4.AR Vision

4.1 Overview

Augmented Reality (AR) technology cleverly integrates virtual information with the real world. It utilizes a wide range of technologies, including multimedia, 3D modeling, real-time tracking and registration, intelligent interaction, and sensing. It simulates computer-generated virtual information, such as text, images, 3D models, music, and video, and then applies it to the real world. The two types of information complement each other, thereby enhancing the real world.

AR systems have three key characteristics: 1) information integration between the real and virtual worlds; 2) real-time interactivity; and 3) the addition of localized virtual objects in a three-dimensional space.

AR technology encompasses new technologies and approaches, including multimedia, 3D modeling, real-time video display and control, multi-sensor fusion, real-time tracking and registration, and scene fusion.

4.2 Usage Instructions

For the PI5/JETSON NANO controller, you must first enter the Docker container. This is not necessary for the Orin board.

When using AR examples, the camera's internal parameters are required; otherwise, the system will not function (the factory image already has the camera's internal parameters calibrated). The internal parameter files are in the same directory as the code.

4.2.1. Camera Intrinsic Calibration

Starting the Monocular Camera

```
#usb camera
ros2 launch usb_cam camera.launch.py
#nuwa camera
ros2 launch ascamera hp60c.launch.py
```

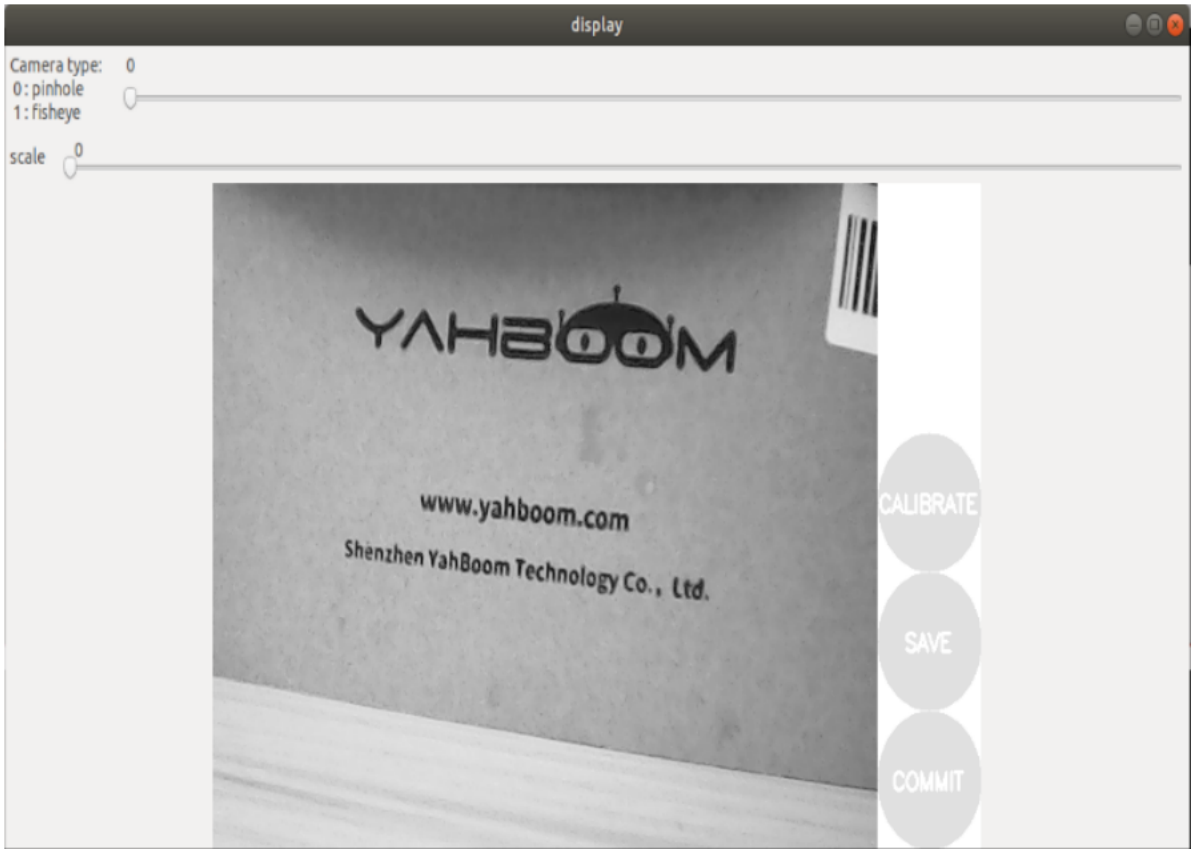
Starting the Calibration Node

```
#usb camera
ros2 run camera_calibration cameracalibrator --size 9x6 --square 0.02 --ros-args
--remap /image:=/usb_cam/image_raw
#nuwa camera
ros2 run camera_calibration cameracalibrator --size 9x6 --square 0.02 --ros-args
--remap /image:=/ascamera_hp60c/camera_publisher/rgb0/image
```

