PROJECT REPORT

Computer Vision Operations using RaspberryPi 2

Guided By
Guojun Yang

Done By
Vignesh Vasudevan
A20387421

For
Prof. Jafar Saniie
ECE 597 – Special Problems
Department of Electrical and Computer Engineering
Illinois Institute of Technology

Table of Contents

1. Introduction

With the fast-paced development of technology in our current information era, computers are becoming more and more efficient at acquiring information from their environment, processing them and consequently, making decisions based on them. One such system, which uses external visual information and processes them to perform certain operations, or to make certain decisions, is done in the area of Digital Image/Video Processing. This area of research is popularly termed as Computer Vision, and it seeks to achieve a fully automated system for Artificial Intelligence which would be very much like the human visual system. [1] Formally, Computer Vision (CV) is a field of study encompassing various disciplines, aiming to give computers a high-level understanding of digital images and videos. Solving even small parts of certain Computer Vision challenges, creates exciting new possibilities in technology, engineering and even entertainment. In order to advance vision research and disseminate vision knowledge, it is highly critical to have a library of programming functions with the optimized and portable code, and hopefully available for free. This was an original goal of Intel team back in 1999 when OpenCV (Open Source Computer Vision Library) was officially launched. Since then, a number of programmers have contributed to the most recent library developments. The latest major change took place in 2009 (OpenCV 2) which includes main changes to the C++ interface. The newest library release can be found on the OpenCV official website. Nowadays the library has >2500 optimized algorithms. It is extensively used around the world, having >2.5M downloads and >40K people in the user group. OpenCV can be used in academic and commercial applications as well. [2]

The aim of this project was use the OpenCV libraries with C++ coding to perform some real-time image processing applications, and gradually proceed to design a system to segment and classify objects in plane in front of a Robotic Arm, in order to automatically let the Arm stow away the objects at particular places. All of the processing was done on a Raspberry Pi 2 with Raspbian Jessie as its OS.

2. Background

To make any kind of progress in this project, it was necessary to get familiar with the working environment and the required tools and equipment. This section briefly summarizes the said tools and the knowledge needed for the completion of the project. It also gives an overview of the work done previously in this field and details about the algorithms used.

2.1 Raspberry Pi 2

All of the work was done using the Raspberry Pi single board computer. Its compact size and surprisingly high computing power makes it one of the most popular main controller to use for any kind of project which involves an automated system. The system-on-chip (SoC) on board the Rpi is responsible for most of its processing power, the SoC being a Broadcom BCM2837. With a CPU clock speed of 900 MHz and an RAM of 1GB, the Rpi 2 is a sufficiently powerful device for the processes of Computer Vision.

The OS installed in the Rpi was the Raspbian Jessie with GUI. This recent model of the Rpi is able to efficiently process and display the GUI without glitches. The Raspbian is a specially designed computer OS to be compatible with the low-performance ARM CPUs on board the Rpis. The functioning of this OS is very similar to a Linux environment, so after the initial setup of the Rpi a complete update of the device and its software were done using UNIX commands.

While the Rpi comes pre-installed with capability to process and compile programs written in different programming languages, the libraries included for the different programs are quite limited. So, to prepare the device to be ready for the Image/Video processing applications of Computer Vision, the required libraries were downloaded and included in the main folder, to make it easily accessible by different programs. Once this was done, the Rpi was finally ready to be used in CV operations. [3]

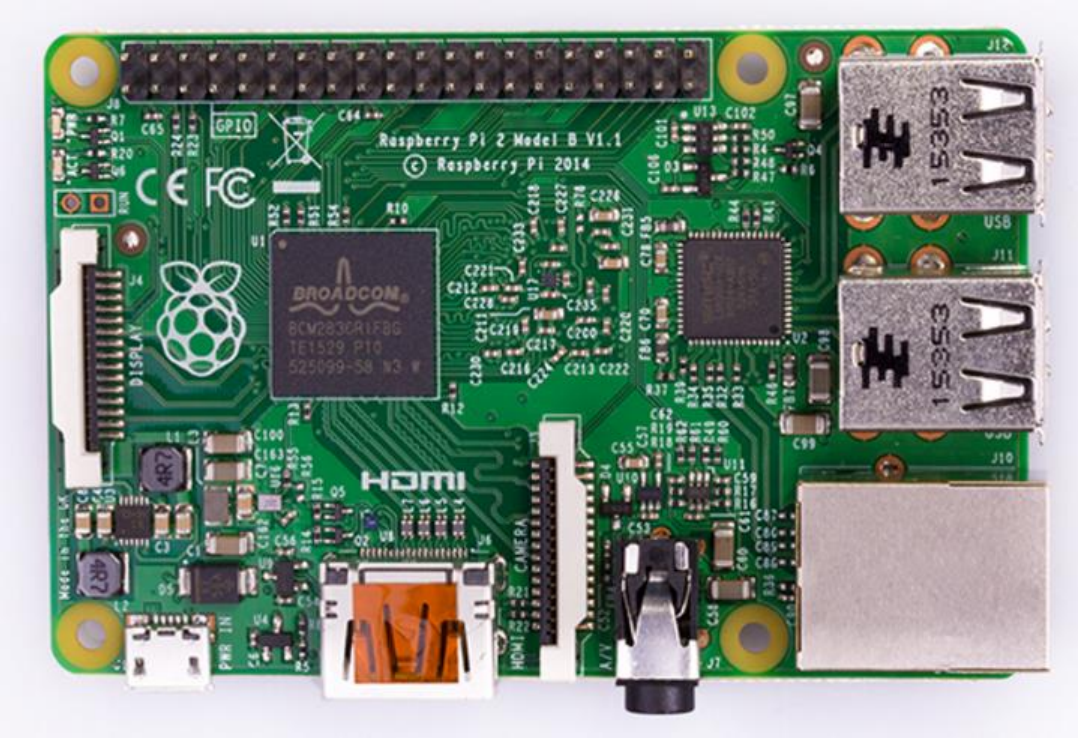


Fig 1. Raspberry Pi 2

2.2 C++ and OpenCV libraries

The OpenCV library is cornerstone of this project. As was previously mentioned, this library makes it possible for the person designing the CV system to achieve greater efficiency in their design by providing optimized CV and image processing operations as pre-defined functions and saves a lot of hassle by making the interface of the visual information and the central control device straightforward. The library was originally written in C and this C interface makes OpenCV portable to some specific platforms such as digital signal processors. Wrappers for languages such as C#, Python, Ruby and Java (using JavaCV) have been developed to encourage adoption by a wider audience. However, since version 2.0, OpenCV includes both its traditional C interface as well as a new C++ interface. This new interface seeks to reduce the number of lines of code necessary to code up vision functionality as well as reduce common programming errors such as memory leaks (through automatic data allocation and de-allocation) that can arise when using OpenCV in C. [4]

C++ is the preferred language used here for the codes. It has imperative, object-oriented and generic programming features, while also providing facilities for low-level memory manipulation. It was designed with a bias toward system programming and embedded, resource-constrained and large systems, with performance, efficiency and flexibility of use as its design highlights. It has a reputation for being a general-purpose programming language.

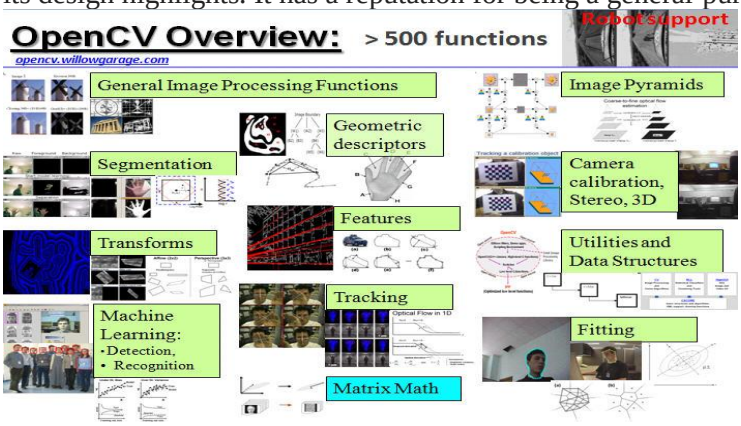


Fig 2. OpenCV functions

2.3 Basic Image Processing Methods

Here, some of the basic image processing methods used in the project is discussed:

- **Filtering:** This is the process by which the given image can be denoised or have just a certain frequency component of the image be extracted from it. Some of the processes used in this project are the Bilateral Filter and the Median filter. Image filtering is useful for many applications, including smoothing, sharpening, removing noise, and edge detection. A filter is defined by a kernel, which is a small array applied to each pixel and its neighbors within an image. In most applications, the center of the kernel is aligned with the current pixel, and is a square with an odd number (3, 5, 7, etc.) of elements in each dimension. The process used to apply filters to an image is known as convolution, and may be applied in either the spatial or frequency domain. [5]

- **Edge Detection:** This operation is the process of extracting edge components from the image. The Canny edge detection algorithm and edge detection using Sobel filters were implemented here.
  It includes a variety of mathematical methods that aim at identifying points in a digital image at which the image brightness changes sharply or, more formally, has discontinuities. The points at which image brightness changes sharply are typically organized into a set of curved line segments termed edges. [6]

- **Morphological Operations**: *Mathematical morphology* is a tool for extracting image components useful in the representation and description of region shape, such as boundaries, skeletons and convex hulls. The language of mathematical morphology is set theory, and as such it can apply directly to binary (two-level) images: a point is either in the set (a pixel is set, or put to foreground) or it isn't (a pixel is reset, or put to background), and the usual set operators (intersection, union, inclusion, complement) can be applied to them. [7]
  Basic operations in mathematical morphology operate on two sets: the first one is the *image*, and the second one is the *structuring element* (sometimes also called the *kernel*, although this terminology is generally reserved for convolutions). The structuring element used in practice is generally much smaller than the image, often a 3x3 matrix.

