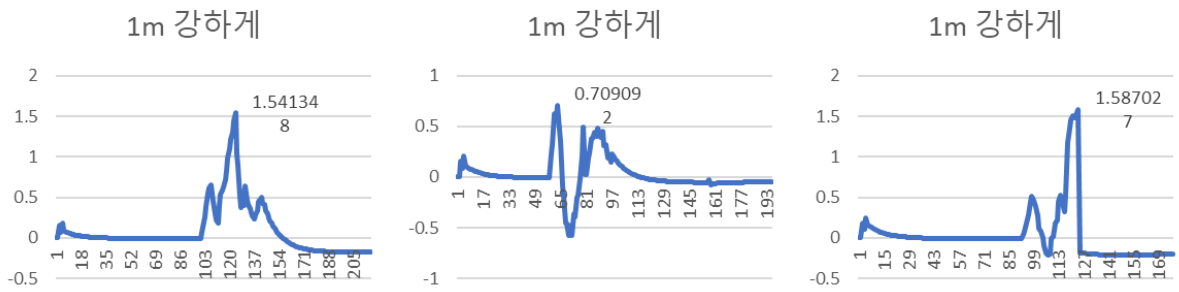
테스트 별 수직속도 그래프(datalog file 첨부: 수직속도, roll, pitch, 전체속도)

테스트 결과

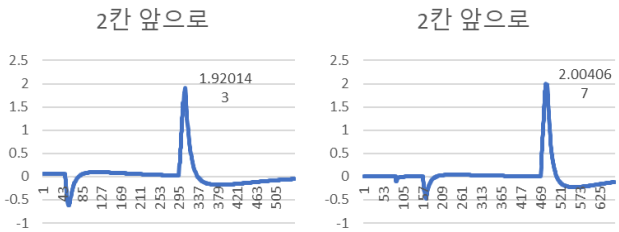
수직 하강(약 1m, +외피) : 3~3.7 / file name : fall



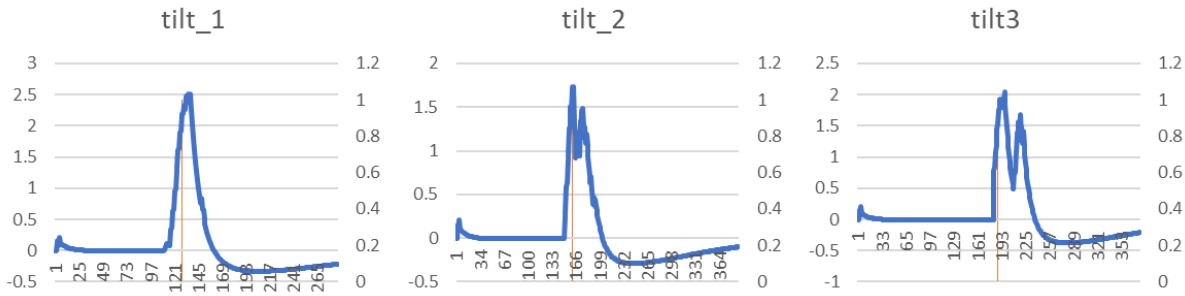
수직 강하게 하강(약 1m, +외피) : 0.7~1.6 / file name : strong_fall



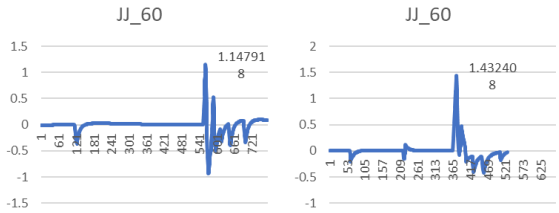
기울며 추락(약 60cm, +사람) : 1.9~2 / file name : 60cm_front

