四川大學

《Matlab 程序设计及应用》

课程训练研究报告

项目名称：

深空无人器翼展振动模量关键参数测量技术

项目组长：Zhong Hanming

课程训练项目要求（清晰写出本课程训练要做的内容和目标）

本项目目标：利用双目视觉非接触式测量手段实现大型天线、太阳翼等模态参数在轨辨识方法，具体在天线或太阳翼表面布设合作目标回光反射标志点作为测点，通过标志点提取技术及三角测量原理实现合作标记点处的振动位移信息计算。

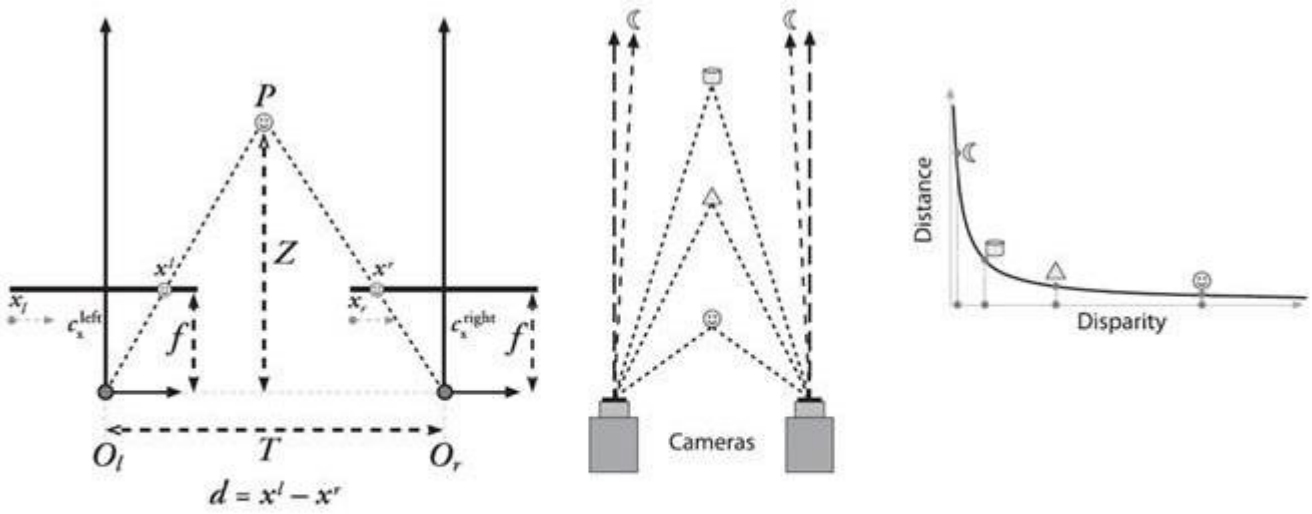
为此我们做了简化研究，通过对黑白棋盘格格子的标定实现双目摄像机的标定，再通过对特征点的提取实现对振动位移信息的分析。

研究报告内容

一． 对黑白棋盘格格子进行标定，从而实现对双目摄像机的标定

1、 原理

通过对两幅图像视差的计算，直接对前方景物（图像所拍摄到的范围）进行距离测量，而无需判断前方出现的是什么类型的障碍物。所以对于任何类型的障碍物，都能根据距离信息的变化，进行必要的预警或制动。双目摄像头的原理与人眼相似。人眼能够感知物体的远近，是由于两只眼睛对同一个物体呈现的图像存在差异，也称“视差”。物体距离越远，视差越小；反之，视差越大。视差的大小对应着物体与眼睛之间距离的远近，这也是 3D 电影能够使人有立体层次感知的原因。



$$\frac{T-(x^l-x^r)}{Z-f}=\frac{T}{Z} \quad \Rightarrow \quad Z=\frac{fT}{x^l-x^r}$$

假设有一个点 p，沿着垂直于相机中心连线方向上下移动，则其在左右相机上的成像点的位置会不断变化，即 d=x1-x2 的大小不断变化，并且点 p 和相机之间的距离 Z 跟视差 d 存在着反比关系。上式中视差 d 可以通过两个相机中心距 T 减去 p 点分别在左右图像上的投影点偏离中心点的值获得，所以只要获取到了两个相机的中心距 T,就可以评估出 p 点距离相机的距离，这个中心距 T 也是双目标定中需要确立的参数之一。

2、过程

第 1 步：准备一张棋盘格，粘贴于墙面，并用直尺测量黑白方格的真实物理长度。

第 2 步：调用双目摄像头，分别从不同角度拍摄得到一系列棋盘格图像。

前两步采用已有图像解决

第 3 步：采用工具箱 TOOLBOX_calib，利用左目图片数据集，进行左目相机标定，得到左目内参矩阵 K1、左目畸变系数向量 D1，并点击 save，得到 mat 文件。

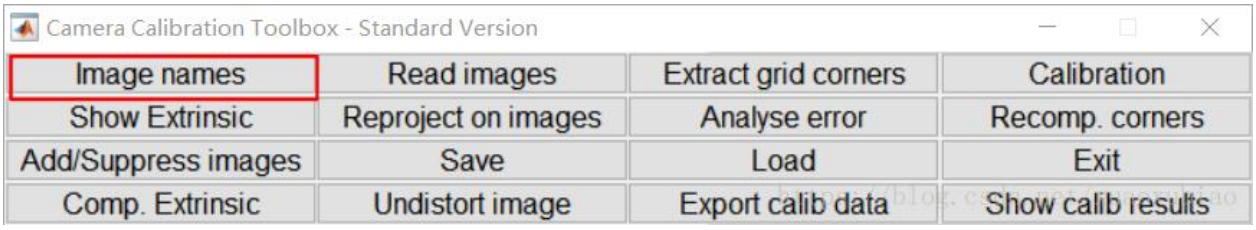


Figure 1 TOOLBOX calib **标定工具箱**

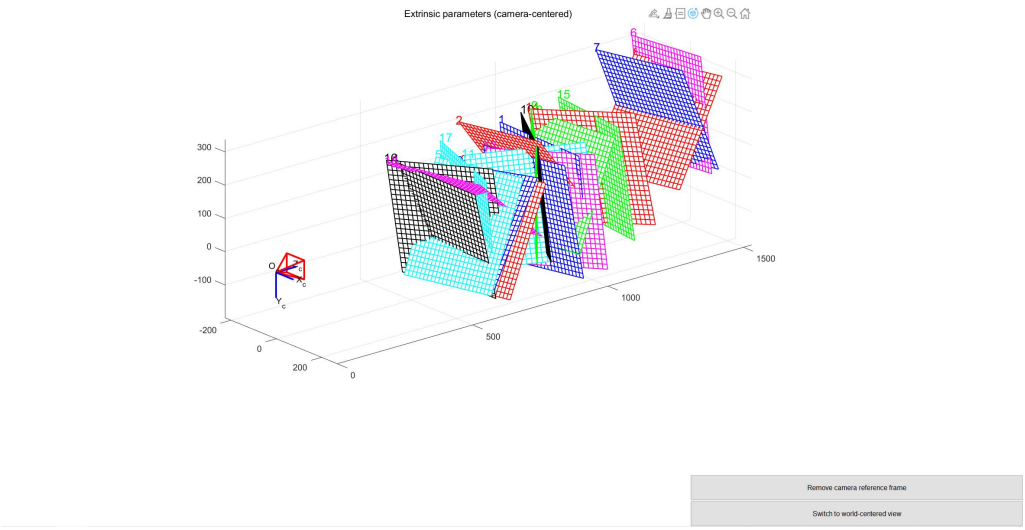


Figure 2 **左边相机的标定结果图**

第 4 步：同上，利用右目图片数据集，进行右目相机标定，得到右目内参矩阵 K2、右目畸变系数向量 D2，

第 5 步：命令行输入 stereo_gui，导入之前得到的文件夹，实现双目标定，点击 show extrinsi 可以显示照片与摄像头的关系图，以及双目摄像头之间的距离