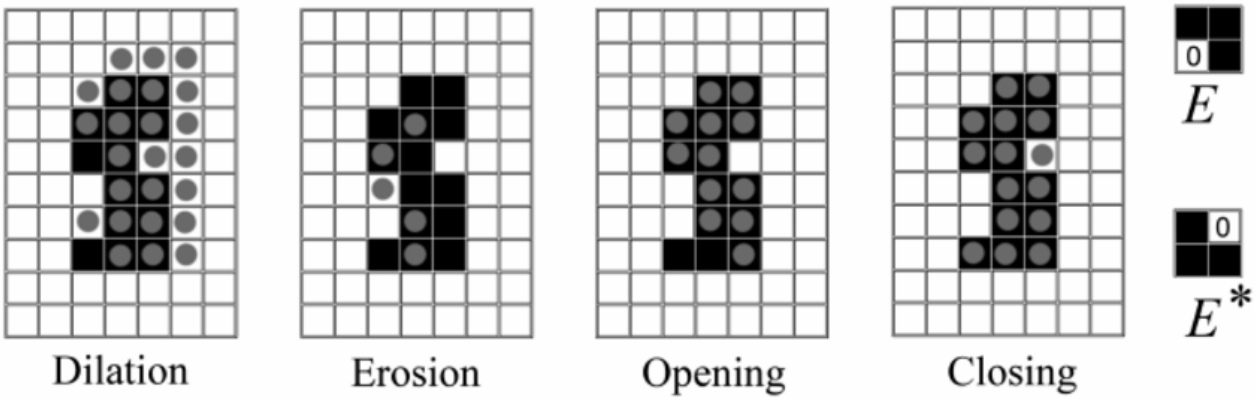


Fig 3. Image Morphological Operations

All the above operations were performed in real time.

2.4 Image Segmentation and Classification

In this project, three different types of segmentation algorithms were implemented. They were:

- **Watershed Segmentation:** In the study of image processing, a watershed is a transformation defined on a grayscale image. The name refers metaphorically to a geological watershed, or drainage divide, which separates adjacent drainage basins. The watershed transformation treats the image it operates upon like a topographic map, with the brightness of each point representing its height, and finds the lines that run along the tops of ridges. There are different technical definitions of a watershed. In graphs, watershed lines may be defined on the nodes, on the edges, or hybrid lines on both nodes and edges. Watersheds may also be defined in the continuous domain. There are also many different algorithms to compute watersheds. Watershed algorithm is used in image processing primarily for segmentation purpose. [8]
  This approach gives you oversegmented result due to noise or any other irregularities in the image. So OpenCV implemented a marker-based watershed algorithm where you specify which are all valley points are to be merged and which are not. It is an interactive image segmentation. What we do is to give different labels for our object we know.

Label the region which we are sure of being the foreground or object with one color (or intensity), label the region which we are sure of being background or non-object with another color and finally the region which we are not sure of anything, label it with 0. That is our marker. Then apply watershed algorithm. Then our marker will be updated with the labels we gave, and the boundaries of objects will have a value of -1. The details and the algorithm followed for its implementation will be discussed in the next section.

- **HSV Color-space based segmentation:** HSV is one of the most common cylindrical-coordinate representations of points in an RGB color model. The representation rearranges the geometry of RGB in an attempt to be more intuitive and perceptually relevant than the cartesian (cube) representation. Developed in the 1970s for computer graphics applications, HSV is used today in color pickers, in image editing software, and less commonly in image analysis and computer vision. The HSV model characterizes the colors as:

- The Hue (H) of a color refers to which pure color it resembles. Hues are described by a number which specifies the position of a pure color on the color wheel, as a fraction between 1 and 0.
- The Saturation (S) of a color describes how white a color is. A pure solid color is fully saturated at value 1, and all other shades are between 1 and 0.
- The Value (V) of a color, also called its lightness, defines how dark the color is. A value of 0 is black, with increasing V moves away from it.

The reason that HSV image segmentation is much easier is that it separates color information (chroma) from intensity or lighting (luma). Because value is separated, it is possible to construct a histogram or thresholding rules using only saturation and hue. This in theory will work regardless of lighting changes in the value channel. Even by singling out only the hue we can still have a very meaningful representation of the base color that will likely work much better than RGB. The end result is a more robust color thresholding over simpler parameters. [9]

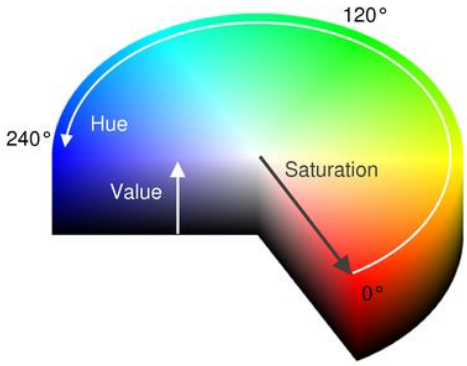


Fig 4. HSV Colorspace

- **Region Growing:** This is a simple region-based image segmentation method. It is also classified as a pixel-based image segmentation method since it involves the selection of initial seed points. This approach to segmentation examines neighboring pixels of initial seed points and determines whether the pixel neighbors should be added to the region. The process is iterated on, in the same manner as general data clustering algorithms. [10]

The main goal of segmentation is to partition an image into regions. Some segmentation methods such as thresholding achieve this goal by looking for the boundaries between regions based on discontinuities in grayscale or color properties. Region-based segmentation is a technique for determining the region directly. The basic formulation is:

$(a)\ \bigcup_{i=1}^{n} R_i = R.$

$(b)\ R_i$ is a connected region, i $= 1, 2, \ldots,$ n

$(c)\ R_i \bigcap R_j = \varnothing$ for all $i = 1, 2, \ldots, n.$

$(d)\ P(R_i) = TRUE$ for $i = 1, 2, \ldots, n.$

$(e)\ P(R_i \bigcup R_j) = FALSE$ for any adjacent region R_i and R_j.

$P(R_i)$ is a logical predicate defined over the points in set R_i and ϕ is the null set.
(a) means that the segmentation must be complete; that is, every pixel must be in a region.
(b) requires that points in a region must be connected in some predefined sense.

(c) indicates that the regions must be disjoint.
(d) deals with properties that must be satisfied by the pixels in a segmented region.
(e) indicates that the sets are different in the sense of predicate P

- **Classification:** Different objects were identified from the segmented images and their information was extracted from it. The objects were then classified according to a set criteria and this information was provided to the computer to make certain decisions based upon them. For classification of the objects in this project, the geometric properties of the objects were exploited to identify it, in addition with matching the contour shapes. The algorithm is summarized in the next section.

2.5 Camera Calibration and Real World Coordinates

- **Camera Calibration:** Geometric camera calibration, also referred to as camera resectioning, estimates the parameters of a lens and image sensor of an image or video camera. We can use these parameters to correct for lens distortion, measure the size of an object in world units, or determine the location of the camera in the scene. These tasks are used in applications such as machine vision to detect and measure objects. They are also used in robotics, for navigation systems, and 3-D scene reconstruction.

  Camera parameters include intrinsics, extrinsics, and distortion coefficients. To estimate the camera parameters, we need to have 3-D world points and their corresponding 2-D image points. We can get these correspondences using multiple images of a calibration pattern, such as a checkerboard. Using the correspondences, we can solve for the camera parameters.r

  Often, we use $[\ u\ v\ 1\ ]^T$ to represent a 2D point position in pixel coordinates. $[x_w y_w z_w 1]^T$ is used to represent a 3D point position in World coordinates. Note: they were expressed in augmented notation of Homogeneous coordinates which is the most common notation in robotics and rigid body transforms. Referring to the pinhole camera model, a camera matrix is used to denote a projective mapping from World coordinates to Pixel coordinates. [11]

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R \quad T] \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

where $K = \begin{bmatrix} \alpha_x & \gamma & u_0 \\ 0 & \alpha_y & v_0 \\ 0 & 0 & 1 \end{bmatrix}$ {Intrinsic Parameter}

$\boldsymbol{R}$ & $\boldsymbol{T}$ are rotational and translational matrices {Extrinsic Parameters}

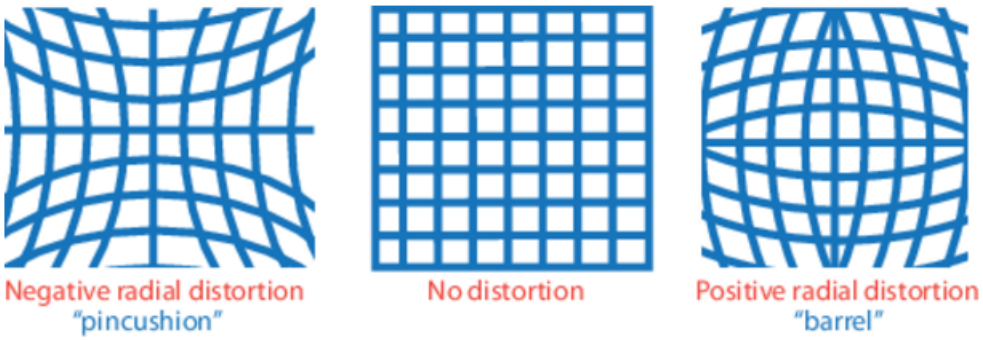


Fig 5. Distortion vs No Distortion in camera

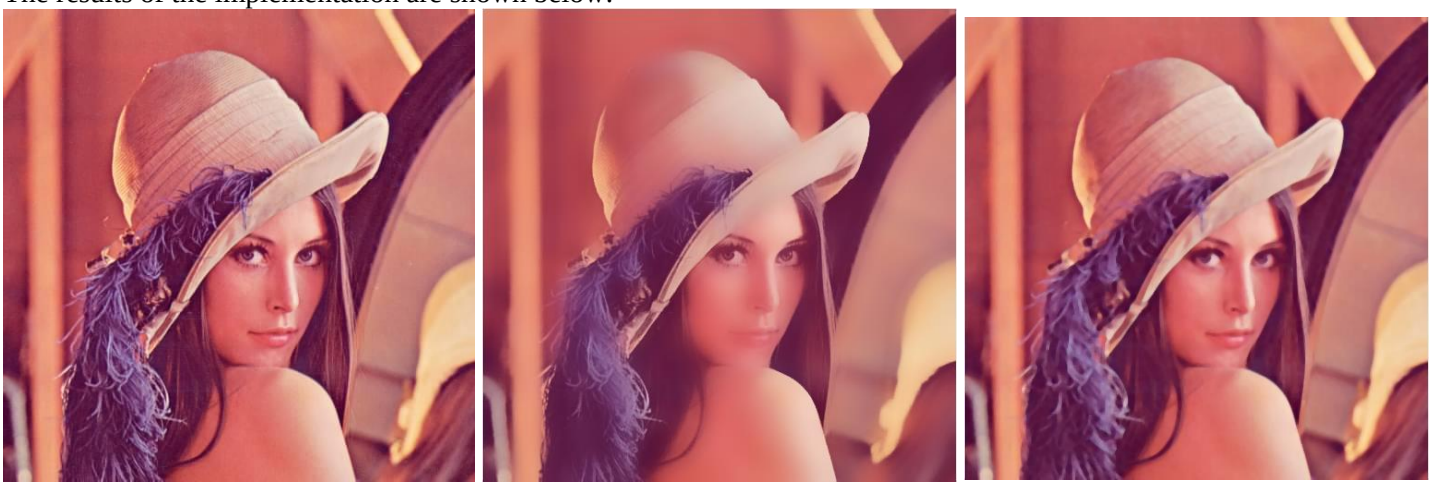
3. Work Done

In this section, all the work done till reaching the final objective and their respective results are presented. First, basic image processing on pre-loaded images are done, after which the processes were applied on real time frames after interfacing with a camera. Finally, all of the learned knowledge was applied to recreate a part of the Robotic Arm classifier computer vision project.

