4. AR Vision

4.1. 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 technical means to simulate computer-generated text, images, three-dimensional models, music, video and other virtual information and apply them 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 characteristics: ① Information integration of the real world and the virtual world; ② Real-time interactivity; ③ Adding and positioning virtual objects in three-dimensional space.

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

4.2. Usage

When using AR cases, the camera's internal parameters must be used, otherwise it cannot be run (the factory image has completed the camera internal parameter calibration). The internal parameter file is in the same directory as the code.

4.2.1, Camera internal parameter calibration

Start the monocular camera

```
ros2 launch orbbec_camera dabai_dcw2.launch.py
```

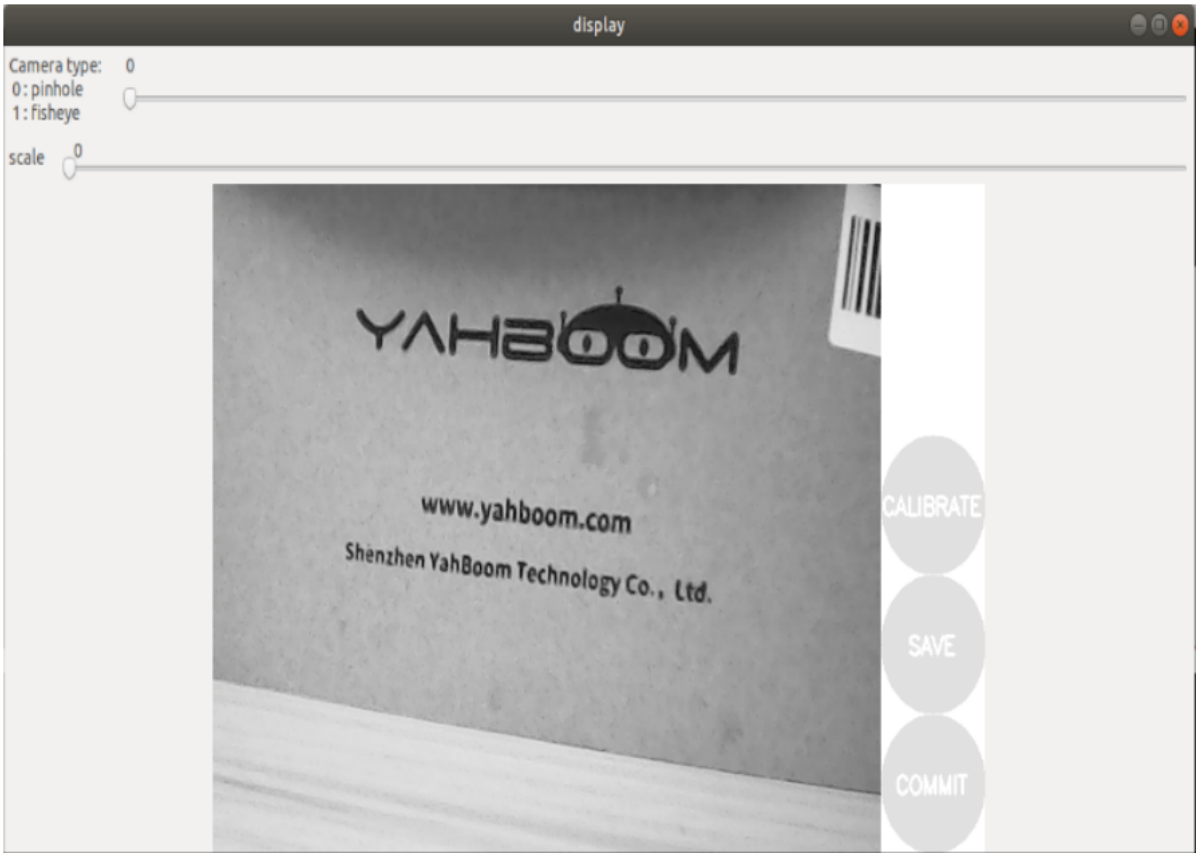
Start the calibration node

```
ros2 run camera_calibration cameracalibrator --size 9x6 --square 0.02 --ros-args
--remap /image:=/camera/color/image_raw
```

size: The number of internal corner points of the calibration chessboard, for example, 9X6, the corner points have six rows and nine columns.

