Open CV

Table of Contents

Background

Introduction

For the earlier stages of the project, Open CV team focused on learning about the basic concepts of image processing, as most members were unfamiliar with the idea.

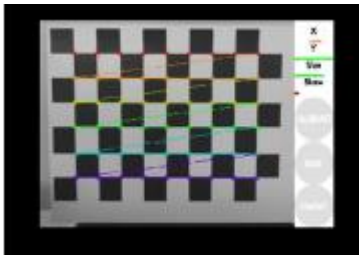
Linux based operating system Ubuntu 16.04.4 LTS version was used as a platform for developing the image processing codes for the robot.

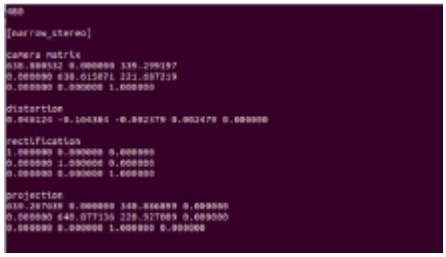
Open CV is an image processing library developed by Intel on Ubuntu. In this case, Open CV version 2.0 was used.

In addition, QTCreator version 5.10.1 was used to run the C++ codes developed.

Setup

Using Robot Operating System (ROS) and a checkerboard, the camera was calibrated. Because the camera used for the image processing takes image that is distorted (especially on the edges), a revision is required to achieve useful information from the images taken by the camera.





The camera takes multiple images of the checkerboard from different angles. Knowing that the checkerboard is actually flat and the size of each square on the checkerboard is the same, the camera's distortion parameters can be calibrated in order to flatten the incoming image data.

In the above picture, printed checkerboard on A4 was used. But for the final calibration, much more precise and larger type of checkerboard was used in order to ensure the precision of the distortion and intrinsic parameters of the camera.

Filters

Various types of filters exist in the open source market for image processing. These are mathematically developed set of codes that edit image data as desired by the user.

These filters can be implemented by running them through QTCreator.

Some examples of such filters are as follows:

-Image Smoothing: makes images smooth (may be used to rid noise)

-Threshold Operation: filter undesired levels of data

-Laplace Operation: shapes within the image may be better defined (rid unnecessary information)

-Canny Edge Detector: define edges of objects

-Hough Line Transform: detect straight lines within image

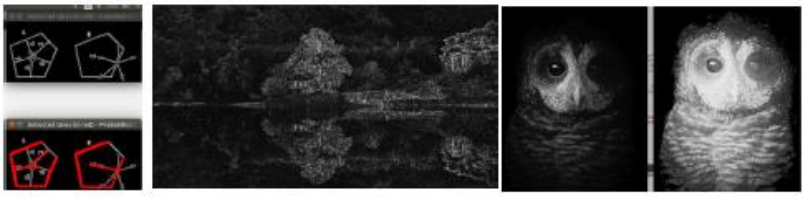
-Histogram Equalization: improve contrast of image

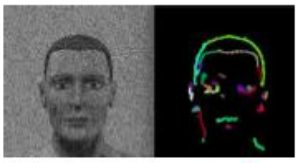
-Contour: shows contour of image

-Convex Hull: determines boundary of objects in image

-Bounding circles and boxes: enclose objects in image with various desired shapes

The following are the results of some filters applied on images





Ball Tracking

Ball tracking requires a set of algorithms that 1) detect color 2) detect shape 3) detect distance.

After calibrating the camera parameters, various filters were used to track the ball.

Multiple filters were used to edit the raw image data.

For example,

Image smoothing was used to refine the image.

Canny edge and polygon filters were used to detect the round shapes around the ball.