3.1 Pre-loaded Image Processing

- **Filtering:** To get familiarized with using the OpenCV environment, a program applying the basic image processing operations on the standard test image 'Lena.jpg' was implemented. The process was to filter the image using the Bilateral Filter and the Median Filter. This was very straight forward with the inbuilt OpenCV functions for the processes.
- The parameters of the bilateral filter was, 30 as the diameter of the pixel neighborhood, 120 as the standard deviation in colorspace and 100 as the standard deviation in the coordinate space.
- The kernel size for the median filter was of 5x5.

The results of the implementation are shown below:



(a) **(b)** **(c)**

Fig 6. (a) Original Image (b) Bilateral Filtered (c) Median Filtered

The bilateral filter gave the image a bit of a smudge effect, whereas the median filter was able to filter out the high frequency components in the image.

- **Watershed Segmentation:** For the implementation of the theory discussed in **2.4**, the color image will first be required to be converted to a 32 bit signed integers (CV_32S), since the watershed flooding requires high accuracy calculations. Also, for the given image, the mask will be pre-defined manually to obtain the best possible result. So, the algorithm followed for implementing this operation is:

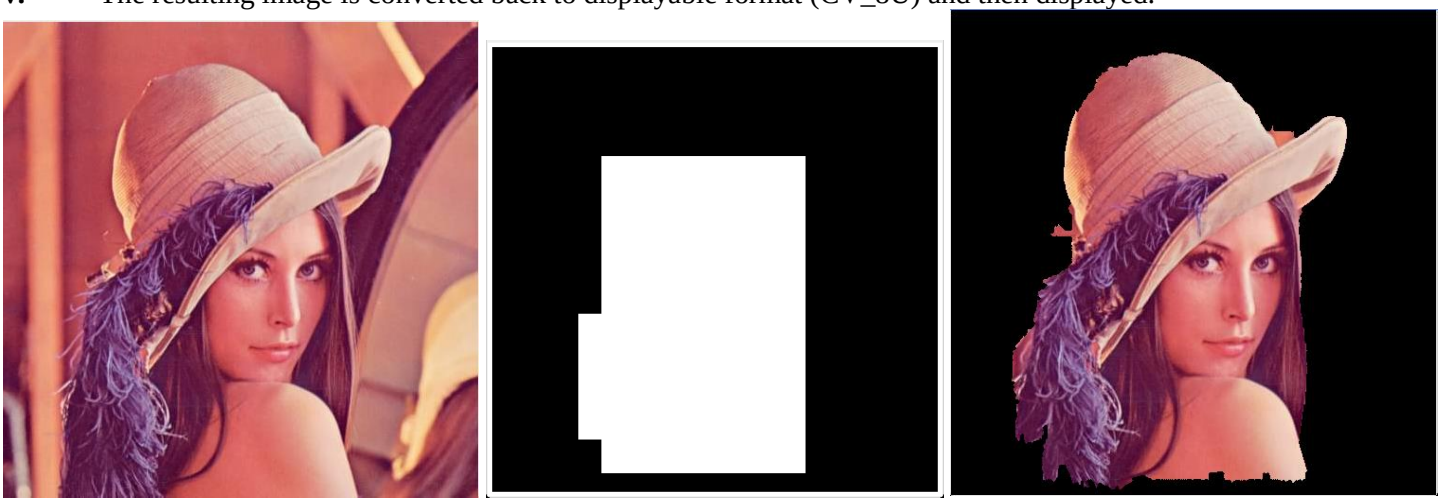
i. A function WatershedSegmenter is defined with the operations of setMarkers (to convert the mask to CV_32S) and process (for the actual Watershed segmentation).

ii. In the main function, the image is first loaded and displayed. Then, the mask is manually defined using the OpenCV Rect() function, which creates an image matrix of the defined size. This was done trial and error. {*Note: the mask doesn't have to be highly precise since the function looks for the minima in the defined mask region. The drawback is that some part of the background could be captured or the segmentation would have rough edges.*}

iii. After the mask is defined, the function WatershedSegmenter is called and used obtain the segmented mask image.

iv. The mask image is thresholded to obtain the watershed mask, which is then bitwise-and operated with the original image.

v. The resulting image is converted back to displayable format (CV_8U) and then displayed.



(a) (b) (c)

Fig 7. (a) Original Image (b) Manually defined Mask (c) Segmented Image

3.2 Real Time Video Processing

- **Edge Detection:** Here, frames from an interfaced camera were captured by the Rpi and processed using OpenCV to find its edges. The frames were first filtered using the Gaussian blur to remove noise so as to reduce the fake edges detected and then it passed through the Canny edge detector to obtain the output. The major challenge here was to learn how to interface an external USB camera to the Rpi in the OpenCV environment. There were some compatibility issues which were later resolved.
  The Gaussian blur parameters were: Kernel = 7x7, standard deviation in x & y = 1.5
  The Canny function parameters used: hysteresis threshold1 = 0, threshold2 = 30, aperture size = 3.

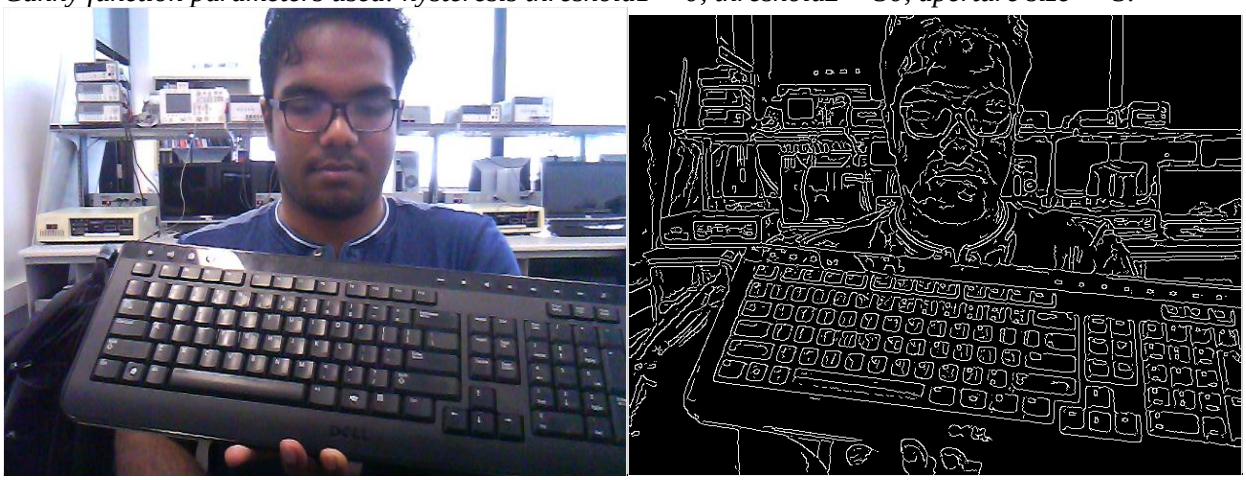
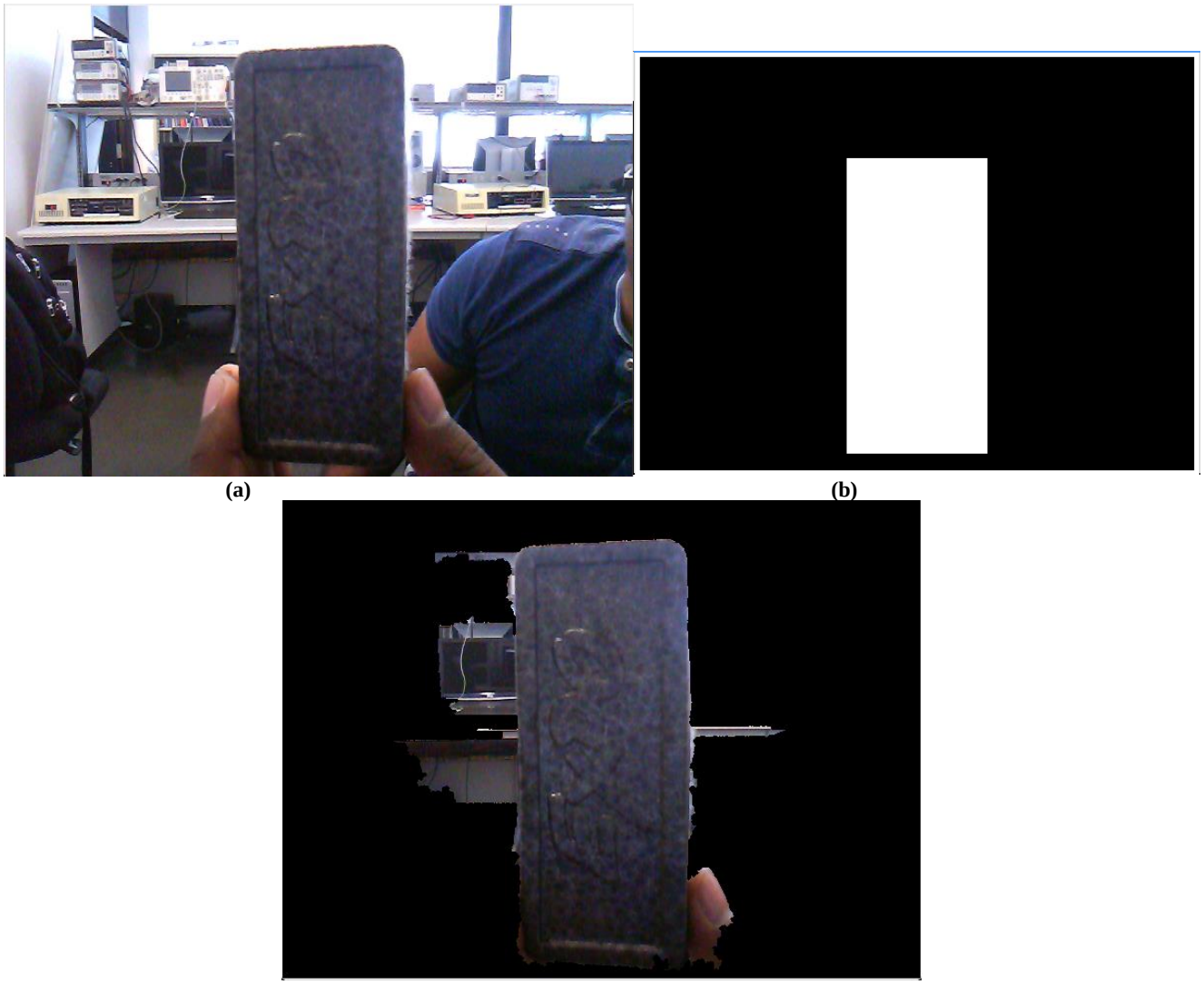