square: The side length of the chessboard, in meters.

image: Image topic released by the camera



Calibration interface

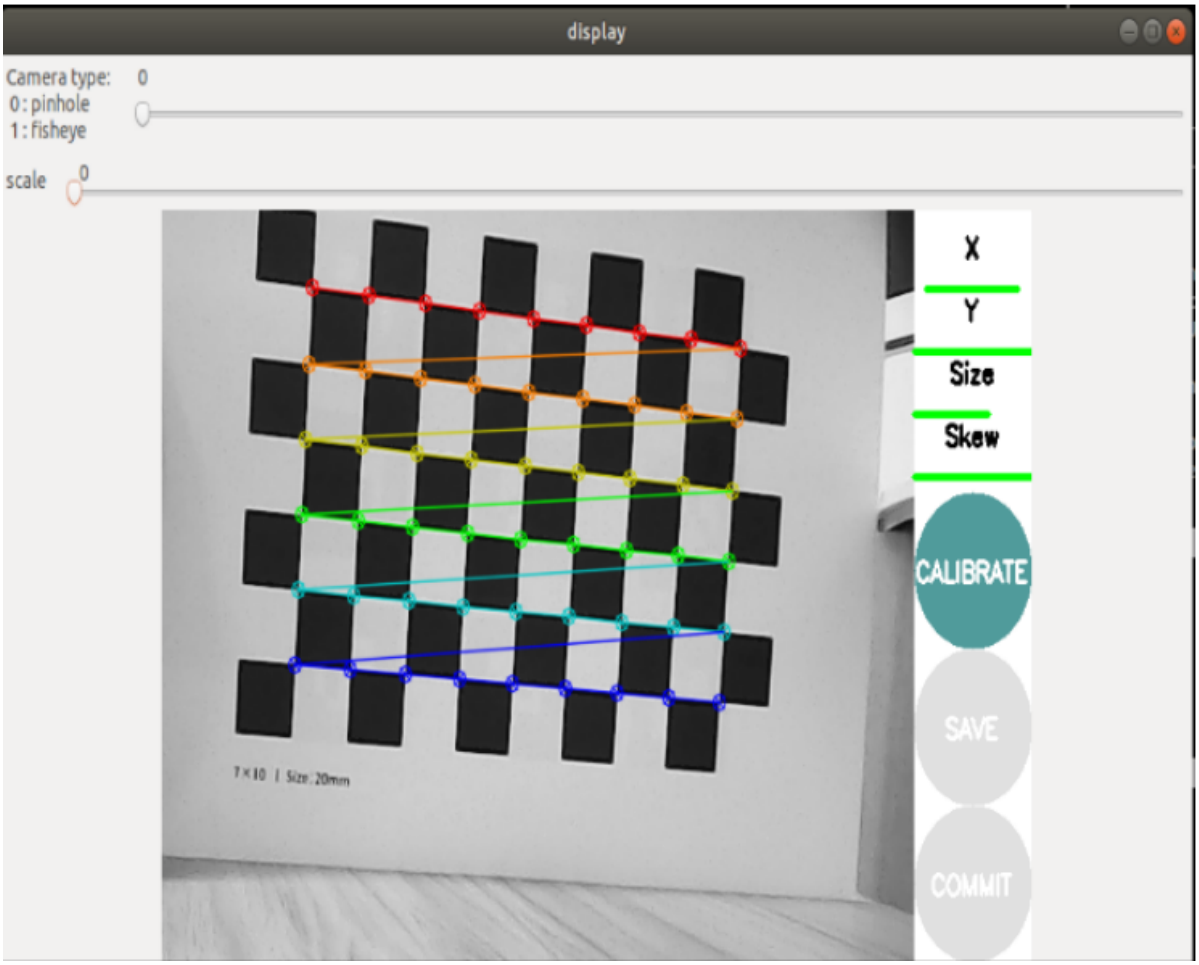
X: Left and right movement of the chessboard in the camera's field of view

Y: Up and down movement of the chessboard in the camera's field of view

Size: Front and back movement of the chessboard in the camera's field of view

Skew: Tilt and rotation of the chessboard in the camera's field of view

After successful startup, place the chessboard in the center of the screen and change different positions. The system will recognize it autonomously. The best situation is that the lines under [X], [Y], [Size], and [Skew] change from red to yellow and then to green as data is collected, and fill as much as possible.