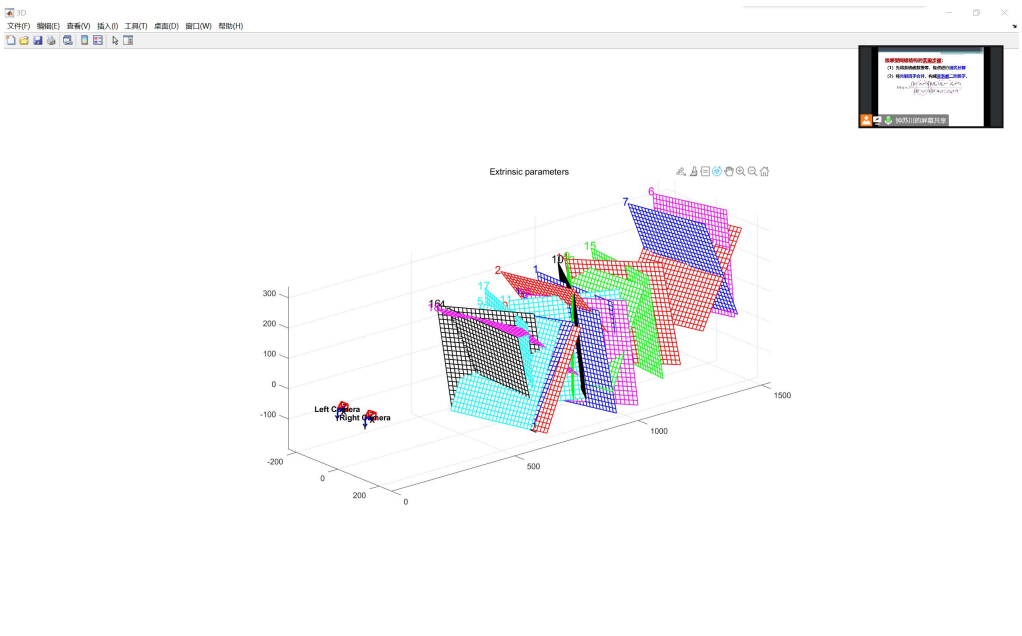


Figure 3 **最终两个相机的标定结果图**

下载工具箱并解压到 toolbox 文件夹，并在 matlab 中添加功能工具箱路径，

在 MATLAB 命令行窗口输入 calib_gui，弹出工具框，点击第一个选项输入图片前缀（right_或 left_）和图片格式（本次采用 jpg 格式），之后选择第一行第三个选项并依次标定即可。

值得注意的是标定过程必须一次标定正确，如果有错误只能重新开始，所以需要比较谨慎。

3、用到的算法和代码

Calib_gui

Stereo_gui

4、结论、总结和拓展

在我们进行实验的时候发现一些缺陷和问题。

第一， 利用本次的标定工具箱，只要标错一个点就必须从头再来。工作量显著增加。

第二， 部分照片存在着较大的畸变，有些照片即使调整了 Kc 即一阶镜头畸变系数得到的结果仍然不够理想。为此我们尝试了大量的 Kc 值，发现部分图像确实利用一阶畸变系数无法校准。

于是我们开始另辟蹊径发现 Matlab 的机器视觉工具箱中正好有相机标定和双目相机的工具箱，并且我们尝试了应用。发现结果比标定工具箱要好得不少，但也出现了新的问题。