Fig 8. (a) Original Image (b) Edge Image

- **Watershed Segmentation:** The algorithm followed here is nearly similar to the one discussed in Sec **3.1**. The only addition here is that the images segmented were real time frames captured from the camera. The manual mask was the major drawback in this procedure since it could only segment the object in the middle of the frame. Following shows the results of this implementation.



(a) **(b)**

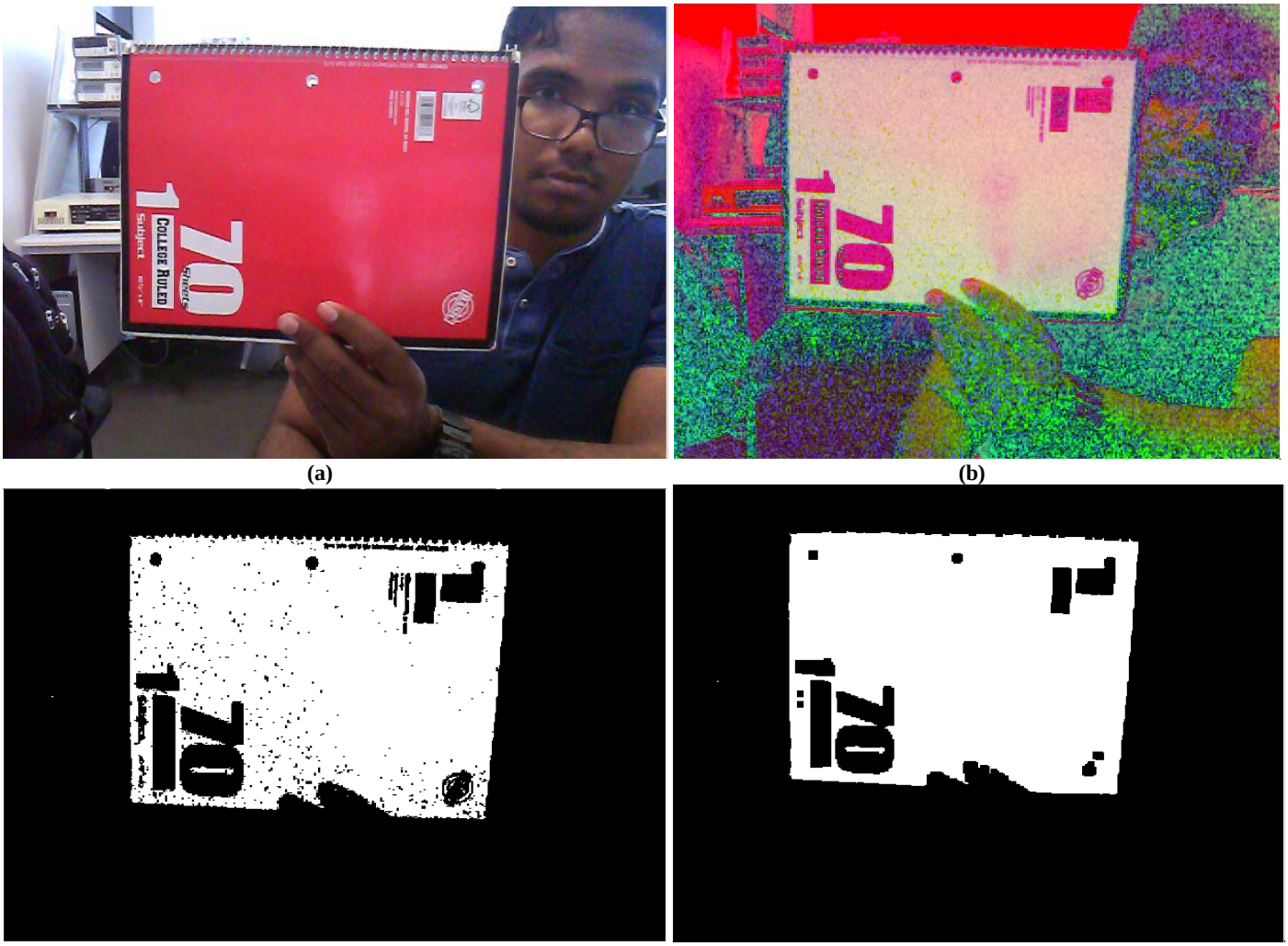
(c)

Fig 9. (a) Original Frame (b) Manually defined Mask (c) Segmented Image

- **HSV Based Segmentation:** Better results were obtained by using HSV colorspace for segmentation. Here, a simple thresholding process is used for segmentation, with the threshold taken to be at the center of the frame. Following is the overview of the algorithm implemented.

i. The device is interfaced with the camera and the captured frames are displayed.
ii. The HSV space values of interest are defined as hsvlow and hsvhigh. {*Note: This is because all the values of the HSV cannot be used as there are some overlapping color shades between the fundamental pure colors for certain HSV values.*}
iii. The frames are converted to the HSV colorspace using the function cvtColor().
iv. The threshold p is determined as the middle of the frame at rows/2 and columns/2.
v. The threshold is applied to the whole frame and the object of interest is extracted and displayed in grayscale.
vi. Since the thresholded image has a lot of errors because of uneven illumination of the frame, a morphological close operation is done with kernel size of 7x7 on the threshold image.
vii. Finally the segmented image in grayscale is displayed.

Following are the results of the segmentation. As we can clearly see, we obtain really better results.



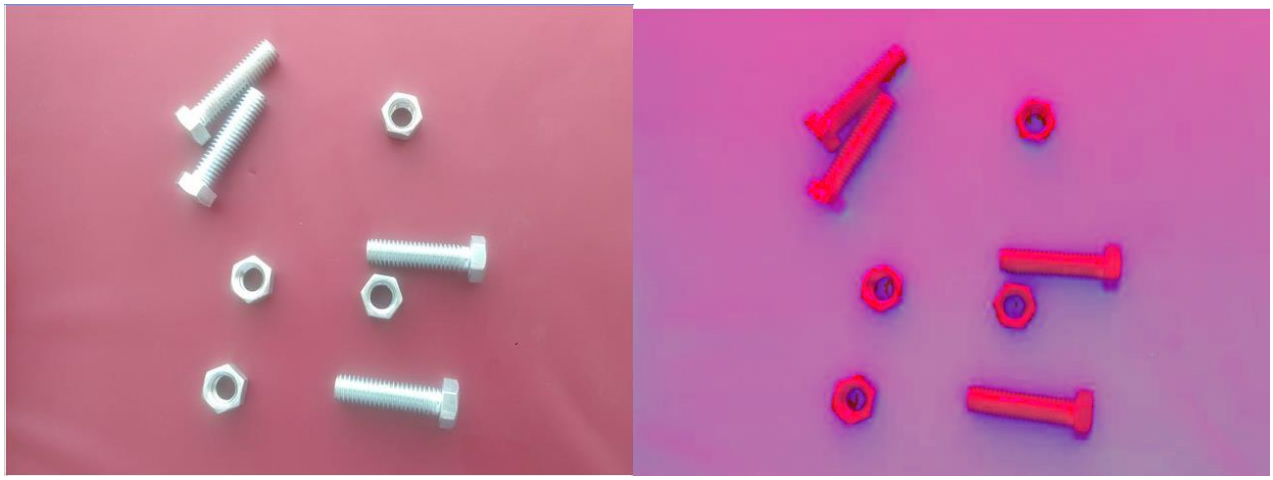
(a) **(b)**

(c) **(d)**

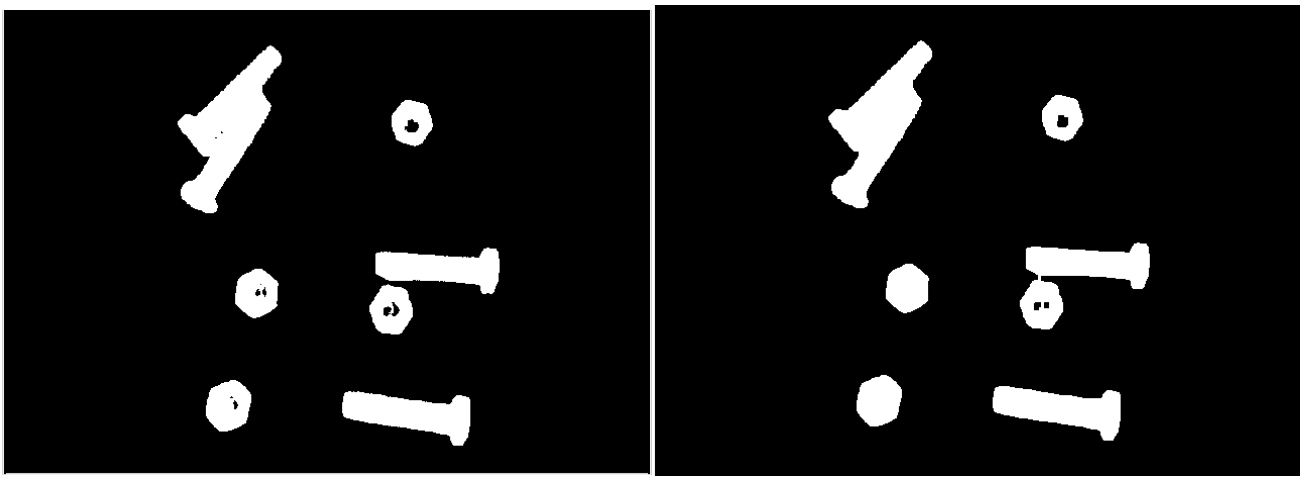
Fig 10. (a) Original Frame (b) HSV Frame (c) Segmented Frame (d) Filled Segmented Frame

3.3 Real Time Object Classifier

- **HSV Based:** Again, an almost similar algorithm as was done in the previous section (Sec **3.2**) was used here. The only difference here was that the threshold was user defined, specifically defined at the point of their mouse-click. The HSV value at that position is then used for the segmentation of the objects. This was done by the OpenCV mousecallback() function.



