4.AR vision

4.1 Overview

Augmented Reality, referred to as "AR",technology is a technology that cleverly integrates virtual information with the real world. It widely uses a variety of technical means such as multimedia, three-dimensional modeling, real-time tracking and registration, intelligent interaction, and sensing.After simulating computer-generated text, images, three-dimensional models, music, videos and other virtual information, it is applied 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 between the real world and the virtual world; ② real-time interactivity; ③ adding 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.

4.2 How to use

When using the AR case, you must have the internal parameters of the camera, otherwise it will not run (the factory image has completed the calibration of the internal parameters of the camera). The internal parameter files are in the same directory as the code.

4.2.1 Camera internal parameter calibration

Start monocular camera

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

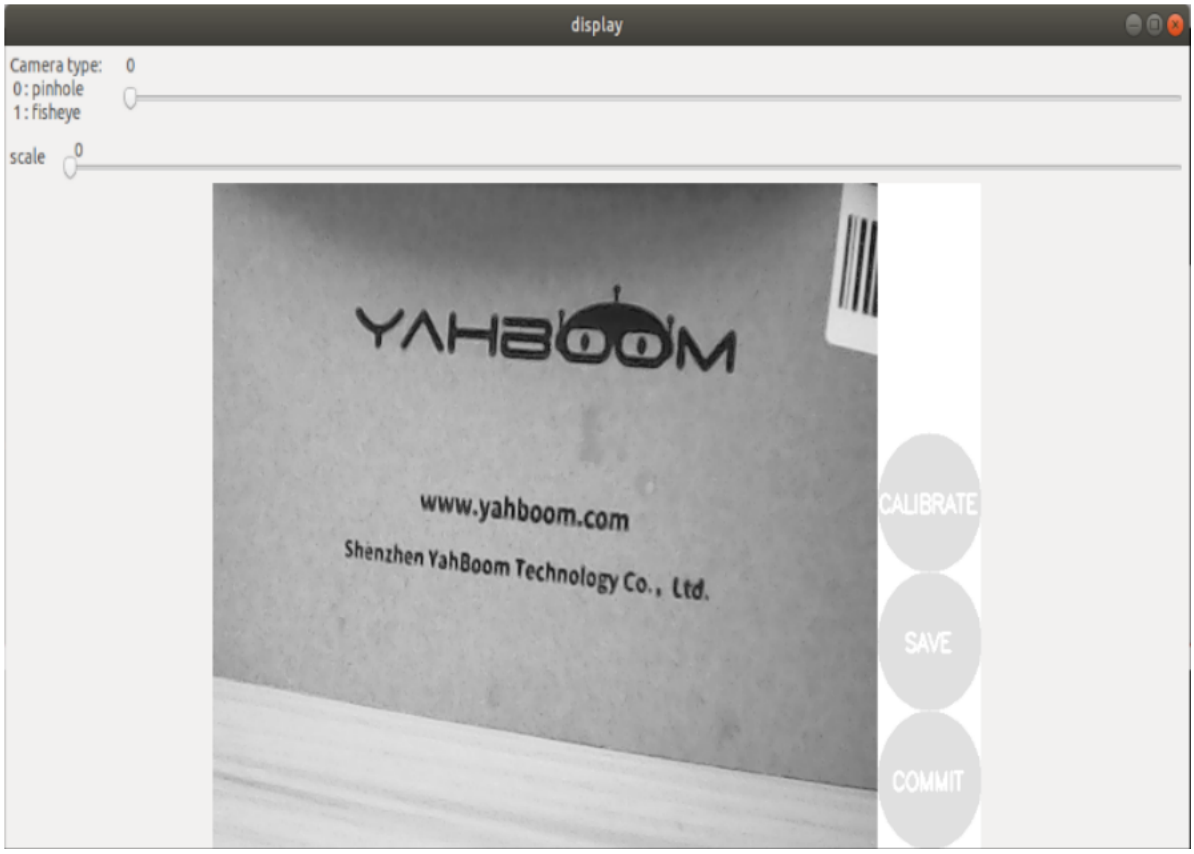
Start calibration node

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

size：Calibrate the number of internal corner points of the checkerboard, for example, 9X6, with a total of six rows and nine columns of corner points.

