33. AR vision

33.1. Overview

Augmented Reality, referred to as "AR", is a technology that cleverly integrates virtual information with the real world. It widely uses multimedia, three-dimensional modeling, real-time tracking and registration, intelligent interaction, sensing and other technologies. It simulates computer-generated text, images, three-dimensional models, music, videos and other virtual information and then applies it to the real world. The two types of information complement each other, thereby achieving "enhancement" of the real world.

The AR system has three outstanding features:

①Integration of information between the real world and the virtual world;

②Has real-time interactivity;

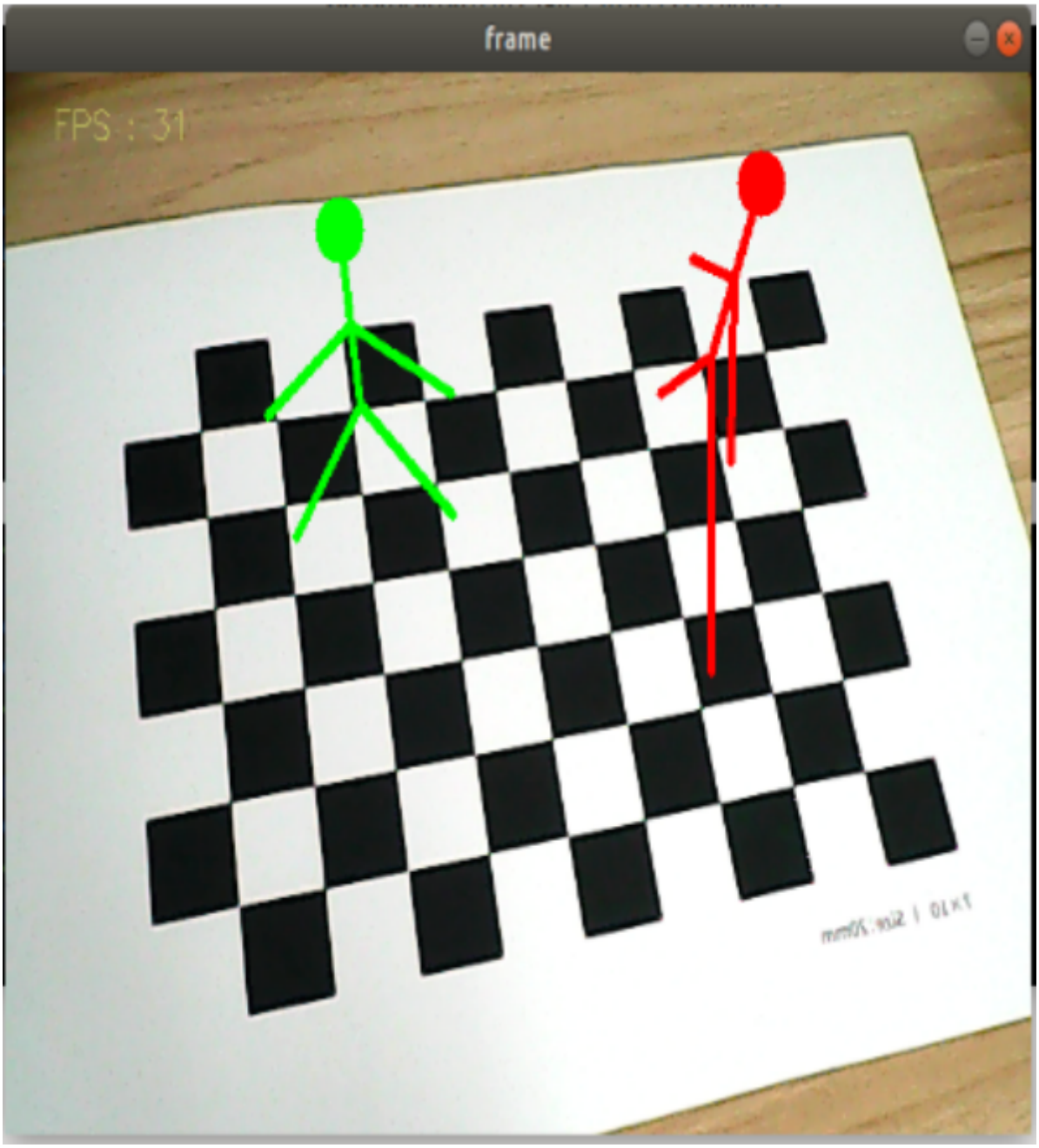
③ is to add positioning virtual objects in the three-dimensional scale space.