(a) **(b)**



(c) (d)

Fig 11. (a) Original Frame (b) HSV Frame (c) Segmented Frame (d) Filled Segmented Frame

➢ **Real World Coordinate Transformation:**

The image pixel coordinates are transformed to Real World coordinates by the extrinsic parameters of the camera, the translational and rotational matrices. Since, most of the operation here is in a 2D plane, and we assume the camera to be fixed in one place, we can forgo the rotational matrix and consider the translational for only 2 dimensions.

That is `[ X ; Y ] = [ a00 a01 ; a10 a11] * [ u ; v ] + [tx ; ty],` where the A and T are the transformation matrices.

To find all the coefficients, we map the points from the 4 corners of the image into the real world system. Here, we are mapping Pixel coordinates to Cm coordinates. This is done manually with a ruler as show in **Fig 12.** The real world coordinates are X, Y and the pixels are u,v . Using the system of equations, we can find out the values of the transformation matrices.

O ---> [0,1] pixels = [0,0] cm *{Origin}* , B ---> [489,0] pixels = [47.8 , 0] cm

C ---> [0,276] pix = [0,26.3] cm , D ---> [489,274] pix = [47.8, 26.7] cm

The matrices obtained are

A = [0.0977505112474437 0 ; 0.0001992745510725 0.09744525547445255]

T = [0 ; - 0.09744525547445255]

These transformations were used to calculate the real world coordinates from the image coordinates.

Fig 12. Real World Coordinates Mapping

➢ **Robotic Arm Classifier:** Here, the vertical camera mounted on the robotic arm set is *used to segment the frames, determine the objects, determine their mass centers, label them, obtain their orientation with respect to the arm and their real-world coordinates.* Region growing segmentation is used in this process. The algorithm and some footnotes regarding the logic and theory behind the steps are given below:

i. A function axisProperties() is defined which returns a ratio of the major by minor axis of the ellipse which best fits the object in question. *{i.e. axisProperties() =* $\frac{majoraxis}{minoraxis}$ *of best fit ellipse.}*

ii. The contours (shapes) of Nuts and Bolts are predefined in the program using a vector of 2d points. {*This shape definition was found out using another code where a single bolt and a single nut were kept isolated in front of the camera and their contour points displayed*}

iii. The camera properties such as brightness, contrast, etc. were set using commands from OpenCV.

iv. The camera calibration matrices were defined, the cameraMatrix and the distCoeffs. These are used to compensate for the distortion of the image due to the camera lens. These parameters are then used to undistort the camera image. {*These matrices are found using a chessboard and point mapping (see Sec* ***2.5****)*}

v. The video capture is started using the interfaced camera and frames are grabbed. Only the region of interest is extracted from the frame, i.e. the region where all the objects are placed.

vi. The image is flipped and the edges of the objects are detected using Sobel high pass filter, by applying it twice, once in the X direction and then the Y direction. Finally, both are added in equal ratio to get well defined edges of the objects. This is the first part of the region growing method.

vii. The morphological operations OPEN, CLOSE and then DILATE are performed on the edge images to grow the object regions and fill up their contours. By these operations, any small undesired objects are filtered out.

viii. Using the obtained grayscale segmented image, the findContours() operation is performed to find the closed boundaries of different objects. From the information obtained by the function, we can determine how many objects there are in the image. Contours of length less than a certain value are erased.

ix. The mass centers of each object is found out using the *moments()* function, which finds points of a contour where most of the other points are concentrated. These moment points are then used to find out the mass centers of the different objects. *{ Moments* $m_{ji} = \sum_{x,y} \left(\text{array}(x,y) \cdot x^j \cdot y^i\right)$ *, Mass Centers* $\bar{x} = \frac{m_{10}}{m_{00}},\ \bar{y} = \frac{m_{01}}{m_{00}}$ *}*

x. Now, for the classification, the object contours are compared to the pre-defined shapes of the Nuts and Bolts using the function *matchShapes()*. Also by using their *axisProperties()* value and the area of the best fit circle on the object, we can identify the objects to be 'Nut', 'Bolt', 'Overlapped' or 'Unknown Object'. The objects are labeled as such.

xi. For the special case of a Bolt, we have to find out the orientation of it with respect to the Robotic Arm. It can be found by using the formula $\alpha = \theta_1 - (90 - \theta_2)$, where $\theta_1 = \frac{y_{coor}}{x_{coor}}$ and θ_2 is the orientation of the best fit rectangle for the object from the horizontal. This will determine the orientation that is needed for the robotic arm to efficiently pick up the bolt.

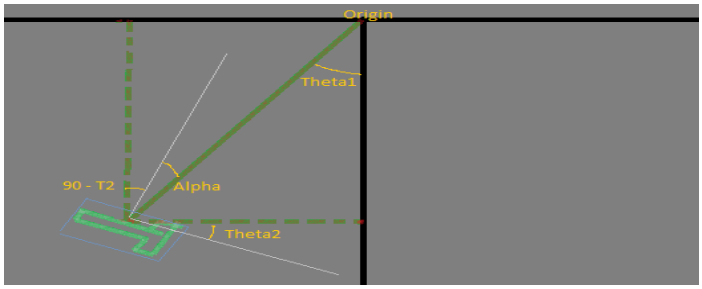
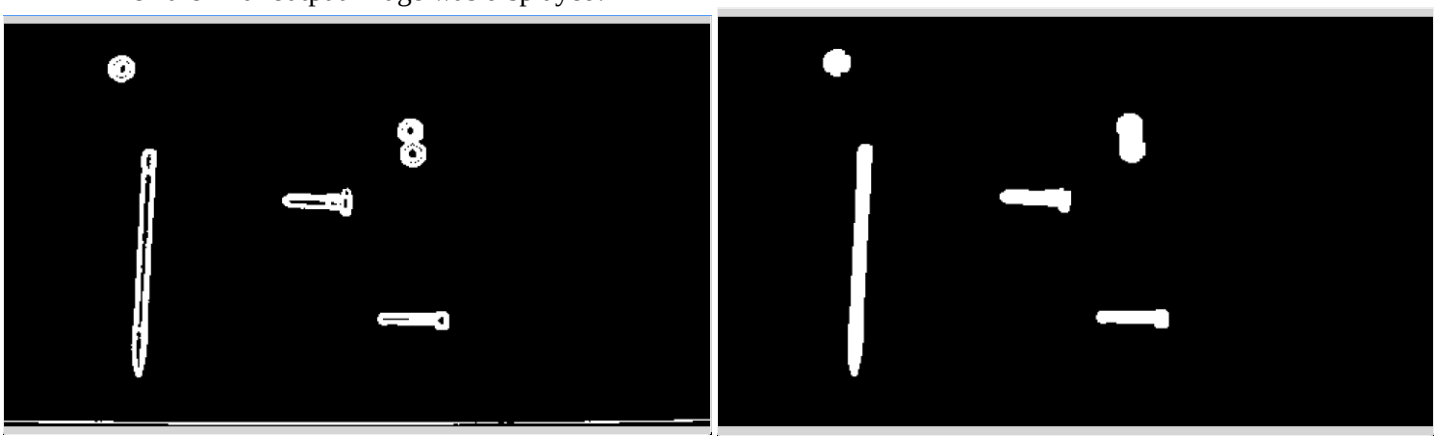


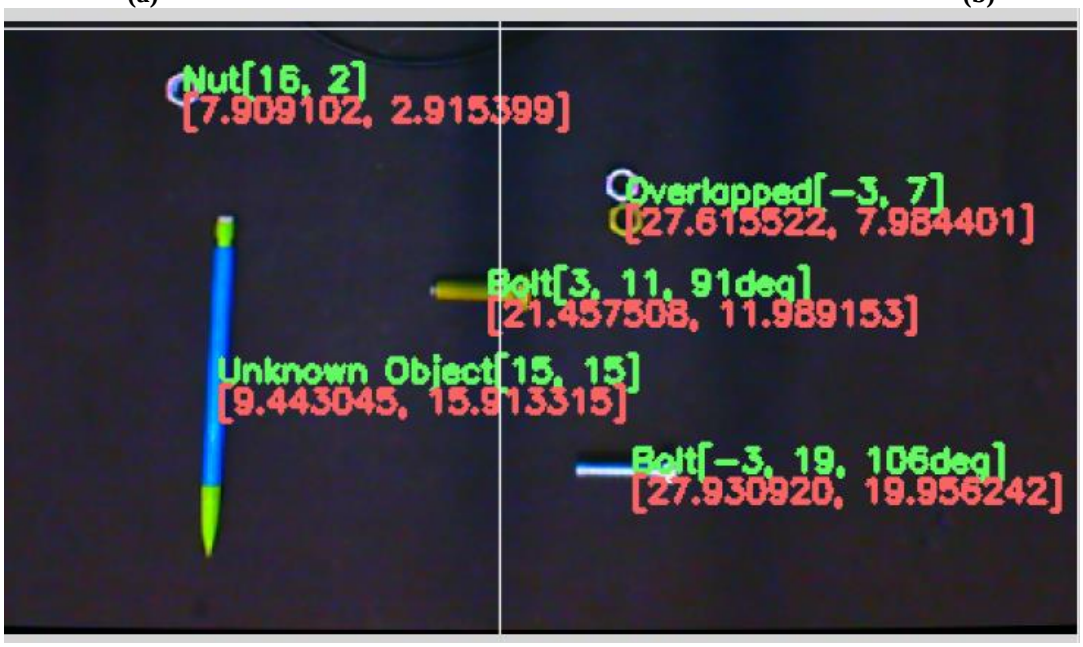
Fig 13. Orientation of Bolt

xii. The Real World Coordinates of the objects are determined by the **A** and **T** matrices determined in the last section. These values are displayed along with the labels of the objects.

xiii. Horizontal and vertical axes are drawn on the image with the origin point being the location of the robotic arm. Then the final output image was displayed.