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

image: Image topic posted by the camera



Calibration screen

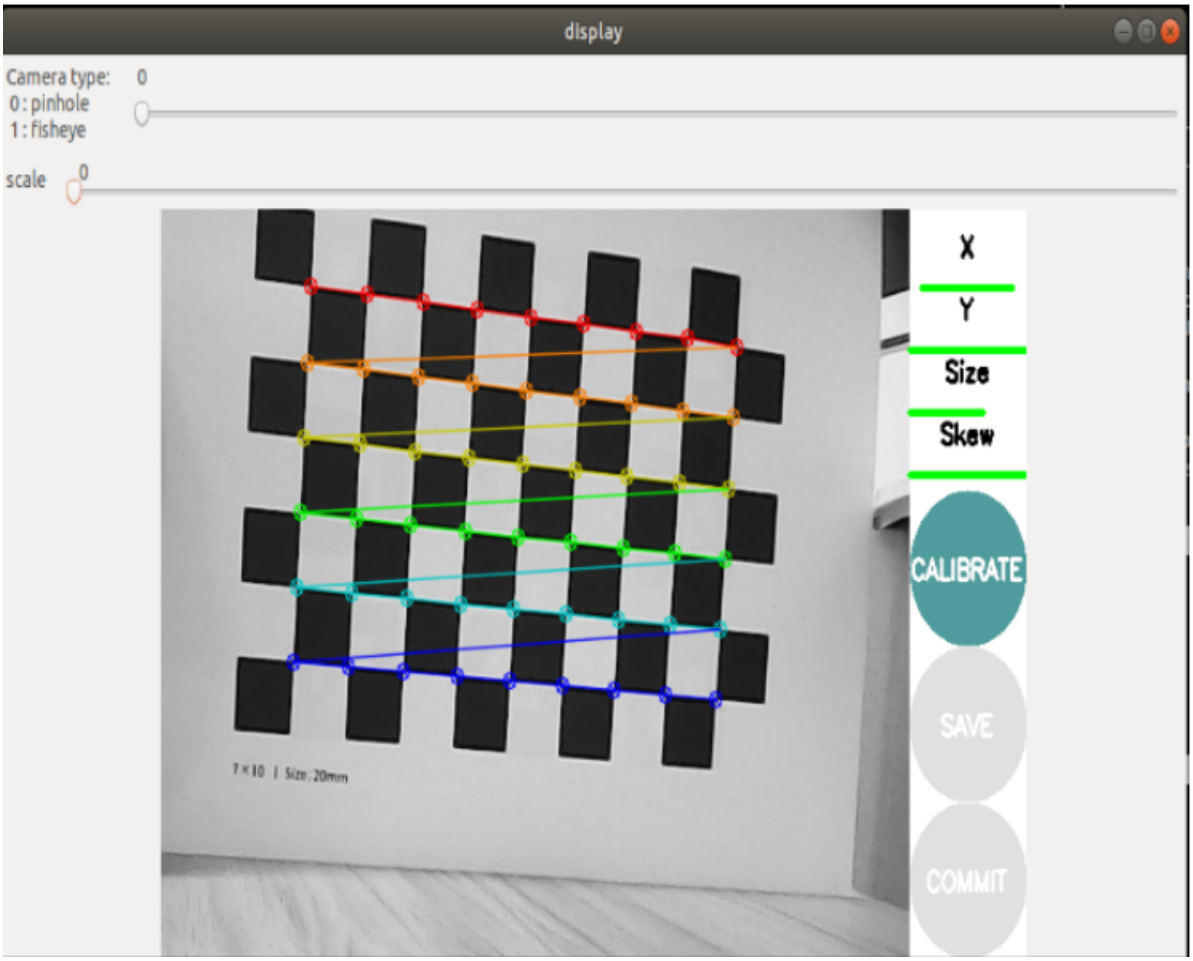
X: The left and right movement of the checkerboard in the camera field of view

Y: The checkerboard moves up and down in the camera field of view

Size: the movement of the checkerboard back and forth in the camera field of view

Skew: The tilt and rotation of the checkerboard in the camera's field of view

After successful startup, place the checkerboard in the center of the screen and change to different positions. The system will identify it independently. The best situation is that the lines under [X], [Y], [Size], and [Skew] will first change from red to yellow and then to green as the data is collected, filling them as fully as possible.



