2022 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

2022 Intel Cup Undergraduate Electronic Design Contest

- Embedded System Design Invitational Contest

作品设计报告

Final Report

Intel Cup Embedded System Design Contest

报告题目：基于 AR 图画处理的自动绘画机

学生姓名: 吴俊涛 任加涵 沈铭

指导教师: 黄继业

参赛学校: 杭州电子科技大学

2022 年英特尔杯大学生电子设计竞赛嵌入式系统专题邀请赛

参赛作品原创性声明

本人郑重声明：所呈交的参赛作品报告，是本人和队友独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果，不侵犯任何第三方的知识产权或其他权利。本人完全意识到本声明的法律结果由本人承担。

参赛队员签名：

日期：　　2022 年 7 月 26 日

基于 AR 图画处理的自动绘画机

摘要

随着社会的发展，人们生活的愈发忙碌，匮乏的娱乐时间让人们追寻起更为简便、短耗时的娱乐活动。基于各种情况，本项目提出了一款基于 AR 图画处理的自动绘画机。旨在给用户带来沉浸式真实绘画体验同时减去绘画所需要耗费的大量时间。

项目主要分成 AR 图画处理模块和机械臂绘制模块，整个项目均部署在信步边缘 AI 计算平台 GNS-V40 上。首先通过摄像机拍摄得到初始画面，经过 AR 算法处理后画面中出现一个动态的 3D 物体，然后使用 OpenVINO 套件部署 Yolov5 目标检测算法，依据检测到的锚框得到图像的主体并使用 Canny 算法得到图像的边缘轮廓线的二维坐标，最后坐标以文件的形式发送给机械臂进行路径规划绘图，从而得到预期的线稿。

我们测试了 AR 增强现实算法的稳定性、YoloV5 目标检测算法的准确性、Canny 算法的鲁棒性和机械臂绘制图像的精度，其结果均在可以容忍的范围内，满足项目的实际需求。实现了真正的设备边缘计算，为市场中缺乏与 AR 相关的自动绘画机相关产品提供了一种可行的解决方案。

关键词： AR 增强现实，OpenVINO，机械臂，Canny 算法，Yolov5 目标检测，自动绘画

AUTOMATIC DRAWING MACHINE BASED ON AR

ABSTRACT

With the development of society, people's life is increasingly busy, and the lack of entertainment time makes people pursue more convenient, short and time-consuming entertainment activities. Based on various situations, this project proposes an automatic painting machine based on AR picture processing. The aim is to give the user an immersive, realistic drawing experience while reducing the amount of time it takes to draw.

The project is mainly divided into AR picture processing module and manipulator drawing module. The whole project is deployed on the walking edge AI computing platform GNS-V40. First, the initial picture is captured by camera, and a dynamic 3D object appears in the picture after AR algorithm is processed. Then, Yolov5 object detection algorithm is deployed by OpenVINO suite, and the image subject is obtained according to the detected anchor frame, and the two-dimensional coordinates of the image edge contour are obtained by Canny algorithm. Finally, the coordinates are sent to the manipulator in the form of a file for path planning and drawing, so as to get the expected line draft.

We have tested the stability of AR augmented reality algorithm, the accuracy of YoloV5 object detection algorithm, the robustness of Canny algorithm and the accuracy of robot arm drawing images. The results are all within the tolerable range and meet the actual requirements of the project. The real edge computing of the device is realized, which provides a feasible solution for the lack of AR related products in the market.

Key words: AR (augmented reality), mechanical arm, OpenVINO,Canny algorithm, Yolov5 target detection, automatic painting

目 录

第一章 绪论

随着社会的不断发展与进步，娱乐项目在我们的生活中已是不可缺少的部分，且越来越多样化。但社会进步的同时，生活节奏也越来越快，学习和工作的强度和压力也相应增加。“996”的工作模式的出现就应证了当代青年巨大的工作压力。寻求解脱、追寻娱乐成为人们的一种发泄压力的一种方式。但是在这样的快生活社会环境之下，很难有时间能够挤出时间出门逛街或者好友聚餐。不仅如此，疫情的阴霾一直笼罩在我们周围，出门更是成为了许多人的奢求。

由此我们想创造一种方便、能让用户沉浸其中的娱乐放松方式。绘画就由此提出。绘画是对生活的观察，是对生活的感悟，它不是一件功利的事情，绘画过程中我们可以享受到沉浸其中的宁静和快感，感受那最为真切抑或是梦幻中的生活。但是绘画是一件极为耗时的事情，在这个忙碌的生活似乎显得格格不入，而且入门门槛也不低，需一个学习过程。这就带来想困扰。

为了解绝这些问题，我们想了很多办法，最终提出了使用 AR 来进行绘图，提供用户沉浸的绘画过程。同时这样一种便捷又新奇的也能够吸引用户。对用户来说，娱乐时间少之又少，绘画又是一件极为耗时的过程，所以我们采用了自动绘画机，用机械臂绘制出用户创作好的 AR 图片。如此，一款基于 AR 绘画创作的自动绘画机便横空出世。

1.1 国内外发展趋势动态分析

1.1.1 AR 技术研究现状及分析

增强现实(Augmented Reality)技术是一种将虚拟信息与真实世界巧妙融合的技术，广泛运用了多媒体、三维建模、实时跟踪及注册、智能交互、传感等多种技术手段，将计算机生成的文字、图像、三维模型、音乐、视频等虚拟信息模拟仿真后，应用到真实世界中，两种信息互为补充，从而实现对真实世界的“增强”。

近年来，全球行业巨头纷纷开展增强现实(AR)产品研发，例如 2012 年谷歌公司率先推出一款 AR 眼镜—Google Project Glass，紧接着微软、苹果、任天堂、华为、腾讯等公司也发布相关 AR 产品。

从增强现实(AR)技术申请情况来看，通过智慧芽“AR”关键词搜索显示，全球增强现实(AR)技术申请量 2016 年以前增长较为平稳，2016 年以后增长较快，2020 年有所回落。随着增强现实(AR)技术申请量的不断增加，行业技术慢慢走向成熟。根据 Gartner 发布的 2020 年新技术成熟度曲线，AR 技术正式脱离该名单，说明 AR 技术趋于稳定，从一项有待观察的技术转变为可以使用的技术。

随着增强现实(AR)技术趋于成熟，其产业应用速度也将加快，增强现实(AR)行业将迎来爆发，其用户规模将持续增加。根据德勤数据显示，到 2025 年，全球近 75%的人口和几乎所有使用社交/通信应用程序的人都将成为 AR 的频繁用户，用户每天将拍摄超过 45 亿个/张 AR 视频/照片。

根据中国信通院数据显示，2020 年全球增强现实(AR)终端出货量约为 63 万台，到 2024 年将达到 4125 万台，年复合增长率达到 188%;2020 年全球增强现实(AR)行业市场规模约为 280 亿元，到 2024 年将达到 2400 亿元，年复合增长率达到 66%。

1.1.2 自动绘画机研究现状及分析

目前国内市场上是没有 AR 绘画机作为成熟的商品出售，通过对国内外 AR 绘画和自动绘画技术的专利检索，大致有以下几种专利：

由杨克俊发明的一种基于触摸显示设备的 AR 互动绘画馆（专利授权号：CN208044543U）。该实用新型提供一种基于触摸显示设备的 AR 互动绘画馆，整个项目包括：触摸屏、处理器、传感器等设备。通过传感器与处理器通信连接，控制主机单元处理内置图像，最终生成 AR 动画传给显示设备。图 1-1 为相关专利附图。