size: The number of internal corner points of the calibration checkerboard. For example, 9x6 means six rows and nine columns of corner points.

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

Image: The image topic published by the camera



Calibration Interface

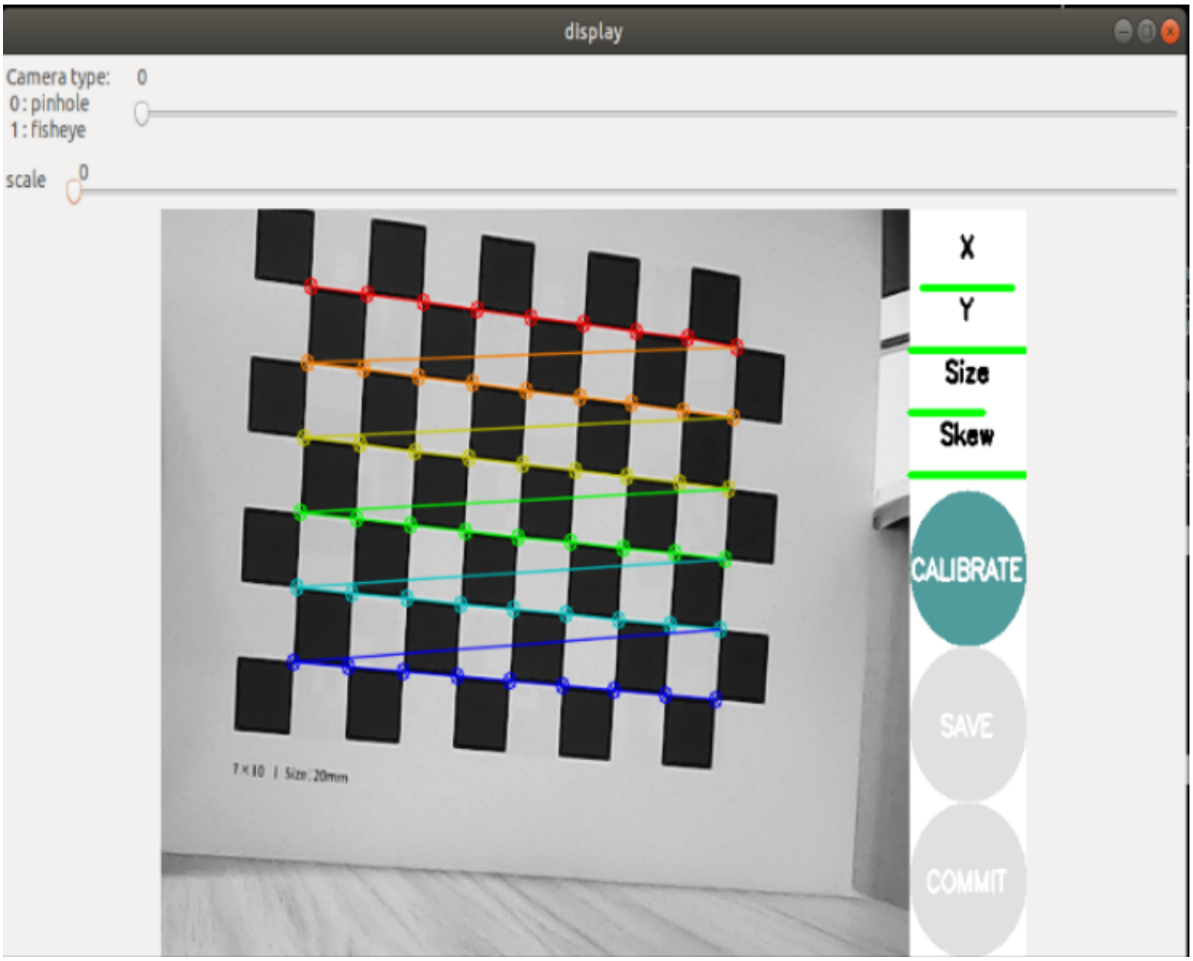
X: Left and right movement of the checkerboard square within the camera's field of view

