Final Report for Capstone Design I

-Open CV part

AILIAN CHI 20150915

- **Step.1 - Prerequisite for starting the Open CV**

Download software

To start using the Open CV, there are several prerequisites. Another operating system Ubuntu has to be used to run this software. Once Ubuntu is ready, download Open CV 3.4.0, robot operating system (ROS), and Qtcreator which is used to edit the code in Open CV.

Camera calibration

Usually cameras shows some distortion to images, which are mainly due to radial distortions and tangential distortion. Straight lines will appear curved because of the distortion, thus, camera calibration is needed to make the curved line straight. Basically, it has two coordinate systems and three projection parameters. We got calibration parameters with ROS using a 6x9 checkerboard. Or another way is to use undistort function in Open CV.

- **Step.2 - Learning basic skills in Ubuntu**

Basic Ubuntu commands and shortcuts:

Getting familiar with the Ubuntu terminal emulator is necessary since all works of Open CV are done in Ubuntu. There are some commands and some terminal shortcuts to execute commands more quickly. For example, sudo, allows you to run programs or other commands with administrative privileges. ls is list all files and folders in current working directory. And more like cp, rm, cd, and mikdir need to be known. For the shortcuts, the most frequently used one is Ctrl + Shift + T, which is open a new terminal. A lot more other shortcuts can be explored to use the Ubuntu easier by searching in the internet.

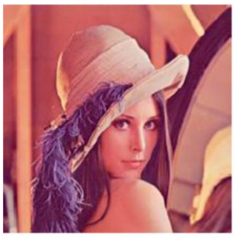
Running the Open CV

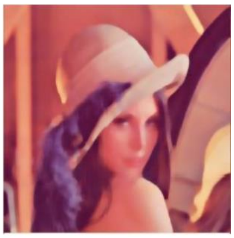
We use Qt creator to run and modify the Open CV code. It is a cross-platform C++, JavaScript and QML integrated development environment which includes a visual debugger. By using this, it is easy to modify and debug. Also, cmake-based build environment can be set easily in QT since Open CV and ROS will be integrated later on.

- **Step.3 - Basic Open CV function**

<u>Smoothing Images</u>

It is used in order to reduce noise. Linear filters are applied to the image that an output pixel's value is a weighted sum of input pixel values. We tried Normalized block filter, Gaussian filter and Median filter for example.


<Source image>


<Destination image>

<u>Thresholding Operation</u>

This function can be used to detect an object based on the range of pixel value by using the function cv::threshold and cv::inRange.

<u>Laplace Operation</u>

The Open CV function Laplacian() implements a discrete analog of the Laplacian operator. And the things on the image are approximately defined and where the contrast is higher than the region is notoriously marked.

<u>Canny Edge Detector, Hough Lines Transform, Hough Circles Transform</u>

Simply speaking, Canny Edge Detector is an algorithm that can detect the edges in an image, and Hough Lines Transform let the image display lines in an image after applying Canny Edge Detector. In the same way, Hough Circles Transform can detect circles in an image.


<Source image>

<Destination image>

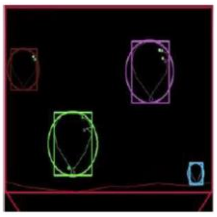
<u>Histogram Equalization</u>

It is a method that improves the contrast in an image, in case that an image does not have much color differences to be analyzed, this method can stretch out the intensity range and the number of pixels is more distributed through the intensity range.

<u>Creating bounding (rotated) boxes and circles (ellipses) for contours</u>

After an image is converted into grayscale and found the contours using the algorithms, bounding rectangular is saved for every contour. Then it becomes easy for us to detect the position of the thing that is bounded. Moreover, by applying cv::minAreaRect and cv::fitEllipse, the bounded box can be rotated and the ellipse is available for contouring so that the position will be detected more accurately.


<Source image>

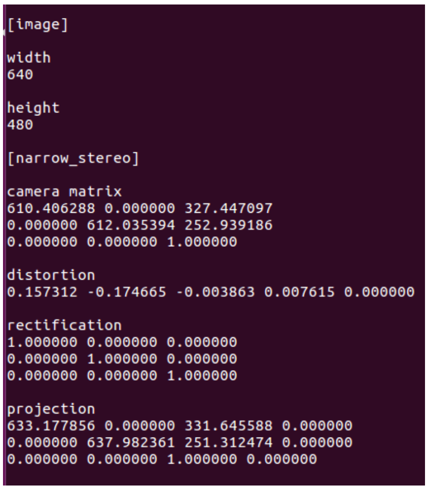

<Output>

- **Step.4 - Understanding and modifying the code**

  We were given the example code for reference. And we need to fully understand the flow of the code so that we can modify it further.

