REPORT: PROJECT 3B – IMAGE STABILIZATION GIMBLE

Done by –

- Rohit Parvatikar(200266206) – rrparvat@ncsu.edu
- Emil Peter(200269126) – epeter@ncsu.edu

CONTENTS

Introduction 3

Flow Diagram 3

System Design 4

Capture the Image 4

BLUR the Image 5

Conversion to HSV format 6

Thresholding of Image 7

Finding the Center of each Reference Objects 8

Finding the Center of Screen 10

Pan, Tilt and Roll Error Calculation 10

Generation of Output Image 11

Timing Information 12

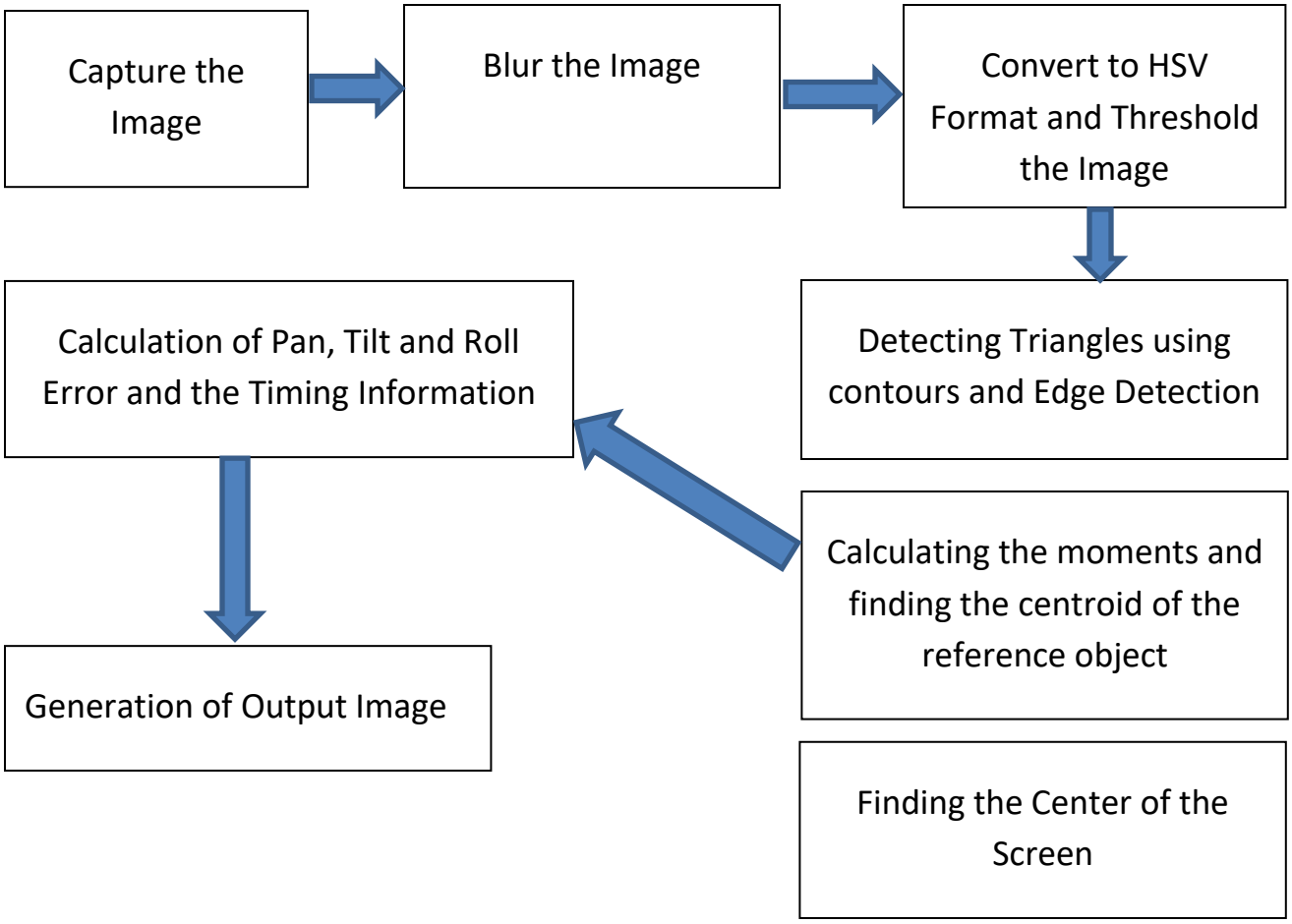
Conclusion 13

INTRODUCTION

In this project, we use C920 Webcam and openCV on Beagle Bone Black to identify two reference triangles in an image and perform the calculation for pan, tilt and roll error. The concept involves identifying the triangles in the image and finding their center respectively. Then, we find the center of the screen and the midpoint of the line between the two triangles and calculate the Pan, Tilt and Roll error based on the values of x and y coordinates of the two points(Centre of the screen and midpoint of the line).

Also, the timing information is calculated based on the time from image acquisition to the error message output.

