

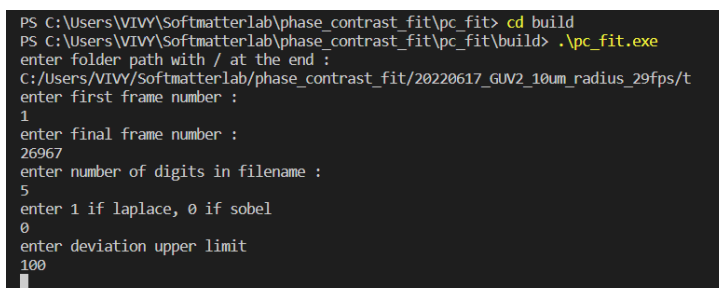
## Introduction

The following manual is an attempt at documenting the in and outs of the code written by Jai Samarth (<https://github.com/JaiSamarth>) during his internship at Soft Matter Biophysics Lab at IISER Mohali in the summer of 2022. This document details the instructions to use the code as well as the algorithmic approach used in it.

This program takes in phase contrast microscopy data in form of BMP images and estimates area of elliptical GUVs by finding their contour and fitting an ellipse over it.

## Part 1 : Using the program

1. Make sure OpenCV and Eigen C++ Libraries are installed on your system. Compile the main.cpp file using CMakeLists.txt and the program will be saved in the build folder by the name pc\_fit.exe
2. Open up a terminal in the same directory as the previously mentioned executable file and run the executable.
3. The program will print few text prompts to initialize few variables. We'll go over them here one by one.
  - Enter folder path with / at the end



```
PS C:\Users\VIVV\Softmatterlab\phase_contrast_fit\pc_fit> cd build
PS C:\Users\VIVV\Softmatterlab\phase_contrast_fit\pc_fit\build> .\pc_fit.exe
enter folder path with / at the end :
C:/Users/VIVV/Softmatterlab/phase_contrast_fit/20220617_GUV2_10um_radius_29fps/t
enter first frame number :
1
enter final frame number :
26967
enter number of digits in filename :
5
enter 1 if laplace, 0 if sobel
0
enter deviation upper limit
100
```

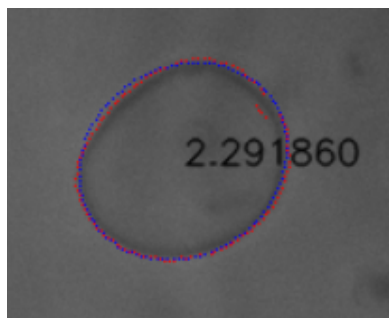
Figure 1: Running pc\_fit.exe

Here we are initializing the prefix of our image file paths before their frame numbers. Enter the folder path here with / at the end assuming the files don't have any additional prefix. If ZEN software is used to generate BMP files from CZI file, it adds a "t" in the beginning of the image names and that must be taken into account as shown in the image above.

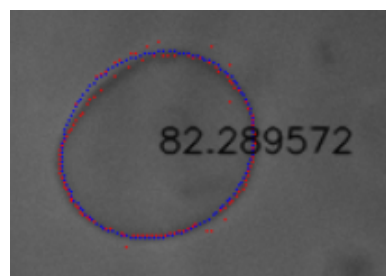
- Enter first frame number  
Here input the integer from where you want to start reading in the images. Usually values like 1 or 0 are used.
- Enter final frame number  
Here input the integer up until you want to read in the images. Usually the number of the final image is used.
- Enter number of digits in filename  
Here input the total number of digits used to specify index of an image. For example "00001.bmp" format uses 5 places to specify an image of index 1.

- Enter 1 if laplace, 0 if sobel

Here to facilitate edge detection for obtaining GUV contour, we can use two types of image derivatives. Laplace and Sobel derivatives. Depending on the image data one might work better than the other. Try both and go ahead with the one which gives better contours. Given below an example showing difference in detected contour (red pixels) between the two filters with deviation in the center.



