Water 期中汇报

我们完成算法分析的基本准则是：**服务于传感器**

Our basic principle for completing algorithm analysis is to serve sensors

整个算法分析的过程是围绕着我们对于传感器的理解进行的，

The entire process of algorithm analysis revolves around our understanding of sensors

我们首先关注的是传送数据的频率，传感器一共会传回 9+4=13 个数据，其中加速度，角速度，欧拉角九个数据是较高频率的，四元数这四个数据只能做到 2HZ，这个过低的频率就容易导致在完成一套测试动作的过程中，最高点的数据并未被记录下来

所以我们做了一个决定——抛弃四元数，选择欧拉角。

Our first focus is on the frequency of data transmission. The sensor will return a total of 13 data points, including acceleration, angular velocity, and Euler angle. Among them, the nine data points of acceleration, angular velocity, and Euler angle are relatively high in frequency, while the quaternion data can only achieve 2HZ. This low frequency can easily lead to the highest point data not being recorded during the completion of a set of testing actions

So we made a decision - to abandon quaternions and choose Euler angles.

这个时候就会遇到一个新的问题——万向节死锁

万向节死锁是机械系统和三维空间旋转控制中的一种现象，主要发生在使用欧拉角描述三维旋转时。当一个旋转轴在一定条件下与另一个旋转轴重合时，系统失去了一个自由度，导致无法独立控制所有旋转方向，这种情况称为万向节死锁。

At this point, a new problem will arise - the deadlock of the universal joint

Universal joint deadlock is a phenomenon in mechanical systems and three-dimensional spatial rotation control, mainly occurring when using Euler angles to describe three-dimensional rotation. When one rotation axis coincides with another rotation axis under certain conditions, the system loses one degree of freedom, resulting in the inability to independently control all rotation directions. This situation is called universal joint deadlock.

辅助理解的三句话：

1. 万向节死锁主要发生在使用欧拉角描述三维旋转时.

2. 当一个旋转轴在一定条件下与另一个旋转轴重合时.
3. 系统失去了一个自由度，导致无法独立控制所有旋转方向.

我们用欧拉角描述物体姿态时必须明确三个轴的顺序（除了三个轴的角度，还必须指定旋转顺序才能完整描述一个欧拉角）

Three sentences to assist understanding:

1. Universal joint deadlocks mainly occur when using Euler angles to describe three-dimensional rotation

When one rotation axis coincides with another rotation axis under certain conditions

3. The system has lost one degree of freedom, resulting in the inability to independently control all rotation directions

When describing the posture of an object using Euler angles, we must specify the order of the three axes (in addition to the angles of the three axes, we must also specify the rotation order to fully describe an Euler angle)

先转动的轴会带着后面的轴转动，但是后转动的轴不会影响前面的轴。

（因为欧拉角描述的是一个变换，而不是旋转过程。

变换意味着一组欧拉角代表着一次变换，而这次变换一定从初始位置开始，按照我们定义的顺序（比如 xyz）旋转的，因此绕前面的轴旋转完之后，这个轴在空间中的位置就确定了，后面的绕其它轴的旋转不会再影响这个轴的位置。