- Click [CALIBRATE] to calculate the camera internal parameters. The more pictures there are, the longer it will take. Just wait. (Sixty or seventy pictures are enough. Too many will cause the system to get stuck easily).
- Click [SAVE] to save the calibration results to [/tmp/calibrationdata.tar.gz] of the current running terminal.

After the calibration is completed, you can move out the [/tmp/calibrationdata.tar.gz] file to check the contents.

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

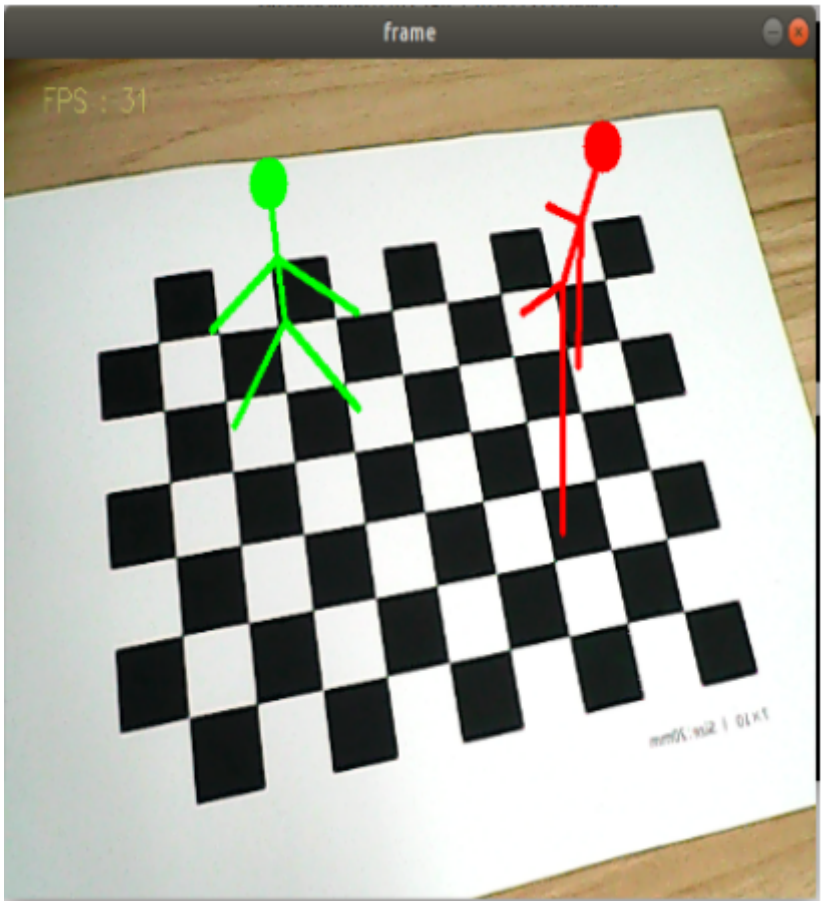
After decompression, there are the pictures just calibrated, an ost.txt file and an ost.yaml file. ost.yaml is the internal parameters of the calibrated camera. Copy the internal reference content here and overwrite it to astra.yaml under ~/dofbot_ws/src/dofbot_visual/AR.

Note: The factory image has been calibrated with parameters, and the program can be started directly. If the camera screen is started, please close the camera process before starting the program.

4.2.2, Program Startup

Terminal input,

```
ros2 run dofbot_pro_vision simple_ar
```



After the program starts, put the chessboard in front of the camera. Note that you need to see the entire chessboard, otherwise the program will exit. After the entire chessboard is recognized, the following 12 effects will be displayed,