FLOW DIAGRAM



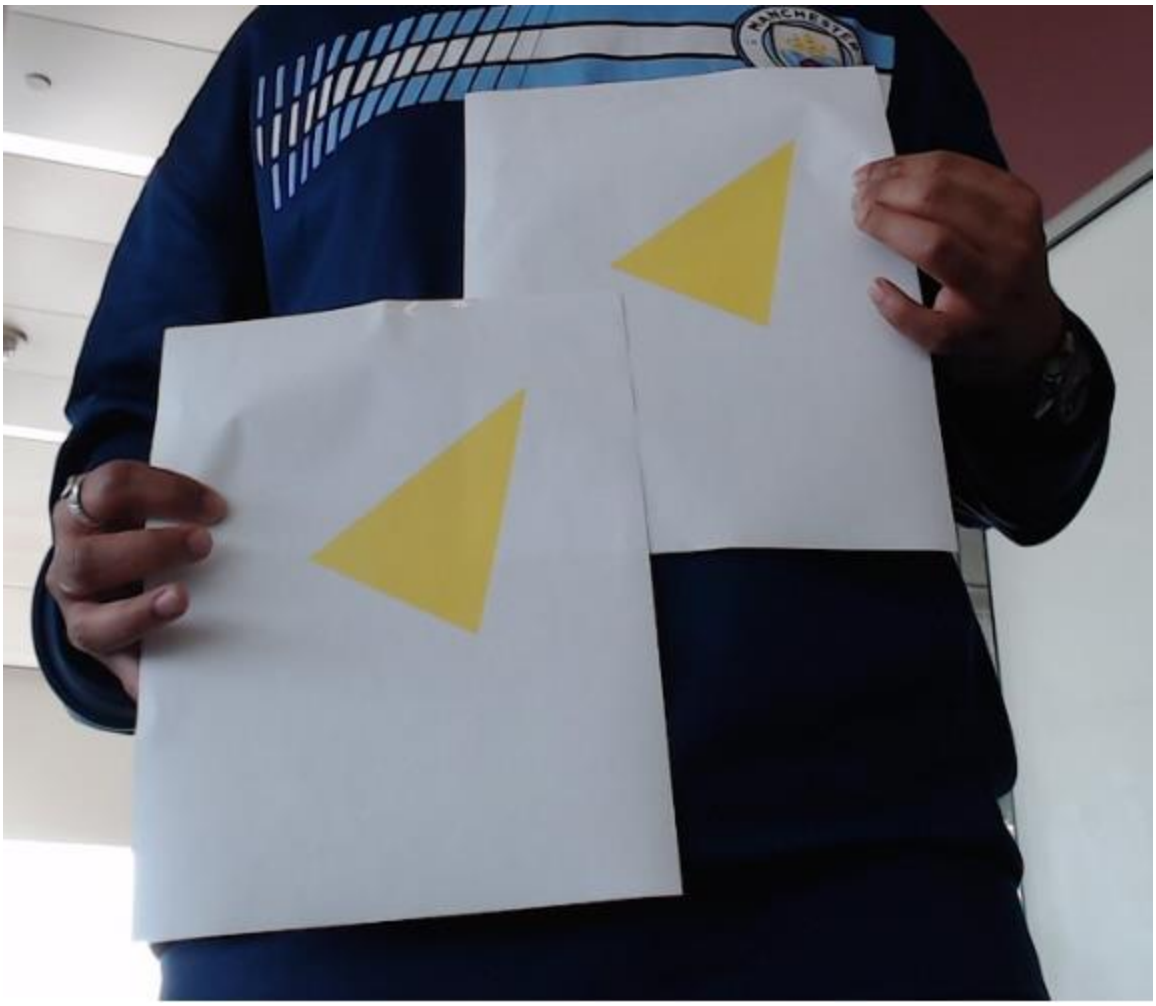
SYSTEM DESIGN

CAPTURE THE IMAGE

We capture the image by using capture() function from VideoCapture class. We set the width and height of the image to 1280 x 720 pixels and then capture the frame using camera(Logitech C920).

The Following code implements the image capture procedure.

```
#if CAPTURE
VideoCapture capture(0);   // capturing from /dev/video0

cout << "Started Processing - Capturing Image" << endl;
// set any  properties in the VideoCapture object
capture.set(CV_CAP_PROP_FRAME_WIDTH,WIDTH);    // width pixels
capture.set(CV_CAP_PROP_FRAME_HEIGHT,HEIGHT);   // height pixels
capture.set(CV_CAP_PROP_GAIN, 0);             // Enable auto gain etc.
if(!capture.isOpened()){   // connect to the camera
   cout << "Failed to connect to the camera." << endl;
}

clock_gettime(CLOCK_THREAD_CPUTIME_ID, &start);

capture >> src;          // capture the image to the frame
if(src.empty()){         // did the capture succeed?
   cout << "Failed to capture an image" << endl;
   return -1;
}
```



BLUR THE IMAGE

After the Image is captured , we perform Blur operations on the image to smoothen it.

For Blur Operation, we use the GaussianBlur() Function. It performs convolution on the input pixels and sums them to give the output values.

The Following code implements the Blurring the Image.

```
// Blurring the image using gaussian blur
GaussianBlur( src, frame, Size( 9, 9 ), 0, 0 );
imwrite( "../RohitProject3/Blur_image.jpg", frame );                    //saving blur image
```

source: https://docs.opencv.org/3.4/dc/dd3/tutorial_gausian_median_blur_bilateral_filter.html