(a) After Sobel derivative



(b) After Laplace derivative

- Enter deviation upper limit

Here specify the upper limit for the deviation of the detected contour from the fitted ellipse. If the deviation goes beyond this number, the program will skip some prespecified frames and open up a new window to take user input for re-adjusting the interest area.

4. After all the parameters are initialized, a window will pop up for the user to select the interest area. Click at the center of the GUV you want to analyze and adjust the sliders to shape the interest area. An example is shown below.

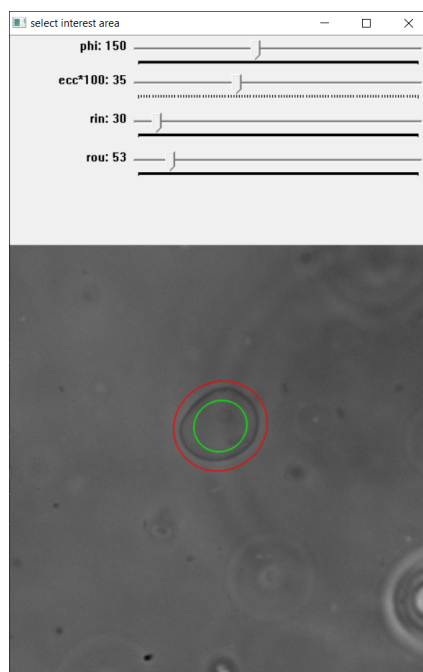


Figure 3: Select interest area

Meaning of sliders :

- phi  
Rotation
- $\text{ecc} * 100$   
Eccentricity times 100. Since floats are not supported in OpenCV sliders.
- rin  
Inner major axis
- rou  
Outer major axis

5. After Step 4, close the window by pressing the escape key. A new window will pop up and will show the contour detected along with the fitted ellipse in two distinct colors for each frame as the program goes through all the images. If the images get distorted, deviation will spike and the program will ask for user input again to re-adjust the interest area., in this case repeat Step 3. After the program goes through all the images it will close the window and print out "program finished" in the terminal. Ellipse fitting data can be accessed from a CSV file named "pc\_fit.csv" in the build folder.

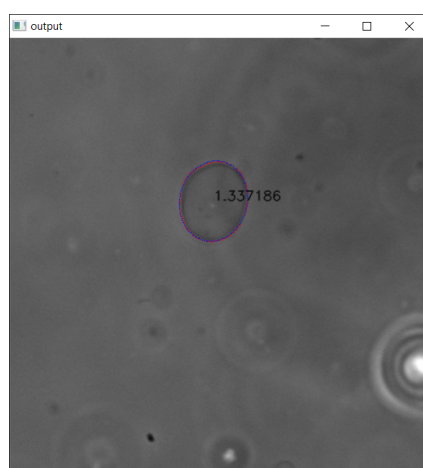


Figure 4: Output window

Detected contour in red, fitted ellipse in blue and deviation at the center

6. If one goes through the code, there are code snippets one can comment and uncomment to modify the output in the previous step. One can view fitted ellipse in form of a continuous curve rather than discrete points or view the interest area as it changes to track the GUV movement.

## Part 2 : Deep dive into workings of the program

Here we go over the algorithmic approach of the program. Syntax for OpenCV is not explained, code comments can be read for explanation and the documentation does a better job of explaining it.

## Image filters

We first use image histogram equalization followed by a CLAHE filter to enhance the image contrast. Next we apply a blur filter to reduce noise and then a gaussian blur filter to smooth out image intensity distribution. Lastly we apply sobel filters along the x and y direction and take their equally weighted sum to generate our final pre-processed image.

Taking sobel image derivatives generates a peak at the GUV boundary as in the phase contrast microscopy technique, GUV boundary is darker than the rest of the system. Sometimes depending on the image data captured, laplace derivative peaks coincide better with the GUV membrane compared to sobel, hence that has also been included in the code.