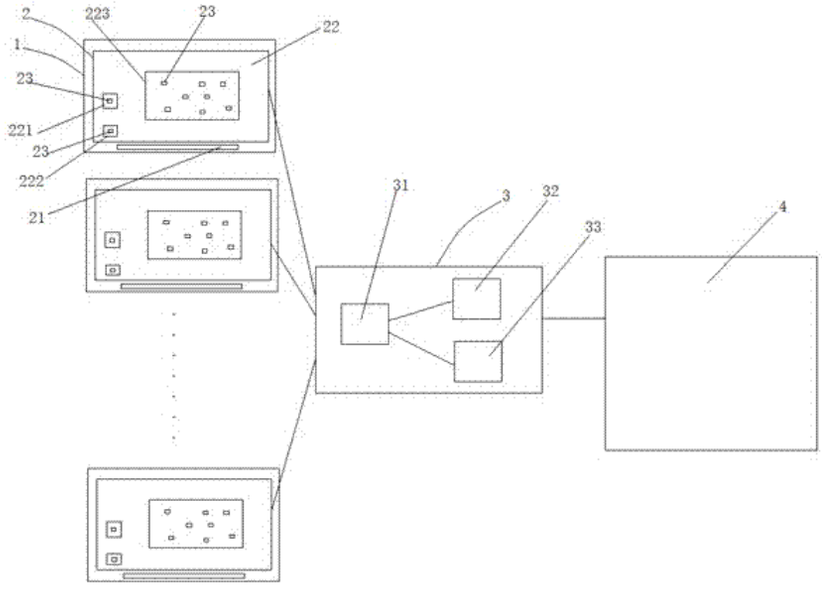


图 1-1 基于触摸显示设备的 AR 互动绘画馆

由郑州科尼科技有限公司提出的 AR 智能互动绘画板（专利授权号：CN208298144U）。该发明包括绘画板本体，绘画板本体包括壳体、画笔以及设置在壳体上的用于显示画笔所绘的图案的显示面板；壳体内设有控制主板；控制主板包括用于控制显示面板显示图案的显示面板控制模块以及用于控制显示面板上所绘图案与使用者互动的 AR 互动模块。图 1-2 为相关专利附图。

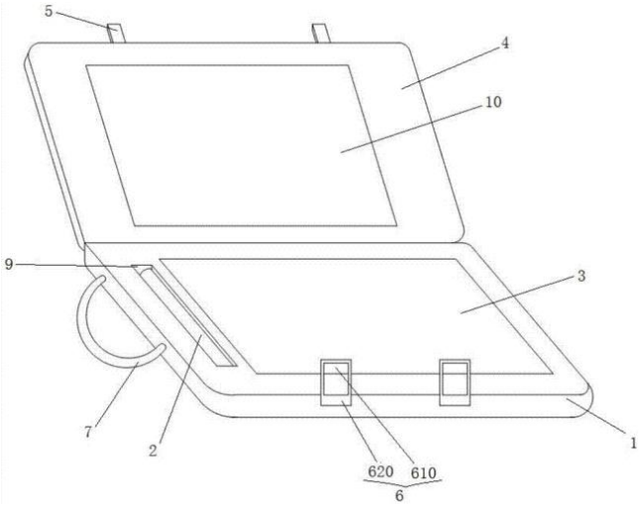


图 1-2 AR 智能互动绘画板

由日本株式会社大气社申请的自动绘画系统（专利授权号：CN111405947A）。该发明以提供能够描绘轮廓顺畅且清楚的图形的自动绘画系统为目的，控制装置构成为，随着基于预

定绘画图形（G）的图形数据通过正交机器人的动作使涂料排出装置与在预定绘画部位描绘的预定绘画图形（G）的轮廓线（g）平行地移动，通过使涂料排出装置连续地进行涂料排出工作的轮廓平行绘画控制的执行绘画出预定绘画图形（G）的轮廓部。图 1-3 为相关专利附图。

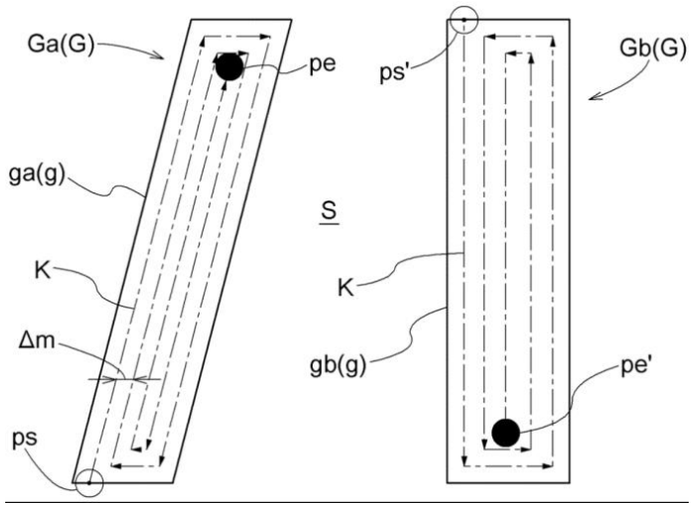


图 1-3 自动绘画系统

由王璐提出的一种全自动绘画装置（专利授权号：CN201456870U）。实用新型提供一种能将电脑中 CNC 机床码(G 码)图案在弧面上进行全自动绘画的装置。该装置包括底座部分、固定部分、旋转控制部分和绘画部分,通过三个步进电机分别控制物体的旋转、笔的摆动、提笔和收笔而实现在物体表面自动绘制图案。该装置通过与电脑相连可自动在物体表面绘制电脑中 CNC 机床码(G 码)图案。图 1-4 为相关专利附图。

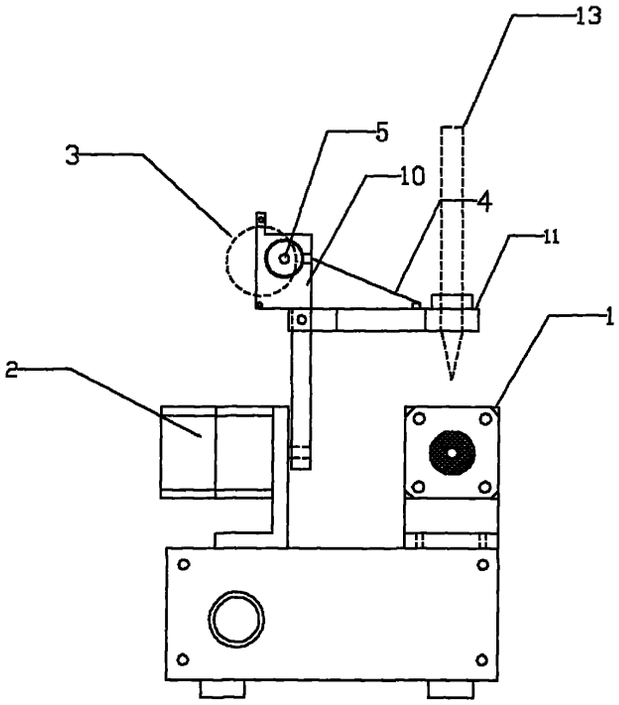


图 1-4 全自动绘画装置

1.2 项目应用前景

项目有很好的应用前景：

（1）可以在景区设立一个 AR 站点，游客可以在景点用 AR 添加上自己喜欢的图案，和景点一起被绘画机绘制在图纸上。

（2）作为一项娱乐 AR 机器，给予人们在忙绿的生活中一方宁静。

（3）室内设计，将设计模块直接用 AR 显示在经过设计师敲定后直接打印设计图纸。

（4）室外绘画，可以搭一套机器，人们可以用 AR 涂鸦、绘画等等，可以重复修改，满意之后用机器在墙上、门上等绘制图案。

第二章 基于 AR 图画处理的自动绘画机设计

2.1 系统方案

本项目总体方案设计如图 2-1 所示。由 AR 绘制模块采集周围环境数据，构建起 AR 环境并传输给 AR 显示模块。根据用户需求进行绘画创作，由绘制模块将实时数据与平台进行传输，根据指令做出图画处理再反馈到显示模块。在用户结束创作后可以选择自动绘制将图画数据传输给绘画机械臂进行自动绘制。