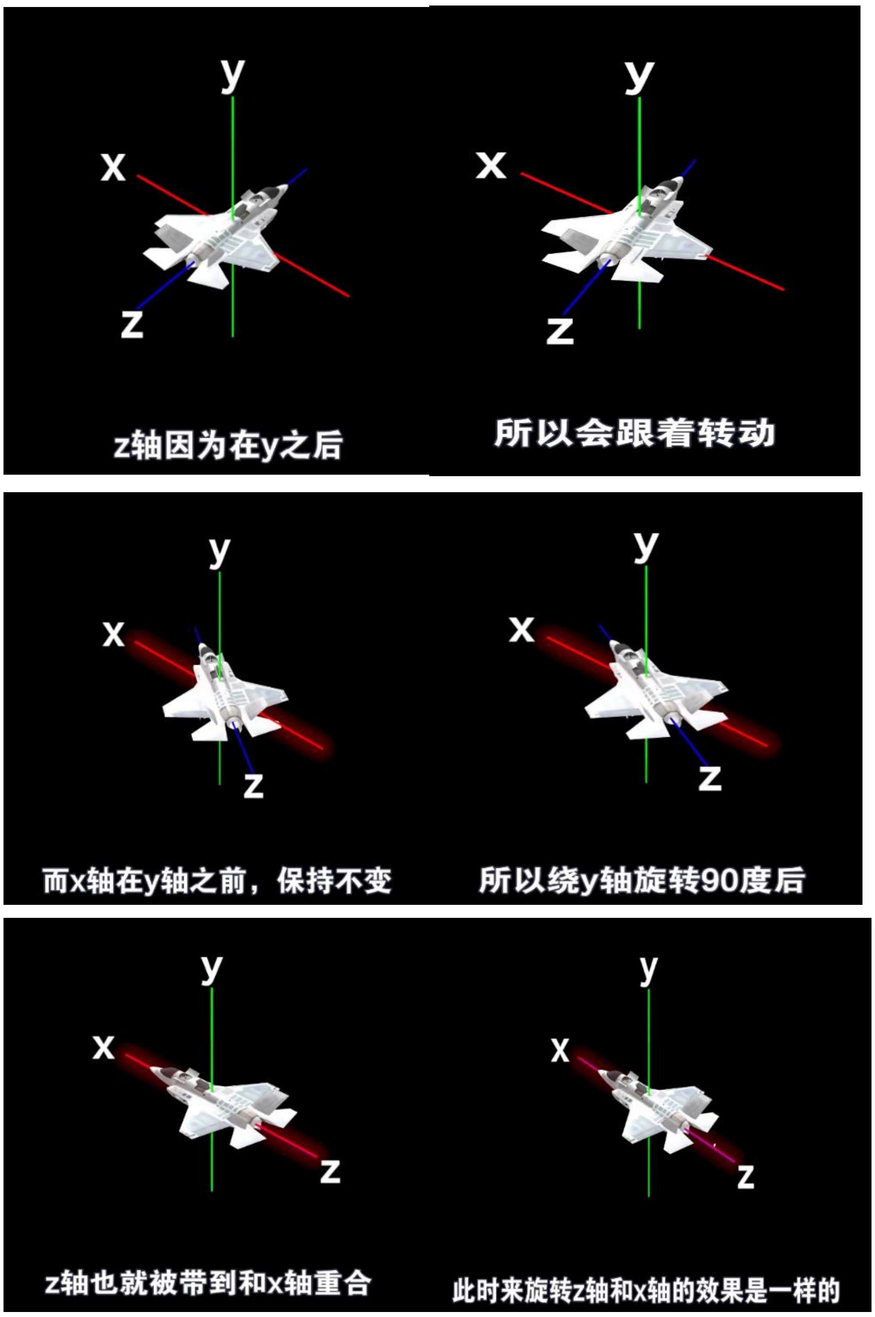
)

The shaft that rotates first will rotate with the shaft that rotates later, but the shaft that rotates later will not affect the shaft in front.

Because Euler angles describe a transformation, not a rotation process.

Transformation means that a set of Euler angles represents a transformation, and this transformation must start from the initial position and rotate in the order we define (such as xyz). Therefore, after rotating around the front axis, the position of this axis in space is determined, and subsequent rotations around other axes will no longer affect the position of this axis.

)



z轴因为在y之后

所以会跟着转动

而x轴在y轴之前，保持不变

所以绕y轴旋转90度后

z轴也就被带到和x轴重合

此时来旋转z轴和x轴的效果是一样的

如何避免万向节死锁?

How to avoid deadlock of universal joints?

1.根据使用场景调整旋转顺序

用欧拉角描述旋转时无法完全避免死锁问题，但可以根据使用场景来减少死锁出现的概率，比如描述船在海上航行的姿态时，可以把中间的旋转轴定义为船的左右方向，因为船几乎很难在这个方向旋转 90 度（船头竖直朝上或朝下）