## Detecting contours

To detect the GUV contour, we need to radially scan the interest area in the image. For this we first generate a polar image of the interest area with rows containing pixel intensity values along a particular theta coordinate and columns representing different theta coordinates. After this we simply store the values of the radial global maxima for different thetas as the GUV contour.

## Calculating deviation

To reduce incorrect GUV contour detection, we need to quantify the quality of a detected contour. Here we chose to quantify it as the root-mean-square of the values of the radial distance between points on the contour and points on the fitted ellipse at different angular coordinates.

## Fitting ellipse on contours

- A C++ implementation of an algorithm proposed [1] and then improved in [2] is used to fit ellipses to contours. It is highly recommended reading them for the explanation and proof of the algorithm.

[1] A. Fitzgibbon, M. Pilu and R. B. Fisher, "Direct least square fitting of ellipses," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 21, no. 5, pp. 476-480, May 1999, doi: 10.1109/34.765658.

[2] <http://autotrace.sourceforge.net/WSCG98.pdf>

- Also credits to Nikolai Chernov for publishing a Matlab implementation of this algorithm upon which this C++ implementation is based.

(<https://www.mathworks.com/matlabcentral/fileexchange/22684-ellipse-fit-direct-method>)

— \* \* \* \* \*

## Appendix

### Code : CMake file

```
cmake_minimum_required(VERSION 3.0.0)
project(pc_fit VERSION 0.1.0)

include(CTest)
enable_testing()

find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
# eigen library path
include_directories( "C:/Users/VIVY/Downloads/eigen-3.4.0/eigen-3.4.0" )
add_compile_options("-Wa,-mbig-obj")
# remove above two lines on linux systems
add_executable(pc_fit main.cpp)
target_link_libraries( ${PROJECT_NAME} ${OpenCV_LIBS} )
set(CPACK_PROJECT_NAME ${PROJECT_NAME})
set(CPACK_PROJECT_VERSION ${PROJECT_VERSION})
include(CPack)
```

### Code : main.cpp

```
/*
This code
takes in .bmp image data
from phase contrast microscopy

identifies contours of elliptical guvs

fits an ellipse over contour

saves ellipse data for each frame in a
csv file

-jsama
*/

#include <iostream>
#include <fstream>
#include <cmath>
#include <vector>
#include "opencv2/opencv.hpp"
#include "opencv2/highgui.hpp"
#include "opencv2/imgproc.hpp"
#include "Eigen/Dense"
#include "Eigen/Eigenvalues"

using namespace cv;
using namespace std;

// constant pi
const double pi = M_PI;
// gui input
double gui_in[6] = {1.0}; // x0, y0, rin, rou, ecc, phi

void detect_click(int event, int x, int y, int flags, void *data)
{
    // copy image
    string *image_path = (string *)data;
    Mat imc = imread(*image_path);
    // left click
    if (event == EVENT_LBUTTONDOWN)
    {
        // get click coords
        gui_in[0] = x;
        gui_in[1] = y;
        // draw ellipse
        ellipse(imc,
```

```

        Point(gui_in[0], gui_in[1]),
        Size(gui_in[2], gui_in[2] * sqrt(1.0 - pow(gui_in[4],2))),
        gui_in[5],
        0,
        360,
        CV_RGB(0,255,0),
        1,
        LINE_AA);
    ellipse(imc,
        Point(gui_in[0], gui_in[1]),
        Size(gui_in[3], gui_in[3] * sqrt(1.0 - pow(gui_in[4],2))),
        gui_in[5],
        0,
        360,
        CV_RGB(255,0,0),
        1,
        LINE_AA);
    // update window
    imshow("select interest area", imc);
}
}

void slider(int val, void *data)
{
    // copy image
    string *image_path = (string *)data;
    Mat imc = imread(*image_path);
    // get trackbar data
    gui_in[2] = getTrackbarPos("rin", "select interest area");
    gui_in[3] = getTrackbarPos("rou", "select interest area");
    gui_in[4] = getTrackbarPos("ecc*100", "select interest area") * 0.01;
    gui_in[5] = getTrackbarPos("phi", "select interest area");
    // draw ellipse
    ellipse(imc,
        Point(gui_in[0], gui_in[1]),
        Size(gui_in[2], gui_in[2] * sqrt(1.0 - pow(gui_in[4],2))),
        gui_in[5],
        0,
        360,
        CV_RGB(0,255,0),
        1,
        LINE_AA);
    ellipse(imc,
        Point(gui_in[0], gui_in[1]),
        Size(gui_in[3], gui_in[3] * sqrt(1.0 - pow(gui_in[4],2))),
        gui_in[5],
        0,
        360,
        CV_RGB(255,0,0),
        1,
        LINE_AA);
    // update window
    imshow("select interest area", imc);
}

void interest_area(Mat img, string path)
{
    // named window
    namedWindow("select interest area");
    // set callback function
    setMouseCallback("select interest area", detect_click, &path);
    // view image
    imshow("select interest area", img);
    // create trackbar
    createTrackbar("phi", "select interest area", 0, 360, slider, &path);
    createTrackbar("ecc*100", "select interest area", 0, 100, slider, &path);
    createTrackbar("rin", "select interest area", 0, min(img.rows, img.cols), slider, &path);
    createTrackbar("rou", "select interest area", 0, min(img.rows, img.cols), slider, &path);
    // close window
    waitKey(0);
    destroyWindow("select interest area");
}
}

