Appendix A C++代码

1. 中值滤波头文件

```
#pragma once
#include<iostream>
#include<opencv2/opencv.hpp>
#include <random>


using namespace cv;
using namespace std;

Mat med_filter(Mat image);
```

2. 中值滤波源文件

```
#include "med_filter.h"

Mat med_filter(Mat image){
	Mat image_output;
	if (image.empty()){		//读取失败
		cout << "读取错误" << endl;
	}

	int a = 7;				//用滤波核大小 7x7 做中值滤波
	medianBlur(image, image_output, a);
	imwrite("G:/cpp/Instrument_homework/ball_med_filter.png", image_output);

	return image_output;
}
```

3. 二值化头文件

```
#pragma once

#include <iostream>
#include <string>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

int OTSU(Mat srcImage);
Mat bin(Mat srcImage, int threshould);
```

4. 二值化源文件

```
#include "binarization.h"

int OTSU(Mat srcImage) {
	int nCols = srcImage.cols;
	int nRows = srcImage.rows;
	int threshold = 0;
	// 初始化统计参数
	int nSumPix[256];
	float nProDis[256];
	for (int i = 0; i < 256; i++) {
		nSumPix[i] = 0;
		nProDis[i] = 0;
	}
	// 统计灰度级中每个像素在整幅图像中的个数
	for (int i = 0; i < nRows; i++) {
		for (int j = 0; j < nCols; j++) {
			nSumPix[(int)srcImage.at<uchar>(i, j)]++;
		}
	}
	// 计算每个灰度级占图像中的概率分布
	for (int i = 0; i < 256; i++) {
		nProDis[i] = (float)nSumPix[i] / (nCols * nRows);
	}
	// 遍历灰度级[0,255],计算出最大类间方差下的阈值
	float w0, w1, u0_temp, u1_temp, u0, u1, delta_temp;
	double delta_max = 0.0;
	for (int i = 0; i < 256; i++) {
		// 初始化相关参数
		w0 = w1 = u0_temp = u1_temp = u0 = u1 = delta_temp = 0;
		for (int j = 0; j < 256; j++) {
			//背景部分
			if (j <= i) {
				// 当前 i 为分割阈值，第一类总的概率
				w0 += nProDis[j];
				u0_temp += j * nProDis[j];
			}
			//前景部分
			else {
				// 当前 i 为分割阈值，第一类总的概率
				w1 += nProDis[j];
				u1_temp += j * nProDis[j];
			}
		}
```

```
            // 分别计算各类的平均灰度
            u0 = u0_temp / w0;
            u1 = u1_temp / w1;
            delta_temp = (float)(w0 * w1 * pow((u0 - u1), 2));
            // 依次找到最大类间方差下的阈值
            if (delta_temp > delta_max) {
                delta_max = delta_temp;
                threshold = i;
            }
        }
        return threshold;
}

Mat bin(Mat srcImage, int threshould) {
        Mat imageoutput;
        imageoutput = cv::Mat::zeros(srcImage.rows, srcImage.cols, CV_8UC1);
        //二值化
        for (int i = 0; i < srcImage.rows; i++) {
            for (int j = 0; j < srcImage.cols; j++) {
                // 满足大于等于阈值置 255
                if (srcImage.at<uchar>(i, j) >= threshould)
                    imageoutput.at<uchar>(i, j) = 255;
                else
                    imageoutput.at<uchar>(i, j) = 0;
            }
        }

        return imageoutput;
}
```

5. 圆心&直径计算头文件

```
#pragma once

#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

Point centre(Mat srcImage, Mat bin);
```