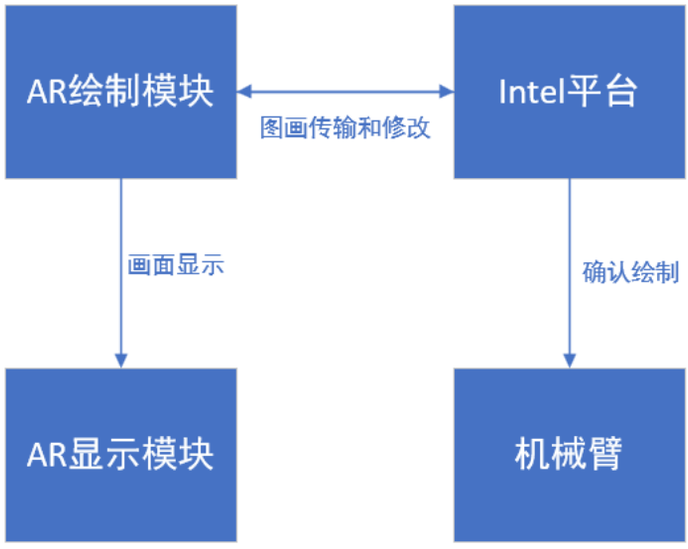


图 2-1 系统框图

2.2 功能与指标

2.2.1 实现功能

由于时间、精力、经济等因素，我们团队经过 3 个月的努力，完成了以功能：

（1）能够自动识别出人、物并使用 AR 自动添加图像，完成图画绘制。

（2）机械臂能够自动绘画出线稿轮廓、细节线条描绘，阴影模糊块，实现高质量复刻图画的素描画。

2.2.2 指标

2.2.2.1 硬件指标

（1）相机指标： SY500W2 30 帧 焦距：4.3 mm

（2）相机画面指标：2592*1944 像素

2.2.2.2 软件指标

（1）识别率指标：为保证 AR 图画能够实时跟踪，特征物识别正确率要在 90%以上。

（2）精确度指标：为保证绘制效果完美，机械臂绘制点的精确度需要在 95%以上。

（3）复刻程度指标：为保证素描图效果好，最终作品要能够在极大程度上还原图画。

第三章 实现原理与软件流程

3.1 实现原理及应用

3.1.1 基于 OpenCV 实现 3D 物品的 AR 实时投影

我们主要使用的工具是 Python、OpenCV，使用 numpy 进行矩阵变换，因为它们都是开源的，易于设置和使用，用它们构建原型较为快速，其主要思想是根据特定平面的位置和方向，在显示屏幕上呈现特定物体的 3D 模型，主要利用射影几何和不同的坐标系和变换矩阵实现。我们实现了在屏幕上投影一个 3D 模型，其位置和方向与某些预定义的平面的位置和方向相匹配。投影是实时完成的，如果参考平面位置或方向发生了，投影模型也会相应地改变。

3.1.1.1 识别目标参考平面。（Target surface）

（1）特征提取：(Feature extraction)

首先在参考图像和目标图像中寻找突出的特征，并以某种方式描述要识别的物体的一部分。当在目标图像和参考图像之间找到一定数量的特征匹配时，我们将认为已经找到了目标参考平面。

（2）特征描述：(Feature description)

我们使用的是在 OpenCV 实验室开发的 ORB 算法，获得的描述符将是二进制字符串。

（3）特征匹配：(Feature matching)

我们使用第一个集合中每个特征的描述符，计算到第二个集合中所有描述符的距离，并返回最接近的描述符作为最佳匹配，由于描述符是二进制字符串，所以使用汉明距离作为描述符之间的距离。我们为找到的最小匹配数定义了一个阈值。如果匹配的数量超过阈值，那么我们假定找到了对象。否则，我们认为没有找到对象。

3.1.1.2 单应性的评估。（Homography estimation）

如图 3-1 所示，我们需要找到从表面平面映射到图像平面的变换，这个变换在视频处理的每一帧中实时更新。

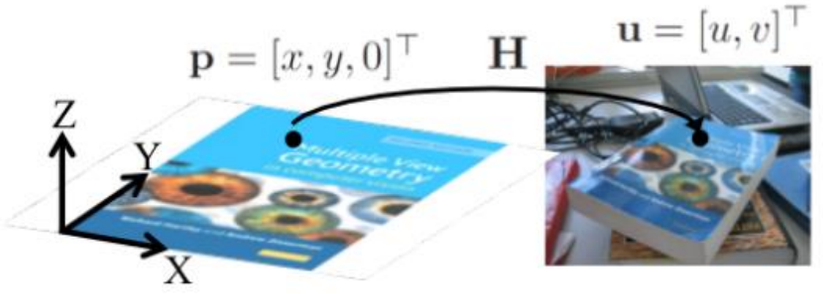


图 3-1 参考平面的坐标变换

参考平面位于世界坐标系中特定的位置和方向，该参考平面在世界坐标系中有一个已知的坐标。我们使用相机为其它拍摄一张照片，我们假设摄像机遵循针孔模型工作（如图

3-2），这大致意味着通过 3D 点 p 的射线和相应的 2D 点 u 相交于 c，即摄像机中心：

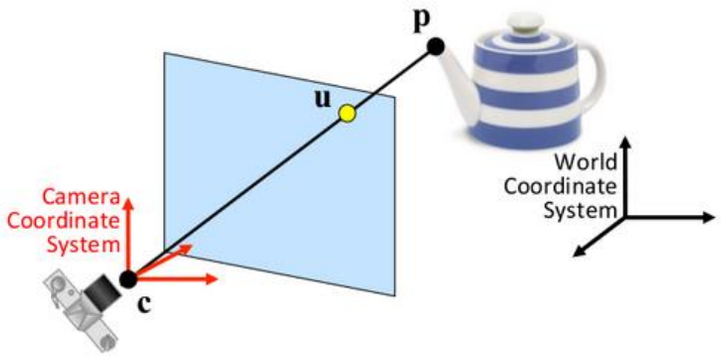


图 3-2 针孔模型

针孔模型的假设简化了计算，那么在 Camera 坐标系中 p 点的 u，v 坐标为：

$$\begin{bmatrix} u\cdot k \\ v\cdot k \\ k \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^{\text{cam}} \\ y^{\text{cam}} \\ z^{\text{cam}} \end{bmatrix}$$

Calibration matrix

(u_0, v_0) Projection of the optical center

f_u, f_v Focal lengths

图 3-3 camera 坐标系 p 点的 u，v 坐标

其中焦距为针孔到像面的距离，光中心的投影为光中心在像面中的位置，k 为比例因子。然而我们知道点 p 在 World 坐标系中的坐标，而不是在 Camera 坐标系中的坐标，因此必须添加另一个转换，将点从 World 坐标系映射到 Camera 坐标系。这个变换告诉我们世界坐标系中 p 点在图像平面中的坐标是：

Calibration matrix　　External calibration matrix

$$\begin{bmatrix} u\cdot k \\ v\cdot k \\ k \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Projection matrix: 3x4 matrix

图 3-4 p 点在图像平面的坐标

由于参考平面上的点的 z 坐标总是等于 0，很容易看出 z 坐标和投影矩阵第三列的乘积总是 0，所以我们可以把这一列和 z 坐标从之前的方程中去掉。将校准矩阵重命名为 A，并考虑到外部校准矩阵为齐次变换：