```

```

void apply_filter(Mat *img, int l)
{
    // copy image
    Mat imf = *img;
    // apply clahe
    equalizeHist(imf, imf);
    Ptr<CLAHE> clahe = createCLAHE();
    clahe->setClipLimit(10);
    clahe->apply(imf, imf);
    // apply blur
    blur(imf, imf, Size(5,5), Point(-1, -1));
    GaussianBlur(imf, imf, Size(5,5), 0, 0);
    if (l == 1)
    {
        // laplace filter
        Mat lap;
        Laplacian(imf, lap, CV_16S, 3, 2, 0, BORDER_DEFAULT);
        convertScaleAbs(lap, imf);
    }
    else
    {
        // sobel filter
        Mat grad_x, grad_y;
        Mat abs_grad_x, abs_grad_y;
        Sobel(imf, grad_x, CV_16S, 1, 0, 3, 1, 0, BORDER_DEFAULT);
        Sobel(imf, grad_y, CV_16S, 0, 1, 3, 1, 0, BORDER_DEFAULT);
        // converting back to CV_8U
        convertScaleAbs(grad_x, abs_grad_x);
        convertScaleAbs(grad_y, abs_grad_y);
        addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, imf);
    }
}

vector<double> linspacef(double a, double b, size_t n)
{
    double spacing = (b-a)/ (double)(n-1);
    vector<double> linspace(n);
    vector<double>::iterator itr;
    double val;
    for (itr = linspace.begin(), val = a; itr != linspace.end(); itr++, val += spacing)
    {
        *itr = val;
    }
    return linspace;
}

void detect_contour(Mat *img, vector<int> &contour_x, vector<int> &contour_y)
{
    int n = 1.5 * (gui_in[2] + gui_in[3]); // set angular resolution
    int m = 2 * (gui_in[3] - gui_in[2]); // set radial resolution
    Mat img_array = *img;
    vector<double> theta = linspacef(0, 2*pi, n);
    vector<double> radius = linspacef(gui_in[2], gui_in[3], m);
    int polar_img[n][m]; // polar image
    // scan interest area
    for (int i = 0; i < n; i++)
    {
        uchar max = 0;
        contour_x.push_back(0);
        contour_y.push_back(0);
        for (int j = 0; j < m; j++)
        {
            // ellipse points
            double ex = radius[j] * cos(theta[i]);
            double ey = ((radius[j] * sqrt(1.0 - pow(gui_in[4],2))) * sin(theta[i]));
            // rotate coordinates
            int x = (int)round(
                ex * cos(gui_in[5] * pi/180.0) - ey * sin(gui_in[5] * pi/180.0) + gui_in[0]
            );
            int y = (int)round(
                ex * sin(gui_in[5] * pi/180.0) + ey * cos(gui_in[5] * pi/180.0) + gui_in[1]
            );
            // max peak