Y: Up and down movement of the checkerboard square within the camera's field of view

Size: Forward and backward movement of the checkerboard square within the camera's field of view

Skew: Tilt and rotation of the checkerboard square within the camera's field of view

After successful startup, place the checkerboard square in the center of the image and change its position to different positions. The system will automatically identify the position. For optimal results, the lines under [X], [Y], [Size], and [Skew] will change from red to yellow and then to green as data is collected, filling the area as much as possible.



- Click [CALIBRATE] to calculate the camera's intrinsic parameters. The more images you have, the longer it will take, so just wait. (Sixty or seventy images is enough; too many can cause the calibration to freeze.)
- Click [SAVE] to save the calibration results to [/tmp/calibrationdata.tar.gz] in the current terminal.

After calibration is complete, you can move the [/tmp/calibrationdata.tar.gz] file to view its contents.

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

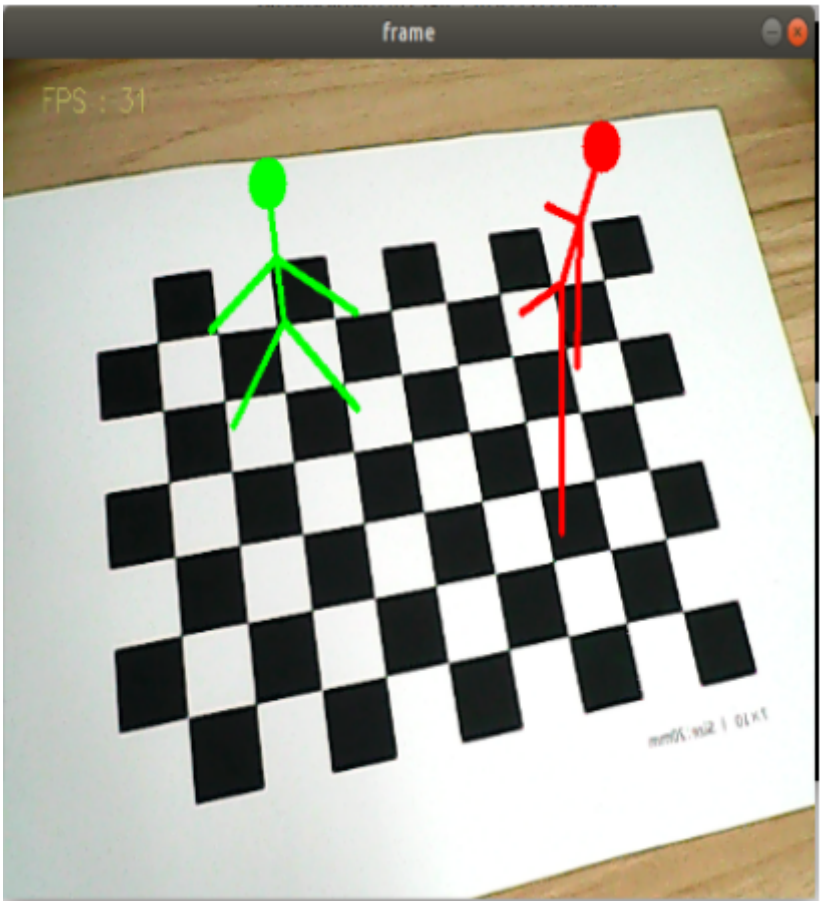
After decompression, the file contains the calibrated images, an ost.txt file, and an ost.yaml file. ost.yaml contains the camera's intrinsic parameters. Copy the internal reference contents here and overwrite them in ~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_vision/config/camera.yaml.