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

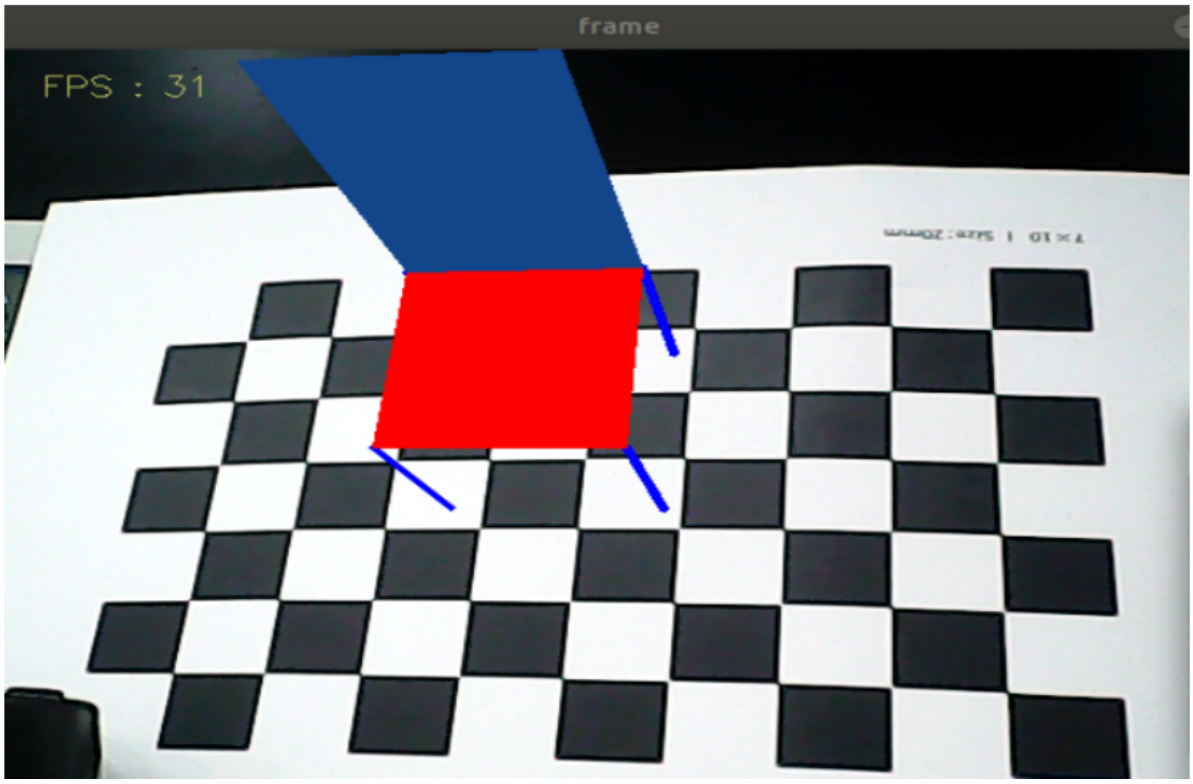
33.2. Start

Terminal input,

```
roslaunch astra_visual astra_AR.launch VideoSwitch:=True
```