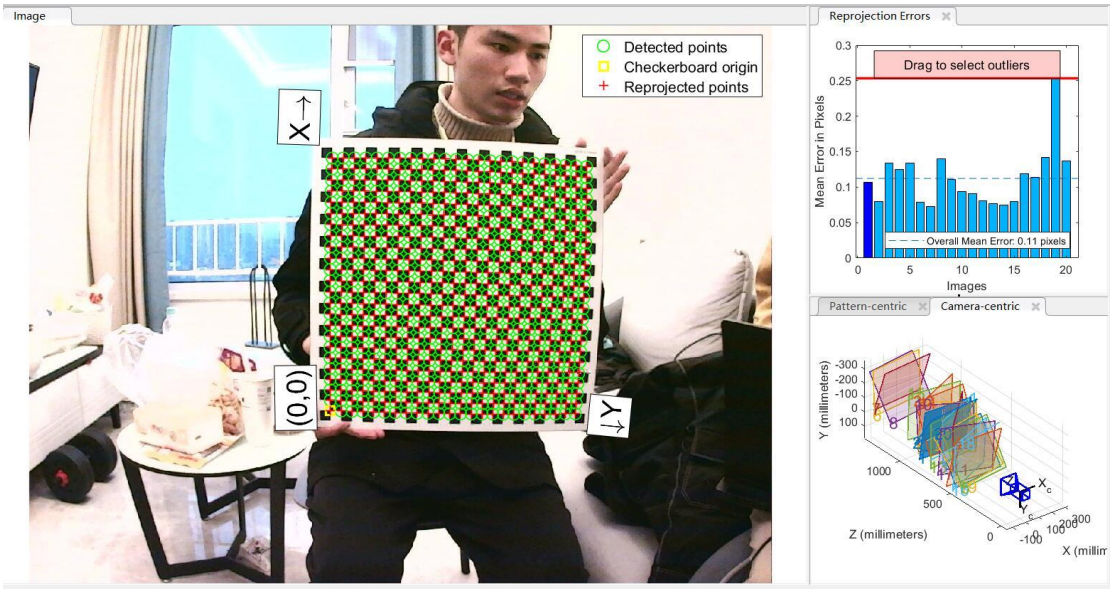


Figure 4 利用 matlab 机器视觉工具箱的左边相机标定

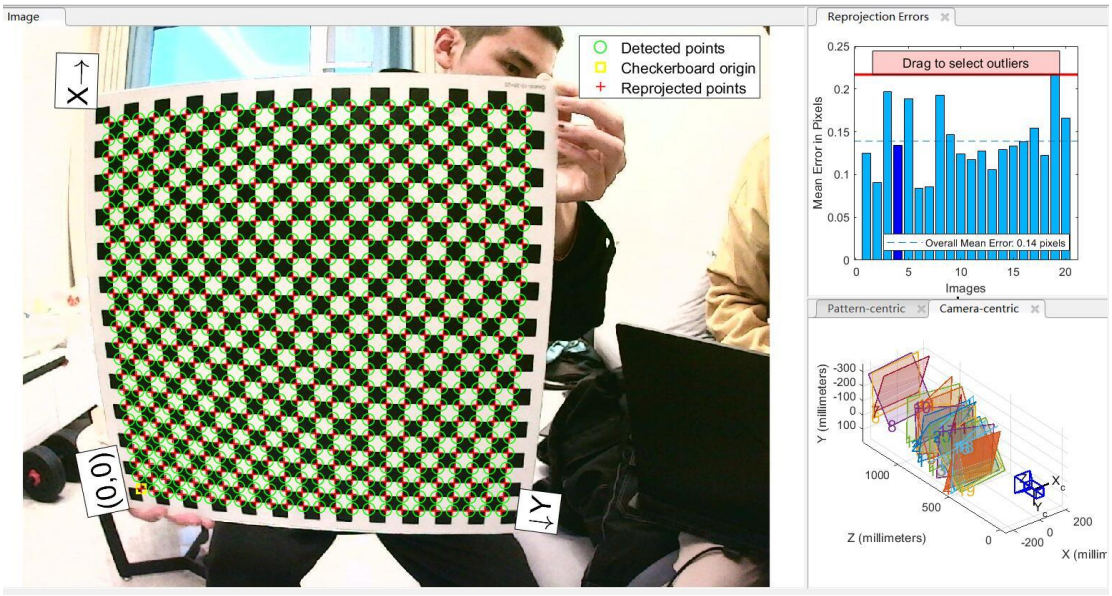


Figure 5 利用 matlab 机器视觉工具箱的右边图像标定

Matlab 的机器视觉工具箱能够很好地对单个相机进行标定，得到了近乎完美的结果，比标定工具箱好上不少，且实现了全自动化操作。

但是当我们期望利用两个相机分别的标定结果进行双目相机标定时出现了问题。具体来说，我们本来想的是利用机器视觉工具箱 app 的结果来进行双目标定，但是发现两个 result 的格式完全不一样无法使用。我们转而使用了 matlab 的双目标定 app，但是这次出现了报错且无法解决。仔细分析后是发现我们本次标定的棋盘是 23x23 的正方形，而 matlab 的机器视觉 app 不能进行正方形棋盘