Note: The factory image has pre-calibrated parameters and can be used directly by launching the program. If the camera image is already active, please close the camera process before launching the program.

4.2.2 Program Startup

In the terminal, enter:

```
ros2 run yahboomcar_vision simple_ar
```



After launching the program, place the checkerboard in front of the camera. Make sure you can see the entire checkerboard, otherwise the program will exit. After the entire checkerboard is recognized, the following 12 effects will be displayed:

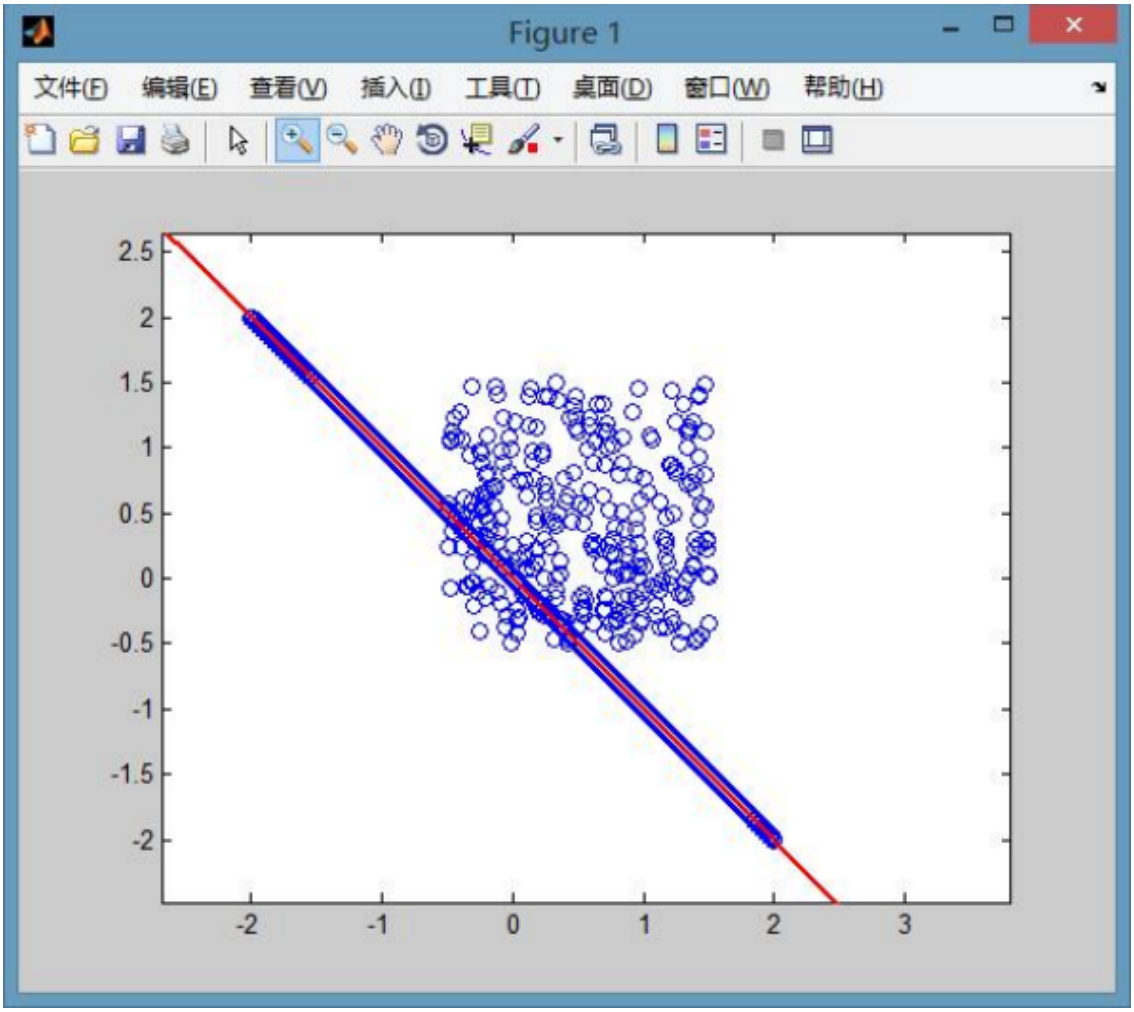
```
["Triangle", "Rectangle", "Parallelogram", "WindMill", "TableTennisTable",
"Ball", "Arrow", "Knife", "Desk", "Bench", "Stickman", "ParallelBars"]
```