  Camera calibration

  When a different camera is used, camera calibration should be done again. And below is the intrinsic data got from the camera.



```
[image]

width
640

height
480

[narrow_stereo]

camera matrix
610.406288 0.000000 327.447097
0.000000 612.035394 252.939186
0.000000 0.000000 1.000000

distortion
0.157312 -0.174665 -0.003863 0.007615 0.000000

rectification
1.000000 0.000000 0.000000
0.000000 1.000000 0.000000
0.000000 0.000000 1.000000

projection
633.177856 0.000000 331.645588 0.000000
0.000000 637.982361 251.312474 0.000000
0.000000 0.000000 1.000000 0.000000
```

Declaration of the tranckbars and some variables

Firstly, the trackbar functions to set HSV colorspace's parameters are declared, and it can change the HSV values with just simply moving the trackbar. Then the data type is changed with intToString and floatToString. In this code, canny edge detector is used to detect the edge of the subjects with specific colors. Thus, the parameters such as threshold, ratio, and the kernel size are set in this step.

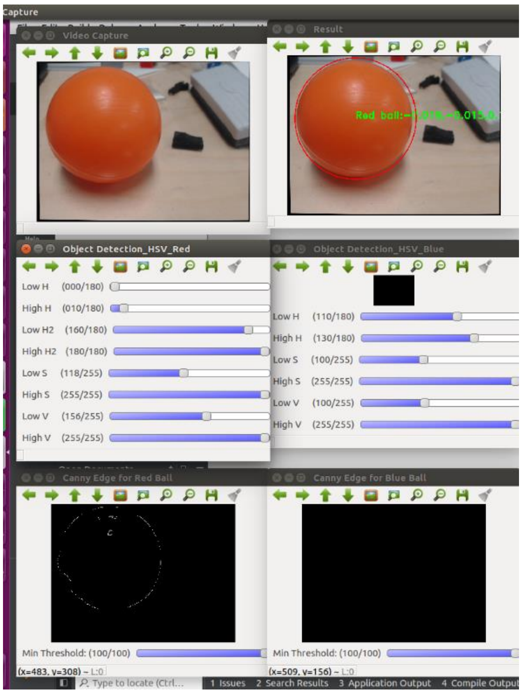
```
void on_canny_edge_trackbar_red(int, void *);
int lowThreshold_r = 100;
int ratio_r = 3;
int kernel_size_r = 3;
void on_canny_edge_trackbar_blue(int, void *);
int lowThreshold_b = 100;
int ratio_b = 3;
int kernel_size_b = 3;
```

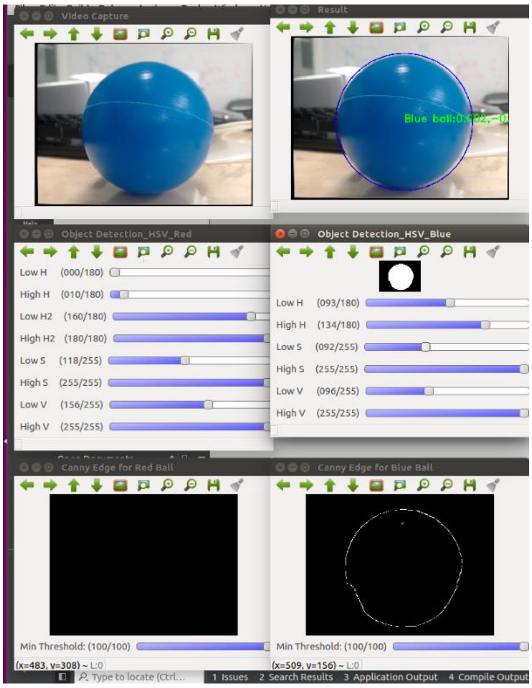
Main function for detecting ball

Basically, the flow of the code is as follows:

Firstly, the initial variables are set such as HSV values for red, blue, green, and the camera intrinsic data. Then webcam is turned on to collect the image data. The image is blurred to reduce the noise and it detects the edges with the canny edge detector. Once it detects the circle of the specific color, it will draw the minimum enclosed circle automatically. And the camera collects the position data of x, y, and z with respect to the center of the camera.