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \mathbf{A}[\mathbf{R}\ \ \mathbf{t}] \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix} = \mathbf{A}[\mathbf{R}_1\ \ \mathbf{R}_2\ \ \mathbf{R}_3\ \ \mathbf{t}] \begin{bmatrix} x \\ y \\ 0 \\ 1 \end{bmatrix}$$

columns of **R**

$$= \mathbf{A}[\mathbf{R}_1\ \ \mathbf{R}_2\ \ \mathbf{t}] \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

图 3-5 外部校准矩阵齐次变换

参考面与像面之间的单应性，匹配估计的矩阵为：

$$\begin{bmatrix} ku \\ kv \\ k \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

图 3-6 匹配估计的矩阵

我们使用随机样本共识(RANSAC)来估计单应矩阵的值。RANSAC 是一种迭代算法，用于在存在大量异常值的情况下进行模型拟合。算法流程大致如图 3-7 所示：

1) Randomly sample 4 matches
2) Estimate Homography **H**
3) Verify homography. Search for other matches consistent with **H**
4) Iterate until convergence

图 3-7 RANSAC 算法流程

3.1.1.3 平面姿态估计（Pose estimation from a plane）

我们不仅要投影参考平面中的点(z 坐标为 0)，还要投影参考空间中的任何点(z 坐标不为 0)。我们已经估计了单应性(H)，它的值是已知的。通过查看摄像机参数或一些常识，我们可以很容易地知道或猜测摄像机校准矩阵(a)的值，摄像机校准矩阵是：

$$\begin{bmatrix} u \cdot k \\ v \cdot k \\ k \end{bmatrix} = \begin{bmatrix} f_u & 0 & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x^{\text{cam}} \\ y^{\text{cam}} \\ z^{\text{cam}} \end{bmatrix}$$

Calibration matrix

(u_0, v_0) Projection of the optical center

f_u, f_v Focal lengths

图 3-8 摄像机校准矩阵

现在我们已经估计了单应矩阵 H 和摄像机校准矩阵 A，我们可以通过用 H 乘以 A 的逆

矩阵恢复 R_1，R_2 和 t：

$$\mathbf{G} = [\mathbf{G}_1\ \mathbf{G}_2\ \mathbf{G}_3] = \mathbf{A}^{-1}\mathbf{H}$$

图 3-9 恢复 R_1\R_2\t 的矩阵 G

则 G_1、G_2、G_3 的值可视为：

$$\begin{aligned}\mathbf{R}_1 &= \mathbf{G}_1\\ \mathbf{R}_2 &= \mathbf{G}_2\\ \mathbf{t} &= \mathbf{G}_3\end{aligned}$$

图 3-10 R1 R2 R3 的计算公式

由于外部校准矩阵[R_1 R_2 R_3 t]是一个齐次变换，它映射了两个不同参考系中的点我们可以确定[R_1 R_2 R_3]必须是正交的。理论上我们可以将 R_3 计算为：

$$\mathbf{R}_3 = \mathbf{R}_1 \times \mathbf{R}_2$$

图 3-11 R_3 的计算公式

下面是对 R 的估计：

$$l = \sqrt{\|\mathbf{G}_1\|\|\mathbf{G}_2\|} \qquad \mathbf{R}_1 = \frac{\mathbf{G}_1}{l}, \mathbf{R}_2 = \frac{\mathbf{G}_2}{l}, \boxed{\mathbf{t} = \frac{\mathbf{G}_3}{l}}$$

$$\mathbf{c} = \mathbf{R}_1 + \mathbf{R}_2\ , \mathbf{p} = \mathbf{R}_1 \times \mathbf{R}_2\ , \mathbf{d} = \mathbf{c} \times \mathbf{p}$$

$$\mathbf{R}_1' = \frac{1}{\sqrt{2}}\left(\frac{\mathbf{c}}{\|\mathbf{c}\|} + \frac{\mathbf{d}}{\|\mathbf{d}\|}\right)$$

$$\mathbf{R}_2' = \frac{1}{\sqrt{2}}\left(\frac{\mathbf{c}}{\|\mathbf{c}\|} - \frac{\mathbf{d}}{\|\mathbf{d}\|}\right)$$

$$\mathbf{R}_3 = \mathbf{R}_1' \times \mathbf{R}_2'$$

$$\boxed{\mathbf{R} = [\mathbf{R}_1'\ \mathbf{R}_2'\ \mathbf{R}_3]}$$

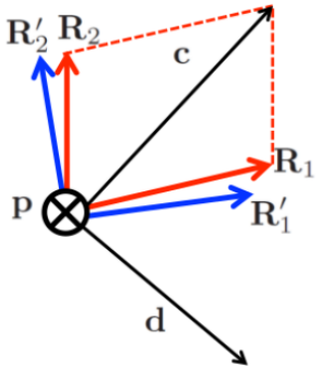


图 3-12 R 的估计

一旦得到了这个基底(R_1',R_2')，那么将 R_3 的值作为 R_1' 和 R_2' 的叉乘就很容易了这个矩阵将是相机校准矩阵 A 与[R_1' R_2' R_3 t]的乘积。

所以，3D 投影矩阵=A*[R_1' R_2' R_3 t]

下面是 3D 投影矩阵的算法流程：

（1）推导出投影成像的数学模型。

（2）通过关键点匹配和 RANSAC 算法对单应性进行启发式估计。（推导出 H）

（3）估计摄像机校准矩阵。（推导出 A）

（4） 根据单应性和摄像机标定矩阵的估计以及 1 推导的数学模型，计算 G_1、G_2 和 t 的值。

（5）在平面 (R_1', R_2') 中找到一个与(G_1,G_2)相似的标准正交基，从中计算 R_3 并更新 t 的值。

3.1.1.4　将 3D 模型投影到图像中并进行绘制。（Project and draw model）

在估计单应性之后，我们将参考曲面的四个角投影到目标图像上，并用一条直线将它们连接起来，我们应该期望得到的直线将参考曲面包围在目标图像中。

AR 的投影的 3D 模型为一只狐狸，效果如下：

图 3-13 AR 投影效果

3.1.2 YoloV5 目标检测算法去除背景信息

快速目标检测首选是 Yolo 系列的模型。下面是 Yolov1 到 Yolov4 的回顾：

Yolov1 的核心思想是将整张图片作为网络的输入，直接在输出层对锚框的位置和类别进行回归。

Yolov2 提出了联合训练算法，这种算法可以把这两种的数据集混合到一起，使用一种分层的观点对物体进行分类，用巨量的分类数据集数据来扩充检测数据集，从而把两种不同的数据集混合起来。

Yolov3 的网络结构是比较经典的 one-stage 结构，分为输入端、Backbone、Neck 和 Prediction 四个部分。

Yolov4 在 Yolov3 的基础上进行了很多的创新，输入端采用 mosaic 数据增强，Backbone 上采用了 CSPDarknet53、Mish 激活函数、Dropblock 等方式，Neck 中采用了 SPP、FPN+PAN 的结构，输出端则采用 CIOU_Loss、DIOU_nms 操作。

Yolov5 是 Yolov4 的进一步升级，主要体现在下面几个方面：

（1）Mosaic 数据增强

Yolov5 的输入端采用了和 Yolov4 一样的 Mosaic 数据增强的方式。4 张图片拼接、随机缩放、随机裁剪、随机排布 4 种方式对数据进行增强，丰富了数据集，同时减少 GPU 的计算。

（2）自适应锚框计算

在 Yolo 系列的算法中，针对不同的数据集，都会有初始设定长宽的锚框。在网络训练中，网络在初始锚框的基础上输出预测框，进而和真实框 ground truth 进行比对，计算两者差距，再反向更新，迭代网络参数，算法步骤如下：