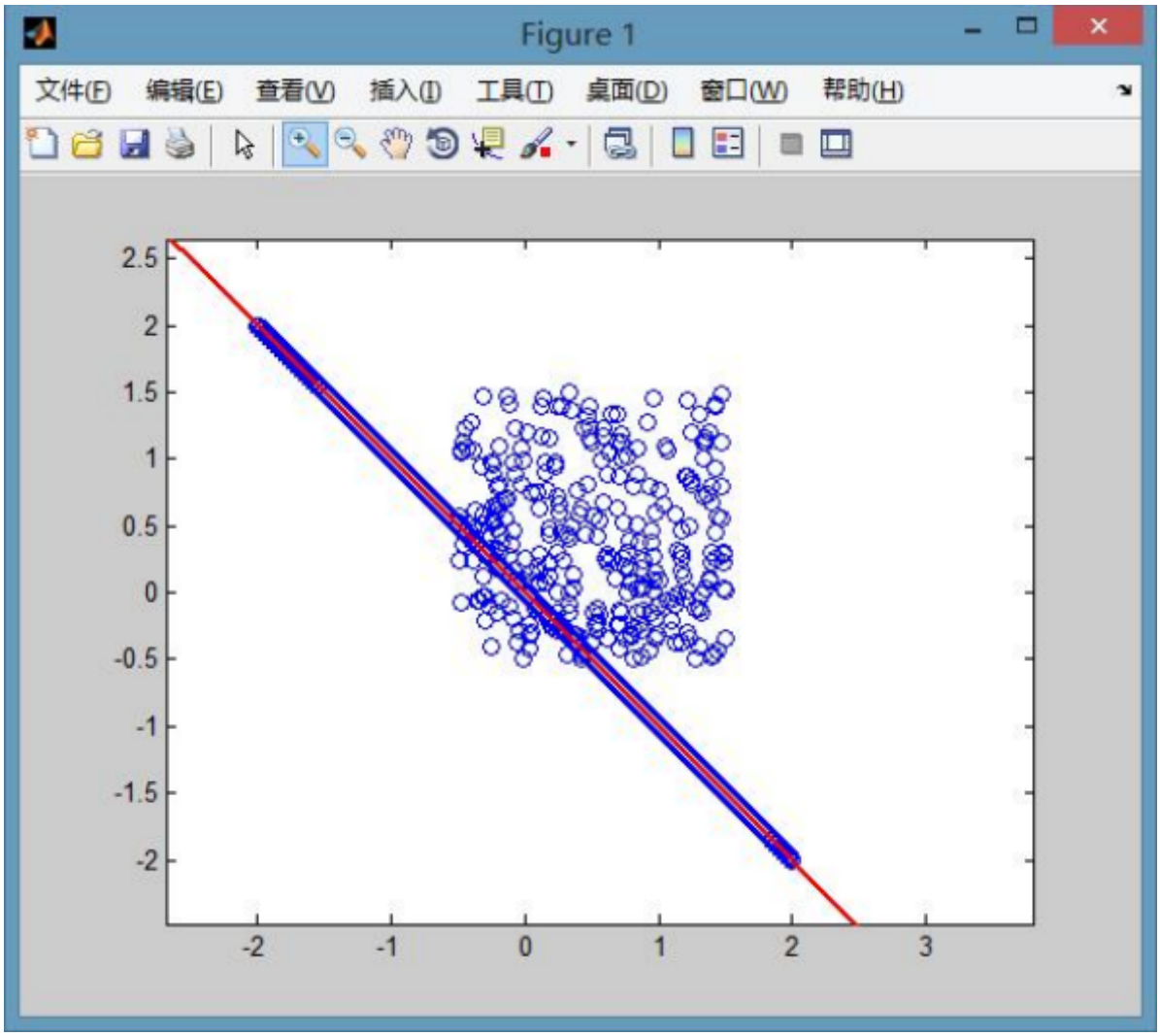
```
["Triangle", "Rectangle", "Parallelogram","WindMill","TableTennisTable",
"Ball","Arrow", "Knife", "Desk","Bench", "Stickman", "ParallelBars"]
```

Click the image and press F to switch the displayed effects.

4.3.1, RANSAC scheme

- Algorithm principle: Use the RANSAC scheme to find the object posture from 3D-2D point correspondence

The RanSaC algorithm (random sampling consistency) was originally a classic algorithm for data processing. Its function is to extract specific components in an object under a large amount of noise. The following figure is an illustration of the effect of the RanSaC algorithm. Some points in the figure clearly satisfy a certain straight line, and another group of points is pure noise. The purpose is to find the equation of the line under a large amount of noise. At this time, the amount of noise data is 3 times that of the line.