6. 圆心&直径计算源文件

```
#include "centre.h"
#include <cmath>

#define pi 3.1415926

Point centre(Mat srcImage, Mat bin) {
	Mat close_dst, open_dst;
	//形态学操作
	Mat kernel = getStructuringElement(MORPH_RECT, Size(3, 3), Point(-1, -1));
	morphologyEx(bin, close_dst, MORPH_CLOSE, kernel, Point(-1, -1));		//
闭操作
	kernel = getStructuringElement(MORPH_RECT, Size(7, 7), Point(-1, -1));	//
开操作
	morphologyEx(bin, open_dst, MORPH_OPEN, kernel, Point(-1, -1));

	Mat dst = Mat::zeros(bin.size(), CV_8UC3);
	vector <vector<Point>> contours;
	vector<Vec4i> hierachy;
	findContours(open_dst, contours, hierachy, RETR_TREE,
CHAIN_APPROX_SIMPLE, Point(0, 0));
	Point cc;
	Mat result_img = srcImage.clone();
	//cvtColor(result_img, result_img, COLOR_GRAY2BGR);

	//RNG rng((int)time(0));
	for (size_t t = 0; t < contours.size(); t++){

		double area = contourArea(contours[t]);			//获取每个轮廓围成的
面积
		if (area < 100)						//如果轮廓围成的面积
小于 100，则过滤
			continue;
		//横纵比过滤
		Rect rect = boundingRect(contours[t]);		//用最小的外接矩形把对象
给包起来

		float h_w = (float)rect.height / rect.width;
		if (h_w > 0.9 && h_w < 1.2){
			drawContours(dst, contours, static_cast<int>(t), Scalar(0, 0, 255), 2, 8,
Mat(), 0, Point(0, 0));
			double length = arcLength(contours[t], true);//周长

			//printf("面积:%f\n", area);
			//printf("周长:%f\n", length);

			//获取圆点坐标
			int x = rect.x + rect.width / 2;  //rect.x 是矩形左上角的横坐标
			int y = rect.y + rect.height / 2; //rect.y 是矩形左上角的纵坐标
			cc = Point(x, y);
			//Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), 
rng.uniform(0, 255));
			circle(result_img, cc, 2, Scalar(0, 0, 255), 2, 8, 0);

			//用面积计算，单位是像素数量
			double radius_area = sqrt(area / pi) * 2;
			printf("\n\n 直径:%f\n\n", radius_area);

			//用周长计算，单位是像素单位
			//double radius_len = length / 2 / pi;
			//printf("\n\n 半径:%f\n\n", radius_len);

		}
	}

	//imshow("contours image", dst);
	//imshow("result image", result_img);
	//waitKey(0);
	imwrite("G:/cpp/Instrument_homework/contour.png", dst);
	imwrite("G:/cpp/Instrument_homework/centre.png", result_img);

	return cc;
}
```

7. 主函数

```
#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include "med_filter.h"
#include "binarization.h"
#include "centre.h"

int main() {
	Mat image, image_filter, image_filter_Gray, image_otsu, image_otsu_filter, 
image_edge_canny, image_edge_sobel;
	double otsu_thresh_val;
	string path_ini;
	const int maxVal = 255;
	path_ini = "G:/cpp/Instrument_homework/ball_ini.png";
	image = imread(path_ini);					//读取
	image_filter = med_filter(image);	//中值滤波
	cvtColor(image_filter, image_filter_Gray, COLOR_RGB2GRAY);	//转换为灰
度图像
	int otsu_threshould = OTSU(image_filter_Gray);			//Otsu 计算阈值
T
	cout << "Threshould\t" << otsu_threshould << endl;
	image_otsu = bin(image_filter_Gray, otsu_threshould);		//二值化
	imwrite("G:/cpp/Instrument_homework/ball_bin.png", image_otsu);
	//imshow("11", image_otsu);

	GaussianBlur(image_otsu, image_otsu_filter, Size(5, 5), 2, 2);	//高斯滤波
	Canny(image_otsu_filter, image_edge_canny, 50, 150);		//Canny 边缘提
取，阈值可调
	//imshow("Canny Edges", image_edge);
	imwrite("G:/cpp/Instrument_homework/ball_edge_canny.png",
image_edge_canny);
	//Sobel 边缘提取
	Mat grad_x, grad_y;

	Sobel(image_otsu_filter, grad_x, CV_8U, 1, 0, 3, 1, 0, BORDER_DEFAULT);
	//x 方向
	Sobel(image_otsu_filter, grad_y, CV_8U, 0, 1, 3, 1, 0, BORDER_DEFAULT);
	//y 方向
	addWeighted(grad_x, 0.5, grad_y, 0.5, 0, image_edge_sobel);			//
合并
	imwrite("G:/cpp/Instrument_homework/ball__edge_sobel.png",
image_edge_canny);

	Point cc = centre(image, image_otsu);
	cout << "\n 圆心:\t(" << cc.x << ", " << cc.y << ")" << endl;

	//waitKey(0);
	return 0;
}
```

Appendix B MySQL

```
DROP TABLE BallMeasurements;
CREATE TABLE BallMeasurements (
    id INT AUTO_INCREMENT PRIMARY KEY,
    ball_id VARCHAR(20),             -- 钢球编号（如自动递增或扫描得到）
    measure_time DATETIME,           -- 测量时间
    diameter_1 FLOAT,                -- 第 1 个方向直径
    diameter_2 FLOAT,                -- 第 2 个方向直径
    diameter_3 FLOAT,                -- 第 3 个方向直径
    operator VARCHAR(50)             -- 操作员信息
);
INSERT INTO BallMeasurements (ball_id, measure_time, diameter_1, diameter_2,
diameter_3, operator)
VALUES
('Ball-001', '2025-05-01 09:15:00', 30.012, 30.009, 30.014, 'Operator-A'),
('Ball-002', '2025-05-01 09:18:00', 35.001, 35.006, 35.004, 'Operator-A'),
('Ball-003', '2025-05-01 09:20:00', 39.998, 40.002, 40.001, 'Operator-B');

SELECT id, ball_id, measure_time, diameter_1, diameter_2, diameter_3, operator,
ROUND((diameter_1+diameter_2+diameter_3)/3, 3) AS avg_diameter
FROM BallMeasurements
```