```

```

        if (img_array.at<uchar>(y,x) >= max)
        {
            max = img_array.at<uchar>(y,x);
            contour_x.back() = x;
            contour_y.back() = y;
        }
    }
}

template <typename t>
t vmean(vector<t> data)
{
    t sum = 0;
    for (size_t i = 0; i < data.size(); i++)
    {
        sum += data[i];
    }
    return (t)(sum/data.size());
}

void direct_ellipse(double ellipse_in[5], vector<int> &contour_x, vector<int> &contour_y)
{
    // centroid of data points
    double xm, ym;
    xm = vmean(contour_x);
    ym = vmean(contour_y);
    // reduced coordinates
    Eigen::VectorXd x_cord(contour_x.size());
    Eigen::VectorXd y_cord(contour_x.size());
    for (size_t i = 0; i < contour_x.size(); i++)
    {
        x_cord(i) = contour_x[i] - xm;
        y_cord(i) = contour_y[i] - ym;
    }
    // design matrix
    Eigen::MatrixXd D1(contour_x.size(), 3);
    Eigen::MatrixXd D2(contour_x.size(), 3);
    D1.col(0) = x_cord.array().pow(2);
    D1.col(1) = x_cord.array() * y_cord.array();
    D1.col(2) = y_cord.array().pow(2);
    D2.col(0) = x_cord;
    D2.col(1) = y_cord;
    D2.col(2) = Eigen::VectorXd::Ones(contour_x.size());
    // scatter matrix
    Eigen::MatrixXd S1(3,3);
    Eigen::MatrixXd S2(3,3);
    Eigen::MatrixXd S3(3,3);
    S1 = D1.transpose() * D1;
    S2 = D1.transpose() * D2;
    S3 = D2.transpose() * D2;
    // constraint
    Eigen::MatrixXd C1(3,3);
    C1 << 0.0,0.0,2.0,0.0,-1.0,0.0,2.0,0.0,0.0;
    // reduced scatter matrix
    Eigen::MatrixXd T = - S3.inverse()*S2.transpose();
    Eigen::MatrixXd M(3,3);
    M = C1.inverse() * (S1 + S2*T);
    // solve eigen values/vectors
    Eigen::EigenSolver<Eigen::MatrixXd> es(M);
    Eigen::MatrixXd eigvec = es.eigenvectors().real();
    // condition
    Eigen::VectorXd cond = 4.0*(eigvec.row(0).array()*eigvec.row(2).array()) - eigvec.row(1).array().pow(2);
    // min pos eigval
    Eigen::VectorXd a1;
    for (int i = 0; i < 3; i++)
    {
        if (cond(i)>0)
        {
            a1 = eigvec.col(i);
            break;
        }
    }
}