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

- The basic assumptions of RANSAC are:
- The data consists of "inside points", for example: the distribution of the data can be explained by some model parameters;
- "Outside points" are data that cannot adapt to the model;
- Data other than this is noise.
- The reasons for the generation of outliers include: extreme values of noise; incorrect measurement methods; incorrect assumptions about the data. RANSAC also makes the following assumptions: given a set of (usually small) inside points, there is a process that can estimate the model parameters; and the model can explain or apply to the inside points.

4.3.2, Source code

Source code location: ~/dofbot_pro_ws/src/dofbot_pro_vision/dofbot_pro_vision/simple_ar.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import os
import sys
import time
import rospy
import rospkg
import cv2 as cv
```

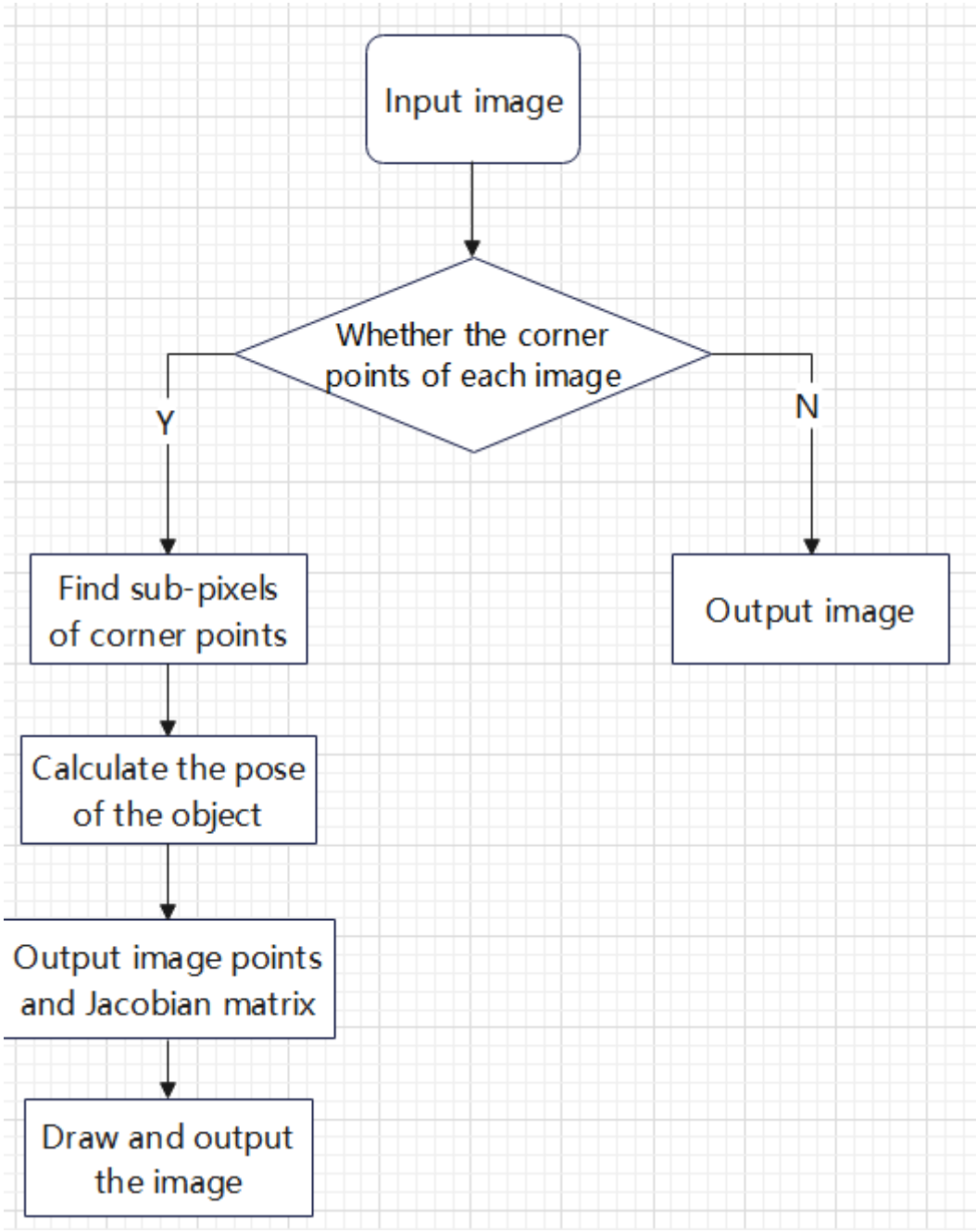
```
import numpy as np
from cv_bridge import CvBridge
from std_msgs.msg import String
from sensor_msgs.msg import CompressedImage,Image