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

4.3.1 RANSAC Scheme

- Algorithm Principle: Using the RANSAC Scheme to Find Object Pose from 3D-2D Point Correspondences

  The RanSaC algorithm (Random Sample Consensus) is a classic data processing algorithm used to extract specific components from an object in the presence of significant noise. The following figure illustrates the RanSaC algorithm's effectiveness. In the figure, some points clearly conform to a line, while another cluster of points is pure noise. The goal is to find the equation of a line in the presence of significant noise. In this case, the amount of noise data is three times the amount of the line itself.




```
    If the least squares method is used, this effect cannot be achieved, and the
straight line will be slightly above the straight line in the figure.
```

- The basic assumptions of RANSAC are:
  - The data consists of "in-place points", i.e., the distribution of the data can be explained by some model parameters;
  - “Outliers” are data that cannot be fitted into the model;
  - Any data other than this is noise.
- Outliers can be caused by extreme noise, incorrect measurement methods, or incorrect assumptions about the data. RANSAC also makes the following assumptions: given a set of (usually small) inliers, there exists a process for estimating model parameters; and the model can explain or apply to the inliers.

4.3.2 Source Code

Source Code Location:~/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_vision/yahboomcar_vision/simple_ar.py

```
#common lib
import os
import sys
import time
import cv2 as cv
import numpy as np
from cv_bridge import CvBridge
```

```
#ros2 lib
import rclpy
from rclpy.node import Node
from std_msgs.msg import String
from sensor_msgs.msg import CompressedImage,Image

print("import done")

cv_edition = cv.__version__
print("cv_edition: ",cv_edition)

class simple_AR(Node):
    def __init__(self,name):
        super().__init__(name)
        #create pub
        self.pub_img = self.create_publisher(Image,'/simpleAR/camera',1)
        #create sub
        self.sub_graphics =
self.create_subscription(String,'/Graphics_topic',self.choose_Graphics,1)

        self.flip = True
        self.index = 0
        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.frame = None
        self.img_name = 'img'
        self.patternSize = (6, 9)
        self.bridge = CvBridge()
        yaml_path =
'/home/jetson/yahboomcar_ros2_ws/yahboomcar_ws/src/yahboomcar_vision/config/came
ra.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)
        camera_type = os.getenv('CAMERA_TYPE', 'usb')
        topic_name = '/ascamera_hp60c/camera_publisher/rgb0/image' if
camera_type == 'nuwa' else '/usb_cam/image_raw'

        self.subscription = self.create_subscription(
            Image,
            topic_name,
            self.image_callback,
            10)


    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 image_callback(self, msg):
        frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
        action = cv.waitKey(10) & 0xFF
        frame =self.process(frame, action)
        cv.imshow('frame', frame)
        #if len(binary) != 0: cv.imshow('frame', ManyImgs(1, ([frame, binary])))
        #else:cv.imshow('frame', frame)
        if action == ord('q') or action == 113:
            cv.destroyAllWindows()

    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
```