的标定，因为有四个顶点都能成为原点 o，于是产生了数组大小不一的错误，无法继续下去。所以我们最终结合了二者取其精华。

二、进行编码特征点提取并且利用双目相机标定的结果来计算三维空间真实坐标

1、特征点提取

1）特征点提取：利用 imfindcircles 函数，发现采用圆形 Hough 变换圆。

三维坐标测量：根据相机的标定结果对校正后的图像进行像素点匹配，之后根据匹配结果计算每个像素的深度，从而得到具体的三维坐标。

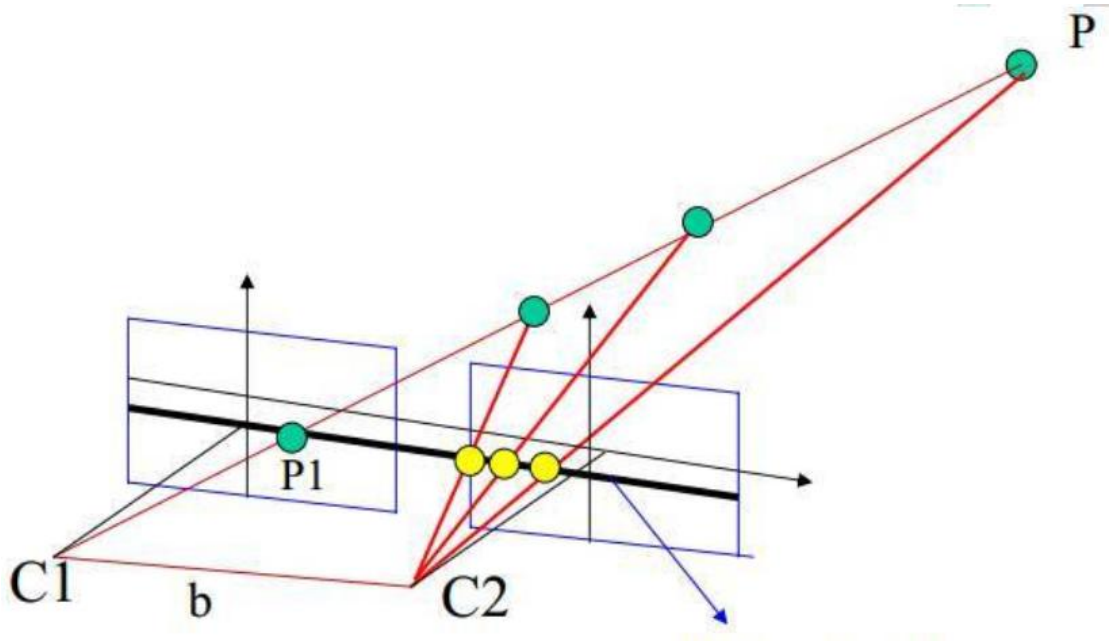


Figure 6 三维坐标测量的原理图

2)、过程

第一步： 对文件中图像进行读取处理；

第二步：利用 imfindcircles 函数为核心进行循环结构设计，同时打印出来圆中心的坐标结果；

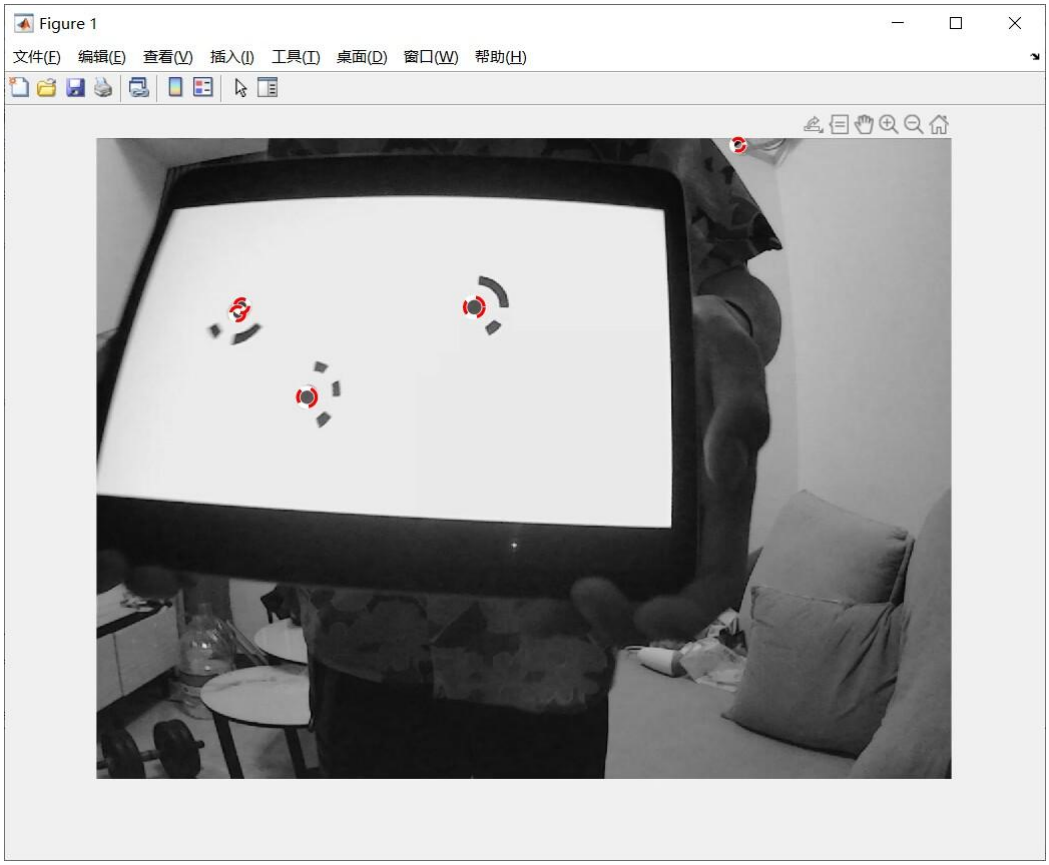


Figure 7 **特征点坐标提取**

第三步：对坐标结果进行处理分析，剔除无效坐标、缺失坐标；

第四步：进行三维空间真实坐标的计算。

3)、用到的算法和代码

```
clear
zuobiao=zeros(126,2);
%%×ó±ßÏà»ú
for(m1=1:42)
m2=m1;
name=strcat('left_',num2str(m2),'.jpg');
A=imread(name);



MyYuanLaiPic = imread(name);%¶ÁÈ¡RGB¸ñÊ½µÄÍ¼Ïñ
MyFirstGrayPic =
rgb2gray(MyYuanLaiPic);%ÓÃÒÑÓÐµÄ°¯Êý½øÐÐRGBµ½»Ò¶ÈÍ¼Ïñ
µÄ×ª»»
[rows , cols , colors] =
size(MyYuanLaiPic);%µÃµ½Ô-À´Í¼ÏñµÄ¾ØÕóµÄ²ÎÊý
MidGrayPic = zeros(rows ,
cols);%ÓÃµÃµ½µÄ²ÎÊý´´½¨Ò»¸öÈ«ÁãµÄ¾ØÕó£¬Õâ¸ö¾ØÕóÓÃ´´æ
```

```
´¢ÓÃÏÂÃæµÄ·½·¨²úÉúµÄ»Ò¶ÈÍ¼Ïñ
MidGrayPic =
uint8(MidGrayPic);%½«´´½¨µÄÈ«Áã¾ØÕó×ª»¯Îªuint8¸ñÊ½£¬Ò
òÎªÓÃÉÏÃæµÄÓï¾ä´´½¨Ö®°óÍ¼ÏñÊÇdoubleÐÍµÄ

for i = 1:rows
    for j = 1:cols
        sum = 0;
        for k = 1:colors
            sum = sum + MyYuanLaiPic(i , j , k) /
3;%½øÐÐ×ª»¯µÄ¹Ø¹ü¹«Ê½£¬sumÃ¿´Î¶¼ÒòÎª°óÃæµÄÊý×Ö¶ø²»ÄÜ³
¬¹ý255
        end
        MidGrayPic(i , j) = sum;
    end
end
imshow(MyYuanLaiPic);
imshow(MidGrayPic);
Rmin = 6;
Rmax = 15;
[centersBright, radiiBright] =
imfindcircles(MidGrayPic,[Rmin
Rmax],'ObjectPolarity','bright');
[centersDark, radiiDark] =
imfindcircles(MidGrayPic,[Rmin
Rmax],'ObjectPolarity','dark');
viscircles(centersBright, radiiBright,'Color','b');
viscircles(centersDark, radiiDark,'LineStyle','--');
centersBright;
centersDark;
B1=3*(m2-1)+1;
B2=3*m2;
disp(centersDark)
disp(m2)
outname=strcat('out-left_',num2str(m2),'.jpg')
print(outname,'-dpng')
end

%%ÓÒ±ßÏà»ú
for(m1=1:42)
m2=m1;
name=strcat('left_',num2str(m2),'.jpg');
A=imread(name);
```

```
MyYuanLaiPic = imread(name);%¶ÁÈ¡RGB¸ñÊ½µÄÍ¼Ïñ
MyFirstGrayPic =
rgb2gray(MyYuanLaiPic);%ÓÃÒÑÓÐµÄ°¯Êý½øÐÐRGBµ½»Ò¶ÈÍ¼Ïñ
µÄ×ª»»
[rows , cols , colors] =
size(MyYuanLaiPic);%µÃµ½Ô-À´Í¼ÏñµÄ¾ØÕóµÄ²ÎÊý
MidGrayPic = zeros(rows ,
cols);%ÓÃµÃµ½µÄ²ÎÊý´´½¨Ò»¸öÈ«ÁãµÄ¾ØÕó£¬Ô¸ö¾ØÕóÓÀ´´æ
´¢ÓÃÏÂÃæµÄ·½·¨²úÉúµÄ»Ò¶ÈÍ¼Ïñ
MidGrayPic =
uint8(MidGrayPic);%½«´´½¨µÄÈ«Áã¾ØÕó×ª»¯Îªuint8¸ñÊ½£¬Ò
òÎªÓÃÉÏÃæµÄÓï¾ä´´½¨Ö®óÍ¼ÏñÊÇdoubleÐÍµÄ

for i = 1:rows
    for j = 1:cols
        sum = 0;
        for k = 1:colors
            sum = sum + MyYuanLaiPic(i , j , k) /
3;%½øÐÐ×ª»¯µÄ¹Ø¼ü¹«Ê½£¬sumÃ¿´Î¶¼ÒòÎª°óÃæµÄÊý×Ö¶ø²»ÄÜ³
¬¹ý255
        end
        MidGrayPic(i , j) = sum;
    end
end
imshow(MyYuanLaiPic);
imshow(MidGrayPic);
Rmin = 6;
Rmax = 15;
[centersBright, radiiBright] =
imfindcircles(MidGrayPic,[Rmin
Rmax],'ObjectPolarity','bright');
[centersDark, radiiDark] =
imfindcircles(MidGrayPic,[Rmin
Rmax],'ObjectPolarity','dark');
viscircles(centersBright, radiiBright,'Color','b');
viscircles(centersDark, radiiDark,'LineStyle','--');
centersBright;
centersDark;
B1=3*(m2-1)+1;
B2=3*m2;
disp(centersDark)
```

```
disp(m2)
outname=strcat('out-left_',num2str(m2),'.jpg')
print(outname,'-dpng')
end
```