class simple_AR:
    def __init__(self):
        rospy.on_shutdown(self.cancel)
        self.index = 0
        self.frame = None
        self.img_name = 'img'
        self.patternSize = (6, 9)
        self.bridge = CvBridge()
        self.flip = rospy.get_param("~flip", False)
        # 加载相机内参矩阵、畸变系数
        # Load the camera internal parameter matrix and distortion coefficient
        yaml_path = rospkg.RosPack().get_path("dofbot_visual") + '/AR/astra.yaml'
        if os.path.exists(yaml_path):
            fs = cv.FileStorage(yaml_path, cv.FileStorage_READ)
            self.cameraMatrix = fs.getNode("camera_matrix").mat()
            self.distCoeffs = fs.getNode("distortion_coefficients").mat()
        else: self.distCoeffs, self.cameraMatrix = (), ()
        self.objectPoints = np.zeros((6 * 9, 3), np.float32)
        self.objectPoints[:, :2] = np.mgrid[0:6, 0:9].T.reshape(-1, 2)
        self.graphics = ["Triangle", "Rectangle", "Parallelogram","WindMill",
                         "TableTennisTable", "Ball", "Arrow", "Knife", "Desk",
                         "Bench", "Stickman", "ParallelBars"]
        self.Graphics = self.graphics[self.index]
        self.axis = np.float32([
            [0, 0, -1], [0, 8, -1], [5, 8, -1], [5, 0, -1],
            [1, 2, -1], [1, 6, -1], [4, 2, -1], [4, 6, -1],
            [1, 0, -4], [1, 8, -4], [4, 0, -4], [4, 8, -4],
            [1, 2, -4], [1, 6, -4], [4, 2, -4], [4, 6, -4],
            [0, 1, -4], [3, 2, -1], [2, 2, -3], [3, 2, -3],
            [1, 2, -3], [2, 2, -4], [2, 2, -5], [0, 4, -4],
            [2, 3, -4], [1, 3, -4], [4, 3, -5], [4, 5, -5],
            [1, 2, -3], [1, 6, -3], [5, 2, -3], [5, 6, -3],
            [3, 4, -5], [0, 6, -4], [5, 6, -4], [2, 8, -4],
            [3, 8, -4], [2, 6, -4], [2, 0, -4], [1, 5, -4],
            [3, 0, -4], [3, 2, -4], [0, 3, -4], [1, 2, -4],
            [4, 2, -4], [5, 3, -4], [2, 7, -4], [3, 7, -4],
            [3, 3, -1], [3, 5, -1], [1, 5, -1], [1, 3, -1],
            [3, 3, -3], [3, 5, -3], [1, 5, -3], [1, 3, -3],
            [1, 3, -6], [1, 5, -6], [3, 3, -4], [3, 5, -4],
            [0, 0, -4], [3, 1, -4], [1, 1, -4], [0, 2, -4],
            [2, 4, -4], [4, 4, -4], [0, 8, -4], [5, 8, -4],
            [5, 0, -4], [0, 4, -5], [5, 4, -4], [5, 4, -5],
            [2, 5, -1], [2, 7, -1], [2, 6, -3], [2, 6, -5],
            [2, 5, -3], [2, 7, -3]
        ])
        self.sub_graphics = rospy.Subscriber('/Graphics_topic', String,
self.choose_Graphics)
        self.pub_img = rospy.Publisher("/simpleAR/camera", Image, queue_size=1)

    def cancel(self):
        self.sub_graphics.unregister()
```

```
        self.pub_img.unregister()
        cv.destroyAllWindows()
        rospy.loginfo("Shutting down this node.")