```



```

    }
    // a2
    Eigen::VectorXd a2 = T * a1;
    // a
    Eigen::VectorXd A(6);
    A << a1, a2;
    // absolute a
    A(5) = A(5) + A(0) * xm * xm + A(1) * xm * ym + A(2) * ym * ym - A(3) * xm - A(4) * ym;
    A(3) = A(3) - 2.0 * xm * A(0) - ym * A(1);
    A(4) = A(4) - 2.0 * ym * A(2) - xm * A(1);
    A = A/A.norm();
    // algebraic to polar
    double a = A(0);
    double b = A(1)/2.0;
    double c = A(2);
    double d = A(3)/2.0;
    double f = A(4)/2.0;
    double g = A(5);
    double det = b*b - a*c;
    ellipse_in[0] = (c*d - b*f)/det;
    ellipse_in[1] = (a*f - b*d)/det;
    double num = 2.0*(a*f*f+c*d*d+g*b*b-2.0*b*d*f-a*c*g);
    double fac = sqrt(pow((a - c), 2) + 4*b*b);
    double ap = sqrt(num / det / (fac - a - c));
    double bp = sqrt(num / det / (-fac - a - c));
    bool width_gt_height = true;
    if (ap<bp)
    {
        width_gt_height = false;
        swap(ap, bp);
    }
    double phi = 0.5 * atan((2.*b) / (a - c));
    if(a>c)
    {
        phi += pi/2;
    }
    if(width_gt_height == false)
    {
        phi += pi/2;
        phi = fmod(phi,pi);
    }
    ellipse_in[2] = ap;
    ellipse_in[3] = bp;
    ellipse_in[4] = phi * 180.0/pi;
}

string padding(int s, int digits)
{
    string str = to_string(s);
    int pad = digits - (int)str.size();
    string name = string(pad, '0').append(str);
    return name;
}

double deviation(double ellipse_in[5], vector<int> &contour_x, vector<int> &contour_y)
{
    // store deviation
    double dev = 0.0;
    // read in ellipse fit
    double xc = ellipse_in[0];
    double yc = ellipse_in[1];
    double ap = ellipse_in[2];
    double bp = ellipse_in[3];
    double ph = ellipse_in[4];
    // theta resolution
    int n = contour_x.size();
    vector<double> theta = linspacef(0, 2*pi, n);
    for (size_t i = 0; i < n; i++)
    {
        // contour points
        double x = contour_x[i];
        double y = contour_y[i];
        // ellipse points

```

```

        double ex = ap * cos(theta[i]);
        double ey = bp * sin(theta[i]);
        // rotate and transform
        double exr = ex*cos(ph*pi/180.0) - ey*sin(ph*pi/180.0) + xc;
        double eyr = ex*sin(ph*pi/180.0) + ey*cos(ph*pi/180.0) + yc;
        // square difference
        dev += pow(x - exr, 2) + pow(y - eyr, 2);
    }
    return sqrt(dev/(double) n);
}

int main(){
    // directory path
    string file_path;
    cout << "enter folder path with / at the end : " << endl;
    cin >> file_path;
    // first frame number
    int first_frame;
    cout << "enter first frame number : " << endl;
    cin >> first_frame;
    // last frame number
    int final_frame;
    cout << "enter final frame number : " << endl;
    cin >> final_frame;
    // number of digits
    int digits;
    cout << "enter number of digits in filename : " << endl;
    cin >> digits;
    // laplace or sobel
    int l;
    cout << "enter 1 if laplace, 0 if sobel" << endl;
    cin >> l;
    // deviation upper limit
    double d;
    cout << "enter deviation upper limit" << endl;
    cin >> d;
    string image_path = file_path + padding(first_frame, digits) + ".bmp";
    // read image
    Mat img = imread(image_path, IMREAD_GRAYSCALE);
    // if fail
    if (img.empty())
    {
        cout << "error in opening image" << endl;
        return 1;
    }
    interest_area(img, image_path);
    bool recenter = false;
    namedWindow("output");
    // store data
    ofstream saved_data;
    saved_data.open("pc_fit.csv");
    saved_data << "t,x0,y0,a,b,phi\n";
    for (size_t i = first_frame; i < final_frame; i++)
    {
        // open image
        image_path = file_path + padding(i, digits) + ".bmp";
        Mat img = imread(image_path, IMREAD_GRAYSCALE);
        if (img.empty())
        {
            cout << "error in opening image : " << i << endl;
        }
        else
        {
            // if previous was rejected
            if (recenter == true)
            {
                interest_area(img, image_path);
                recenter = false;
            }
            // apply filters
            apply_filter(&img, l);

            // detect contour

