A binocular camera finds coordinates by color

1.Connect your camera

To get the image, you need to catch image or video first. Find the IP camera app, and type the ip address:

```
video1="http://admin:admin@192.168.0.101:8081/"
video2="http://admin:admin@192.168.0.102:8081/"

camera1 = cv2.VideoCapture(video1)
camera2=cv2.VideoCapture(video2)
#等待两秒
```

2.Calibrate the two camera（opencv or matlab）

Using openCv

There are two steps:

1.find the parameter of the camera itself.

2.find the relative location between right camera and left camera.

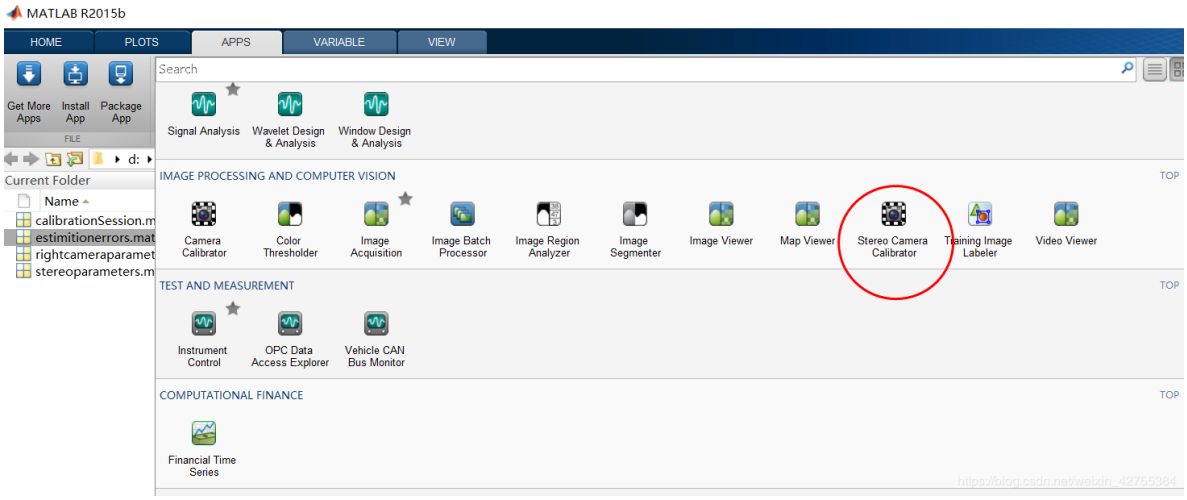
```
import cv2
import numpy as np
import glob

# find corner point of board
# threshold
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
#the dimension of the board,w stands for width,h stands for height
w = 8
h = 6
# Checkerboard points in world coordinates, such as (0,0,0), (1,0,0),
(2,0,0)...,(8,5,0), minus the Z coordinate, is written as a two-dimensional
matrix
```