- Step1：读取训练集中所有图片的 w、h 以及检测框的 w、h。
- Step2：将读取的坐标修正为绝对坐标。
- Step3：使用 Kmeans 算法对训练集中所有的检测框进行聚类，得到 k 个 anchors。

- Step4：通过遗传算法对得到的 anchors 进行变异，如果变异后效果好将其保留，否则跳过。

- Step5：将最终得到的最优 anchors 按照面积返回。

（3）自适应图片缩放

在常用的目标检测算法中，不同的图片长宽都不相同，因此常用的方式是将原始图片统一缩放到一个标准尺寸，再送入检测网络中。letterbox 自适应图片缩放技术尽量保持高宽比，缺的用灰边补齐达到固定的尺寸。

Yolov5 网络的整体结构图如下：

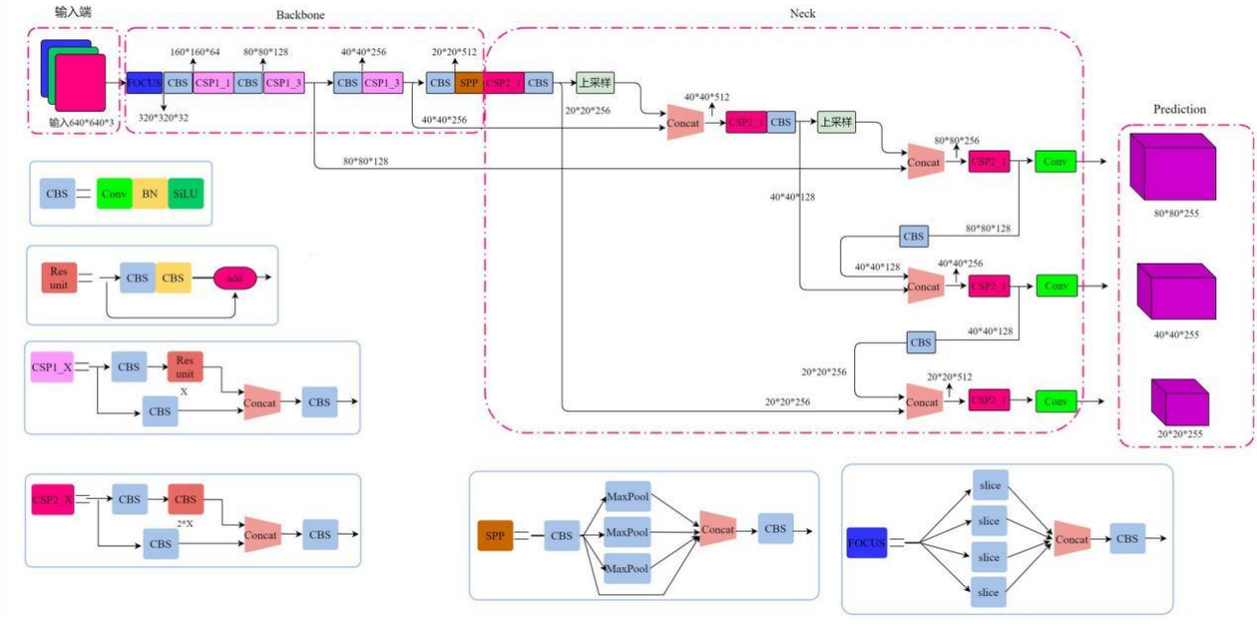


图 3-14 Yolov5 网络的整体结构图

上图即 Yolov5 的网络结构图，可以看出，还是分为输入端、Backbone、Neck、Prediction 四个部分。

（1）BackBone 模块

主要进行特征提取，将图像中的物体信息通过卷积网络进行提取，用于后面目标检测。

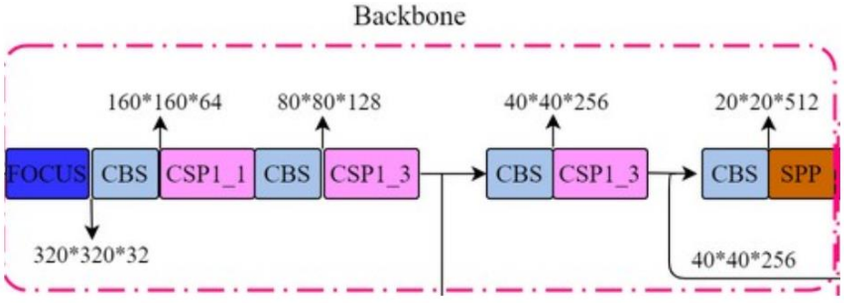


图 3-15 BackBone 模块

（2）Focus 模块

Focus 层原理和 PassThrough 层很类似。它采用切片操作把高分辨率的图片拆分成多个低分辨率的图片/特征图，即隔列采样+拼接。

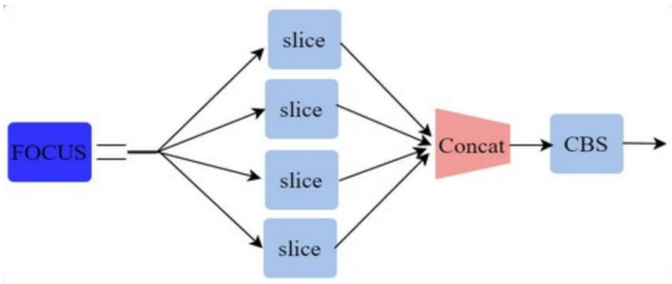


图 3-16 Focus 模块

（3）SPP 模块

空间金字塔池化，能将任意大小的特征图转换成固定大小的特征向量。

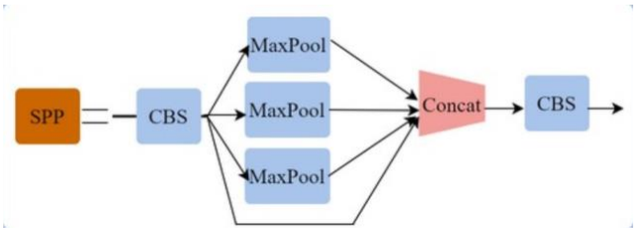


图 3-17 SPP 模块

（4）Neck 模块

对特征进行混合与组合，增强网络的鲁棒性，加强物体检测能力，并且将这些特征传递给 Head 层进行预测。

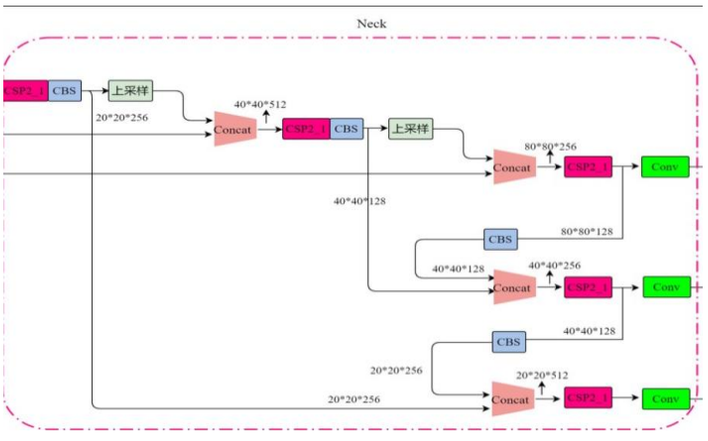


图 3-18 Neck 模块

（5）NMS 非极大值抑制

判断相邻网格识别的是否是同一物体，消除掉多余检测框。

（6）Prediction

进行最终的预测输出。

目标检测的主体为一个人，效果图如下：

图 3-19 目标检测最终效果图

3.1.3 Canny 算法提取边缘

采用数字图像处理中的 Canny 算法，可以快速得到图片的边缘，算法的步骤为 ：

（1）使用高斯滤波器来平滑图像，达到滤除噪声的效果。

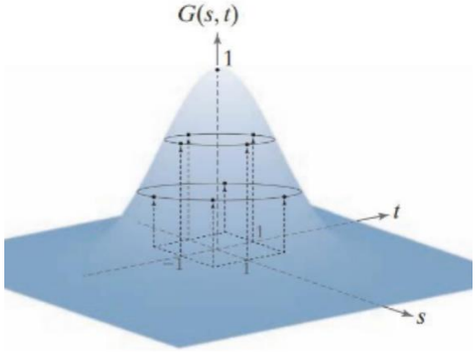


图 3-20 高斯滤波器

$$G(s,t) = Ke^{-\frac{s^2+t^2}{2\sigma^2}}, K = \frac{1}{2\pi\sigma^2}$$

if $r^2 = s^2 + t^2$

then $G(r) = Ke^{-\frac{r^2}{2\sigma^2}}$

r 为距离高斯核的中心的欧式距离

（2）计算图像中每个像素点的梯度大小和方向。

在图像中，用梯度来表示灰度值的变化程度和方向。