CONVERSION TO HSV FORMAT

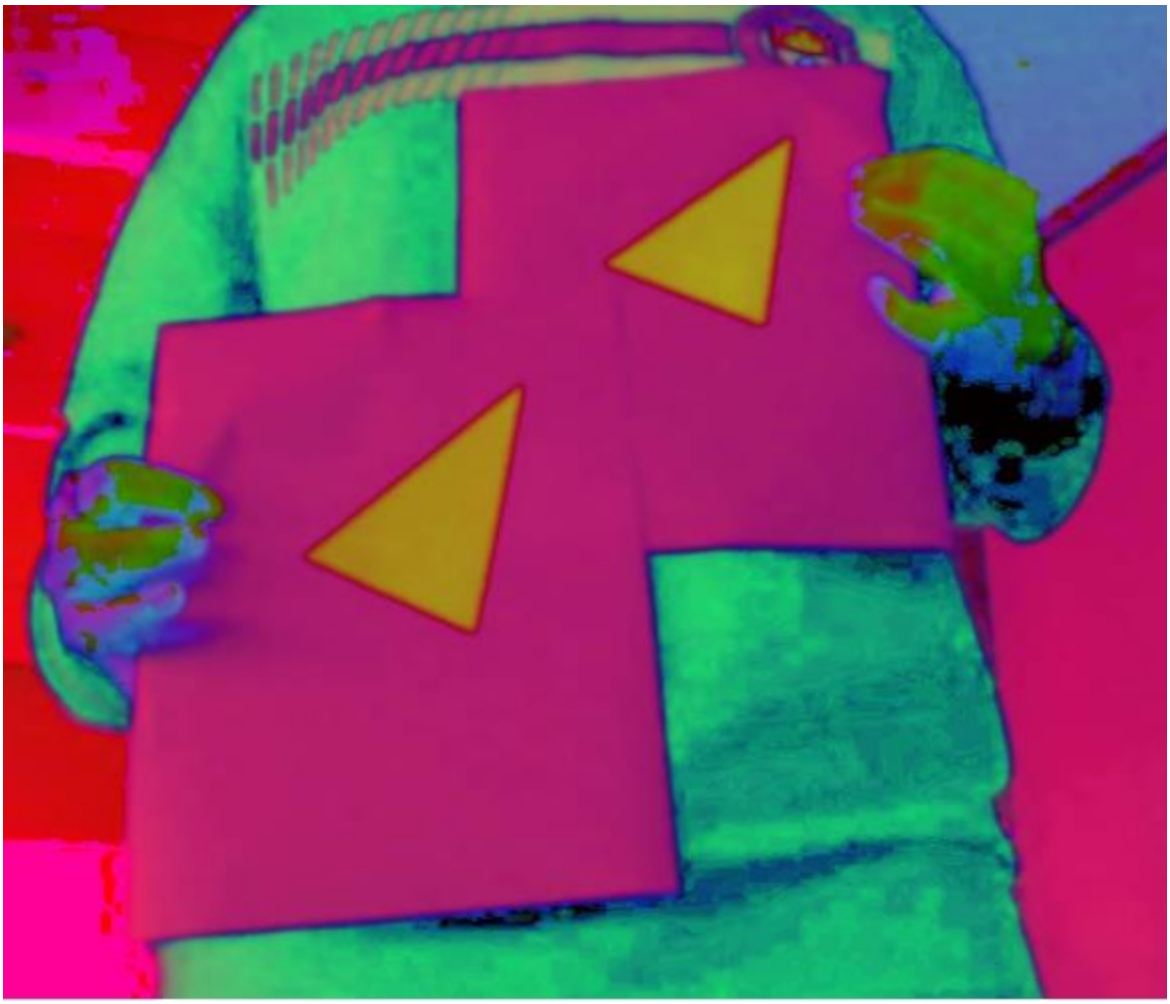
The next step in the image processing is to perform the conversion of the Blurred Image into HSV format. This step is done to find the minimum and maximum values in order to avoid erroneous values after thresholding.

For this we use inRange() Function.

The following code implements the conversion to HSV format.

```
// Detect the object based on HSV Range Values
inRange(frame_HSV, Scalar(20, 100, 100), Scalar(50, 255, 255), frame_threshold);
imwrite( "../RohitProject3/Threshold_image.jpg", frame_threshold );              //saving image after applying threshold
```