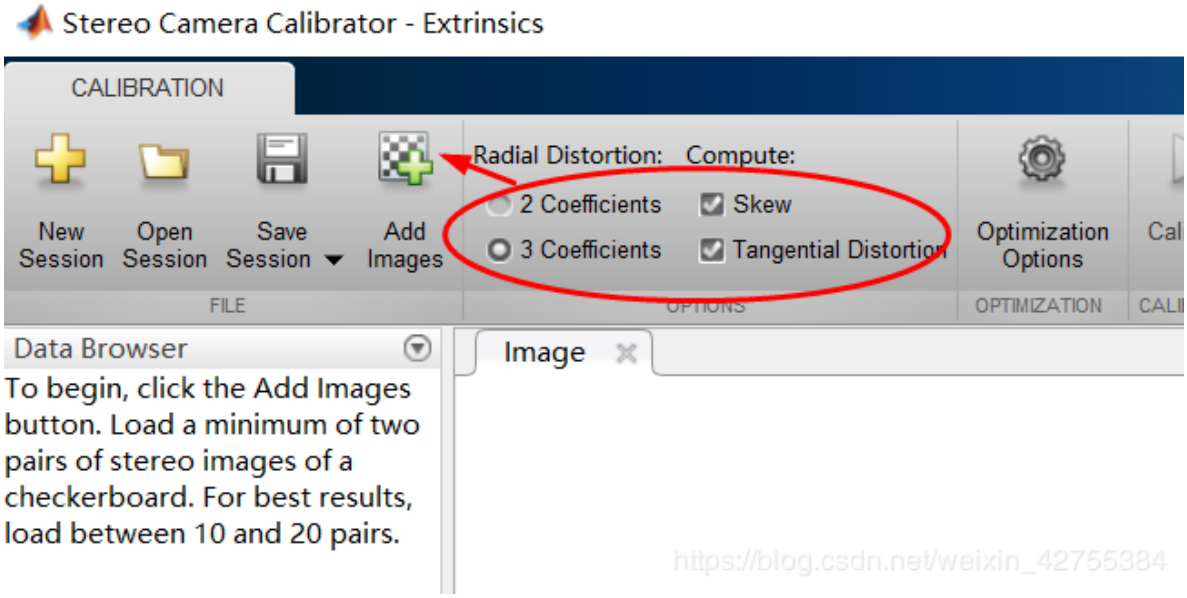
```
objp = np.zeros((w*h,3), np.float32)
objp[:,:2] = np.mgrid[0:w,0:h].T.reshape(-1,2)
# Stores the world coordinates and image coordinates of the checkerboard corner
points
objpoints = [] # Three dimensional points in the world coordinate system
imgpoints = [] # Two dimensional points in the image plane

images = glob.glob('./calibrateImages/Left/*.bmp')
for fname in images:
    img = cv2.imread(fname)
    gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    # Find the corner of the checkerboard
    ret, corners = cv2.findChessboardCorners(gray, (w,h),None)
    # If you find enough pairs, store them
    if ret == True:
        cv2.cornerSubPix(gray,corners,(11,11),(-1,-1),criteria)
        objpoints.append(objp)
        imgpoints.append(corners)
        # Display the corners on the image
        cv2.drawChessboardCorners(img, (w,h), corners, ret)
        cv2.imshow('findCorners',img)
        cv2.waitKey(1)
cv2.destroyAllWindows()

ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints,
gray.shape[::-1], None, None)
print (("ret:"),ret)
print (("mtx:\n"),mtx)        # Internal parameter matrix
print (("dist:\n"),dist)      #    distortion cofficients =
(k_1,k_2,p_1,p_2,k_3)
print (("rvecs:\n"),rvecs)    # rotation vector  # extrinsic parameter
print (("tvecs:\n"),tvecs)    # transformation vector  # extrinsic parameter
fw = open("leftparam.txt","w")
fw.write(("ret:")+str(ret)+"\n")
fw.write(("mtx:\n")+str(mtx)+"\n")        # camera matrix
fw.write(("dist:\n")+str(dist)+"\n")      #   distortion cofficients =
(k_1,k_2,p_1,p_2,k_3)
fw.write(("rvecs:\n")+str(rvecs)+"\n")    # rotation vector
fw.write(("tvecs:\n")+str(tvecs)+"\n")    # transformation vector
fw.close()

# elimate the distortion
img2 = cv2.imread('./calibrateImages/Left/img_0001.bmp')
h,w = img2.shape[:2]
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(mtx,dist,(w,h),0,(w,h)) # Free
proportional parameter
dst = cv2.undistort(img2, mtx, dist, None, newcameramtx)
# Crop the image according to the previous ROI area
x,y,w,h = roi
dst = dst[y:y+h, x:x+w]
cv2.imwrite('calibresultright.jpg',dst)
# cv2.imshow('calibresult',dst)
cv2.waitKey(1)
# cv2.destroyAllWindows()

# Backprojection error
total_error = 0
for i in range(len(objpoints)):
```