2.使用其他描述旋转的方式

使用四元数，旋转矩阵等其它方式描述旋转

1. Adjust the rotation order according to the usage scenario

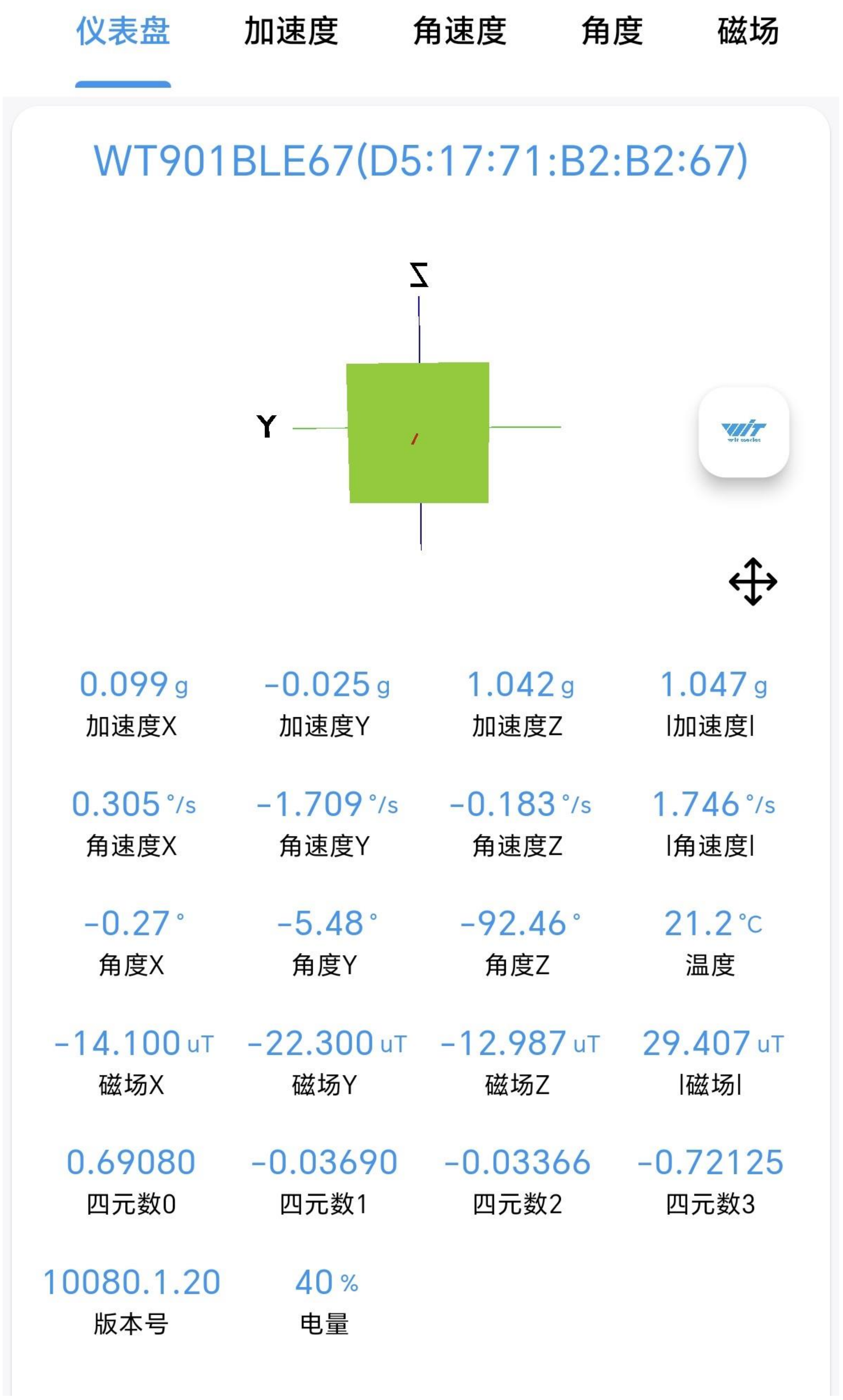
Describing rotation using Euler angles cannot completely avoid deadlock problems, but the probability of deadlock occurrence can be reduced according to the usage scenario. For example, when describing the posture of a ship sailing at sea, the middle rotation axis can be defined as the left and right directions of the ship, because it is almost difficult for the ship to rotate 90 degrees in this direction (with the bow facing vertically up or down)

2. Use other methods to describe rotation

Using quaternions, rotation matrices, and other methods to describe rotation

再次回到传感器上，经过多次试验之后我们发现，绕 x 轴的转动是不受影响的，绕 z 轴的转动在角度上同样可以正常获取，但绕 y 轴会受到万向节死锁的影响，如果按照同一种方向佩戴传感器，在我们测外展内收的这一环节，就是绕 y 轴旋转的，这就导致无法准确获取旋转角度，最终我们采用了解决问题的第一种方式——根据场景调整旋转，改变一个传感器的初始绑定角度使得在内收外展动作时，传感器的自身坐标是绕 x 轴旋转的。

Returning to the sensor again, after multiple experiments, we found that the rotation around the x-axis is not affected, and the rotation around the z-axis can also be obtained normally in terms of angle. However, the rotation around the y-axis will be affected by the deadlock of the universal joint. If we wear the sensor in the same direction, it will rotate around the y-axis in the process of measuring the outward and inward movements, which will result in the inability to accurately obtain the rotation angle. Finally, we adopted the first solution to the problem - adjusting the rotation according to the scene, changing the initial binding angle of a sensor so that during the inward and outward movements, the sensor's own coordinates rotate around the x-axis.