- Click [CALIBRATE] to calculate the internal parameters of the camera. The more pictures you have, the longer it will take. Just wait. (Sixty or seventy is enough, too many can easily get stuck).
- Click [SAVE] to save the calibration results to [/tmp/calibrationdata.tar.gz] of the currently running terminal.

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

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

After decompression, there are the image just calibrated, an ost.txt file and an ost.yaml file. ost.yaml is the internal parameter for calibrating the camera. Copy the content of the internal parameters here to astra.yaml under /home/yahboom/dofbot_ws/src/dofbot_visual/AR.

4.2.2 Start up

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

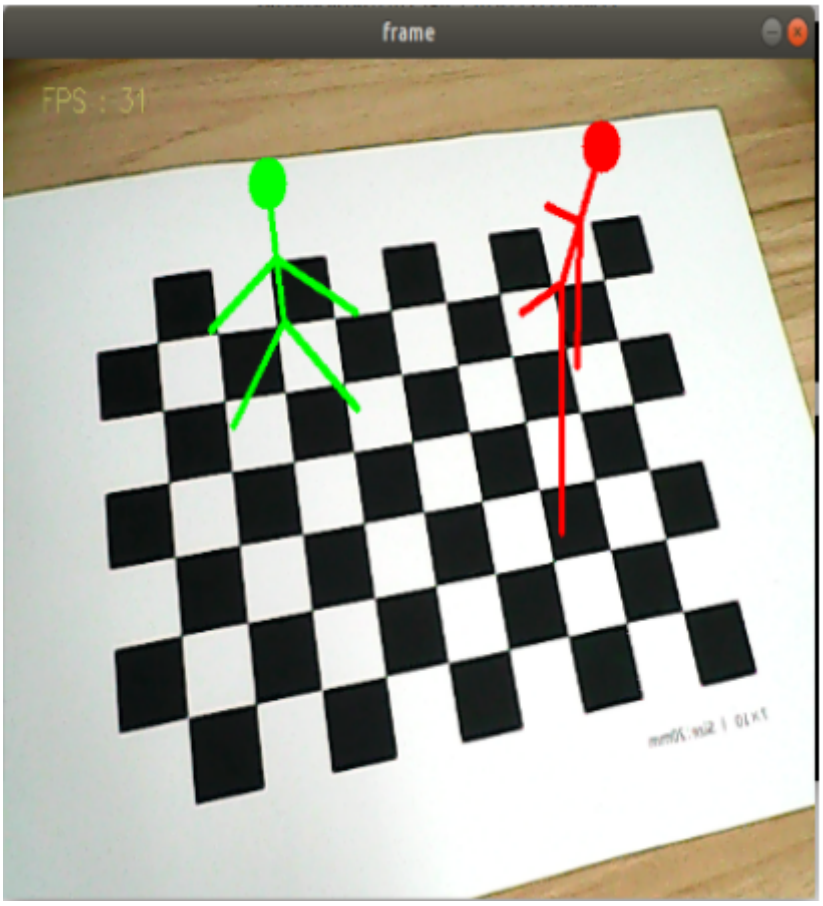
```
["Triangle", "Rectangle", "Parallelogram","WindMill","TableTennisTable",
"Ball","Arrow", "Knife", "Desk","Bench", "Stickman", "ParallelBars"]
```

Click on the image and press F to switch the displayed effect.