(a) **(b)**



(c)

Fig 14. (a) Edge Image (b) Filled and Segmented Image (c) Final Output

4. Conclusion

After the preliminary of learning to operate the Raspberry Pi and OpenCV on C++, I was able to complete the objectives of the project efficiently. By learning about the segmentation algorithms and real time application of image processing techniques, I was introduced to the basics of designing Computer Vision systems. The obtained new skillsets, that are, the functioning of a new programming language (especially a popular one such as C++) and the various libraries included by it for advanced systems design has broadened my horizons, and will surely be of use in future.

In this project, I managed to implement several established image processing techniques both in real time and normally. The implementation of different types of segmentation techniques gave me insight into their respective pros and cons. Comparing the techniques, I was able to decide the most apt method for the final implementation, 'Object Classifier'. This project is a part of the overall project using the Robotic Arm. Future work can include using the implementations described here to complete the Robotic Arm classifier using the Raspberry Pi.

5. References

[1] The British Machine Vision Organization, "What is Computer Vision?"

[2] Ivan Culjak, David Abram, Tomislav Pribanic, "A brief introduction to OpenCV", *MIPRO, 2012 Proceedings of the 35th International Convention*

[3] Raspberry Pi Foundation, "Raspberry Pi 2", *Raspberrypi.org*

[4] Wikipedia, "OpenCV"

[5] Wikipedia, "Image Filtering"

[6] Wikipedia, "Edge Detection"

[7] Mathworks, "Image Morphological Operations"

[8] Wikipedia, "Watershed (Image Processing)"

[9] Wikipedia, "HSL and HSV"

[10] Wikipedia, "Region Growing (Image Segmentation)"

[11] Mathworks, "Camera Calibration"

[12] OpenCV Docs, "Functions and commands"

6. Appendix

A. ImgFilt.cpp

```
#include <opencv2/highgui/highgui.hpp>
#include <opencv/cv.h>
#include <iostream>


int main(int argc, char* argv[])
{
      cv::Mat img = cv::imread(argv[1],CV_LOAD_IMAGE_UNCHANGED);
      cv::Mat eimg,img2;

      cv::namedWindow("Color image",CV_WINDOW_AUTOSIZE);
      cv::imshow("Color image", img);

      cv::bilateralFilter(img,eimg,30,120,100);
         cv::namedWindow("Bilateral Filtered",CV_WINDOW_AUTOSIZE);
         cv::imshow("Bilateral Filtered", eimg);

      cv::medianBlur(img,img2,5);
         cv::namedWindow("Median Filtered",CV_WINDOW_AUTOSIZE);
         cv::imshow("Median Filtered", img2);


         cv::waitKey(0);
      cv::destroyWindow("Color image");
      cv::destroyWindow("Bilateral Filtered");
      cv::destroyWindow("Median Filtered");
```

```
      return 0;
}
```

B. imgseg.cpp

```
#include <opencv2/opencv.hpp>
#include <string>

using namespace cv;
using namespace std;

class WatershedSegmenter{
private:
    cv::Mat markers;
public:
    void setMarkers(cv::Mat& markerImage)
    {
        markerImage.convertTo(markers, CV_32S);
    }

    cv::Mat process(cv::Mat &image)
    {
        cv::watershed(image, markers);
        markers.convertTo(markers,CV_8U);
        return markers;
    }
};


int main(int argc, char* argv[])
{
    cv::Mat image = cv::imread(argv[1]);
    cv::Mat blank(image.size(),CV_8U,cv::Scalar(0xFF));
    cv::Mat dest;
    imshow("originalimage", image);


    cv::Mat markers(image.size(),CV_8U,cv::Scalar(-1));
    //Rect(topleftcornerX, topleftcornerY, width, height);
    //top rectangle
    markers(Rect(0,0,image.cols, 5)) = Scalar::all(1);
    //bottom rectangle
    markers(Rect(0,image.rows-5,image.cols, 5)) = Scalar::all(1);
    //left rectangle
    markers(Rect(0,0,5,image.rows)) = Scalar::all(1);
    //right rectangle
    markers(Rect(image.cols-5,0,5,image.rows)) = Scalar::all(1);
    //centre rectangle
    int centreW = image.cols/4;
    int centreH = image.rows/4;
```

```
    markers(Rect((image.cols/2)-(centreW),(image.rows/2)-(centreH), centreW*1.8,
centreH*2.8)) = Scalar::all(2);
    markers(Rect((image.cols/2)-(centreW*1.2),(image.rows/2)+50, centreW*1.8,
centreH*1.5-50)) = Scalar::all(2);
    markers.convertTo(markers,CV_BGR2GRAY);
    imshow("markers", markers);

    //Create watershed segmentation object
    WatershedSegmenter segmenter;
    segmenter.setMarkers(markers);
    cv::Mat wshedMask = segmenter.process(image);
    cv::Mat mask;
    convertScaleAbs(wshedMask, mask, 1, 0);
    double thresh = threshold(mask, mask, 1, 255, THRESH_BINARY);
    bitwise_and(image, image, dest, mask);
    dest.convertTo(dest,CV_8U);

    imshow("final_result", dest);
    cv::waitKey(0);

    return 0;
}
```

C. realtimedges.cpp

```
#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
#include<opencv2/objdetect/objdetect.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

int main(int, char**)
{
    cv::VideoCapture cap;
    cap.open(0);
    if(!cap.isOpened())
       {
     cout << "Error opening camera!";
     return -1;
 }

    cv::Mat edges;
    for(;;)
    {
        cv::Mat frame;
        cap >> frame;
        cv::cvtColor(frame, edges, CV_BGR2GRAY);
        cv::GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
```

```
        cv::Canny(edges, edges, 0, 30, 3);
        cv::imshow("Live image", frame);
        cv::imshow("Edges", edges);
        if(waitKey(10) >= 0)
     {     cv::destroyAllWindows();
           break;}
    }

    return 0;
}
```

D. rltmseg01.cpp

```
#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
#include<opencv2/objdetect/objdetect.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>

using namespace cv;
using namespace std;

class WatershedSegmenter{
private:
    cv::Mat markers;
public:
    void setMarkers(cv::Mat& markerImage)
    {
        markerImage.convertTo(markers, CV_32S);
    }

    cv::Mat process(cv::Mat &image)
    {
        cv::watershed(image, markers);
        markers.convertTo(markers,CV_8U);
        return markers;
    }
};


int main(int, char**)
{
    cv::VideoCapture cap;
    cap.open(0);
    if(!cap.isOpened())
       {
     cout << "Error opening camera!";
     return -1;
 }
```

```
    for(;;)
    {
        cv::Mat frame;
        cap >> frame;
        cv::imshow("Live image", frame);
           cv::Mat image = frame;
        cv::Mat blank(image.size(),CV_8U,cv::Scalar(0xFF));
        cv::Mat dest;

        cv::Mat markers(image.size(),CV_8U,cv::Scalar(-1));
        //Rect(topleftcornerX, topleftcornerY, width, height);
        //top rectangle
        markers(Rect(0,0,image.cols, 5)) = Scalar::all(1);
        //bottom rectangle
        markers(Rect(0,image.rows-5,image.cols, 5)) = Scalar::all(1);
        //left rectangle
        markers(Rect(0,0,5,image.rows)) = Scalar::all(1);
        //right rectangle
        markers(Rect(image.cols-5,0,5,image.rows)) = Scalar::all(1);
        //centre rectangle
        int centreW = image.cols/4;
        int centreH = image.rows/4;
        markers(Rect((image.cols/2)-(centreW/2),(image.rows/2)-(centreH),
centreW, centreH*2.8)) = Scalar::all(2);
        markers.convertTo(markers,CV_BGR2GRAY);
        imshow("markers", markers);

        //Create watershed segmentation object
        WatershedSegmenter segmenter;
        segmenter.setMarkers(markers);
        cv::Mat wshedMask = segmenter.process(image);
        cv::Mat mask;
        convertScaleAbs(wshedMask, mask, 1, 0);
        double thresh = threshold(mask, mask, 1, 255, THRESH_BINARY);
        bitwise_and(image, image, dest, mask);
        dest.convertTo(dest,CV_8U);

        imshow("final_result", dest);


      if(cv::waitKey(30) >= 0)
      {     cv::destroyAllWindows();
            break;}

    }

    return 0;
}
```

E. rltmseg02HSV.cpp

```
#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
#include<opencv2/objdetect/objdetect.hpp>
```

```
#include<opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>


using namespace std;
using namespace cv;

Mat edges,frame,bw,HSV_image, bw_filt;


Scalar hsvlow(0,0,0),hsvhigh(180,255,255);


int main ( int argc, char **argv )
{
    VideoCapture cap;
    cap.open(0);
    if(!cap.isOpened())
       {
     cout << "Error opening camera!";
     return -1; }

    namedWindow("Live image");
    namedWindow("segmented");

    Mat kernel2 = Mat::ones(7,7,CV_8U);

    int loop;
    float change;


    for(;;)
    { cv::Mat frame;
     cap >> frame;
     Scalar hsvlow(0,0,0),hsvhigh(180,255,255);

     cv::imshow("Live image", frame);
     cvtColor(frame, HSV_image, CV_BGR2HSV);
     imshow("HSV image",HSV_image);

     Vec3b p = HSV_image.at<Vec3b>(frame.rows/2,frame.cols/2);

     //cout << hsvlow  << hsvhigh << "\n";
        for (loop=0;loop<3;loop++) {
                change=p[loop]*0.3;
           if((loop==0))
                 {   change = p[loop]*0.08;
               hsvhigh[loop] = p[loop] + change;
               hsvlow[loop]  = p[loop] - change; }
           if((loop==1) && (hsvlow[loop]<127))
           {     hsvlow[loop] = 127;}
           if((loop==1) && (hsvlow[loop]>=127))
           {     hsvlow[loop] = p[loop] - change;}
           if((loop==2) && (hsvlow[loop]<150))
```

```
            {     hsvlow[loop] = 150;}
            if((loop==2) && (hsvlow[loop]>=150))
            {     hsvlow[loop] = p[loop] - change;}

         }
         inRange( HSV_image, hsvlow,hsvhigh,bw);
         imshow("segmented",bw);
       morphologyEx(bw,bw_filt,MORPH_CLOSE,kernel2);
       imshow("segmented and filled",bw_filt);
       if(cv::waitKey(90) >= 0)
       {     cv::destroyAllWindows();
             break;}

}

    return 0;
}
```