opencv 部分：

```
Mat jiaozheng( Mat image )
{
    Size image_size = image.size();
    float                       intrinsic[3][3]                       =
{589.2526583947847,0,321.8607532099886,0,585.7784771038199,251.033852
8599469,0,0,1};
    float distortion[1][5] = {-0.5284205687061442, 0.3373615384253201,
-0.002133029981628697, 0.001511983002864886, -0.1598661778309496};
    Mat intrinsic_matrix = Mat(3,3,CV_32FC1,intrinsic);
    Mat distortion_coeffs = Mat(1,5,CV_32FC1,distortion);
    Mat R = Mat::eye(3,3,CV_32F);
    Mat mapx = Mat(image_size,CV_32FC1);
    Mat mapy = Mat(image_size,CV_32FC1);

initUndistortRectifyMap(intrinsic_matrix,distortion_coeffs,R,intrinsi
c_matrix,image_size,CV_32FC1,mapx,mapy);
    Mat t = image.clone();
    cv::remap( image, t, mapx, mapy, INTER_LINEAR);
    return t;
}

//opencv2.4.9 vs2012
#include <opencv2\opencv.hpp>
#include <fstream>
#include <iostream>

using namespace std;
using namespace cv;

Point2f    xyz2uv(Point3f    worldPoint,float    intrinsic[3][3],float
translation[1][3],float rotation[3][3]);
Point3f uv2xyz(Point2f uvLeft,Point2f uvRight);

//图片对数量
int PicNum = 14;
```

```
//左相机内参数矩阵
float leftIntrinsic[3][3] = {4037.82450,          0,     947.65449,
                                0,    3969.79038,        455.48718,
                                0,             0,             1};
//左相机畸变系数
float leftDistortion[1][5] = {0.18962, -4.05566, -0.00510, 0.02895, 0};
//左相机旋转矩阵
float leftRotation[3][3] = {0.912333,      -0.211508,      0.350590,
                          0.023249,      -0.828105,     -0.560091,
                          0.408789,       0.519140,     -0.750590};
//左相机平移向量
float leftTranslation[1][3] = {-127.199992, 28.190639, 1471.356768};

//右相机内参数矩阵
float rightIntrinsic[3][3] = {3765.83307,          0,     339.31958,
                                0,  3808.08469,        660.05543,
                                0,             0,             1};
//右相机畸变系数
float rightDistortion[1][5] = {-0.24195, 5.97763, -0.02057, -0.01429,
0};
//右相机旋转矩阵
float rightRotation[3][3] = {-0.134947,         0.989568,
    -0.050442,
                           0.752355,         0.069205,     -0.655113,
                          -0.644788,        -0.126356,     -0.753845};
//右相机平移向量
float rightTranslation[1][3] = {50.877397, -99.796492, 1507.312197};


int main()
{
    //已知空间坐标求成像坐标
    Point3f point(700,220,530);
    cout<<"左相机中坐标："<<endl;
    cout<<xyz2uv(point,leftIntrinsic,leftTranslation,leftRotation)<<e
ndl;
    cout<<"右相机中坐标："<<endl;
    cout<<xyz2uv(point,rightIntrinsic,rightTranslation,rightRotation)
<<endl;

    //已知左右相机成像坐标求空间坐标
    Point2f                              l                              =
xyz2uv(point,leftIntrinsic,leftTranslation,leftRotation);
    Point2f                              r                              =
```

```
xyz2uv(point,rightIntrinsic,rightTranslation,rightRotation);
    Point3f worldPoint;
    worldPoint = uv2xyz(l,r);
    cout<<"空间坐标为:"<<endl<<uv2xyz(l,r)<<endl;

    system("pause");

    return 0;
}


//*************************************
// Description: 根据左右相机中成像坐标求解空间坐标
// Method:    uv2xyz
// FullName:  uv2xyz
// Access:    public
// Parameter: Point2f uvLeft
// Parameter: Point2f uvRight
// Returns:   cv::Point3f
//*************************************
Point3f uv2xyz(Point2f uvLeft,Point2f uvRight)
{
    //  [u1]      |X|                  [u2]      |X|
    //Z*|v1| = Ml*|Y|                Z*|v2| = Mr*|Y|
    //  [ 1]      |Z|                  [ 1]      |Z|
    //           |1|                             |1|
    Mat mLeftRotation = Mat(3,3,CV_32F,leftRotation);
    Mat mLeftTranslation = Mat(3,1,CV_32F,leftTranslation);
    Mat mLeftRT = Mat(3,4,CV_32F);//左相机M矩阵
    hconcat(mLeftRotation,mLeftTranslation,mLeftRT);
    Mat mLeftIntrinsic = Mat(3,3,CV_32F,leftIntrinsic);
    Mat mLeftM = mLeftIntrinsic * mLeftRT;
    //cout<<"左相机M矩阵 = "<<endl<<mLeftM<<endl;

    Mat mRightRotation = Mat(3,3,CV_32F,rightRotation);
    Mat mRightTranslation = Mat(3,1,CV_32F,rightTranslation);
    Mat mRightRT = Mat(3,4,CV_32F);//右相机M矩阵
    hconcat(mRightRotation,mRightTranslation,mRightRT);
    Mat mRightIntrinsic = Mat(3,3,CV_32F,rightIntrinsic);
    Mat mRightM = mRightIntrinsic * mRightRT;
    //cout<<"右相机M矩阵 = "<<endl<<mRightM<<endl;

    //最小二乘法A矩阵
    Mat A = Mat(4,3,CV_32F);
```

```
    A.at<float>(0,0)    =    uvLeft.x    *    mLeftM.at<float>(2,0)    -
mLeftM.at<float>(0,0);
    A.at<float>(0,1)    =    uvLeft.x    *    mLeftM.at<float>(2,1)    -
mLeftM.at<float>(0,1);
    A.at<float>(0,2)    =    uvLeft.x    *    mLeftM.at<float>(2,2)    -
mLeftM.at<float>(0,2);

    A.at<float>(1,0)    =    uvLeft.y    *    mLeftM.at<float>(2,0)    -
mLeftM.at<float>(1,0);
    A.at<float>(1,1)    =    uvLeft.y    *    mLeftM.at<float>(2,1)    -
mLeftM.at<float>(1,1);
    A.at<float>(1,2)    =    uvLeft.y    *    mLeftM.at<float>(2,2)    -
mLeftM.at<float>(1,2);

    A.at<float>(2,0)    =   uvRight.x   *   mRightM.at<float>(2,0)    -
mRightM.at<float>(0,0);
    A.at<float>(2,1)    =   uvRight.x   *   mRightM.at<float>(2,1)    -
mRightM.at<float>(0,1);
    A.at<float>(2,2)    =   uvRight.x   *   mRightM.at<float>(2,2)    -
mRightM.at<float>(0,2);

    A.at<float>(3,0)    =   uvRight.y   *   mRightM.at<float>(2,0)    -
mRightM.at<float>(1,0);
    A.at<float>(3,1)    =   uvRight.y   *   mRightM.at<float>(2,1)    -
mRightM.at<float>(1,1);
    A.at<float>(3,2)    =   uvRight.y   *   mRightM.at<float>(2,2)    -
mRightM.at<float>(1,2);

    //最小二乘法 B 矩阵
    Mat B = Mat(4,1,CV_32F);
    B.at<float>(0,0)    =    mLeftM.at<float>(0,3)    -    uvLeft.x    *
mLeftM.at<float>(2,3);
    B.at<float>(1,0)    =    mLeftM.at<float>(1,3)    -    uvLeft.y    *
mLeftM.at<float>(2,3);
    B.at<float>(2,0)    =   mRightM.at<float>(0,3)   -   uvRight.x   *
mRightM.at<float>(2,3);
    B.at<float>(3,0)    =   mRightM.at<float>(1,3)   -   uvRight.y   *
mRightM.at<float>(2,3);

    Mat XYZ = Mat(3,1,CV_32F);
    //采用 SVD 最小二乘法求解 XYZ
    solve(A,B,XYZ,DECOMP_SVD);

    //cout<<"空间坐标为 = "<<endl<<XYZ<<endl;
```

```
    //世界坐标系中坐标
    Point3f world;
    world.x = XYZ.at<float>(0,0);
    world.y = XYZ.at<float>(1,0);
    world.z = XYZ.at<float>(2,0);

    return world;
}

//************************************
// Description: 将世界坐标系中的点投影到左右相机成像坐标系中
// Method:    xyz2uv
// FullName:  xyz2uv
// Access:    public
// Parameter: Point3f worldPoint
// Parameter: float intrinsic[3][3]
// Parameter: float translation[1][3]
// Parameter: float rotation[3][3]
// Returns:   cv::Point2f
// Author:    小白
// Date:      2017/01/10
// History:
//************************************
Point2f    xyz2uv(Point3f    worldPoint,float    intrinsic[3][3],float
translation[1][3],float rotation[3][3])
{
    //    [fx s x0]                          [Xc]       [Xw]       [u]
1      [Xc]
    //K = |0 fy y0|       TEMP = [R T]     |Yc| = TEMP*|Yw|     | | =
—*K *|Yc|
    //    [ 0 0 1 ]                          [Zc]       |Zw|       [v]
Zc  [Zc]
    //                                                 [1 ]
    Point3f c;
    c.x = rotation[0][0]*worldPoint.x + rotation[0][1]*worldPoint.y +
rotation[0][2]*worldPoint.z + translation[0][0]*1;
    c.y = rotation[1][0]*worldPoint.x + rotation[1][1]*worldPoint.y +
rotation[1][2]*worldPoint.z + translation[0][1]*1;
    c.z = rotation[2][0]*worldPoint.x + rotation[2][1]*worldPoint.y +
rotation[2][2]*worldPoint.z + translation[0][2]*1;

    Point2f uv;
    uv.x    =    (intrinsic[0][0]*c.x    +    intrinsic[0][1]*c.y    +
```

```
intrinsic[0][2]*c.z)/c.z;
    uv.y    =    (intrinsic[1][0]*c.x    +    intrinsic[1][1]*c.y    +
intrinsic[1][2]*c.z)/c.z;

    return uv;
}
```