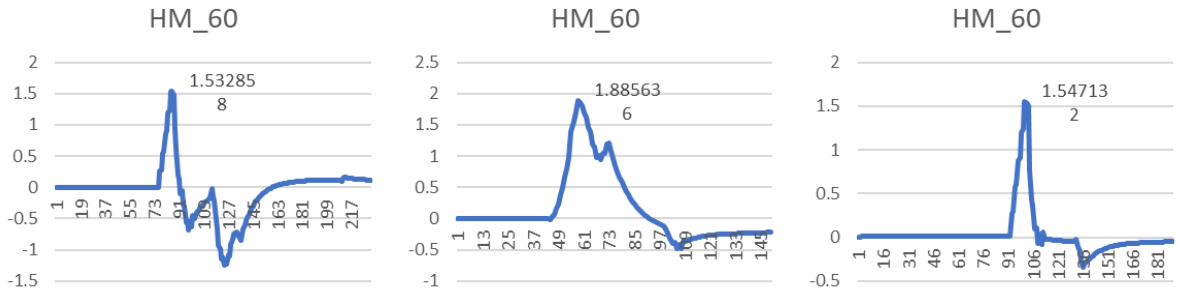


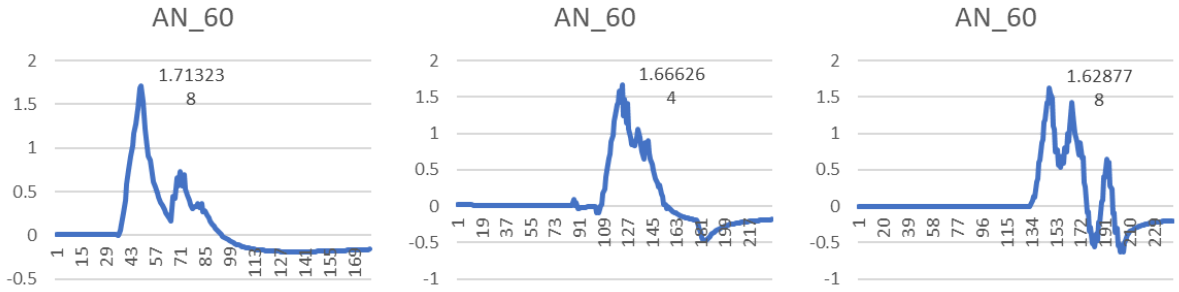
기울며 추락(약 80cm, +마네킹) : 1.7~2.5 / file name : linear



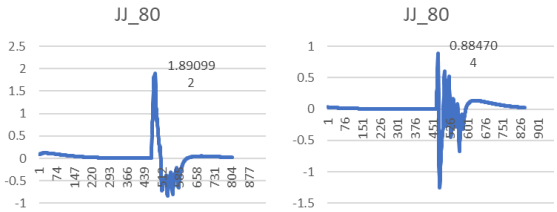
사다리에서 점프(약 60cm, +사람) : 진주 1.1~1.4, 해민 1.5~1.8, 안나 1.6~1.7 / file name : name_60