    def choose_Graphics(self, msg):
        if not isinstance(msg, String): return
        if msg.data in self.graphics: self.Graphics = msg.data
        else: self.graphics_update()

    def graphics_update(self):
        self.index += 1
        if self.index >= len(self.graphics): self.index = 0
        self.Graphics = self.graphics[self.index]

    def process(self, img, action):
        if self.flip == True: img = cv.flip(img, 1)
        if action == ord('f') or action == ord('F'): self.graphics_update()
        gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
        # 查找每个图片的角点
        # Find the corner 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)
        # 查找角点亚像素
        # 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))
            # 计算对象姿态solvePnPRansac
            # Compute object pose solvePnPRansac
            retval, rvec, tvec, inliers = cv.solvePnPRansac(
                self.objectPoints, corners, self.cameraMatrix, self.distCoeffs)
            # 输出图像点和雅可比矩阵
            # 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)
        cv.putText(frame, self.Graphics, (240, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9,
(0, 0, 255), 1)
        self.pub_img.publish(self.bridge.cv2_to_imgmsg(img, "bgr8"))
        return img

    def draw(self, img, corners, image_Points):
        # drawContours函数中绘图颜色顺序是bgr
        # drawContours the color order of the drawing is BGR
        img_pts = np.int32(image_Points).reshape(-1, 2)
        if self.Graphics == "Triangle":
            cv.drawContours(img, [np.array([img_pts[14], img_pts[15],
img_pts[23]])], -1, (255, 0, 0), -1)
        elif self.Graphics == "Rectangle":
            cv.drawContours(img, [np.array([img_pts[12], img_pts[13],
img_pts[15], img_pts[14]])], -1, (0, 255, 0), -1)
        elif self.Graphics == "Parallelogram":
```

```
            cv.drawContours(img, [np.array([img_pts[12], img_pts[10],
img_pts[15], img_pts[9]])], -1, (65, 105, 225), 1)
        elif self.Graphics == "WindMill":
            cv.drawContours(img, [np.array([img_pts[60], img_pts[38],
img_pts[61], img_pts[21]])], -1, (0, 0, 255), -1)
            cv.drawContours(img, [np.array([img_pts[10], img_pts[14],
img_pts[58], img_pts[21]])], -1, (0, 0, 255), -1)
            cv.drawContours(img, [np.array([img_pts[62], img_pts[63],
img_pts[23], img_pts[21]])], -1, (0, 0, 255), -1)
            cv.drawContours(img, [np.array([img_pts[25], img_pts[64],
img_pts[65], img_pts[21]])], -1, (0, 0, 255), -1)
            cv.line(img, tuple(img_pts[64]), tuple(img_pts[35]), (0, 255, 0), 3)
        elif self.Graphics == "TableTennisTable":
            cv.line(img, tuple(img_pts[0]), tuple(img_pts[60]), (255, 0, 0), 3)
            for i in range(1, 4):
                cv.line(img, tuple(img_pts[i]), tuple(img_pts[65 + i]), (255, 0,
0), 3)
            cv.drawContours(img, [np.array([img_pts[60], img_pts[66],
img_pts[67], img_pts[68]])], -1, (0, 255, 0), -1)
            cv.drawContours(img, [np.array([img_pts[23], img_pts[69],
img_pts[71], img_pts[70]])], -1, (0, 0, 255), -1)
        elif self.Graphics == "Ball": cv.circle(img, tuple(img_pts[22]), 30, (0,
0, 255), -1)
        elif self.Graphics == "Arrow":
            cv.drawContours(img, [np.array([img_pts[13], img_pts[34],
img_pts[36]])], -1, (0, 255, 0), -1)
            cv.drawContours(img, [np.array([img_pts[37], img_pts[15],
img_pts[10], img_pts[38]])], -1, (0, 255, 0), -1)
        elif self.Graphics == "Knife":
            cv.drawContours(img, [np.array([img_pts[58], img_pts[24],
img_pts[35], img_pts[47]])], -1, (160, 252, 0),
                            -1)
            cv.drawContours(img, [np.array([img_pts[40], img_pts[38],
img_pts[21], img_pts[41]])], -1, (30, 144, 255),
                            -1)
            cv.drawContours(img, [np.array([img_pts[42:46]])], -1, (0, 0, 255),
-1)
        elif self.Graphics == "Desk":
            for i in range(4):
                cv.line(img, tuple(img_pts[4 + i]), tuple(img_pts[12 + i]), (163,
148, 128), 3)
            cv.drawContours(img, [np.array([img_pts[14], img_pts[12],
img_pts[13], img_pts[15]])], -1, (0, 199, 140),
                            -1)
        elif self.Graphics == "Bench":
            for i in range(4):
                cv.line(img, tuple(img_pts[48 + i]), tuple(img_pts[52 + i]),
(255, 0, 0), 3)
            cv.drawContours(img, [img_pts[52:56]], -1, (0, 0, 255), -1)
            cv.drawContours(img, [img_pts[54:58]], -1, (139, 69, 19), -1)
        elif self.Graphics == "Stickman":
            cv.line(img, tuple(img_pts[18]), tuple(img_pts[4]), (0, 0, 255), 3)
            cv.line(img, tuple(img_pts[18]), tuple(img_pts[6]), (0, 0, 255), 3)
            cv.line(img, tuple(img_pts[18]), tuple(img_pts[21]), (0, 0, 255), 3)
            cv.line(img, tuple(img_pts[21]), tuple(img_pts[19]), (0, 0, 255), 3)
            cv.line(img, tuple(img_pts[21]), tuple(img_pts[20]), (0, 0, 255), 3)
```