F. rltmseg03HSV.cpp

```
#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
#include<opencv2/objdetect/objdetect.hpp>
#include<opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>


using namespace std;
using namespace cv;

Mat edges,frame,bw,HSV_image,bw_filt;

void mouseEvent (int evt, int x, int y, int flags, void*);

Scalar hsvlow(0,0,0),hsvhigh(180,255,255);


int main ( int argc, char **argv )
{

    namedWindow("Input image");
    namedWindow("segmented");

    Mat kernel1 = Mat::ones(3,3,CV_8U);

    frame = imread(argv[1], CV_LOAD_IMAGE_COLOR);

    Scalar hsvlow(0,0,0),hsvhigh(180,255,255);

    cvtColor(frame, HSV_image, CV_BGR2HSV);
```

```
    setMouseCallback( "Input image", mouseEvent, &frame );

    //cout << hsvlow  << hsvhigh << "\n";

    char key = 0;
    while ((int)key != 27)
    { imshow("Input image", frame);
      imshow("HSV image",HSV_image);
      key =  waitKey(1);
    }

    return 0;
}
void mouseEvent (int evt, int x, int y, int flags, void*)
{
     if(evt == CV_EVENT_LBUTTONDOWN)
     {     Vec3b p = HSV_image.at<Vec3b>(y,x);
           int loop;
           float change;
           Mat kernel2 = Mat::ones(5,5,CV_8U);
           for (loop=0;loop<3;loop++) {
                change=p[loop]*0.3;
                if((loop==0))
                     {   change = p[loop]*0.15;
                    hsvhigh[loop] = p[loop] + change;
                    hsvlow[loop]  = p[loop] - change; }
                if((loop==1) && (hsvlow[loop]<127))
                {     hsvlow[loop] = 127;}
                if((loop==1) && (hsvlow[loop]>=127))
                {     hsvlow[loop] = p[loop] - change;}
                if((loop==2) && (hsvlow[loop]<150))
                {     hsvlow[loop] = 150;}
                if((loop==2) && (hsvlow[loop]>=150))
                {     hsvlow[loop] = p[loop] - change;}
           }
           inRange( HSV_image, hsvlow,hsvhigh,bw);
           threshold (bw, bw, 70, 255, CV_THRESH_BINARY_INV);
           imshow("segmented", bw);
           morphologyEx(bw,bw_filt,MORPH_CLOSE,kernel2);

           imshow("segmented and filled",bw_filt);



     }
}
```