$$\textbf{梯度大小：}\ G=\sqrt{G_x^2+G_y^2}$$

$$\textbf{梯度方向：}Angle(\theta)=tan^{-1}(\frac{G_y}{G_x})$$

（3）使用非极大值抑制，消除边缘检测带来的不利影响。

遍历图像中的所有像素点，判断当前像素点是否是周围像素点中具有相同方向梯度的最大值。如果是梯度最大的像素点，就保留，否则就抑制。这一步的目的是将模糊（blurred）的边界变得清晰（sharp）。保留了每个像素点上梯度强度的极大值，而删掉其他的值。

（4）双阈值法检测确定真实和潜在的边缘。

minVal 表示最小的阈值，maxVal 表示最大的阈值。如果梯度值大于 maxVal，则处理为边界，如果 minVal<梯度值<maxVal，则连有边界就保留。如果像素的梯度值<minVal，则舍弃该像素。

（5）通过抑制孤立的弱边缘完成边缘检测。

一般可定义只要其中邻域像素其中一个为强边缘像素点，则该弱边缘就可以保留为强边缘，即真实边缘点。由真实边缘引起的弱边缘像素点将连接到强边缘像素点，而噪声响应则未连接。通过查看弱边缘像素及其 8 个邻域像素，可根据其与强边缘的连接情况来进行判断。

作品的效果展示：

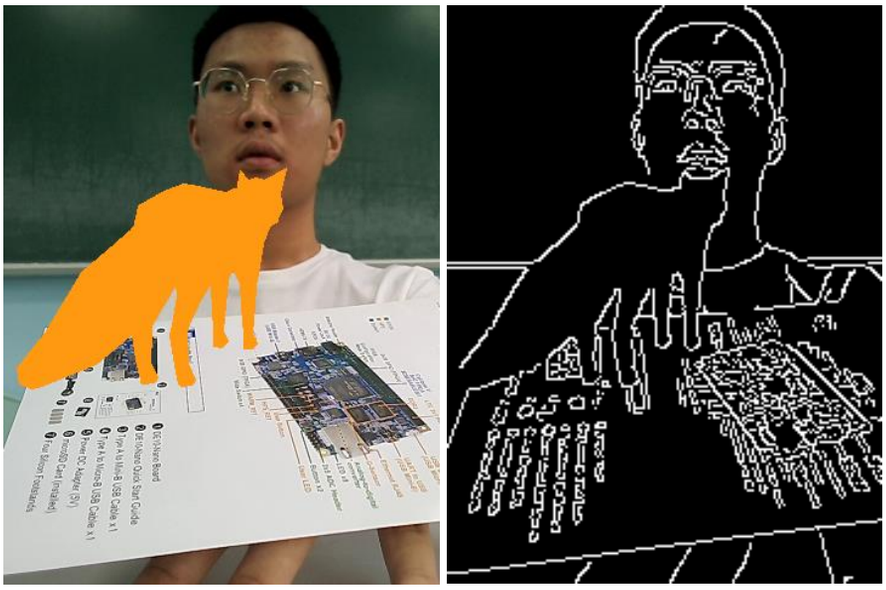


图 3-21 最终图画效果图

3.1.4 机械臂绘图的点的序列规划

由于经过 AR 处理之后的图片所给出的数据是二值化之后的根据 X 轴坐标从小至大所排列的像素坐标数据，所以我们还需对点的坐标数据进行矢量化，并对其重新排列，处理，使得其符合机械臂绘画时的笔顺以及加快绘画的速度。具体操作步骤如下:

（1）预处理

根据图片的分辨率创建一个二维数组 img 用于保存像素点的信息，同时创建一个同样大小的二维数组 flag 用于保存点是否已经被绘画。初始化两个栈，分别用于保存目标点四周八个像素点的情况 以及分支点序列。

（2）栅格化处理

将像素点根据其 X,Y 坐标重新填入到数组 img 中，使其能够以像素的形式还原图片。

（3）寻找未使用的点

按照 X，Y 从小到大的顺序遍历 img 数组，寻找 img 数组中为 1，且 flag 数组中为 0，即未绘画的像素点，进行第四步。

（4）判断四周八个像素点状况

判断根据四周八个像素点中的未绘画的像素点，并将未绘画的像素点的坐标序列存入栈中返回，进行分类讨论:

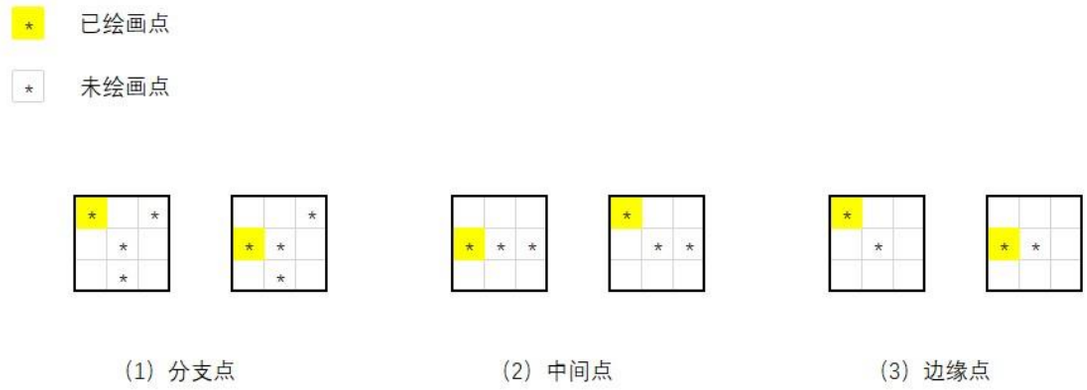
若未绘画的点数量大于等于 2 个，则为分支点，将 flag 标志置为 1，输出至绘画点序列文件中，并其坐标存储进分支点栈。最后将未绘画点出栈，继续进行第四步。

若未绘画的点等于 1 个，则为中间点，将 flag 标志置为 1，输出至绘画点序列文件中，最后将未绘画点出栈，继续进行第四步。

若未绘画的点等于 0 个，则为边缘点，进一步判断该点 flag 标志，以排除特殊情况干扰：

若 flag 标志位为 0，即未经过绘画，则将 flag 标志置为 1，输出至绘画点序列文件中，并设置提笔标志，表示一根线条绘画完毕，之后将分支点出栈，继续进行第四步操作，绘画分支点。若分支点栈空则跳转至第三步。