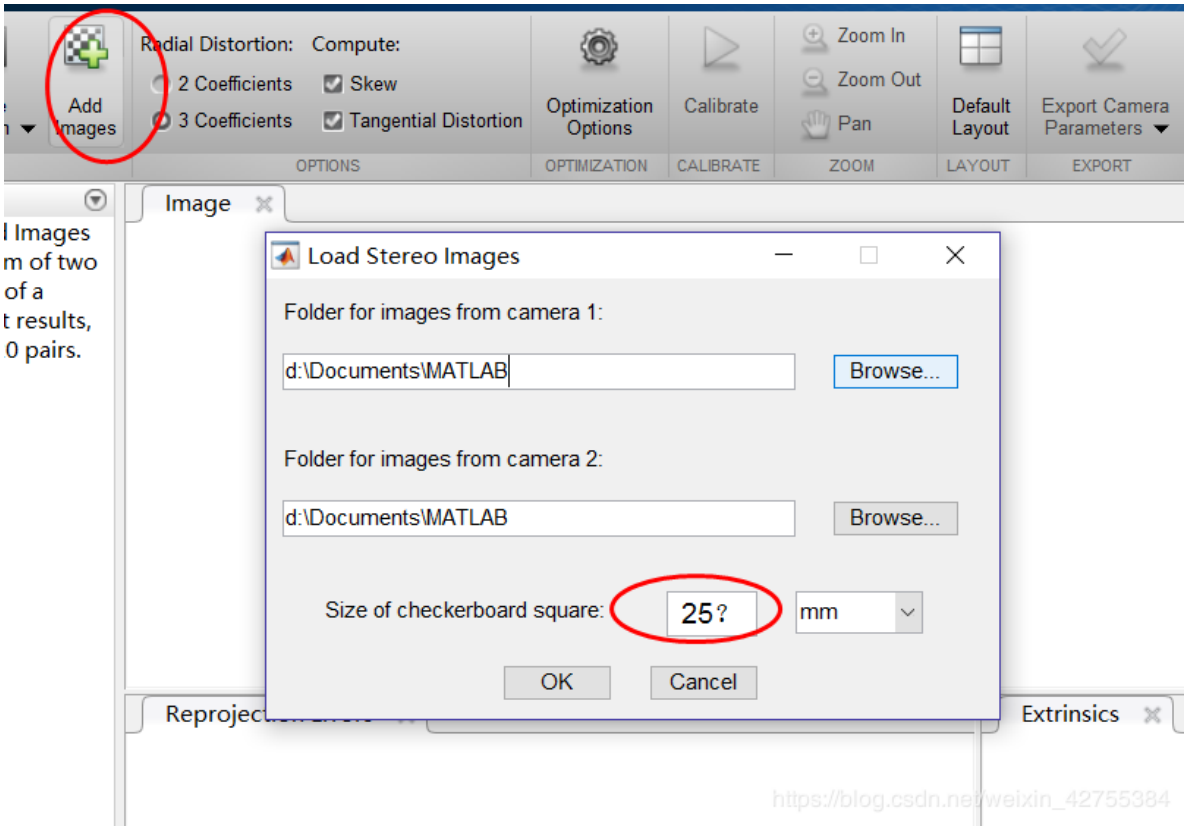
```
    imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx,
dist)
    error = cv2.norm(imgpoints[i],imgpoints2, cv2.NORM_L2)/len(imgpoints2)
    total_error += error
print (("left total error: "), total_error/len(objpoints))
```

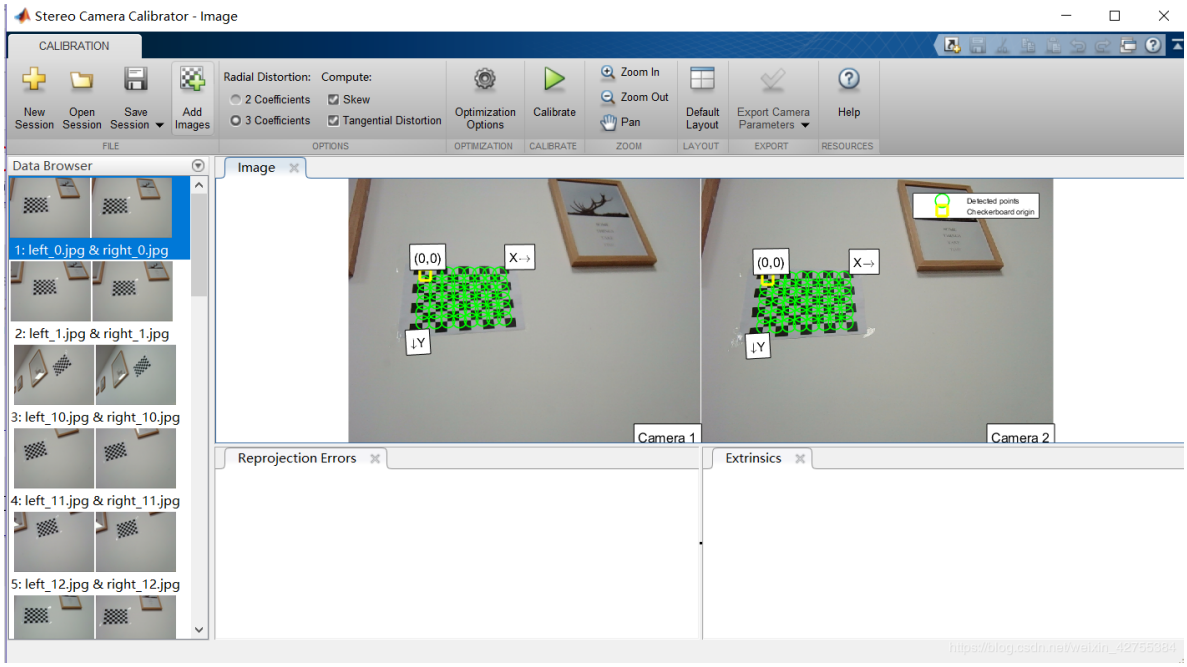
MATLAB:

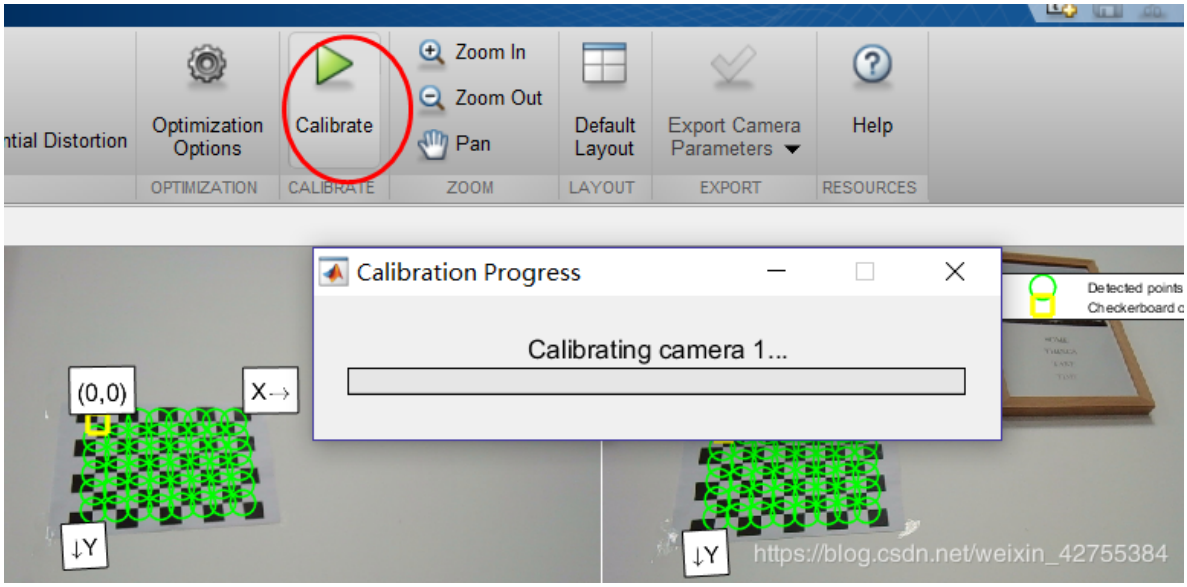
Using "camera calibrator ",get the parameter like this:

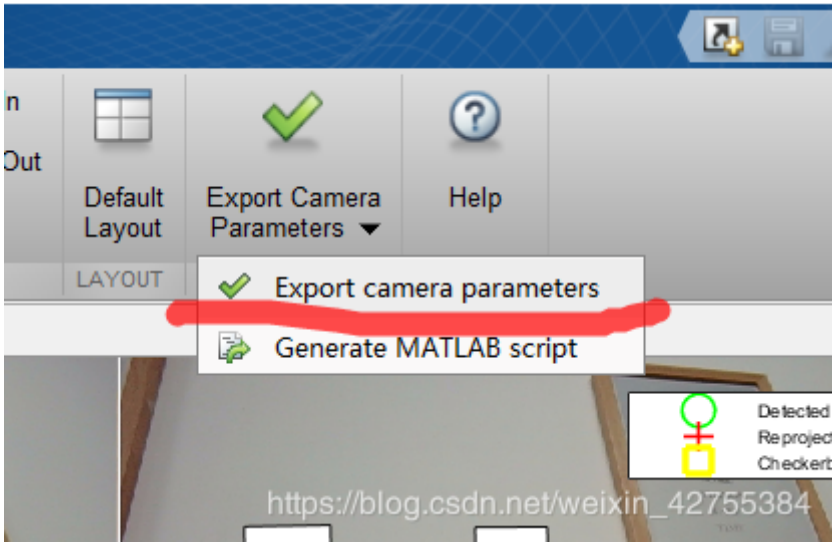


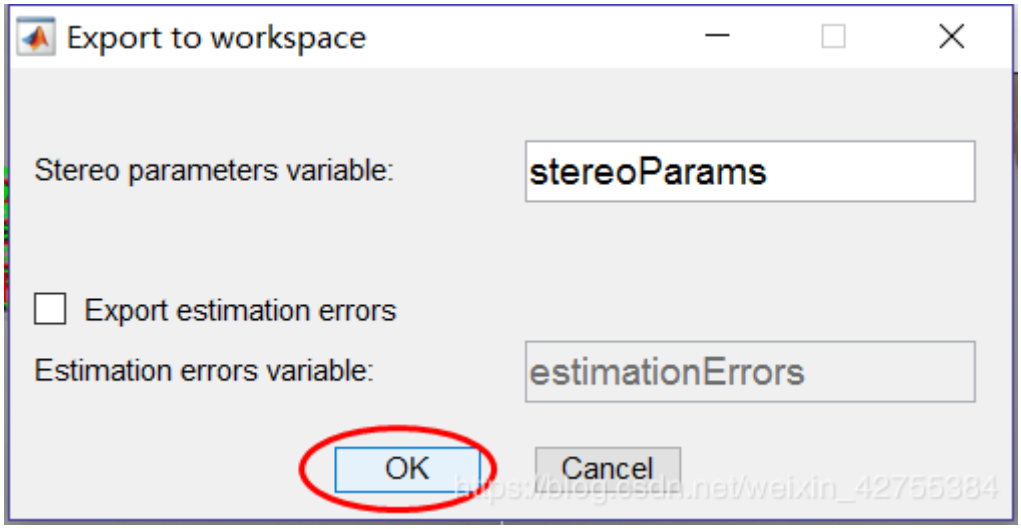


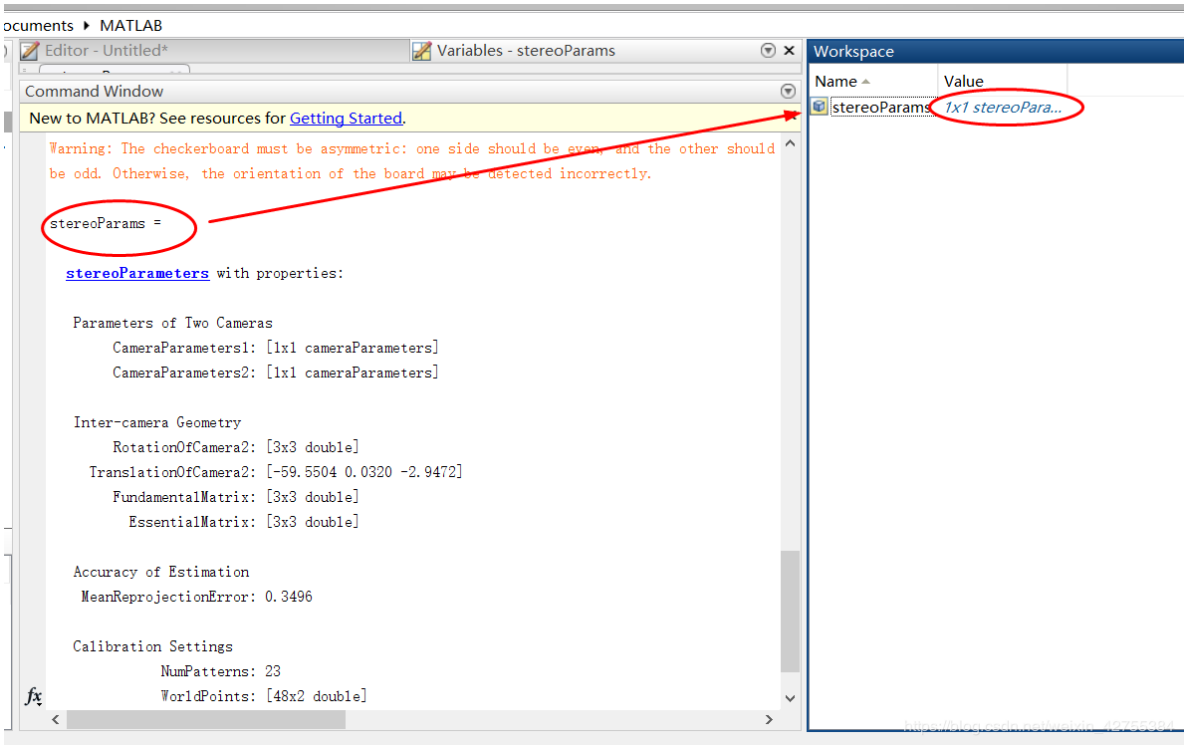

Add Images
Radial Distortion:
Compute:
2 Coefficients
3 Coefficients
Skew
Tangential Distortion
Optimization Options
Calibrate
Zoom In
Zoom Out
Pan
Default Layout
Export Camera Parameters
OPTIONS
OPTIMIZATION
CALIBRATE
ZOOM
LAYOUT
EXPORT
Image
Load Stereo Images
Folder for images from camera 1:
d:\Documents\MATLAB
Browse...
Folder for images from camera 2:
d:\Documents\MATLAB
Browse...
Size of checkerboard square:
25?
mm
OK
Cancel
Extrinsics

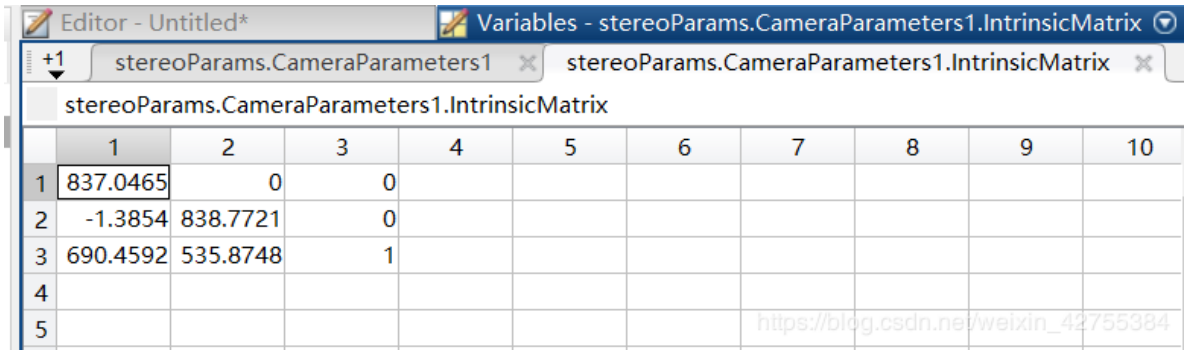

Stereo Camera Calibrator - Image
CALIBRATION
New Session
Open Session
Save Session
Add Images
Radial Distortion:
Compute:
2 Coefficients
3 Coefficients
Skew
Tangential Distortion
Optimization Options
Calibrate
Zoom In
Zoom Out
Pan
Default Layout
Export Camera Parameters
Help
FILE
OPTIONS
OPTIMIZATION
CALIBRATE
ZOOM
LAYOUT
EXPORT
RESOURCES
Data Browser
1: left_0.jpg & right_0.jpg
2: left_1.jpg & right_1.jpg
3: left_10.jpg & right_10.jpg
4: left_11.jpg & right_11.jpg
5: left_12.jpg & right_12.jpg
Image
(0,0)
Camera 1
Camera 2
Reprojection Errors
Extrinsics


Optimization Options
Calibrate
Zoom In
Zoom Out
Pan
Default Layout
Export Camera Parameters
Help
OPTIMIZATION
CALIBRATE
ZOOM
LAYOUT
EXPORT
RESOURCES
Calibration Progress
Calibrating camera 1...
(0,0)
X→
↓Y
Detected points
Checkerboard
https://blog.csdn.net/weixin_42755384


Default Layout
Export Camera Parameters
Help
LAYOUT
Export camera parameters
Generate MATLAB script
https://blog.csdn.net/weixin_42755384


Export to workspace
Stereo parameters variable:
stereoParams
Export estimation errors
Estimation errors variable:
estimationErrors
OK
Cancel






```python
# filename: camera_configs.py
import cv2
import numpy as np

left_camera_matrix = np.array([[3213.08, 0., 600.64723],
                               [0.,3060.75, 186.58058],
                               [0., 0., 1.]])
left_distortion = np.array([[-0.1426, -0.4962, -0.0150, -0.0744, 0.00000]])

right_camera_matrix = np.array([[1004.37, 0., 400.00856],
                                [0., 1004.52, 269.37140],
                                [0., 0., 1.]])
right_distortion = np.array([[-0.5450, -0.2870, -0.0110, -0.0329, 0.00000]])

R = np.array(([0.9999,-0.0019,0.0101],# Rotation vector
              [0.0021,0.9999,-0.0138],
              [-0.0100,0.0138,0.9999]))
T = np.array([-150.5252,-4.888,11.9268])  # transformation vector

size = (1280,760)  # image size
```

finally we can get the parameter like this.

3.1 The feature track algorithm

load the camera configurates, find the center of red object ,through x1,y1,x2,y2 get the world coordinates.

```
from collections import  deque
import numpy as np
import xyz_calculate
#import imutils
import cv2
import time
def truncate(self, n, decimals=0):
    n = float(n)
    multiplier = 10 ** decimals
    return int(n * multiplier) / multiplier
#Set the red threshold in rad space
redLower = np.array([170, 100, 100])
redUpper = np.array([179, 255, 255])
#Initializes the trace list
mybuffer = 64
pts = deque(maxlen=mybuffer)
#open camera（you need two camera connected first）
video1="http://admin:admin@192.168.137.13:8081/"
video2="http://admin:admin@192.168.137.21:8081/"

camera1 = cv2.VideoCapture(video1)
camera2=cv2.VideoCapture(video2)
#wait for a minute
time.sleep(1)
# Go through each frame and check the red cap
while True:
    #read a frame in one loop
    (ret, frame) = camera1.read()
    (ret2,frame2)=camera2.read()
    #judge if both camera are opened,if so, go on
    if not ret and ret2:
        print ('No Camera')
        break
    #frame = imutils.resize(frame, width=600)
    #transform the picture from rgb space to hsv space
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    hsv2=cv2.cvtColor(frame2,cv2.COLOR_BGR2HSV)

    #The mask is constructed according to the threshold value
    mask = cv2.inRange(hsv, redLower, redUpper)
    #Corrosion operation
    mask2=cv2.inRange(hsv2,redLower,redUpper)
    mask = cv2.erode(mask, None, iterations=2)

    mask2=cv2.erode(mask2,None,iterations=2)
    #The purpose of the expansion operation is to remove the noise
    mask = cv2.dilate(mask, None, iterations=2)

    mask2=cv2.dilate(mask2,None,iterations=2)
    #contour detection
    cnts = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)[-2]
```

```
 cnts2=cv2.findContours(mask.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
[-2]
    #Initializes the center of mass of the cap circle
    center = None
    #If there is an outline
    if len(cnts) and len(cnts)==len(cnts2) :
        #Find the contour with the largest area
        c = max(cnts, key = cv2.contourArea)
        c2=max(cnts2, key = cv2.contourArea)
        #Determine the circumferential circle of the contour with the largest
area
        ((x, y), radius) = cv2.minEnclosingCircle(c)
        ((x2,y2),radius2)=cv2.minEnclosingCircle(c2)
        #Calculate the moment of the contour
        M = cv2.moments(c)
        M2=cv2.moments(c2)
        #Calculate the center of mass
        center = (int(M["m10"]/M["m00"]), int(M["m01"]/M["m00"]))
        center2= (int(M["m10"]/M["m00"]), int(M["m01"]/M["m00"]))
        #The drawing is performed only if the radius is greater than 10
        if radius > 10:
            cv2.circle(frame, (int(x), int(y)), int(radius), (0, 255, 255), 2)
            cv2.circle(frame, center, 5, (0, 0, 255), -1)
            xw,yw,zw=xyz_calculate.calculate_world_xyz(x,y,x2,y2)
            cv2.putText(frame, "X,Y,Z:
"+str(int(xw))+","+str(int(yw))+","+str(int(zw))+"!",
(int(x),int(y)),cv2.FONT_HERSHEY_SIMPLEX, 1, (0,255,0),2)
            #Add the center of mass to the PTS, and to the left of the list
            pts.appendleft(center)
    #Traverse the tracking points and plot the tracks in sections
    for i in range(1, len(pts)):
        if pts[i - 1] is None or pts[i] is None:
            continue
        #Calculate the thickness of the small line segment
        thickness = int(np.sqrt(mybuffer / float(i + 1)) * 2.5)
        #draw a little line segment
        cv2.line(frame, pts[i - 1], pts[i], (0, 0, 255), thickness)

    #res = cv2.bitwise_and(frame, frame, mask=mask)
    cv2.imshow('Frame', frame)
    #Keyboard detection, esc key exit detected
    k = cv2.waitKey(5)&0xFF
    if k == 27:
        break
#release camera process
camera1.release()
#close all the windows
cv2.destroyAllWindows()
```

and the coordinates function:

```
import camera_configs
```

```
import numpy as np
#cam=camera_configs.right_camera_matrix
# r=camera_configs.R
# t=camera_configs.T
cam=np.array([[988.4811,-4.6932,600.3510],[0,989.7077,360.3455],[0,0,1]])
r=np.array([[0.9990,-0.0096,0.0439],[-0.0105,0.9998,-0.0196],
[-0.0437,0.0201,0.9988]])
t=np.array([[-150],[-4.4888],[11.9268]])



def calculate_world_xyz(x1,y1,x2,y2):

    x1=x1-cam[0][2]
    y1=y1-cam[1][2]
    x2=x2-cam[0][2]
    y2=y2-cam[1][2]

    z=cam[1][1]*(cam[1][1]*t[0]-x2*t[2])/\
      (x2*(r[2][0]*x1+r[2][1]*y1+cam[1][1]*r[2][2])-cam[1][1]*
       (r[0][0]*x1+r[0][1]*y1+cam[1][1]*r[0][2]))
    y=z*y1/cam[1][1]
    x=z*x1/cam[0][0]

    return (x,y,z)
```

3.2Using disparity(another way,may be can combine with the first way):

```
import numpy as np
import cv2
import camera_configs

cv2.namedWindow("left")
cv2.namedWindow("right")
cv2.namedWindow("depth")
cv2.moveWindow("left", 0, 0)
cv2.moveWindow("right", 600, 0)
cv2.createTrackbar("num", "depth", 0, 10, lambda x: None)
cv2.createTrackbar("blockSize", "depth", 5, 255, lambda x: None)

# Add a click event to print the distance to the current point
def callbackFunc(e, x, y, f, p):
    if e == cv2.EVENT_LBUTTONDOWN:
        print(x, y)
        print(threeD[y][x])


cv2.setMouseCallback("depth", callbackFunc, None)

while True:
    # ret1, frame1 = camera1.read()
    # ret2, frame2 = camera2.read()
    # if not ret1 or not ret2:
```

```
    #     break

    frame1 = cv2.imread("./img/01.bmp")
    frame2 = cv2.imread("./img/02.bmp")

    # Reconstruct the image according to the correct map
    img1_rectified = cv2.remap(frame1, camera_configs.left_map1,
camera_configs.left_map2, cv2.INTER_LINEAR)
    img2_rectified = cv2.remap(frame2, camera_configs.right_map1,
camera_configs.right_map2, cv2.INTER_LINEAR)

    # The image is set as grayscale for StereoBM
    imgL = cv2.cvtColor(img1_rectified, cv2.COLOR_BGR2GRAY)
    imgR = cv2.cvtColor(img2_rectified, cv2.COLOR_BGR2GRAY)
    # Two trackbars are used to adjust different parameters to see the effect
    num = cv2.getTrackbarPos("num", "depth")
    blockSize = cv2.getTrackbarPos("blockSize", "depth")
    if blockSize % 2 == 0:
        blockSize += 1
    if blockSize < 5:
        blockSize = 5

    # Generate the difference graph according to the Block Maching method
    stereo = cv2.StereoBM_create(numDisparities=16 * num, blockSize=blockSize)
    # stereo = cv2.StereoSGBM_create(numDisparities=16 * num,
blockSize=blockSize)
    disparity = stereo.compute(imgL, imgR)

    disp = cv2.normalize(disparity, disparity, alpha=0, beta=255,
norm_type=cv2.NORM_MINMAX, dtype=cv2.CV_8U)
    #the threeD is the coordinates space of the whole image, you can click the
image to get every points image
    threeD = cv2.reprojectImageTo3D(disparity.astype(np.float32) / 16.,
camera_configs.Q)
    cv2.imshow("depth", disp)

    key = cv2.waitKey(1)
    if key == ord("q"):
        break
    elif key == ord("s"):
        cv2.imwrite("./snapshot/BM_left.jpg", imgL)
        cv2.imwrite("./snapshot/BM_right.jpg", imgR)
        cv2.imwrite("./snapshot/BM_depth.jpg", disp)

cv2.destroyAllWindows()
```

Documents

color track:

https://blog.csdn.net/sinat_31135199/article/details/51252393

http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html#converting-colorspaces

monocular vision:

https://www.fdxlabs.com/calculate-x-y-z-real-world-coordinates-from-a-single-camera-using-opencv/

underwater *binocular vision* (possibly can work for your next experiment):

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6211033/

github code:

https://github.com/Cuirongcheng/Image-matching-and-ranging-based-on-binocular-stereo-vision

https://github.com/siwufeiwu/Binocular-vision-ranging