2、 计算对应特征点的三维坐标

通过查阅题目中给出的" stereo_triangulation"帮助文档可以知道，该函数内部参数主要有

输入部分：

xL:左图像素坐标的 2xN 矩阵

xR:右侧图像像素坐标的 2xN 矩阵

om,T:左右相机之间的旋转矢量和平移矢量(立体标定输出)

fc_left, cc_left,…:左相机固有参数(立体标定输出)

fc_right, cc_right,…:右侧相机的固有参数(立体标定输出)

输出部分：

XL:左侧相机参考系中点坐标的 3xN 矩阵

XR:右相机参考系中点坐标的 3xN 矩阵

在第一问中通过 stereo_gui 工具箱，我们可以得出相机的内参矩阵，将得到的内参矩阵带入函数，同时将提取的左右特征点坐标分别用矩阵 xL 和矩阵 xR 表示，代码如下：

```
clc;
clear;
xL = readtable ('xL.txt');
xL = table2array(xL)
xR = readtable('xR.txt');
xR = table2array(xR)
load('Calib_Results_stereo.mat');
[XL,XR]                                                    =
stereo_triangulation(xL,xR,om,T,fc_left,cc_left,kc_left,alpha_c_le
ft,fc_right,cc_right,kc_right,alpha_c_right);
```

同时我们对其中内置的"stereo_triangulation"再次进行分析，并做了部分注释：

```
function                    [XL,XR]                        =
stereo_triangulation(xL,xR,om,T,fc_left,cc_left,kc_left,alpha_c_le
ft,fc_right,cc_right,kc_right,alpha_c_right),

%                           [XL,XR]                        =
stereo_triangulation(xL,xR,om,T,fc_left,cc_left,kc_left,alpha_c_le
ft,fc_right,cc_right,kc_right,alpha_c_right),
%
% Function that computes the position of a set on N points given the
left and right image projections.
```

```
% The cameras are assumed to be calibrated, intrinsically, and
extrinsically.
%
% Input:
%           xL: 2xN matrix of pixel coordinates in the left image
%           xR: 2xN matrix of pixel coordinates in the right image
%            om,T: rotation vector and translation vector between
right and left cameras (output of stereo calibration)
%            fc_left,cc_left,...: intrinsic parameters of the left
camera  (output of stereo calibration)
%           fc_right,cc_right,...: intrinsic parameters of the right
camera (output of stereo calibration)
%
% Output:
%
%           XL: 3xN matrix of coordinates of the points in the left
camera reference frame
%           XR: 3xN matrix of coordinates of the points in the right
camera reference frame
%
% Note: XR and XL are related to each other through the rigid motion
equation: XR = R * XL + T, where R = rodrigues(om)
%          For          more          information,          visit
http://www.vision.caltech.edu/bouguetj/calib_doc/htmls/example5.ht
ml
%
%
% (c) Jean-Yves Bouguet - Intel Corporation - April 9th, 2003



%--- Normalize the image projection according to the intrinsic
parameters of the left and right cameras
%相机标定内参中的γ (alpha_c)若为零，表示像素坐标系 u、v 轴成直角，
若不为零像素坐标系 u、v 轴不成直角
%在引用 alpha_c 计算时，需要注意 alpha_c=alpha_c/fc(1),即 γ=γ/fx

alpha_c_left = alpha_c_left/fc_left(1);
alpha_c_right = alpha_c_right/fc_right(1);

%option 1 Common camera model 没有径向畸变 K3,即 K3=0
xt = normalize_pixel(xL,fc_left,cc_left,kc_left,alpha_c_left);
xtt = normalize_pixel(xR,fc_right,cc_right,kc_right,alpha_c_right);
%option 2 Special camera model 带有有径向畸变 K3
```

```
%                                   xt                                  =
normalize_pixel_fisheye(xL,fc_left,cc_left,kc_left,alpha_c_left);
%                                   xtt                                 =
normalize_pixel_fisheye(xR,fc_right,cc_right,kc_right,alpha_c_righ
t);

%--- Extend the normalized projections in homogeneous coordinates
xt = [xt;ones(1,size(xt,2))];
xtt = [xtt;ones(1,size(xtt,2))];

%--- Number of points:
N = size(xt,2);

%--- Rotation matrix corresponding to the rigid motion between left
and right cameras:
R = rodrigues(om);


%--- Triangulation of the rays in 3D space:

u = R * xt;

n_xt2 = dot(xt,xt);
n_xtt2 = dot(xtt,xtt);

T_vect = repmat(T, [1 N]);

DD = n_xt2 .* n_xtt2 - dot(u,xtt).^2;

dot_uT = dot(u,T_vect);
dot_xttT = dot(xtt,T_vect);
dot_xttu = dot(u,xtt);

NN1 = dot_xttu.*dot_xttT - n_xtt2 .* dot_uT;
NN2 = n_xt2.*dot_xttT - dot_uT.*dot_xttu;

Zt = NN1./DD;
Ztt = NN2./DD;

X1 = xt .* repmat(Zt,[3 1]);
X2 = R'*(xtt.*repmat(Ztt,[3,1])  - T_vect);


%--- Left coordinates:
```

```
XL = 1/2 * (X1 + X2);

%--- Right coordinates:
XR = R*XL + T_vect;
```