WT901BLE67(D5:17:71:B2:B2:67)

0.099 g	-0.025 g	1.042 g	1.047 g
加速度X	加速度Y	加速度Z	|加速度|
0.305 °/s	-1.709 °/s	-0.183 °/s	1.746 °/s
角速度X	角速度Y	角速度Z	|角速度|
-0.27 °	-5.48 °	-92.46 °	21.2 ℃
角度X	角度Y	角度Z	温度
-14.100 uT	-22.300 uT	-12.987 uT	29.407 uT
磁场X	磁场Y	磁场Z	|磁场|
0.69080	-0.03690	-0.03366	-0.72125
四元数0	四元数1	四元数2	四元数3
10080.1.20	40 %		
版本号	电量		

基于上述的分析，我们认为胯关节处有一个默认的固定不动的传感器，在我们进行六个动作的过程中，每个动作在一个轴上有大幅度的角度变换，我们用收到的 6 个传感器的数据扔给 12 个函数，通过让患者中间的停顿来保证排序后的中间时刻为初始状态角度值，然后用 MAX-MID,MID-MIN 来计算运动幅度并返回。

Based on the above analysis, we believe that there is a default fixed sensor at the hip joint. During our six movements, each movement has a significant angle transformation on an axis. We throw the data from the six sensors we receive to 12 functions, ensuring that the sorted intermediate moments are the initial state angle values by letting the patient pause in between. Then, we use MAX-MID and MID-MIN to calculate the motion amplitude and return it.

分析到这里基本就可以开始愉（zhe）悦（mo)的代码编写工作了，在我们的算法工程师一顿操作之后代码基本成型，过程中的问题在于传感器有 0**->**-0　-0**->**0　-180 **->**180 180**->**-180 的变化，我们的初始想法是遇到负数加 360，

后来在测试时发现这样的情况：初始 0　　增大最大到 30　　减小最小到-20

如果加 360 会导致正确的 MAX=30 会被 340 覆盖

再次进行试验发现突变最多只有一个方向，所以+360 与否取决于做什么动作（内收外展不需要，前屈后伸不需要，内旋外旋需要）

At this point in the analysis, we can basically start writing code. After our algorithm engineer's operation, the code was basically formed. The problem during the process was that the sensor had a 0->-0->0->0-180->180->-180 change. Our initial idea was to add 360 when encountering negative numbers,

Later, during testing, it was discovered that the initial value of 0 increased to a maximum of 30 and decreased to a minimum of -20

If 360 is added, it will cause the correct MAX=30 to be overwritten by 340

Further experiments revealed that the mutation can only occur in one direction at most, so whether or not+360 is determined by the action taken (adduction and abduction not required, flexion and extension not required, internal rotation and external rotation required)