source: https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html



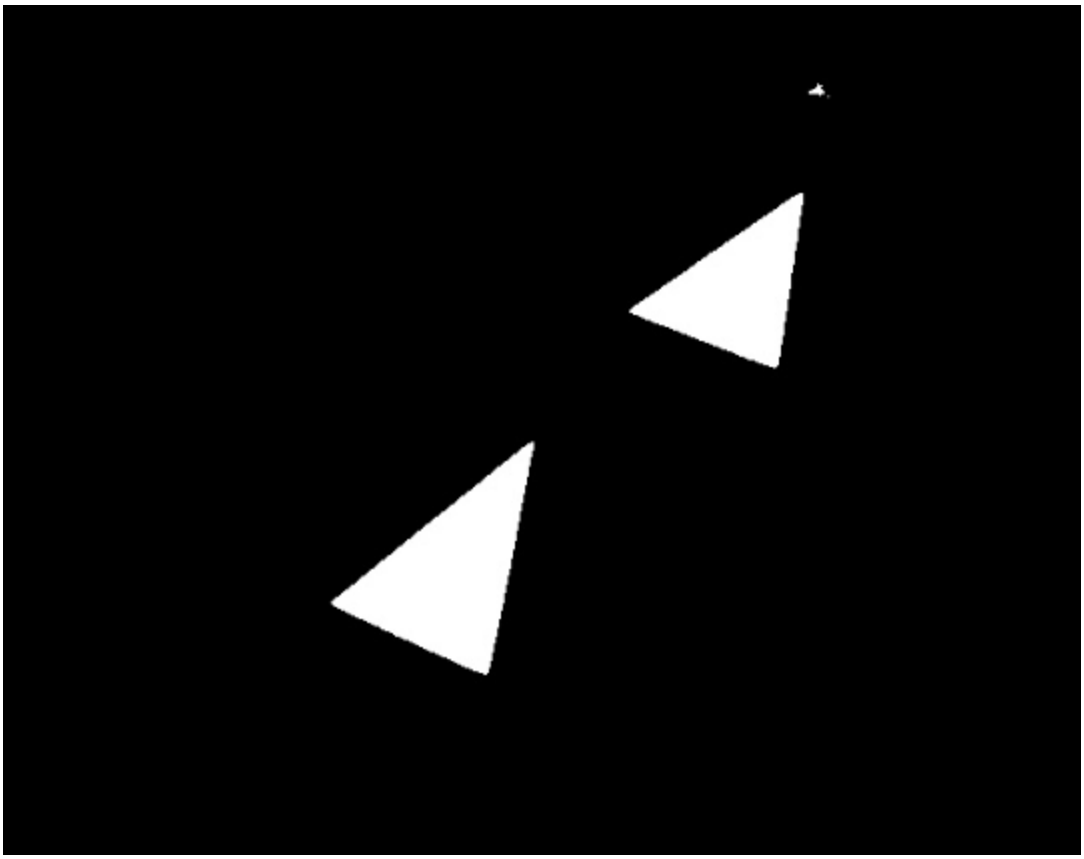
THRESHOLDING OF IMAGE

Then, we perform Thresholding on the resultant HSV image using Threshold Binary type. For this operation, we use threshold() Function.

The Following code implements the thresholding the Image.

```
// Detect the object based on HSV Range Values
inRange(frame_HSV, Scalar(20, 100, 100), Scalar(50, 255, 255), frame_threshold);
imwrite( "../RohitProject3/Threshold_image.jpg", frame_threshold );              //saving image after applying threshold
```