以上就是代码部分的内容，最终得到输出
XL:左侧相机参考系中点坐标的 3xN 矩阵；
XR:右相机参考系中点坐标的 3xN 矩阵；

4)、结论、总结和拓展

在进行第二部分的时候是最为困难且遇到最大问题的部分。同时有一些总结和拓展，具体来说有如下一些部分：

1. 圆提取不全且容易提取到边缘目标：imfindcircles 函数掌握不精，理论上来说我们能通过优化参数来进行更为细致的提取，但是时间有些且经验不足走了很多弯路；
2. 圆提取的时候，根据其他组的成功经验尝试了 surf 算法，但是不熟悉，不过也拓展了思路，但是 surf 算法存在以下的缺点：第一，圆提取过多无法进行有效筛选，具体表现为三个特征点的都会提取出众多坐标，且调参的效果不好；第二，无法实现自动化，必须手动提取，因为 surf 算法提取的圆很多，需要手动一张一张图片进行手动三个点分别提取，工作量巨大，84 张图片总共 252 个点，只能手动记录，效率低下；
3. 在计算三维坐标的时候没有发现 calib_toolbox 的自带的三维坐标计算，所以利用了 opencv 的三维坐标计算代码。之后跑通了之后又用回了 matlab 的 calib——toolbox 工具箱的算法
4. 圆提取可以利用 matlab 来调用 OpenCV 来进行，我们借鉴了别人成功的思路，利用 opencv 复现了 Arc-support Line Segments Revisited: An Efficient High-quality Ellipse Detection 的椭圆检测算法，可做了尝试后发现效果不好后放弃。

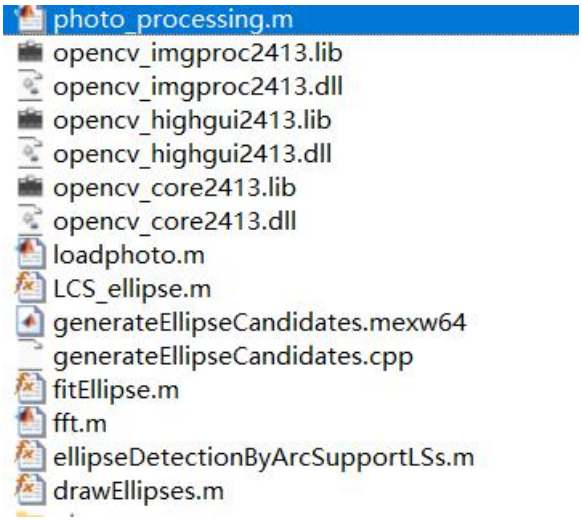


Figure 8 复现了论文调用了 OpenCV 后封装好进行椭圆提取，但提取效果不佳

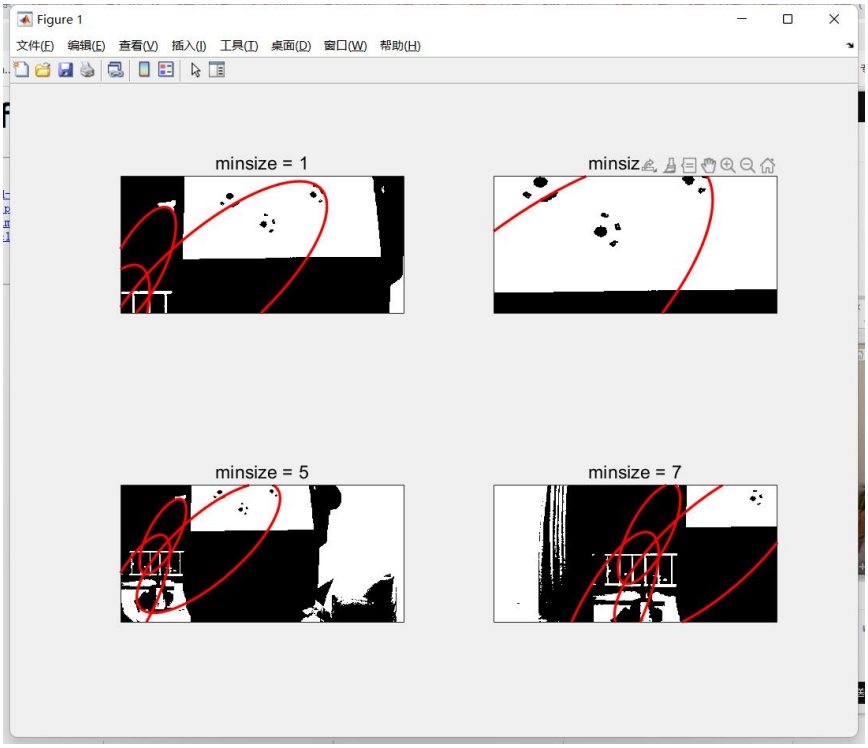


Figure 9 调用 OpenCV 进行提取

三．对三维坐标进行频谱分析，分析其振动频率

1. 原理

对于求到的三维离散坐标，利用快速傅里叶变换 FFT 分析其频谱。 FFT 的基本思想是把原始的 N 点序列，依次分解成一系列的短序列。充分利用 DFT 计算式中指数因子 所具有的对称性质和周期性质，进而求出这些短序列相应的 DFT 并进行适当组合，达到删除重复计算，减少乘法运算和简化结构的目的。此后，

在这思想基础上又开发了高基和分裂基等快速算法，当 N 是素数时，可以将 DFT 算转化为求循环卷积，从而更进一步减少乘法次数，提高速度。 增加 N 可以减少窗函数频谱的主瓣宽度，改善频率分辨率