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

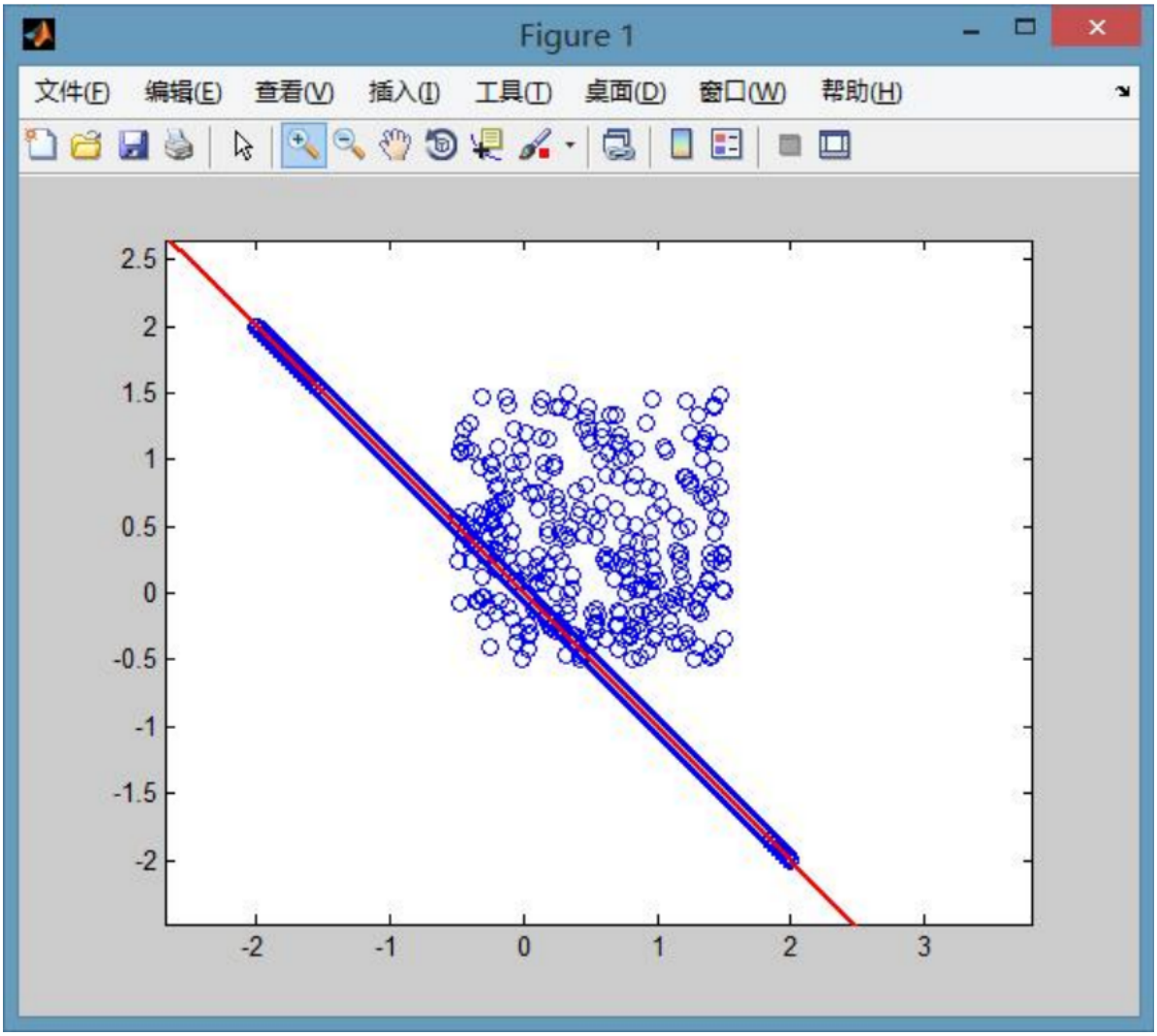


33.3. Source code analysis

33.3.1. Algorithm principle

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

The RanSaC algorithm (Random Sampling Consistent) was originally a classic algorithm used for data processing. Its function is to extract specific components in objects in the presence of a large amount of noise. The figure below is an illustration of the effect of the RanSaC algorithm. There are some points in the picture that obviously satisfy a certain straight line, and there are other points that are pure noise. The purpose is to find the equation of the straight line in the presence of a large amount of noise, when the amount of noise data is 3 times that of the straight line.



If the least squares method is used, such an effect cannot be obtained, and the straight line will be slightly higher than the straight line in the picture.

The basic assumptions of RANSAC are:

(1) The data consists of "internal points". For example, the distribution of data can be explained by some model parameters;

(2) "Outlier points" are data that cannot fit the model;

(3) Other data are noise.

The causes of outliers include: extreme values of noise; wrong measurement methods; wrong assumptions about the data.

RANSAC also makes the following assumption: given a set of (usually small) internal points, there is a process that can estimate the parameters of the model; and the model can explain or be applicable to the internal points.

33.3.2. Core code

Code location:/home/yahboom/orbbec_ws/src/astra_visual/AR

```
def process(self, img, action):
        if action == ord('f') or action == ord('F'):
            self.index += 1
            if self.index >= len(self.graphics): self.index = 0
            self.Graphics = self.graphics[self.index]
```

```
        if self.flip: img = cv.flip(img, 1)
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        # Find the corner points 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 sub-pixels
        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, )
            img = self.draw(img, corners, image_Points)
        return img
```

key function

- findChessboardCorners()

```
def findChessboardCorners(image, patternSize, corners=None, flags=None):
'''
Find image corners
:param image: Input the original chessboard image. The image must be an 8-
bit grayscale or color image.
:param patternSize: (w,h), the number of interior corners in each row and
column on the chessboard. w = the number of black and white pieces on a row
of the chessboard
Amount-1, h=the number of black and white blocks on a column of the
chessboard-1.
For example: 10x6 chessboard, then (w,h)=(9,5)
:param corners: array, output array of detected corner points.
:param flags: int, different operation flags, which can be 0 or a
combination of the following values:
CALIB_CB_ADAPTIVE_THRESH Use adaptive thresholding to convert the image to
black and white instead of using a fixed threshold.
CALIB_CB_NORMALIZE_IMAGE Use histogram equalization before binarizing the
image using fixed or adaptive thresholding.
picture.
CALIB_CB_FILTER_QUADS Use additional criteria (such as contour area,
perimeter, square shape) to filter out the
Segment extracted false quadrilateral.
CALIB_CB_FAST_CHECK Runs a fast check mechanism on the image to find the
corners of the chessboard, returning if no corners are found
A quick reminder.
Calling under degenerate conditions can be greatly accelerated when no
chessboard is observed.
:return: retval, corners
'''
pass
```

- cornerSubPix()

```
def cornerSubPix(image, corners, winSize, zeroZone, criteria):
'''
Sub-pixel corner detection function
:param image: input image
:param corners: pixel corners (used as both input and output)
:param winSize: The 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 for stopping optimization
:return: sub-pixel corner point
'''
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: object point list
:param imagePoints: list of corner points
:param cameraMatrix: camera matrix
:param distCoeffs: distortion coefficient
:param rvec:
:param tvec:
:param useExtrinsicGuess:
:param iterationsCount:
:param reprojectionError:
:param confidence:
:param inliers:
:param flags:
:return: retval, rvec, tvec, inliers
'''
pass
```