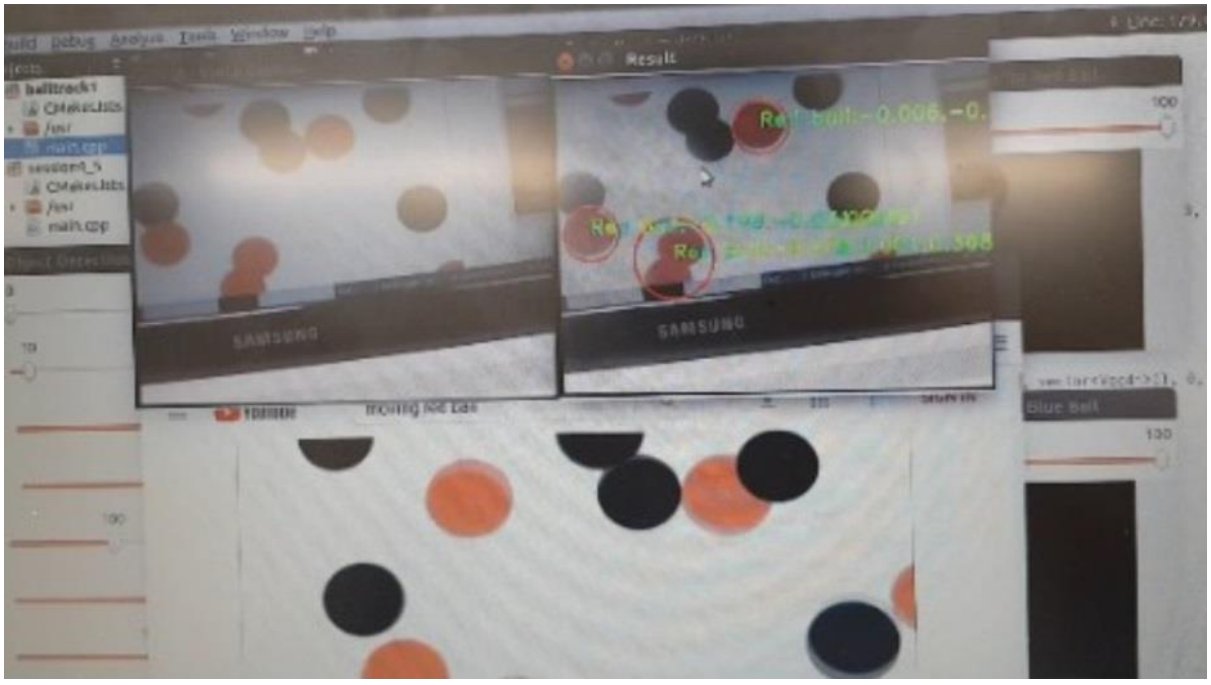
Contours were used to draw contour lines around the ball.

Circular fitting was also used. Only circles larger than a certain radius were detected.

The center of each circle was also detected. This information is important for later improvement in ball detection.

Morph and erode filters were used to merge similar colors within the ball so that only 1 ball is detected for 1 ball, rather than multiple smaller balls being detected erroneously.

RGB colorspace was converted to HSV for color detection, as it is more accurate.

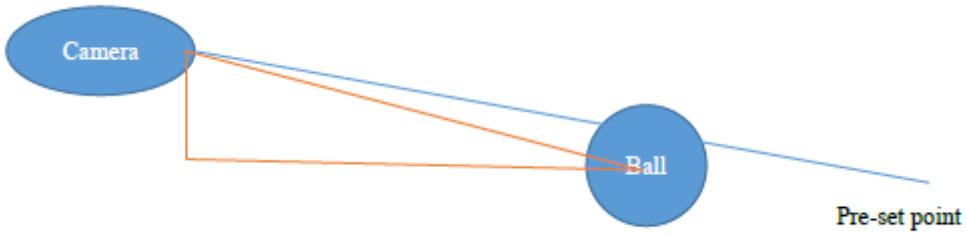


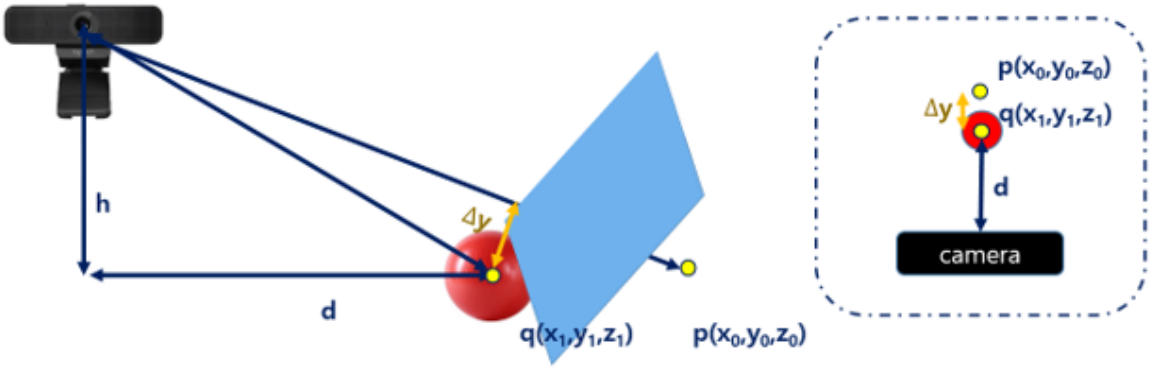
Improvements

Part i – Ball distance

Initially, the distance of the ball from the camera was calculated by using the radius information of the ball. However, because the circular fitting of the ball was not very precise, the distance acquired from the radius information also had much errors. In order to overcome this problem, a new method was developed: using center information of the ball to measure the distance with trigonometry.