增加 N 也会增加旁瓣的能量，导致更多的频率泄露，影响信号中较弱频率分量的分辨。

采样定理告诉我们，采样频率要大于信号频率的两倍。 假设采样频率为 Fs，信号频率为 F，那就因该 Fs > 2F。 采样得到的数字信号，就可以做 FFT 变换了。N 个采样点，经过 FFT 之后，就可以得到 N 个点的 FFT 结果。结果结果是左右对称的，所以上面图中的结果只取了一半 256/2=128。为了方便进行 FFT 运算，通常 N 取 2 的整数次方。 分辨率：例如某点 n 所表示的频率为：Fn=(n-1)*Fs/N。则 Fn 所能分辨到频率为 Fs / N。如果采样频率 Fs 为 1024Hz，采样点数为 1024 点，则可以分辨到 1Hz。 幅值：那么 FFT 之后结果就是一个为 N 点的复数。每一个点就对应着一个频率点。这个点的模值，就是该频率值下的幅度特性。

2.过程

第一步：将不同时刻 x、y、z 的坐标读成三个向量

第二步：利用 fft 分别分析三个方向的振动情况

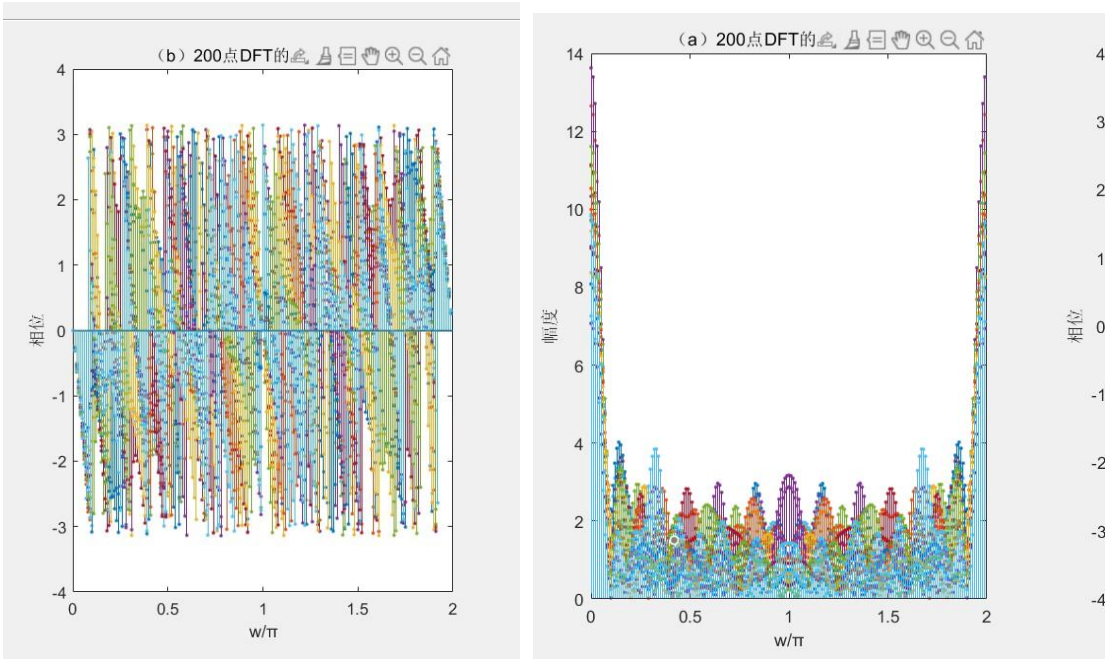
3.代码（以 x 方向为例）

```
clear
zn=Ztotal                          %输入时域序列向量
yn=Ytotal;
xn=Xtotal;
Xk200= fft(xn, 200);                    % 计算 xn 的 200 点 fft
Xk1500 = fft(xn, 1500);                    % 计算 xn 的 1500 点 fft

% 以下为绘图部分
k = 0 : 199;
wk = 2*k/200;                           %计算 200 点 DFT 对应的采样点频率
subplot(1,2,1);
stem(wk, abs(Xk200), '.');                 %绘制 200 点 DFT 的幅频特性图
```

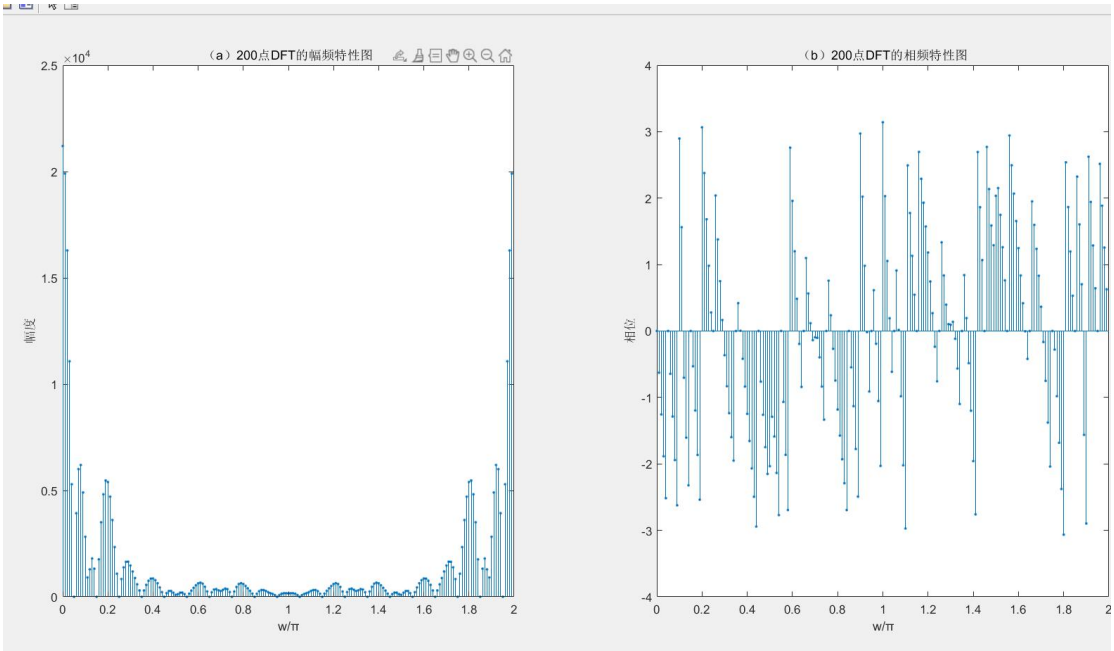
```
title('（a）200 点 DFT 的幅频特性图');
 xlabel('w/π');
 ylabel(' 幅度 ');

subplot(1,2,2);
stem(wk, angle(Xk200), '.');          %绘制 200 点 DFT 的相频特性图
line([0,2], [0,0]);
title('（b）200 点 DFT 的相频特性图');
xlabel('w/π');
ylabel(' 相位 ');
```

4.结果与结论

X 与 Y 方向频谱分析结果



Z 方向的频谱分析结果



结论：X、Y 方向规律不明显、Z 方向成一定规律的振动

展望：

1、 可以计算出具体的振动频率，结合拍摄时间等

2、 X、Y 方向也并不是无规律振动，而是以手臂为支点的小规模转动，可以根据现有的数据将规律建模分析