```

```

vector<int> contour_x;
vector<int> contour_y;
detect_contour(&img, contour_x, contour_y);

// fit ellipse
double ellipse_in[5]; // x0, y0, a, b, phi
direct_ellipse(ellipse_in, contour_x, contour_y);

// view
Mat imc = imread(image_path, IMREAD_COLOR);

// contour
// view detected contour
for (size_t i = 0; i < contour_x.size(); i++)
{
    imc.at<Vec3b>(contour_y[i], contour_x[i])[0] = 0;
    imc.at<Vec3b>(contour_y[i], contour_x[i])[1] = 0;
    imc.at<Vec3b>(contour_y[i], contour_x[i])[2] = 255;
}

// ellipse fit in discrete form
// view fitted ellipse in discrete form
double xc = ellipse_in[0];
double yc = ellipse_in[1];
double ap = ellipse_in[2];
double bp = ellipse_in[3];
double ph = ellipse_in[4];
int n = contour_x.size();
vector<double> theta = linspacef(0, 2*pi, n);
for (size_t i = 0; i < n; i++)
{
    double x = contour_x[i];
    double y = contour_y[i];
    double ex = ap * cos(theta[i]);
    double ey = bp * sin(theta[i]);
    double exr = ex*cos(ph*pi/180.0) - ey*sin(ph*pi/180.0) + xc;
    double eyr = ex*sin(ph*pi/180.0) + ey*cos(ph*pi/180.0) + yc;
    imc.at<Vec3b>(eyr, exr)[0] = 255;
    imc.at<Vec3b>(eyr, exr)[1] = 0;
    imc.at<Vec3b>(eyr, exr)[2] = 0;
}

// ellipse fit
// view fitted ellipse in curve form
// ellipse(imc,
// Point(ellipse_in[0],
// ellipse_in[1]),
// Size(ellipse_in[2],
// ellipse_in[3]),
// ellipse_in[4],
// 0,
// 360,
// Scalar(0,255,0),
// 1,
// LINE_AA);

// interest area
// visualize interest area for debugging
// ellipse(imc,
// Point(gui_in[0], gui_in[1]),
// Size(gui_in[2], gui_in[2] * sqrt(1.0 - pow(gui_in[4],2))),
// gui_in[5],
// 0,
// 360,
// CV_RGB(0,255,0),
// 1,
// LINE_AA);
// ellipse(imc,
// Point(gui_in[0], gui_in[1]),
// Size(gui_in[3], gui_in[3] * sqrt(1.0 - pow(gui_in[4],2))),
// gui_in[5],
// 0,
// 360,

```

```
// CV_RGB(255,0,0),
// 1,
// LINE_AA);

// deviation
// output contour deviation as
//text on image for debugging
putText(imc,
to_string(deviation(ellipse_in, contour_x, contour_y)),
Point(gui_in[0],
gui_in[1]),
FONT_HERSHEY_SIMPLEX,
0.5, Scalar(1,1,1), 1, LINE_AA);

imshow("output", imc);
waitKey(1);

// rejection criteria
// tweak for each dataset
if (deviation(ellipse_in, contour_x, contour_y) > d)
{
    recenter = true;
    // skip 50 frames
    i += 49;
}
else
{
    // update center and phi
    gui_in[0] = ellipse_in[0];
    gui_in[1] = ellipse_in[1];
    gui_in[5] = ellipse_in[4];
    saved_data
    << i << ", "
    << ellipse_in[0] << ", "
    << ellipse_in[1] << ", "
    << ellipse_in[2] << ", "
    << ellipse_in[3] << ", "
    << ellipse_in[4] << "\n";
}
}
}
// waitkey(0);
destroyWindow("output");
cout << "program finished" << endl;
return 0;
}
```