source: https://docs.opencv.org/3.4/db/d8e/tutorial_threshold.html



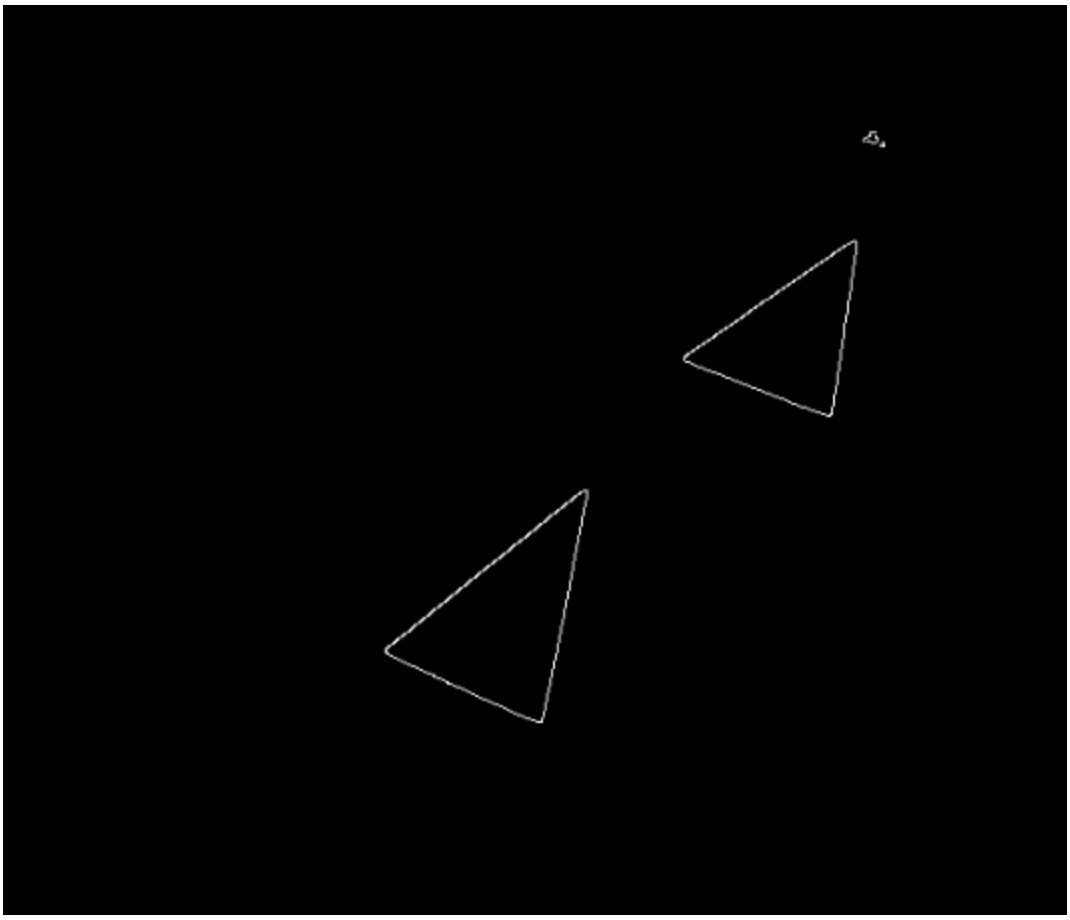
FINDING THE CENTER OF EACH REFERENCE OBJECTS

We find the center of the two reference triangles in the space. First, using canny edge detector we get the edges of the two Triangles.

The Following code implements the edge detection mechanism.

```
// Detect edges using canny
Canny( frame_threshold, canny_output, 50, 150, 3 );
imwrite( "../RohitProject3/Edges_image.jpg", canny_output );                    //saving edges of image
```

source: https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html



Then we find the number of contours using findContours() function and get the centroid using Point2f vector. We detect the triangles by approximating the polygons obtained from the contours. The next step is to draw the triangles , encircle them and draw circle around the center of each triangle.

The Following code implements the Functionality.

```
// Get the centroid of figures.
vector<Point2f> mc(contours.size());
for( int i = 0; i<contours.size(); i++)
{ mc[i] = Point2f( mu[i].m10/mu[i].m00 , mu[i].m01/mu[i].m00 ); }
```

source: https://www.learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/

```
// Draw contours