The result of the example code is as below:



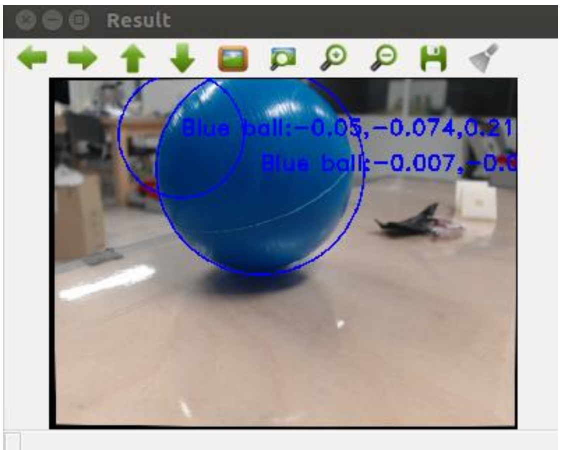


Modification of the code

1. Adjust the min_tracking_ball_size

The demonstration place is 3m wide and 5m long, and only half part of the region is arranged balls. And the green ball has to be detected in order to find the basket. Thus, the camera should at least detect the ball at 4 meters away. However, the distance it can detect at first was only 2-3 meters.

In this case, the minimum tracking size of the ball can be adjusted. Initially it was 20 pixel, which is about 0.52cm. Later the value of 10 is tried, and it could detect further. But the noise also increased which affects the accuracy as shown below:



2. Solve the multi-detection problem

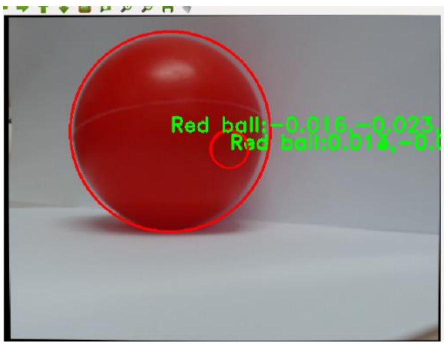
Example code worked well in Open CV, but when it comes to integrating the data in ROS, we found that the output data is not a single value even it seems to be only one contour. And it becomes severe as it gets close to the camera. The problem is it cannot smartly avoid the extra circles even they overlaps. Therefore, an algorithm that can erase the useless data is constructed as below. Its idea is when the sum of the two radius is greater than the distance between two centers, remove the circle with smaller radius, and it will keep comparing until only one value is left at last. Surely, there are many other ideas to solve this problem!

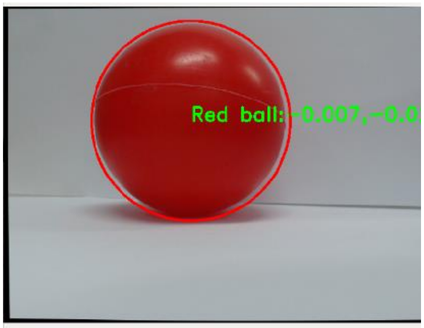
```
size_t contour_b = contours_b.size();
for( size_t i = 0; i< contour_b; i++ ){
            if(radius_b[i] > iMin_tracking_ball_size){
                for(size_t j=0; j<contour_b; j++){
                    for(size_t k=0; k<contour_b; k++){
                        one = center_b[j];
                        two = center_b[k];
                        r1 = radius_b[j];
                        r2 = radius_b[k];

                        x1 = one.x;
                        y1 = one.y;
                        x2 = two.x;
                        y2 = two.y;
                         l = sqrt((x1-x2)*(x1-x2)+(y1-y2)*(y1-y2));

                        if(r1+r2>l){
                            if(r1>r2){ //!!!
                                //max_element
                                radius_b.erase(radius_b.begin()+k);
                                center_b.erase(center_b.begin()+k);
                                contours_b.erase(contours_b.begin()+k);
                                contour_b--;
                                j--;
```

After the upgraded code is applied, it shows only one contour successfully. The comparison is as below:





Finally, the Open CV part of this semester is completed here!