G. rbarm01.cpp

```
#include <opencv2/core/core.hpp>
#include <opencv2/opencv.hpp>
#include <opencv2/objdetect/objdetect.hpp>
```

```
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include <string>
#include <sstream>
#include <stdio.h>


#define PI 3.14159265

using namespace cv;
using namespace std;

double axisProperties(vector<Point> contour);


int main(int, char**)
{
    VideoCapture cap;
    cap.open(0);

    Mat imgGray,edges;
    int ddepth = CV_16S;
    int delta = 0;
    int scale = 1;
    vector< vector<Point> > contours, contours2;
    vector <Point> shapeBolt2(25); vector <Point> shapeBolt1(36);
    vector <Point> shapeNut(28);
    vector <Vec4i> hierarchy;

      //----------------Shape definitions-----------------------------------------
----------------------------------
    shapeBolt2[0] = Point(274,138);     shapeBolt2[1] = Point(274,140);
    shapeBolt2[2] = Point(273,141);     shapeBolt2[3] = Point(273,145);
    shapeBolt2[4] = Point(274,146);     shapeBolt2[5] = Point(274, 169);
    shapeBolt2[6] = Point(273, 170);     shapeBolt2[7] = Point(273, 179);
    shapeBolt2[8] = Point(272, 180);     shapeBolt2[9] = Point(272, 183);
    shapeBolt2[10] = Point(273, 184);     shapeBolt2[11] = Point(273, 185);
    shapeBolt2[12] = Point(274, 186);     shapeBolt2[13] = Point(279, 186);
    shapeBolt2[14] = Point(280, 185);    shapeBolt2[15] = Point(280, 180);
    shapeBolt2[16] = Point(281, 179);    shapeBolt2[17] = Point(281, 165);
    shapeBolt2[18] = Point(282, 164);    shapeBolt2[19] = Point(282, 149);
    shapeBolt2[20] = Point(283, 148);    shapeBolt2[21] = Point(283, 147);
    shapeBolt2[22] = Point(285, 145);    shapeBolt2[23] = Point(285, 139);
    shapeBolt2[24] = Point(284, 138);

    shapeBolt1[0] = Point(247, 139);     shapeBolt1[1] = Point(246, 140);
    shapeBolt1[2] = Point(243, 140);     shapeBolt1[3] = Point(243, 141);
    shapeBolt1[4] = Point(242, 142);     shapeBolt1[5] = Point(242, 145);
    shapeBolt1[6] = Point(243, 146);     shapeBolt1[7] = Point(244, 146);
    shapeBolt1[8] = Point(245, 147);     shapeBolt1[9] = Point(245, 151);
    shapeBolt1[10] = Point(246, 152);     shapeBolt1[11] = Point(246, 173);
    shapeBolt1[12] = Point(245, 174);     shapeBolt1[13] = Point(245, 182);
```

```
    shapeBolt1[14] = Point(246, 183);    shapeBolt1[15] = Point(246, 184);
    shapeBolt1[16] = Point(247, 185);    shapeBolt1[17] = Point(247, 186);
    shapeBolt1[18] = Point(248, 187);    shapeBolt1[19] = Point(252, 187);
    shapeBolt1[20] = Point(253, 186);    shapeBolt1[21] = Point(253, 185);
    shapeBolt1[22] = Point(255, 183);    shapeBolt1[23] = Point(255, 166);
    shapeBolt1[24] = Point(256, 165);    shapeBolt1[25] = Point(256, 157);
    shapeBolt1[26] = Point(257, 156);    shapeBolt1[27] = Point(257, 148);
    shapeBolt1[28] = Point(258, 147);    shapeBolt1[29] = Point(260, 147);
    shapeBolt1[30] = Point(261, 146);    shapeBolt1[31] = Point(261, 142);
    shapeBolt1[32] = Point(260, 141);    shapeBolt1[33] = Point(260, 140);
    shapeBolt1[34] = Point(257, 140);    shapeBolt1[35] = Point(256, 139);


    shapeNut[0] = Point(247,119);
    shapeNut[1] = Point(245,121);
    shapeNut[2] = Point(245,122);
    shapeNut[3] = Point(244,123);
    shapeNut[4] = Point(244,124);
    shapeNut[5] = Point(243,125);
    shapeNut[6] = Point(243,131);
    shapeNut[7] = Point(244,131);
    shapeNut[8] = Point(245,132);
    shapeNut[9] = Point(245,134);
    shapeNut[10] = Point(246,133);
    shapeNut[11] = Point(248,135);
    shapeNut[12] = Point(249,135);
    shapeNut[13] = Point(250,136);
    shapeNut[14] = Point(251,135);
    shapeNut[15] = Point(252,135);
    shapeNut[16] = Point(254,133);
    shapeNut[17] = Point(254,131);
    shapeNut[18] = Point(251,128);
    shapeNut[19] = Point(252,127);
    shapeNut[20] = Point(253,127);
    shapeNut[21] = Point(254,126);
    shapeNut[22] = Point(255,126);
    shapeNut[23] = Point(257,124);
    shapeNut[24] = Point(257,123);
    shapeNut[25] = Point(258,122);
    shapeNut[26] = Point(258,121);
    shapeNut[27] = Point(256,119);


    //----------------------Set Camera Properties-----------------------------
----------------------------------
    cap.set(CV_CAP_PROP_FOCUS, 0);
    cap.set(CV_CAP_PROP_BRIGHTNESS, -15);
    cap.set(CV_CAP_PROP_SATURATION, 200);
    cap.set(CV_CAP_PROP_CONTRAST, 10);

    if(!cap.isOpened())
       {
        cout << "Error opening camera!";
        return -1;
       }
```

```
    //-------------------Camera Calibration Matrices-----------------------------
-------------------------------------------------------------------------------
    Mat cameraMatrix = Mat::eye(3,3,CV_64F);
    cameraMatrix.at<double>(0,0) = 6.0486080554129342e+002;
cameraMatrix.at<double>(0,1) = 0;     cameraMatrix.at<double>(0,2) =
3.1950000000000000e+002;
    cameraMatrix.at<double>(1,0) = 0;   cameraMatrix.at<double>(1,1) =
6.0486080554129342e+002;     cameraMatrix.at<double>(1,2) =
2.3950000000000000e+002;
    cameraMatrix.at<double>(2,0) = 0;   cameraMatrix.at<double>(2,1) = 0;
cameraMatrix.at<double>(2,2) = 1;

    Mat distCoeffs = Mat::zeros(5,1,CV_64F);
    distCoeffs.at<double>(0,0) = 8.2620139698452666e-002;
distCoeffs.at<double>(1,0) = -2.7675886003881384e-001; distCoeffs.at<double>(2,0)
= 0;
    distCoeffs.at<double>(3,0) = 0; distCoeffs.at<double>(4,0) =
5.9528994991108919e-001;



    for(;;)
    {
        Mat frame,framed,imgfilt, imgc;
        cap >> framed;
     int rcol = 640; int rlin = 480;
     int x1 = 60; int y1 = 0;
        int Bolts = 0, Nuts = 0;
     Rect roi(x1,y1,rcol - 150,rlin - 200);
     double theta1, theta2, angle;
     Point2d aux, auxtemp; Point2d pt1 = Point2d(226,0); Point2d pt2 =
Point2d(226,280);
     Point2d pt3 = Point2d(0,3); Point2d pt4 = Point2d(490,3);
     float RWx , RWy;

     undistort(framed,frame,cameraMatrix,distCoeffs,cameraMatrix);

     //--------Edge Detection-----------------------------------------------
     frame = frame(roi);
     flip(frame,frame,-1);
        cvtColor(frame, imgGray, CV_BGR2GRAY);
        GaussianBlur(imgGray, imgGray, Size(3,3), 0, 0);

     Mat gdx,gdy,abs_gdx,abs_gdy;
     Sobel(imgGray, gdx, ddepth, 1, 0, 3, scale, delta, BORDER_DEFAULT);
     convertScaleAbs(gdx,abs_gdx);

     Sobel(imgGray, gdy, ddepth, 0, 1, 3, scale, delta, BORDER_DEFAULT);
     convertScaleAbs(gdy,abs_gdy);

     addWeighted(abs_gdx,0.5,abs_gdy,0.5, 0, edges);
     threshold(edges,edges,50,255,1);
     bitwise_not(edges,edges);
```

```
        namedWindow("Edges", WINDOW_NORMAL);
        imshow("Edges", edges);

      //-------------Morphological Operations-----------------------------------
-----
      Mat element = getStructuringElement(MORPH_RECT, Size(3,3), Point(-1,-1));
      morphologyEx(edges,imgfilt,MORPH_OPEN,element);
      morphologyEx(imgfilt,imgfilt,MORPH_CLOSE,element,Point(-1,-1),3);
      dilate(imgfilt,imgfilt, getStructuringElement(MORPH_ELLIPSE, Size(1,1)));




      //-------------------Contour Operations-----------------------------------
------------
      findContours(imgfilt.clone(), contours, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);

      for (int i = 0; i < contours.size(); i++){
            double area  = contourArea(contours[i]);

            if (area > 0 && area <= 200)
                  drawContours(imgfilt, contours, i, CV_RGB(0,0,0), -1);
      }

      //----------------------Classifier-------------------------------------------
---------------------
      findContours(imgfilt.clone(), contours2, hierarchy, CV_RETR_CCOMP,
CV_CHAIN_APPROX_SIMPLE);

      vector<Point2d> massCenter(contours2.size());
      vector<Moments> moment(contours2.size());

      for (int i = 0; i < contours2.size(); i++)
      {
            moment[i] = moments(contours2[i], false);
            massCenter[i] = Point2d(moment[i].m10/moment[i].m00,
moment[i].m01/moment[i].m00);
      }


      if (!contours2.empty() && !hierarchy.empty())
      {
            aux = Point2d(0,0); auxtemp = Point2d(0,0);

            int idx = 0;
            for(; idx >= 0 ; idx = hierarchy[idx][0])
            {

                  double compareBolt1 = matchShapes(contours2[idx], shapeBolt1,
CV_CONTOURS_MATCH_I3, 0.0);
                  double compareBolt2 = matchShapes(contours2[idx], shapeBolt2,
CV_CONTOURS_MATCH_I3, 0.0);
```

```
                if ((compareBolt1 < 0.55) || (compareBolt2 < 0.55) &&
(axisProperties(contours2[idx]) >= 2.50) && (axisProperties(contours2[idx]) < 10)
&& (arcLength(contours2[idx], true) >= 100) && (arcLength(contours2[idx], true)
<= 190))
                     Bolts = Bolts + 1;
                double compareNut = matchShapes(contours2[idx], shapeNut,
CV_CONTOURS_MATCH_I3, 0.0);
                if ((compareNut < 0.55)&&((axisProperties(contours2[idx]) < 1.3)
&& (arcLength(contours2[idx], true) <290)))
                     Nuts = Nuts + 1;

                aux.x = 0; aux.y = 0;

                aux.x = (-1 * (massCenter[idx].x - 250));
                aux.y = ((massCenter[idx].y - 5 ));

                //Real World Coordinate Conversion
                RWx = massCenter[idx].x*0.0977505112474437 + massCenter[idx].y*0
+ 0;
                RWy = massCenter[idx].x*0.0001992745510725 +
massCenter[idx].y*0.09744525547445255 - 0.09744525547445255;


                auxtemp.x = (-1 * (massCenter[idx].x - 226));
                auxtemp.y = ((massCenter[idx].y + 40 ));

                /*if(auxtemp.x < 0 && auxtemp.y < 180)
                {     aux.x = auxtemp.x*0.81+5; aux.y = auxtemp.y-5; }
                else if (auxtemp.x > 0 && auxtemp.y <180)
                {     aux.x = auxtemp.x*0.8;  aux.y =auxtemp.y*1.05; }
                else if (auxtemp.x > 0 && auxtemp.y >180)
                {     aux.x = auxtemp.x*0.85;  aux.y =auxtemp.y*1.04; }
                else if (auxtemp.x < 0 && auxtemp.y >180)
                {     aux.x = auxtemp.x*0.90;  aux.y =auxtemp.y; }*/

                aux.x = aux.x*(46.8/490); aux.y = aux.y*(26.7/280);

                string xy = "[" + to_string(cvRound(aux.x)) + ", " +
to_string(cvRound(aux.y)) + "]";
                string RWxy = "[" + to_string(RWx) + ", " + to_string(RWy) + "]";

                Point2f circleCenter;
                float circleRadius;
                minEnclosingCircle(contours2[idx], circleCenter, circleRadius);

                if ((PI*circleRadius*circleRadius > 2500 &&
axisProperties(contours2[idx]) < 3.5)||(PI*circleRadius*circleRadius > 940 &&
axisProperties(contours2[idx]) < 2.5)||(PI*circleRadius*circleRadius > 2000 &&
axisProperties(contours2[idx]) < 3.0))
                {
                     putText(frame, "Overlapped" + xy, massCenter[idx],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(100,250,100,255), 2);
                     putText(frame, RWxy,
Point2d(massCenter[idx].x,massCenter[idx].y + 14), FONT_HERSHEY_SIMPLEX, 0.5,
Scalar(100,100,250,255), 2);
```

```
                    //cout << PI*circleRadius*circleRadius << "\n";
                    //cout << (axisProperties(contours2[idx])) << "\n";
               }
               else if ((axisProperties(contours2[idx]) >= 2.50) &&
(axisProperties(contours2[idx]) < 10) && (arcLength(contours2[idx], true) >= 100)
&& (arcLength(contours2[idx], true) <= 190))
               {
                    RotatedRect rec = fitEllipse(contours2[idx]);
                    theta1 = atan(auxtemp.y/auxtemp.x) * 180/ PI;
                    theta2 = (rec.angle);
                    angle = theta1 - (90 - theta2);
                    if (angle < 0)
                         angle = 180 - abs(angle);
                    else if(angle > 180)
                         angle = abs(angle) - 180;

                    string xy = "[" + to_string(cvRound(aux.x)) + ", " +
to_string(cvRound(aux.y)) + ", " + to_string(cvRound(angle)) + "deg]";

                    string test = to_string(cvRound(theta1)) + ", " +
to_string(cvRound(theta2)) + ", " + to_string(cvRound(angle));
                    putText(frame, "Bolt" + xy, massCenter[idx],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(100,250,100,255), 2);
                    putText(frame, RWxy,
Point2d(massCenter[idx].x,massCenter[idx].y + 14), FONT_HERSHEY_SIMPLEX, 0.5,
Scalar(100,100,250,255), 2);
               }
               else if ((axisProperties(contours2[idx]) < 1.3) &&
(arcLength(contours2[idx], true) <290))
               {
                    putText(frame, "Nut" + xy, massCenter[idx],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(100,250,100,255), 2);
                    putText(frame, RWxy,
Point2d(massCenter[idx].x,massCenter[idx].y + 14), FONT_HERSHEY_SIMPLEX, 0.5,
Scalar(100,100,250,255), 2);
                    angle = 0;
               }
               else
               {
                    putText(frame, "Unknown Object" + xy, massCenter[idx],
FONT_HERSHEY_SIMPLEX, 0.5, Scalar(100,250,100,255), 2);
                    putText(frame, RWxy,
Point2d(massCenter[idx].x,massCenter[idx].y + 14), FONT_HERSHEY_SIMPLEX, 0.5,
Scalar(100,100,250,255), 2);
                    //cout << PI*circleRadius*circleRadius << "\n";
                    //cout << (axisProperties(contours2[idx])) << "\n";
               }
          }
     }

     cv::line(frame, pt1, pt2, Scalar(250,250,250));
     cv::line(frame, pt3, pt4, Scalar(250,250,250));
     namedWindow("Live image", WINDOW_NORMAL);
     imshow("Live image", frame);
```

```
      namedWindow("ALL", WINDOW_NORMAL);
      threshold(imgfilt,imgc, 0, 255, 0);
      imshow("ALL", imgc);
      //cout << "Number of Bolts = " << Bolts << ",";
         //cout << "Number of Nuts = " << Nuts << "\n";

         if(waitKey(4) >= 0)
      {     cv::destroyAllWindows();
            break;}
    }

    return 0;
}


////-------------------------------------------------------------------------
-------------------------------------------------/////

double axisProperties(vector<Point> contour)
{
      float majorAxis, minorAxis;
      Point center;
      if (contour.size()>=5)
      {     RotatedRect minEllipse = fitEllipse(contour);
            majorAxis = max(minEllipse.size.height, minEllipse.size.width);
            minorAxis = min(minEllipse.size.height, minEllipse.size.width);

            return (majorAxis/minorAxis);    }
}

///----------------------------------MEASURES FOR REAL WORLD COORDINATE----------
-------------------------------------------------------------------------------
----------//
/*
Origin at left top corner (near Robo Arm) ----> Coordinates with respect to
bottom left corner (away from Robo Arm) = [6.5cm , 26.8cm]
Rough sketch>    --------0rigin---------------Robo Arm----------------------B
            |         |                                                    |
      26.8cm ->|    |                                                  |
            |         |                                                    |
            |         |                                                    |
            |--6.5cm--C----------------------------------------------------D

O ---> [0,1] pixels =  [0,0] cm
B ---> [489,0] pixels =  [47.8 , 0] cm
C ---> [0,276] pix  =  [0,26.3] cm
D ---> [489,274] pix =  [47.8, 26.7] cm

[ X ; Y ] = [ a00 a01 ; a10 a11] * [ u ; v ] + [tx ; ty]

*/
```