Mat drawing(canny_output.size(), CV_8UC3, Scalar(255,255,255));
vector<Point> approx_triangle;
for( int i = 0; i<contours.size(); i++ )
{
    Scalar color = Scalar(167,151,0); // B G R values
    approxPolyDP(contours[i], approx_triangle, 0.05*arcLength(contours[i],true), true);
    if (approx_triangle.size()==3)
    {
        drawContours(drawing, contours, i, color, 2, 8, hierarchy, 0, Point());
        minEnclosingCircle(contours[i], center_circle, radius);
        circle( drawing, center_circle, radius, color, 1, 8, 0 );
        if (count==0)
            center1=mc[i];
        else if (count==1)
            center2=mc[i];
        count++;
    }
}
```

FINDING THE CENTER OF SCREEN

In this step we find the Center of the Image(frame) by calculating the Width and Height of the Image and encircle the center.

```
Point center(WIDTH/2,HEIGHT/2);                                    //finding center of image
circle(drawing, center, 5, Scalar(128,0,0), 1);
```

PAN, TILT AND ROLL ERROR CALCULATION

Consider the center of the reference triangles as C1(a1,b1) and C2(a2,b2). Once we get the midpoint of the line between the two reference triangles(x1, y1) and the center of the screen coordinates(x2, y2), we find the Pan, Tilt and Roll Error in the following way:

Pan Error = x1 – x2;