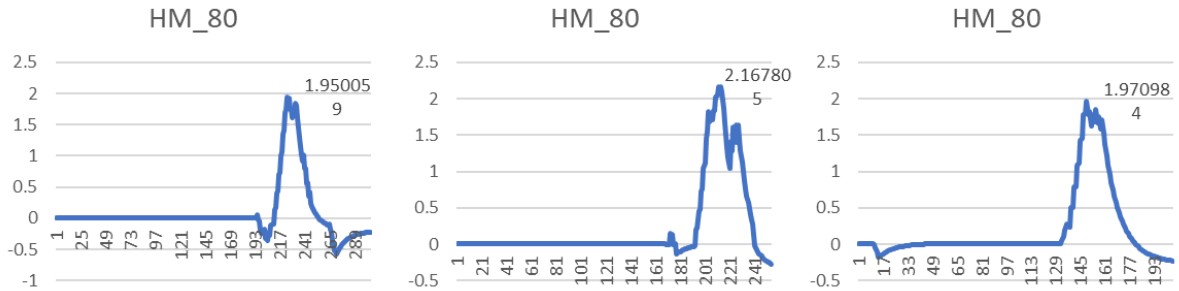


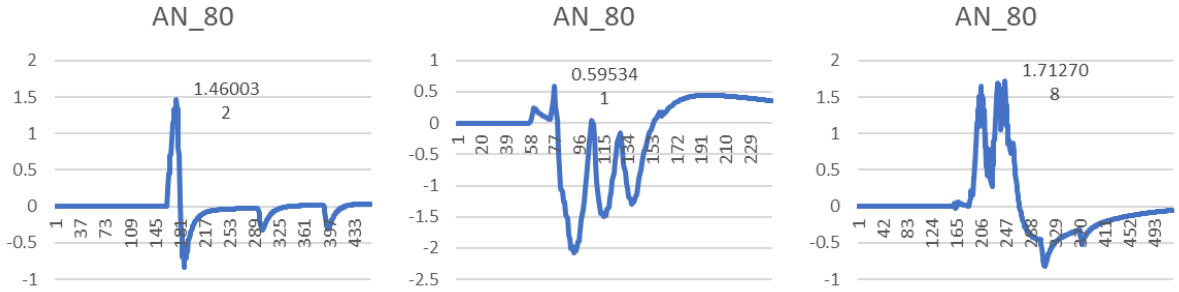




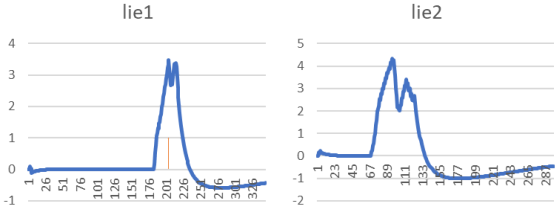
사다리에서 점프(약 80cm, +사람) : 진주 0.8~1.8, 해민 1.9~2.1, 안나 1.4~1.7 / file name : name_80



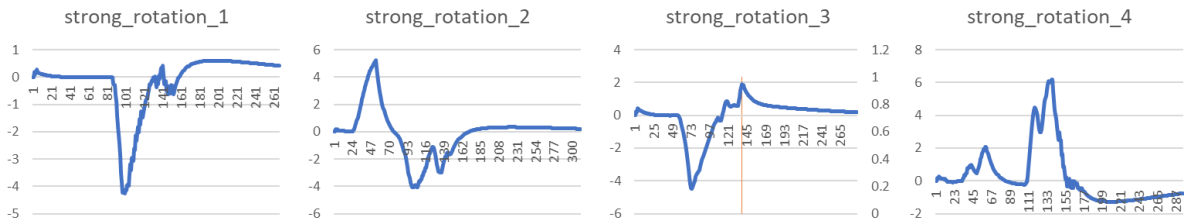




회전(수평, 아래로, +마네킹) : 3.5~4.5 / file name : lie

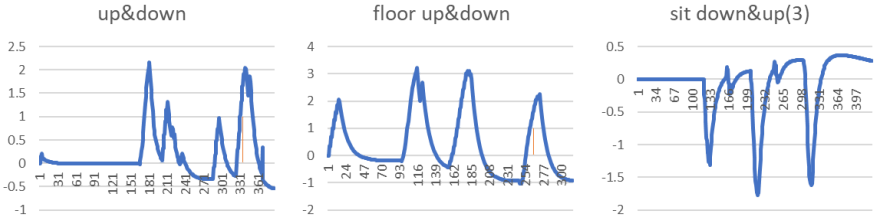


강한 회전(수평, 옆으로, +외피) : 절대값으로 4~6 / file name : strong_rot



강한 회전(수평, 아래로, +외피) : 절대값으로 7 / file name : strong_lie

up&down(외피, 사람) : 1.5~3.5 / file name : chair_updown floor_updown sit_updown



수직속도의 세기 : 강하게 하강 < 하강(사람) < 하강(외피) < 강한 회전

테스트 결과 결론

수직속도	1m(4.43)	1.5m(5.42)
하강	3.52	4.46
	3.70	2.92
	2.96	4.33
강하게 하강	1.54	1.43
	0.71	0.63
	1.59	1.51

v = √(2∗g∗h) 로 연산하였을 시 1m = 4.43, 1.5m = 5.42의 결과값 나옴

테스트 결과 데이터 양상(흐름)이 테스트 시마다 유사하고 실제 테스트 결과와 연산 결과 값이 동일하지 않지만

공기저항과 정형화 되지 않은 테스트 환경을 고려하면 어느정도 유사하다고 판단됩니다.

허나 강하게 외력을 주어 추락시킬 경우 힘이 추가되어 수직속도의 값이 자유낙하의 경우보다 큰 값을 가질것이라 예상하였으나

테스트 분석 결과 외력이 들어간 수직하강의 경우 값이 더 작게 나왔고,

외력, 원심력이 동반 된 경우 동일한 테스트인 경우에도 데이터 양상(흐름)에 공통점이 바로 파악되지 않아 분석에 어려움이 있어 바로 사용하기에는 보완이 필요하다고 판단됩니다.

외력이 들어간 경우의 추락 테스트를 하셨다면 결과 공유와 설명 부탁드리고,

혹시 개발당시 고려되지 않았던 사항이면 알고리즘 보완에 도움을 요청드립니다.