4.2.3 Program startup

Terminal input,

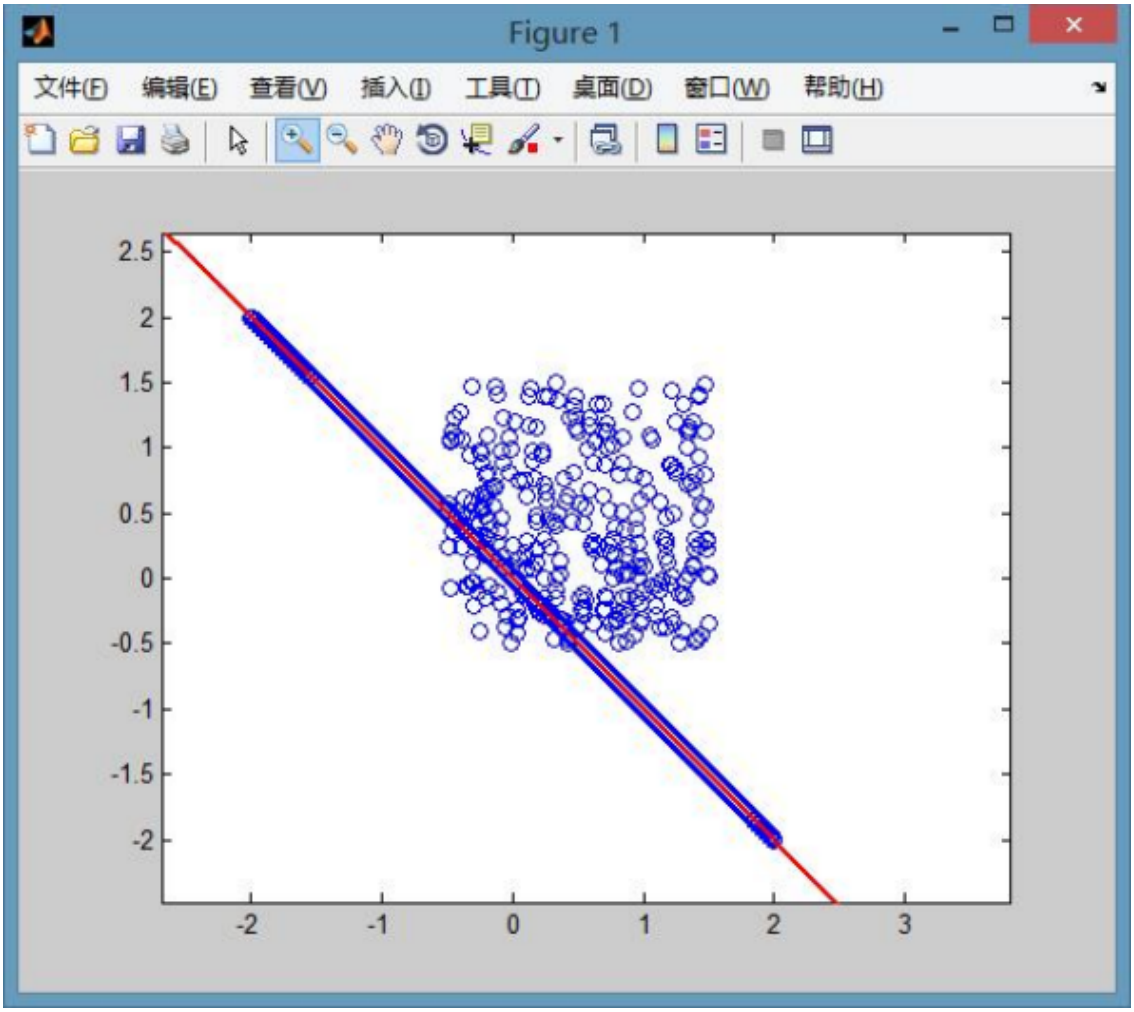
```
roscore
rosrun dofbot_visual simple_AR.py
```



4.3.1 RANSAC solution

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

  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:
  - The data consists of "internal points", for example: the distribution of the data can be explained by some model parameters;
  - "Outliers" are data that cannot fit the model;
  - Data other than this is noise.
- The causes of outliers include: extreme values of noise; wrong measurement methods; wrong assumptions about 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.

4.3.2 Source code

Source code location: /home/yahboom/dofbot_ws/src/dofbot_visual/AR/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)
        # Load camera internal parameter matrix and distortion coefficient
        # 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)
        # Find the corner points of each image
        # 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)
        # Find corner subpixels
        # 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
            # Compute object pose solvePnPRansac
            retval, rvec, tvec, inliers = cv.solvePnPRansac(
                self.objectPoints, corners, self.cameraMatrix, self.distCoeffs)
            # Output image points and Jacobian matrix
            # 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):
        # The drawing color order in the drawContours function is 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()
```