The graphical explanation of this method is as follows:





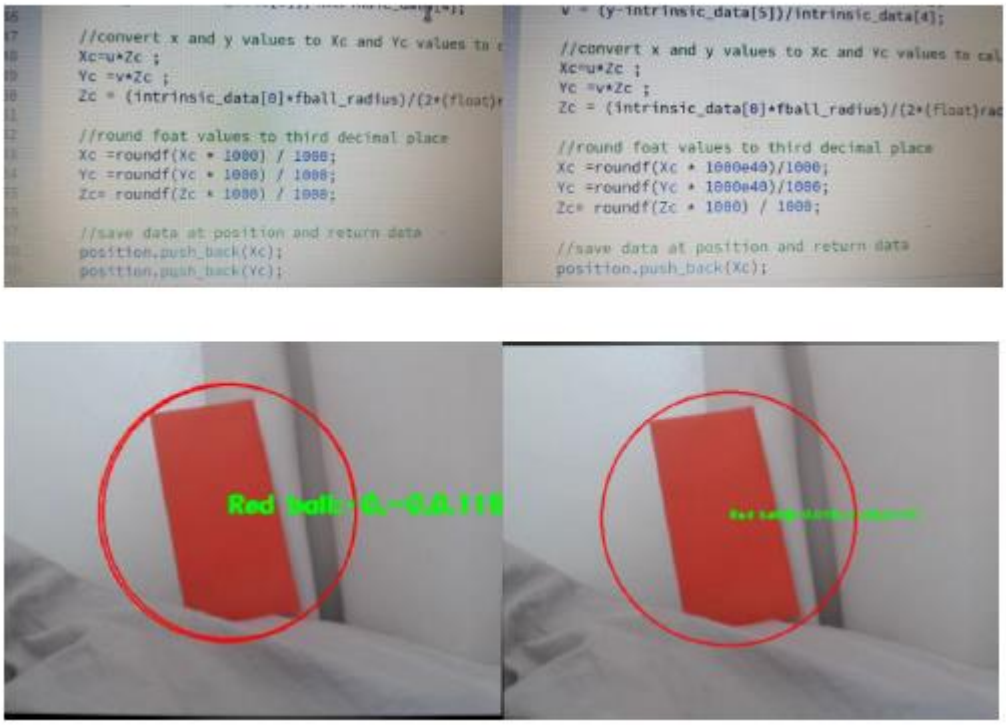
From experimental procedure, an arbitrary point on the camera image on the floor is set (or a line of set of points). Also from actual experimental procedure, the horizontal distance from the camera to such point can be physically measured.

By using this method, the precision of distance from camera to ball improved significantly.

Part ii – Interface

In order to work more efficiently, information such as distance, center, radius, etc. were displayed on the screen as follows:

<Before> <After>



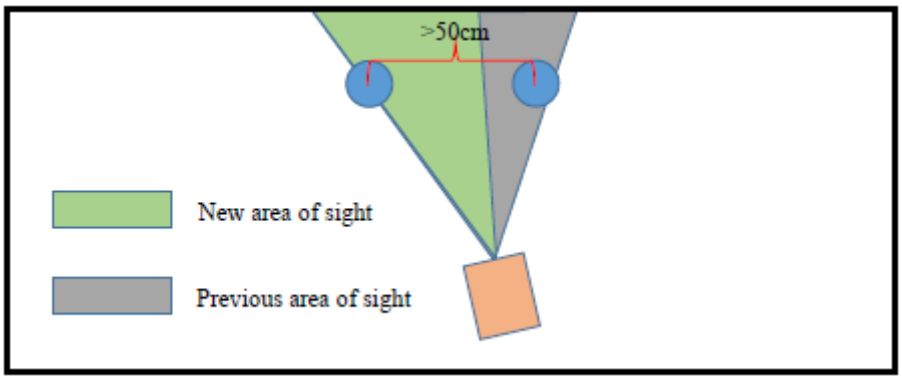
Part iii – Green Ball

New set of codes for detecting green ball to find the basket location was added to the previous set of codes.

Part iv – Priority

During test runs, when the camera detected 2 blue balls with similar distances, it had trouble deciding which ball to pick up first. There arose a problem regarding priority of ball selection.

At first, reducing the angle for the view of camera was considered as shown below. However, this would slow the entire process because the robot has a reduced sight. So another solution was considered.



The solution was developed by reducing the view so that the camera is able to see just 1 ball at a time. The reduced image only affects the priority in ball selection and does not actually affect the camera view. Therefore, the robot remembers that another ball exists in the deleted image region and proceeds to picking it up once the prioritized ball is collected.

The graphical explanation is as shown below: