6. AR vision

6. AR vision
6.1. Overview
6.2. How to use
6.3. Source code analysis
6.3.1. Algorithm principle
6.3.2. core code

Feature pack: ~/yahboomcar_ws/src/yahboomcar_visual

This section can be run on a virtual machine or on the mainboard of the car. Let's take jetson as an example.

6.1. Overview

Augmented Reality, referred to as "AR", is a technology that ingeniously integrates virtual information with the real world. It widely uses multimedia, 3D modeling, real-time tracking and registration, intelligent interaction, sensing and other technologies. The method is to simulate and simulate virtual information such as text, images, three-dimensional models, music, and videos generated by the computer, and then apply it to the real world. The two kinds of information complement each other, thereby realizing the "enhancement" of the real world.

The AR system has three prominent features: (1) the information integration of the real world and the virtual world; (2) real-time interactivity; (3) the addition of positioning virtual objects in the three-dimensional scale space.

Augmented reality technology includes multimedia, three-dimensional modeling, real-time video display and control, multi-sensor fusion, real-time tracking and registration, scene fusion and other new technologies and new means.

6.2. How to use

When using the AR case, you must have the internal parameters of the camera, otherwise it will not work. The internal parameter file is in the same directory as the code (under the AR folder of the function package); different cameras correspond to different internal parameters.

The internal parameter calibration can be quickly calibrated with a checkerboard. The specific method can be seen in the lesson [Astra Camera Calibration].

```
#Raspberry Pi 5 master needs to enter docker first, please perform this step
#If running the script into docker fails, please refer to ROS/07, Docker tutorial
~/run_docker.sh
```

Start the monocular camera

```
roslaunch usb_cam usb_cam-test.launch
```

<PI5 needs to open another terminal and enter the same docker container

1. In the above steps, a docker container has been opened. You can open another terminal on the host (car) to view:

```
docker ps -a
```

```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND       CREATED      STATUS       PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9    "/bin/bash"   3 days ago   Up 9 hours             ecstatic_lewin
jetson@ubuntu:~$
```

2. Now enter the docker container in the newly opened terminal:

```
docker exec -it 5b698ea10535 /bin/bash
```

```
jetson@ubuntu:~$ docker ps -a
CONTAINER ID   IMAGE                                COMMAND       CREATED      STATUS       PORTS     NAMES
5b698ea10535   yahboomtechnology/ros-foxy:3.3.9    "/bin/bash"   3 days ago   Up 9 hours             ecstatic_lewin
jetson@ubuntu:~$ docker exec -it 5b698ea10535 /bin/bash
---------------------------------------------------------
my_robot_type: x3 | my_lidar: a1 | my_camera: astrapro
---------------------------------------------------------
root@ubuntu:/#
```

After successfully entering the container, you can open countless terminals to enter the container.

Start the calibration node

```
rosrun camera_calibration cameracalibrator.py image:=/usb_cam/image_raw
camera:=/usb_cam --size 9x6 --square 0.02
```

After calibration, move the [calibrationdata.tar.gz] file to the [home] directory.

```
sudo mv /tmp/calibrationdata.tar.gz ~
```

After decompression, open [ost.yaml] in the folder, find the camera internal parameter matrix and distortion coefficient and modify it to the corresponding location of the [astra.yaml] file, just modify the contents of two [data]. For example: the following.

```
camera_matrix :  !! opencv - matrix
  rows :  3
  cols :  3
  dt :  d
  data : [ 615.50506 ,    0.       ,  365.84388 ,
           0. , 623.69024 , 238.778 ,
           0. , 0. , 1. ]
distortion_model :  plumb_bob
distortion_coefficients :  !! opencv - matrix
  rows :  1
  cols :  5
  dt :  d
  data : [ 0.166417 ,  - 0.160106 ,  - 0.008776 ,  0.025459 ,  0.000000 ]
```

A total of 12 effects.

```
[ "Triangle" ,  "Rectangle" ,  "Parallelogram" , "WindMill" , "TableTennisTable"
,  "Ball" ,  "Arrow" ,  "Knife" ,  "Desk" ,
"Bench" ,  "Stickman" ,  "ParallelBars" ]
```

start command

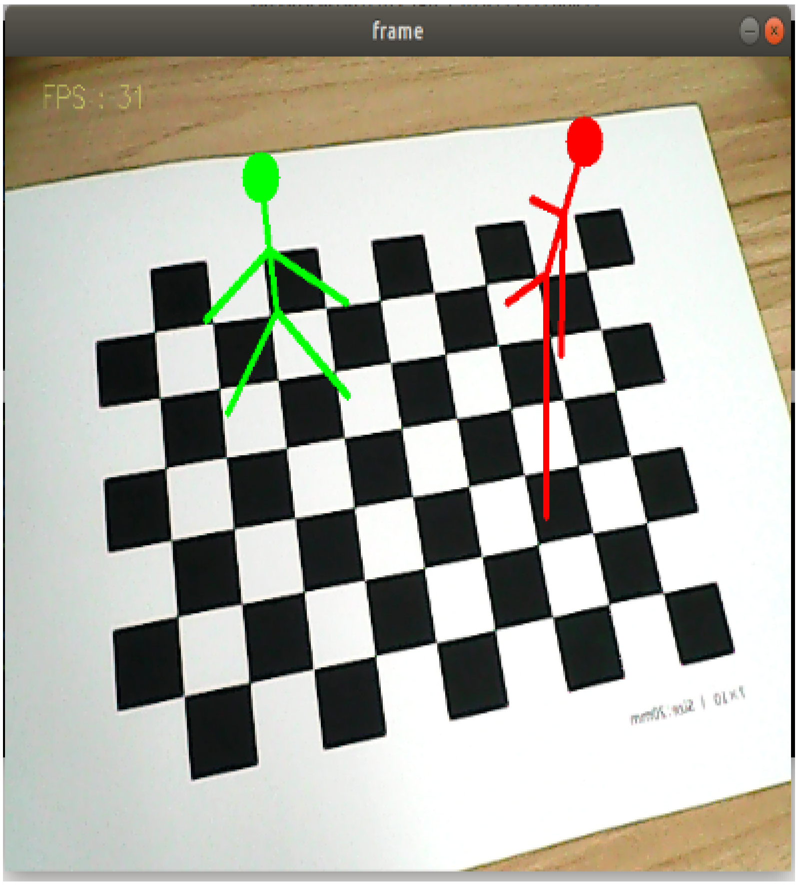
```
roslaunch yahboomcar_visual simple_AR.launch display:=true flip:=false
```

- display parameter: True; the image window is displayed locally; Fasle, it is not displayed.
- flip parameter: whether to switch the screen horizontally, the default is OK.

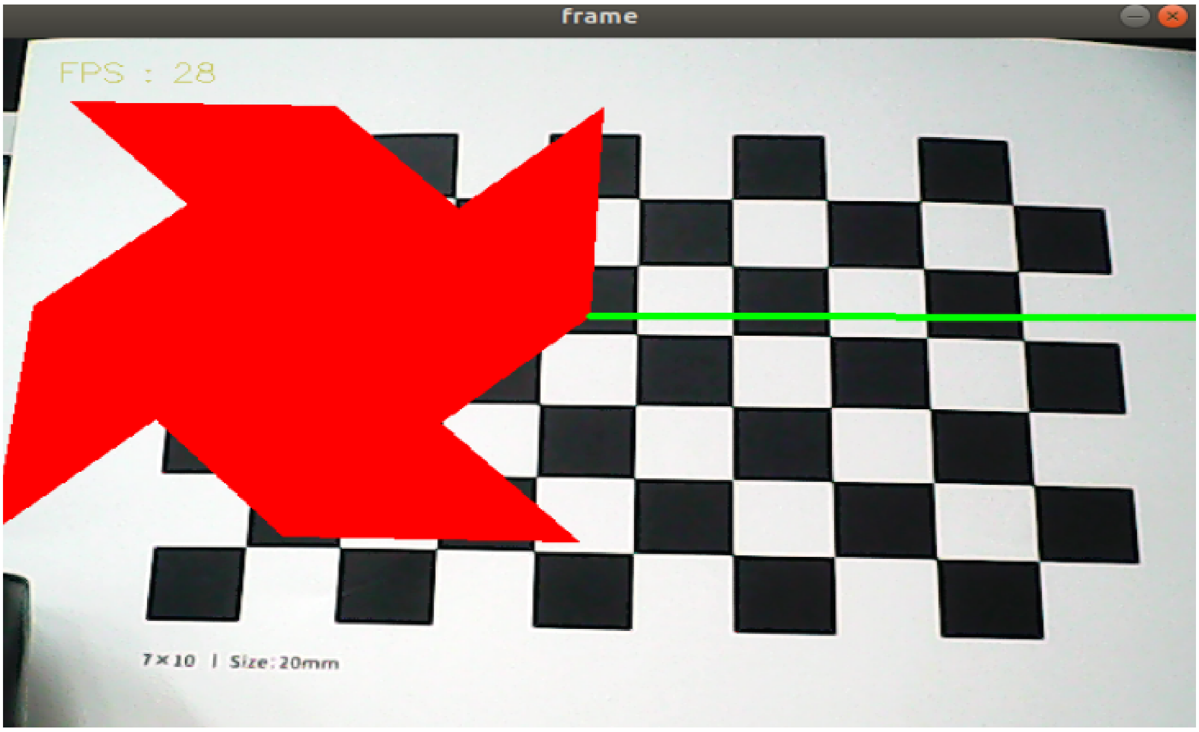
Set parameters according to your needs, you can also directly modify the launch file, and you don't need to attach parameters when starting. When the screen is not turned on, you can use the network monitoring method to view

```
Open the IP of the device: 8080
```

1. When the screen is displayed (that is, display is true), press the [q] key to exit, and the [f] key to switch between different effects. You can also use the command line to switch.

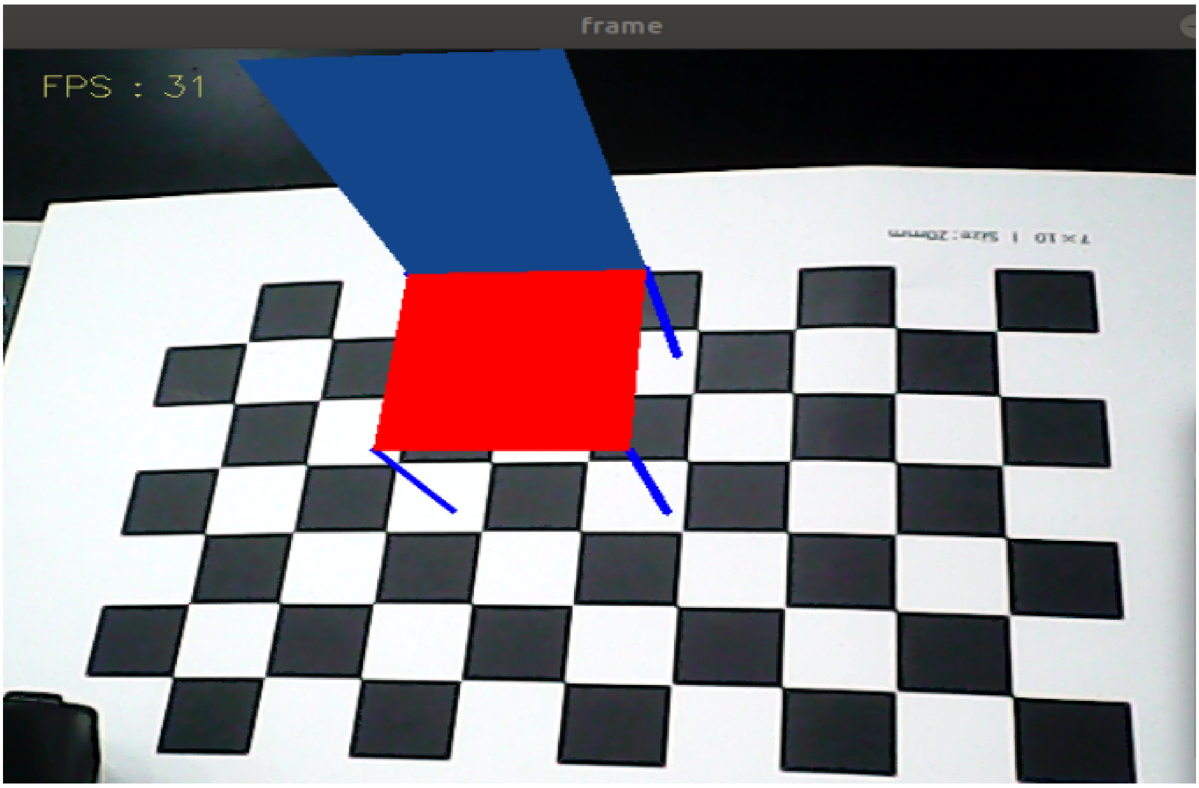


Use the [f] or [F] key to switch between different effects.



2. When the screen is not displayed (ie display is false), the effect can only be switched through the command line

```
jetson@yahboom: ~ 80x4
------------------------
jetson@yahboom:~$ rostopic pub /Graphics_topic std_msgs/String "data: 'Bench'"
publishing and latching message. Press ctrl-C to terminate
```

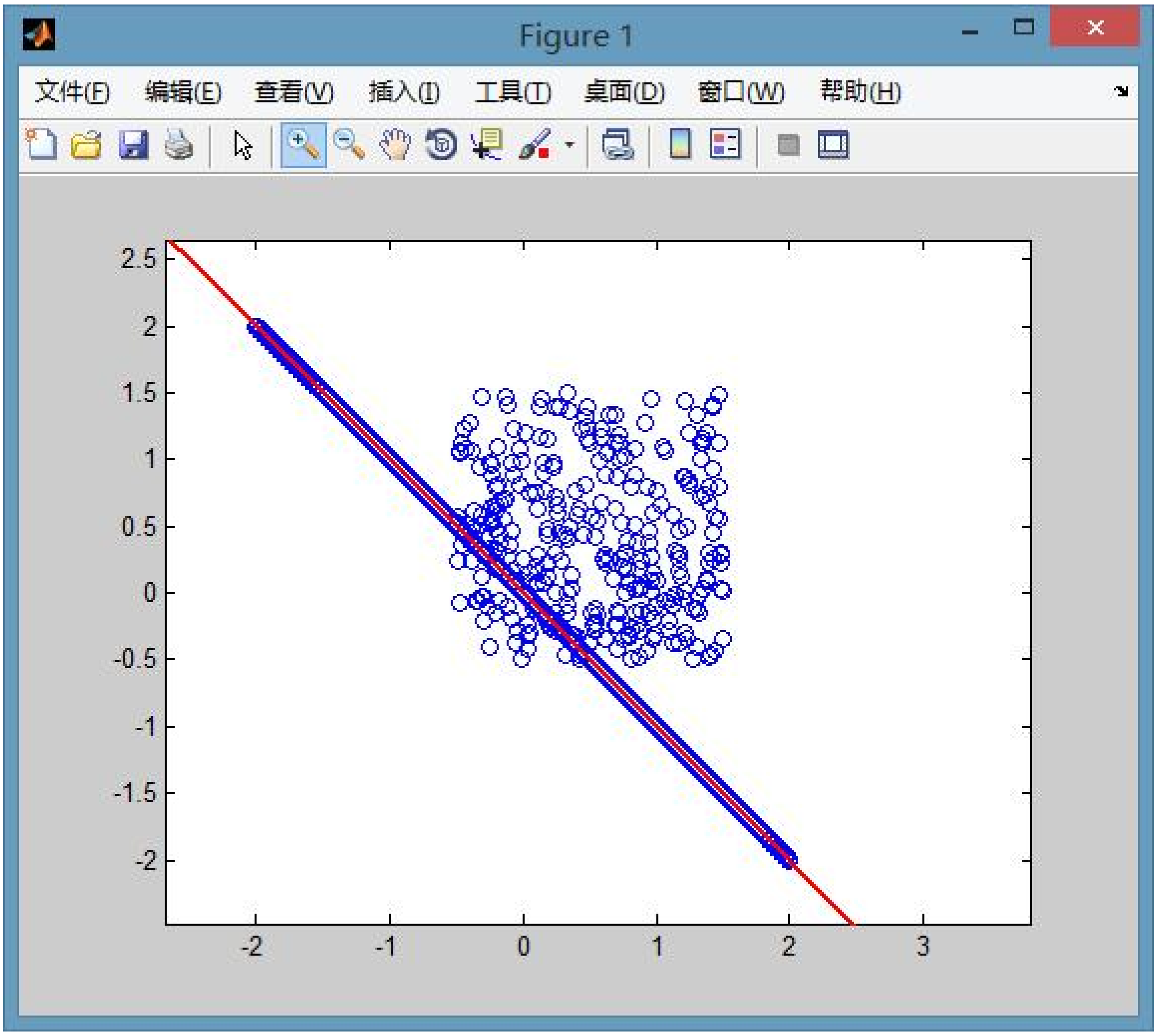


6.3. Source code analysis

6.3.1. Algorithm principle

Find object poses from 3D-2D point correspondences using the RANSAC scheme.

The RanSaC algorithm (Random Sampling Consistency) was originally a classic algorithm for data processing. Its function is to extract specific components in objects in the presence of a large amount of noise. The following figure is an illustration of the effect of the RanSaC algorithm. There are some points in the figure that obviously satisfy a straight line, and another group of points is pure noise. The goal is to find the equation of the line in the presence of a lot of noise, where the amount of noisy data is 3 times that of the line.

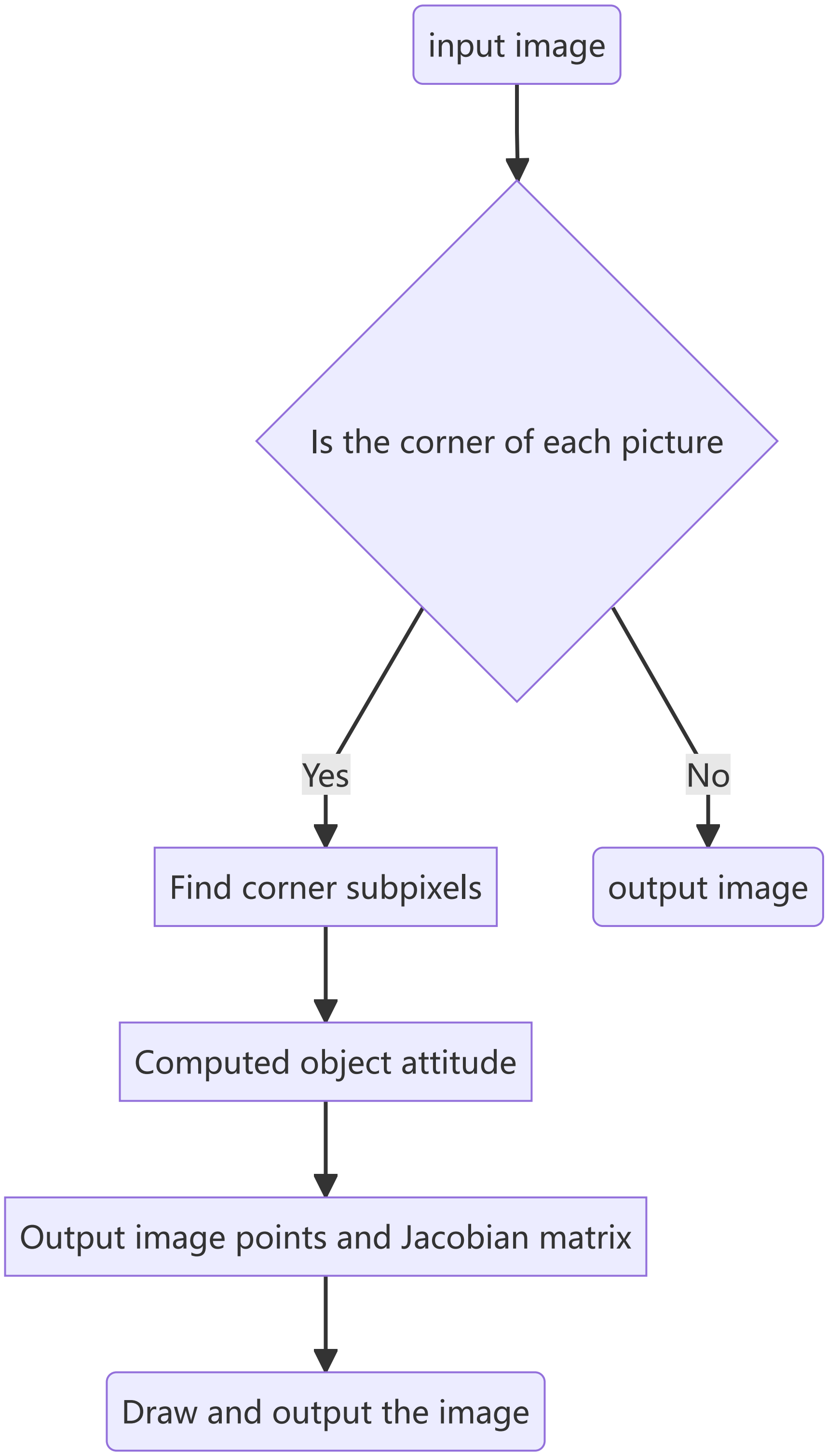


If this effect cannot be obtained by the least squares method, the straight line will be a little higher than the straight line in the figure.

The basic assumptions of RANSAC are: (1) The data consists of "inside points", for example: the distribution of the data can be explained by some model parameters; (2) "outlier points" are data that cannot fit the model; (3) Except Data other than this is noise. The causes of outliers are: extreme values of noise; wrong measurement methods; wrong assumptions about the data. RANSAC also makes the following assumptions: given a set of (usually small) intra-office points, there is a process by which the model parameters can be estimated; and the model can explain or apply to intra-office points.

6.3.2. core code

Design Flow:



launch file

```
< launch >
    < arg  name = "flip"  default = "False" />
    < arg  name = "display"  default = "False" />
    < node  name = "simple_AR"  pkg = "yahboomcar_visual"  type = "simple_AR.py"
output = "screen"  args = "$(arg display)" >
        < param  name = "flip"  type = "bool"  value = "$(arg flip)" />
        < remap  from = "/simpleAR/camera"  to = "/simpleAR/camera" />
    </ node >
    <!-- web_video_server -->
    < node  pkg = "web_video_server"  type = "web_video_server"  name =
"web_video_server"  output = "screen" />
</ launch >
```

python main function

```
    def  process ( self ,  img ):
        if  self . flip  ==  'True' :  img  =  cv . flip ( img ,  1 )
        gray  =  cv . cvtColor ( img ,  cv . COLOR_BGR2GRAY )
        # Find the corners of each image
        retval ,  corners  =  cv . findChessboardCorners (
            gray ,  self . patternSize ,  None ,
            flags = cv . CALIB_CB_ADAPTIVE_THRESH  +  cv .
CALIB_CB_NORMALIZE_IMAGE  +  cv . CALIB_CB_FAST_CHECK )
        # Find corner subpixels
        if  retval :
            corners  =  cv . cornerSubPix (
                gray ,  corners , ( 11 ,  11 ), ( - 1 ,  - 1 ),
                ( cv . TERM_CRITERIA_EPS  +  cv . TERM_CRITERIA_MAX_ITER ,  30 ,
 0.001 ))
            # Calculate object pose solvePnPRansac
            retval ,  rvec ,  tvec ,  inliers  =  cv . solvePnPRansac (
                self . objectPoints ,  corners ,  self . cameraMatrix ,  self .
distCoeffs )
            # output image points and jacobian matrix
            image_Points ,  jacobian  =  cv . projectPoints (
                self . __axis ,  rvec ,  tvec ,  self . cameraMatrix ,  self .
distCoeffs , )
            # draw the image
            img  =  self . draw ( img ,  corners ,  image_Points )
        return  img
```

key function

https://docs.opencv.org/3.0-alpha/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html

- findChessboardCorners()

```
def  findChessboardCorners ( image ,  patternSize ,  corners = None ,  flags =
None ):
    '''
Find image corners
:param image: Input the original checkerboard image. The image must be an 8-bit
grayscale or colormap.
```

```
:param patternSize: (w,h), the number of interior corners of each row and column
on the board.  w = the number of black and white blocks on a row of the board -
1, h = the number of black and white blocks on a column of the board - 1.
For example: 10x6 chessboard, then (w,h)=(9,5)
:param corners: array, output array of detected corners.
:param flags: int, different operation flags, can be 0 or a combination of the
following values:
CALIB_CB_ADAPTIVE_THRESH Converts the image to black and white using adaptive
thresholding instead of using a fixed threshold.
    CALIB_CB_NORMALIZE_IMAGE Use histogram equalization to equalize the image
before binarizing it with fixed threshold or adaptive thresholding.
    CALIB_CB_FILTER_QUADS uses additional criteria (such as contour area,
perimeter, square shape) to filter out false quads extracted during the contour
retrieval stage.
    CALIB_CB_FAST_CHECK Runs a quick check mechanism on the image to find the
corners of the board, and returns a quick reminder if no corners are found.
    Calls in degenerate conditions can be greatly accelerated when the
checkerboard is not observed.
    :return: retval, corners
'''
    pass
```

- cornerSubPix()

We need to use cornerSubPix() to further optimize the detected corners, so that the accuracy of the corners can reach sub-pixel level.

```
def  cornerSubPix ( image ,  corners ,  winSize ,  zeroZone ,  criteria ):
    '''
Subpixel corner detection function
:param image: input image
:param corners: pixel corners (both as input and output)
:param winSize: area size is NXN; N=(winSize*2+1)
:param zeroZone: Similar to winSize, but always has a smaller range, Size(-1,-1)
means ignore
:param criteria: criteria to stop optimization
:return: subpixel corner
'''
    pass
```

- solvePnPRansac()

```
def  solvePnPRansac ( objectPoints ,  imagePoints ,  cameraMatrix ,  distCoeffs ,

                   rvec = None ,  tvec = None ,  useExtrinsicGuess = None ,
 iterationsCount = None ,
                   reprojectionError = None ,  confidence = None ,  inliers =
None ,  flags = None ):
    '''
    Calculate object pose
:param objectPoints: list of object points
:param imagePoints: list of corner points
:param cameraMatrix: camera matrix
:param distCoeffs: Distortion coefficients
    :param rvec:
```

```
    :param tvec:
    :param useExtrinsicGuess:
    :param iterationsCount:
:param reprojectionError:
    :param confidence:
    :param inliers:
    :param flags:
    :return: retval, rvec, tvec, inliers
'''
    pass
```

Find object poses from 3D-2D point correspondences using the RANSAC scheme. This function estimates the object pose given a set of object points, their corresponding image projections, and the camera matrix and distortion coefficients. This function finds a pose that minimizes the re-projection error, the re-observation error, that is, the observed pixel point projection imagePoints and the object projection (`projectPoints ()`) sum of squared distances between objectPoints. The use of RANSAC can avoid the effect of outliers on the results.

- projectPoints()

```
def  projectPoints ( objectPoints ,  rvec ,  tvec ,  cameraMatrix ,  distCoeffs ,
 imagePoints = None ,  jacobian = None ,  aspectRatio = None ):
    '''
Output image points and Jacobian matrix
    :param objectPoints:
:param rvec: rotation vector
:param tvec: translation vector
:param cameraMatrix: camera matrix
:param distCoeffs: Distortion coefficients
    :param imagePoints:
    :param jacobian:
param aspectRatio:
    :return: imagePoints, jacobian
'''
    pass
```