```
            cv.line(img, tuple(img_pts[21]), tuple(img_pts[22]), (0, 0, 255), 3)
            cv.circle(img, tuple(img_pts[22]), 15, (0, 0, 255), -1)
            cv.line(img, tuple(img_pts[74]), tuple(img_pts[72]), (0, 255, 0), 3)
            cv.line(img, tuple(img_pts[74]), tuple(img_pts[73]), (0, 255, 0), 3)
            cv.line(img, tuple(img_pts[74]), tuple(img_pts[37]), (0, 255, 0), 3)
            cv.line(img, tuple(img_pts[37]), tuple(img_pts[76]), (0, 255, 0), 3)
            cv.line(img, tuple(img_pts[37]), tuple(img_pts[77]), (0, 255, 0), 3)
            cv.line(img, tuple(img_pts[37]), tuple(img_pts[75]), (0, 255, 0), 3)
            cv.circle(img, tuple(img_pts[75]), 15, (0, 255, 0), -1)
        elif self.Graphics == "ParallelBars":
            for i in range(4):
                cv.line(img, tuple(img_pts[4 + i]), tuple(img_pts[12 + i]), (255,
0, 0), 3)
            cv.line(img, tuple(img_pts[8]), tuple(img_pts[9]), (0, 0, 255), 3)
            cv.line(img, tuple(img_pts[10]), tuple(img_pts[11]), (0, 0, 255), 3)
        return img

if __name__ == '__main__':
    rospy.init_node("simple_AR", anonymous=False)
    capture = cv.VideoCapture(0)
    capture.set(6, cv.VideoWriter_fourcc('M', 'J', 'P', 'G'))
    capture.set(cv.CAP_PROP_FRAME_WIDTH, 640)
    capture.set(cv.CAP_PROP_FRAME_HEIGHT, 480)
    print("capture get FPS : ", capture.get(cv.CAP_PROP_FPS))
    pTime, cTime = 0, 0
    ar = simple_AR()
    while capture.isOpened():
        ret, frame = capture.read()
        action = cv.waitKey(1) & 0xFF
        if action == ord('q') or action == ord("Q"): break
        frame = ar.process(frame, action)
        cTime = time.time()
        fps = 1 / (cTime - pTime)
        pTime = cTime
        text = "FPS : " + str(int(fps))
        cv.putText(frame, text, (20, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0,
255), 1)
        if len(sys.argv) != 1:
            if sys.argv[1]=="true" or sys.argv[1]=="True": cv.imshow('frame',
frame)
        else:cv.imshow('frame', frame)
    capture.release()
    cv.destroyAllWindows()
```

Program flow chart,


Input image
Whether the corner points of each image
Y
N
Find sub-pixels of corner points
Output image
Calculate the pose of the object
Output image points and Jacobian matrix
Draw and output the image