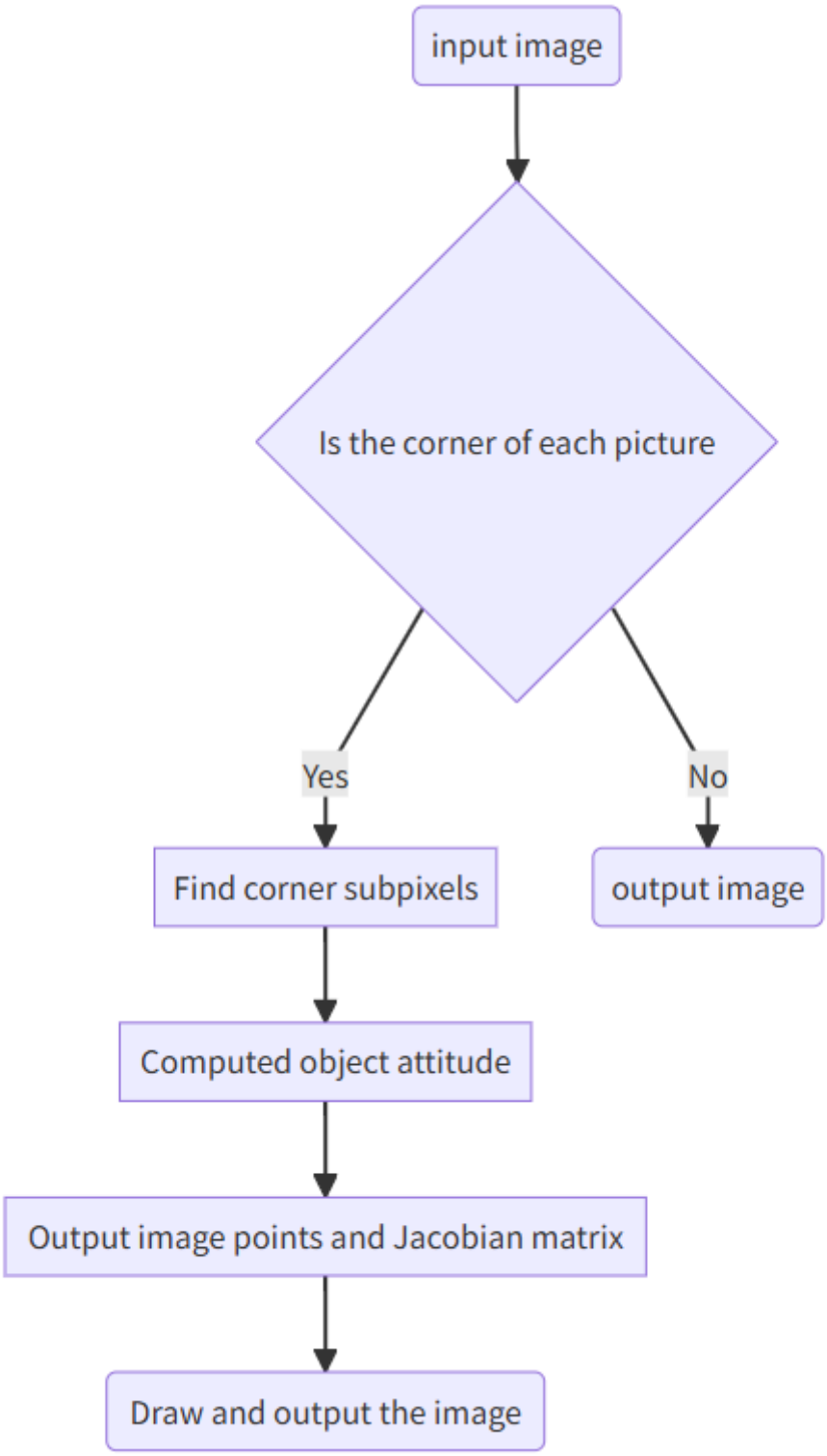
```
            image_Points, jacobian = cv.projectPoints(
                self.axis, rvec, tvec, self.cameraMatrix, self.distCoeffs, )
            img = self.draw(img, corners, image_Points)
        cv.putText(img, self.Graphics, (240, 30), cv.FONT_HERSHEY_SIMPLEX, 0.9,
(0, 0, 255), 1)
        self.pub_img.publish(self.bridge.cv2_to_imgmsg(img, "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

def main():
    print("start")
    rclpy.init()
    ar = simple_AR('simple_AR')
    rclpy.spin(ar)
```

Program flow chart,


input image
Is the corner of each picture
Yes
No
Find corner subpixels
output image
Computed object attitude
Output image points and Jacobian matrix
Draw and output the image