若 flag 标志位为 1，即已经过绘画，为特殊状态误判(详见第 5 点)，不进行绘画与输出，直接继续将分支点出栈，进行第四步。若分支点栈空则跳转至第三步。



（5）特殊状态误判

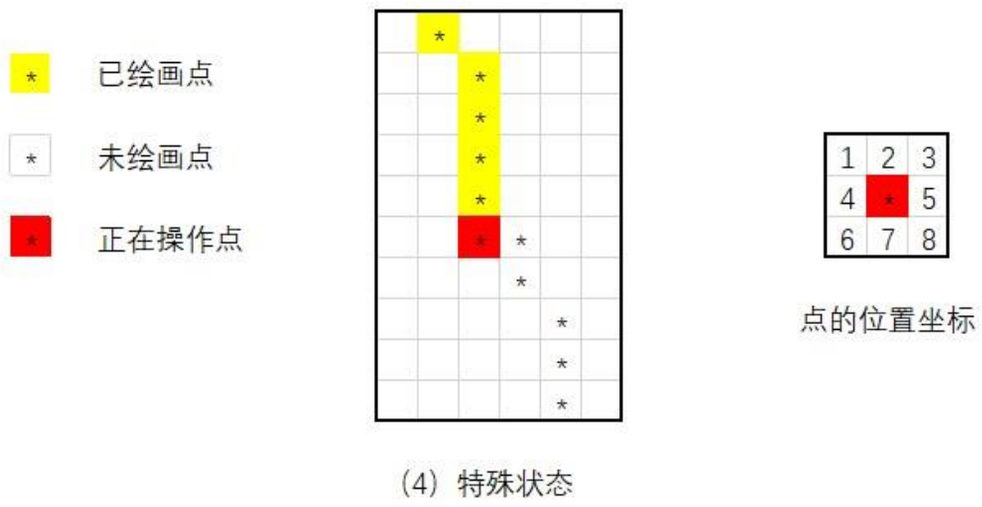


图 3-22 像素点类型

可以看见上图中虽然红点并非分支点，但是根据算法会被归类为分支点之列。所以在返回四周点的坐标栈时，角上的，即 1，3，6，8 位置的未绘画点会处于栈的底部，而边上的，即 2，4，5，7 位置的未绘画点会处于栈的顶部，使得边上的未绘画点会先于角上的未绘画点处理，于是在对边上点进行操作时，其下一个点一定会是角上的点。以此来消除特殊情况。在第四步中对于四周无未绘画点的情况一共只有两种，一种是一条直线的末端，另一种就是此特殊情况，所以需要特判来消除。

3.2 硬件框图

信步边缘 AI 计算平台 GNS-V40 连接摄像机、鼠标、显示器和键盘，拍摄者手持 AR 投影的参考平面，AR 算法将在参考平面上投影一个 3D 模型的立体投影效果，然后利用 YoloV5 算法获取拍摄者的主体锚框，对图像根据锚框进行裁剪来去除背景等无用信息，将裁剪后的图像经过 Canny 算法处理后快速得到边缘二值图像，将边缘线条上点的二维坐标提供给机械臂进行点的序列规划绘制具有 AR 增强现实效果的线稿。

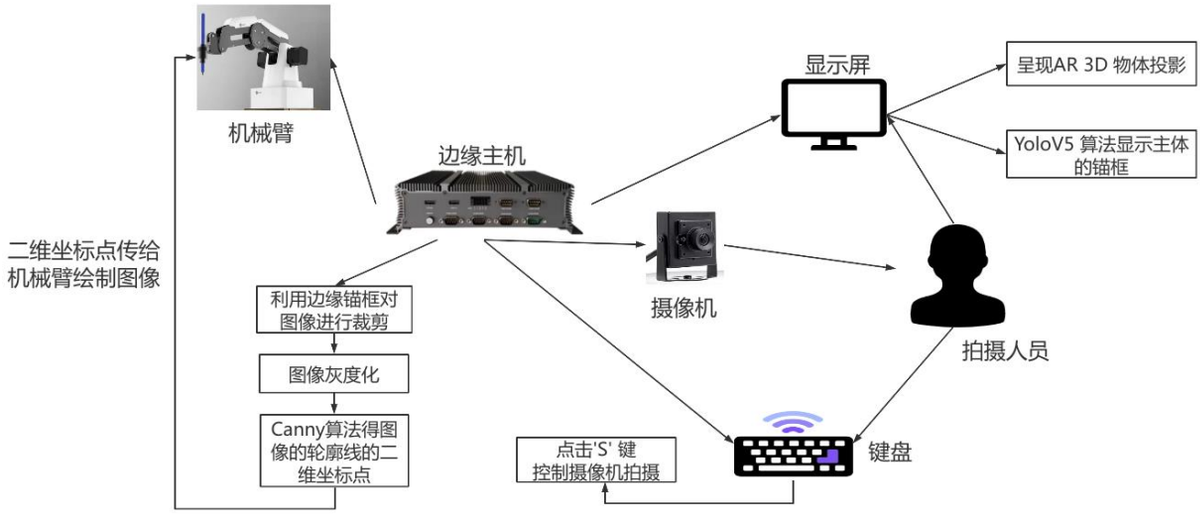

机械臂
边缘主机
显示屏
呈现AR 3D 物体投影
YoloV5 算法显示主体的锚框
二维坐标点传给机械臂绘制图像
利用边缘锚框对图像进行裁剪
图像灰度化
Canny算法得图像的轮廓线的二维坐标点
摄像机
拍摄人员
点击'S' 键控制摄像机拍摄
键盘

图 3-23 硬件框图

第四章 作品测试与分析

4.1 系统测试方案

4.1.1 AR 增强现实算法稳定性测试

我们项目的增强现实的稳定性主要和摄像机的视角和拍摄人员拿的参考平面角度有关，我们对拍摄人员拿参考平面的不同角度对 AR 生成的效果进行了测试，通过记录 AR 模型稳定投影的时间来评估 AR 增强现实算法的稳定性。

4.1.2 YoloV5 目标检测算法准确性和稳定性测试

由于 YoloV5 目标检测的对象为一个人，我们测试需要改变摄像机画面中拍摄人物的数量和拍摄环境的光照和背景信息来对目标检测算法的准确性和稳定性进行评估。

4.1.3 Canny 边缘检测算法的鲁棒性测试

测试不同的图像的边缘生成的效果，在不同的光照环境下使用 Canny 算法生成边缘图像，来检验 Canny 算法提取图像边缘的良好的鲁棒性。

4.1.4 机械臂绘制图像的精度测试

机械臂加速度的大小影响着绘画的速度，但过大的加速度还会降低机械臂的精度，影响图像的绘制。我们将同一张图片使用不同的加速度来进行绘制，再将其放置在一起比较，最后综合考虑绘制时间以及精度来选择出最为合适的加速度。

4.2 测试设备

为了验证系统的可行性、准确性和鲁棒性，我们应用的测试设备有（如图 4-1 所示）

（1）高清工业相机

实时获取拍摄人员的画面，以便接收并处理。

（2）高清显示屏、鼠标和键盘

用来显示 AR 增强现实的 3D 物体和拍摄人员，控制拍摄的正常进行。

（3）信步边缘 AI 计算平台 GNS-V40

主要的计算平台，包括 AR 3D 物体的实时投影绘制、YoloV5 目标检测、Canny 算法和绘制图像点的序列规划。

（4）DOBOT 魔术师轻量机械臂

根据 Canny 算法得到的点的坐标完成绘制图像的任务。

图 4-1 测试环境

4.3 测试数据

4.3.1 AR 增强现实算法稳定性测试

为了验证我们的 AR 增强现实算法能够适应不同倾斜角度的参考平面，我们在不同参考平面的角度下测试了 3D 模型稳定存在的时间，从而验证算法的稳定性。

测试组别	参考屏幕的倾斜角度	AR 模型的稳定时间
A1	90°	4.13s
A2	45°	>10s
A3	0°	1.65s

表 4-1 稳定性结果

由于测试的参考平面在拍摄时是近乎 45 度倾斜拍摄，在测试时参考平面倾斜 45 度 AR 投影能稳定存在，进一步表明了算法的稳定性和可行性。

4.3.2 YoloV5 目标检测算法准确性和稳定性测试

由于我们的目标检测算法的任务是准确识别拍摄人员，且其他物体不对识别造成干扰，我们测试了画面中不同的的背景干扰下的 YoloV5 识别效果。

测试组别	画面中人物数量（/人）	YoloV5 锚框持续时间
A1	1	>10s
A2	2	3s
A3	3	1s

表 4-2 识别效果

通过测试结果可以看出当画面中人物数量为一人时，YoloV5 持续时间大于 10s（稳定存在），而我们的目标是锚框能将画面中的一个拍摄人员持续包含在内，结果证明了 YoloV5 算法对于本项目的的可行性、准确性和稳定性。