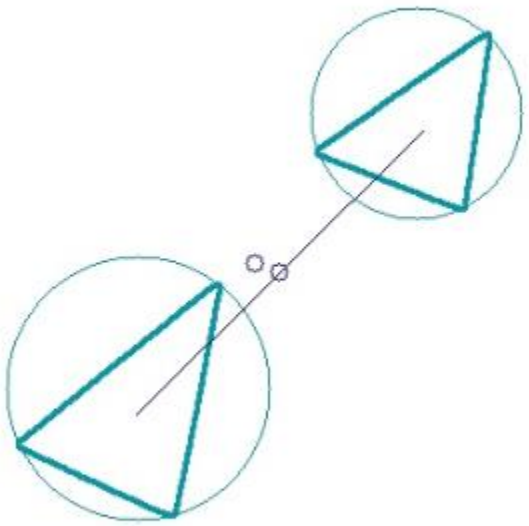
Tilt Error = y1 – y2;

Roll Error = (180/PI) * atanf(opposite_side/base); or (180/PI) * atanf(base/opposite_side); depending on the relationship of Pan and Tilt Error.

```
error.x=(midpoint.x-center.x);                                    //Calculating pan error
error.y=(midpoint.y-center.y);                                    //Calculating roll error

base=center1.x-center2.x;
opposite_side=center1.y-center2.y;

//Calculating roll error
if (abs(error.x)<abs(error.y))
    roll_error=atanf((float)opposite_side/base);
else
    roll_error=atanf((float)base/opposite_side);

clock_gettime(CLOCK_THREAD_CPUTIME_ID, &end);

//Calculating time taken
diff = 1000000000 * (end.tv_sec - start.tv_sec) +
    end.tv_nsec - start.tv_nsec;

printf("Pan error: %d Tilt error: %d Roll error: %lf\n",error.x,error.y,roll_error*(180/PI) );
printf("Time taken: %10.3f ms\n",(diff/1000000.0) );
```

GENERATION OF OUTPUT IMAGE

A line is drawn between the centers of the two triangles which is used as a reference for calculations. After all the calculations are done and the values are displayed on the PUTTY/TERA TERM terminal we use imwrite() function to write the final image to the output jpeg file.



TIMING INFORMATION

The timing information is calculated as the time from image acquisition to the output error values. We find it using clock_gettime() function.

The Following code implements the timing information.

```
clock_gettime(CLOCK_THREAD_CPUTIME_ID, &start);

capture >> src;          // capture the image to the frame
if(src.empty()){         // did the capture succeed?
    cout << "Failed to capture an image" << endl;
    return -1;
}
cout << "Processing - Performing Image Processing" << endl;
```

```
clock_gettime(CLOCK_THREAD_CPUTIME_ID, &end);

//Calculating time taken
diff = 1000000000 * (end.tv_sec - start.tv_sec) +
    end.tv_nsec - start.tv_nsec;

printf("Pan error: %d Tilt error: %d Roll error: %lf\n",error.x,error.y,roll_error*(180/PI) );
printf("Time taken: %10.3f ms\n",(diff/1000000.0) );
```

The Following screenshots show the timing information of the system in two scenarios –

- The file is being read
- The camera is capturing the image

```
debian@beaglebone:~/Desktop/RohitProject3$ ./test1
Reading Image from file
Pan error: 14 Tilt error: 5 Roll error: -45.349362
Time taken:    437.860 ms
debian@beaglebone:~/Desktop/RohitProject3$
```

```
debian@beaglebone:~/Desktop/RohitProject3$ ./test1
Started Processing - Capturing Image
Processing - Performing Image Processing
Pan error: 14 Tilt error: 5 Roll error: -45.349362
Time taken:    983.559 ms
```

CONCLUSION

OpenCV is a very powerful programming platform for Image Processing Applications and we learn how to interface openCV on Beagle Bone to exploit its computational throughput. We use various image processing techniques such as Blurring, Edge Detection, Thresholding and Polygon Approximation and Detection to perform the required operations to get the desired intermediate output. Also, we learnt how to find the Pan, Tilt and Roll Error for the image.