代码(code):

```
import java.util.ArrayList;

import java.util.Collections;

import java.util.HashMap;
import java.util.Map;


class Sensor{
    Double AccX,AccY,AccZ;
    Double GyroX,GyroY,GyroZ;
    Double Roll,Pitch,Yaw;
    Sensor(double Roll,double Pitch,double Yaw){
        this.Roll = Roll;
        this.Pitch = Pitch;
        this.Yaw = Yaw;
    }
}
class Flexion{
    public static double Left(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor R2,Sensor
R3){
        double a = L1.Roll;
        return a;
    }
    public static double Right(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor
R2,Sensor R3){
        double a = R1.Roll;
        return a;
    }
}
class Extension{
    public static double Left(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor R2,Sensor
R3){
        double a = L1.Roll;
        return a;
    }
    public static double Right(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor
R2,Sensor R3){
        double a = R1.Roll;
        return a;
    }
}
class Abduction{
    public static double Left(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor R2,Sensor
R3){
        double a = L2.Roll;
        return a;
    }
    public static double Right(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor
R2,Sensor R3){
        double a = R2.Roll;
        return a;
    }
}
class Adduction{
    public static double Left(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor R2,Sensor
R3){
        double a = L2.Roll;
        return a;
    }
    public static double Right(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor
R2,Sensor R3){
        double a = R2.Roll;
        return a;
    }
}
class ExRotation{
    public static double Left(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor R2,Sensor
R3){
        double a = L3.Yaw;
        if(a<0) a += 360;
        return a;
    }
    public static double Right(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor
R2,Sensor R3){
        double a = R3.Yaw;
        if(a<0) a += 360;
        return a;
    }
}
class InRotation{
    public static double Left(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor R2,Sensor
R3){
        double a = L3.Yaw;
        if(a<0)   a += 360;
        return a;
    }
    public static double Right(Sensor L1,Sensor L2,Sensor L3,Sensor R1,Sensor
R2,Sensor R3){
        double a = R3.Yaw;
        if(a<0)   a += 360;
        return a;
    }
}
public class Analysis {
    static ArrayList<Double> FlexionLeft = new ArrayList<>();
    static ArrayList<Double> FlexionRight = new ArrayList<>();

    static ArrayList<Double> ExtensionLeft = new ArrayList<>();
    static ArrayList<Double> ExtensionRight = new ArrayList<>();

    static ArrayList<Double> AbductionLeft = new ArrayList<>();
    static ArrayList<Double> AbductionRight = new ArrayList<>();

    static ArrayList<Double> AdductionLeft = new ArrayList<>();
    static ArrayList<Double> AdductionRight = new ArrayList<>();

    static ArrayList<Double> ExRotationLeft = new ArrayList<>();
    static ArrayList<Double> ExRotationRight = new ArrayList<>();

    static ArrayList<Double> InRotationLeft = new ArrayList<>();
    static ArrayList<Double> InRotationRight = new ArrayList<>();

    static double GetMaxMid(ArrayList<Double> Angle){
        Collections.sort(Angle);
        int n = Angle.size();
        int delta = n / 20;
        int MIN = 0 + delta;
        int MID = n / 2;
        int MAX = n - delta;
        double res = Angle.get(MAX) - Angle.get(MID);
//          if(res < 0) res += 360;
        return res;
    }
    static double GetMidMin(ArrayList<Double> Angle){
        Collections.sort(Angle);
        int n = Angle.size();
        int delta = n / 20;
        int MIN = 0 + delta;
        int MID = n / 2;
        int MAX = n - delta;
        double res = Angle.get(MID) - Angle.get(MIN);
//          if(res < 0) res += 360;
        return res;
    }

    public static void Calculate(ArrayList<Double> Data){

        Sensor L1 = new Sensor(Data.get(6),Data.get(7),Data.get(8));
        Sensor L2 = new Sensor(Data.get(15),Data.get(16),Data.get(17));
        Sensor L3 = new Sensor(Data.get(24),Data.get(25),Data.get(26));
        Sensor R1 = new Sensor(Data.get(33),Data.get(34),Data.get(35));
        Sensor R2 = new Sensor(Data.get(42),Data.get(43),Data.get(44));
        Sensor R3 = new Sensor(Data.get(51),Data.get(52),Data.get(53));
```

```
            FlexionLeft.add(Flexion.Left(L1,L2,L3,R1,R2,R3));
            FlexionRight.add(Flexion.Right(L1,L2,L3,R1,R2,R3));

            ExtensionLeft.add(Extension.Left(L1,L2,L3,R1,R2,R3));
            ExtensionRight.add(Extension.Right(L1,L2,L3,R1,R2,R3));

            AbductionLeft.add(Abduction.Left(L1,L2,L3,R1,R2,R3));
            AbductionRight.add(Abduction.Right(L1,L2,L3,R1,R2,R3));

            AdductionLeft.add(Adduction.Left(L1,L2,L3,R1,R2,R3));
            AdductionRight.add(Adduction.Right(L1,L2,L3,R1,R2,R3));

            ExRotationLeft.add(ExRotation.Left(L1,L2,L3,R1,R2,R3));
            ExRotationRight.add(ExRotation.Right(L1,L2,L3,R1,R2,R3));

            InRotationLeft.add(InRotation.Left(L1,L2,L3,R1,R2,R3));
            InRotationRight.add(InRotation.Right(L1,L2,L3,R1,R2,R3));

//          ExternalRotationLeft.add(ExternalRotation.Left(L1,L2,L3,R1,R2,R3));
        }
        public static HashMap<String,Double> Statistic(){
            HashMap<String,Double> res = new HashMap<>();
            res.put("左前屈",GetMidMin(FlexionLeft));
            res.put("右前屈",GetMidMin(FlexionRight));

            res.put("左后伸",GetMaxMid(ExtensionLeft));
            res.put("右后伸",GetMaxMid(ExtensionRight));

            res.put("左外展",GetMidMin(AbductionLeft));
            res.put("右外展",GetMaxMid(AbductionRight));

            res.put("左内收",GetMaxMid(AdductionLeft));
            res.put("右内收",GetMidMin(AdductionRight));

            res.put("左外旋",GetMaxMid(ExRotationLeft));
            res.put("右外旋",GetMidMin(ExRotationRight));

            res.put("左内旋",GetMidMin(InRotationLeft));
            res.put("右内旋",GetMaxMid(InRotationRight));

            return res;
        }
}
```