4.3.3 Canny 边缘检测算法的鲁棒性测试

数字图像画面的复杂性对 Canny 算法的鲁棒性效果影响最大，我们测试了不同的复杂的图像下的生成效果，下面是我们的实验结果：

测试组别	不同的复杂程度的图像	Canny 算法生成的视觉效果
A1	物体简笔画	（1）
A2	单一人物	（2）
A3	风景画	（3）
A4	AR+人	（4）

表 4-3 实验结果

（1）测试组 A1

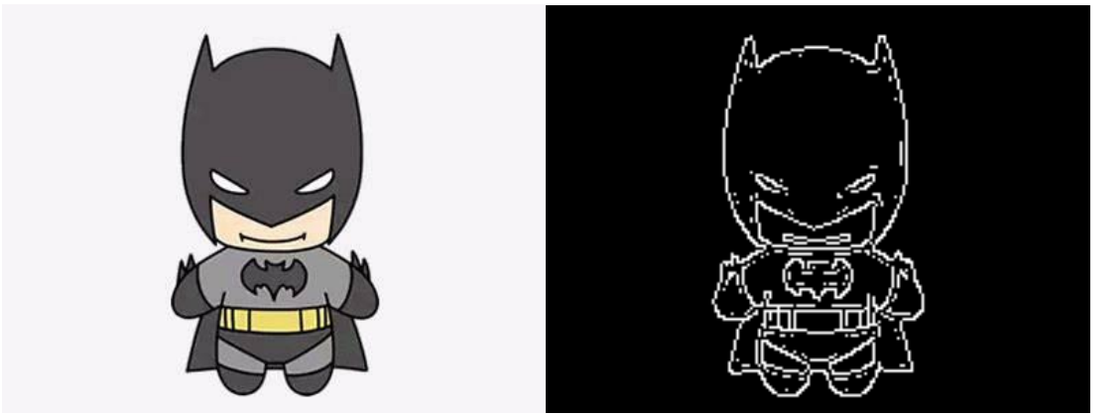


图 4-2 测试组 A1

（2）测试组 A2

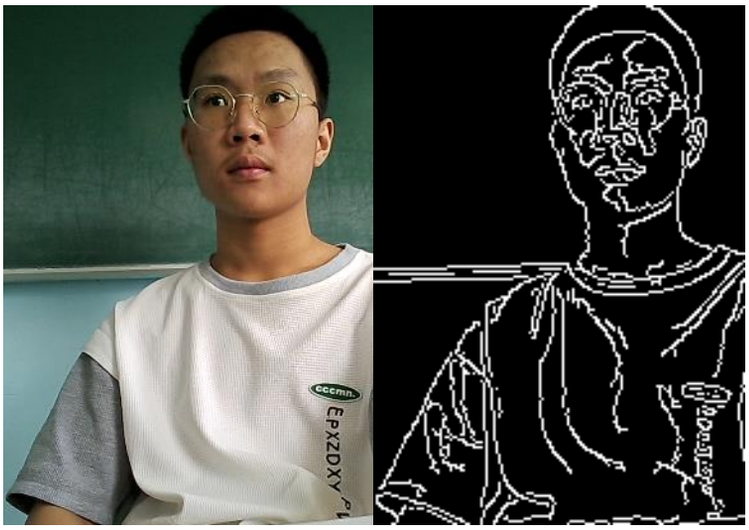
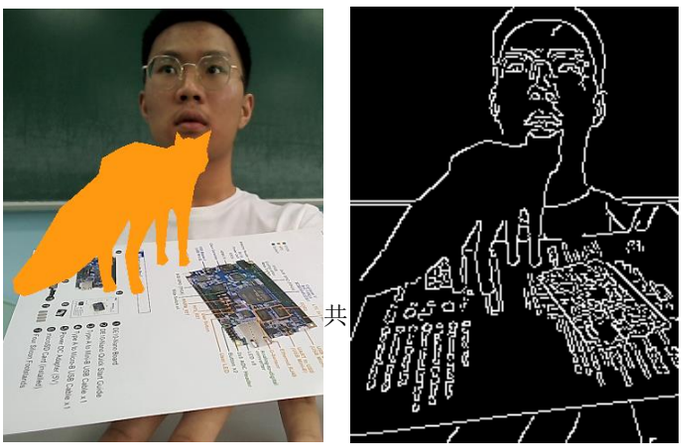


图 4-3 测试组 A2

（3）测试组 A3

图 4-4 测试组 A3

（4）测试组 A4



共

图 4-5 测试组 A4

我们比较了不同复杂程度的画面下 Canny 算法的生成效果，理论上 Canny 算法对于物体线条轮廓明显的图效果较好，但对于复杂物体如人，风景可能会丢失部分细节。但对于我们的项目，Canny 算法的速度快，且画面细节丢失在可接受的范围内，Canny 算法在提取边缘轮廓方面是可行的。

4.3.4 机械臂绘制图像的精度测试

机械臂的加速度大小影响着绘制时间的长短，但过大的加速度还会使绘制的精度有所下降。在速度与精度之间寻找一个合适的加速度尤为重要，我们的实验结果如下：

测试组别	点的数量	加速度设置(mm/s²)	绘制时间（s）	绘制效果
A1	1384	200	384	(1)
A1	1384	500	313	(1)
A1	1384	800	274	(1)
A1	1384	1000	250	(1)
A2	4605	200	1098	(2)
A2	4605	500	896	(2)
A2	4605	800	783	(2)
A2	4605	1000	715	(2)

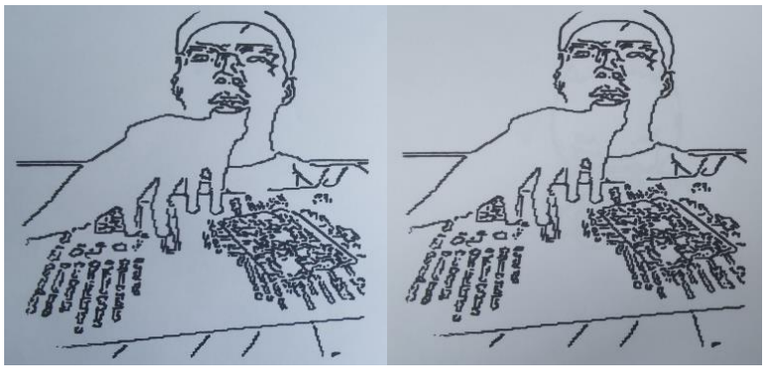
表 4-4 实验结果

（1） 测试组 A1

图 4-6 测试组 A1

从左到右的加速度分别为 200，500，800，1000(mm/s²),可以看见，随着加速度的变大，绘制的时间也在缩短，但当机械臂的加速度达到 1000 时，机械臂的抖动较为剧烈，导致绘制的图片粗糙感较强，绘制精度有些许下降。

（2） 测试组 A2



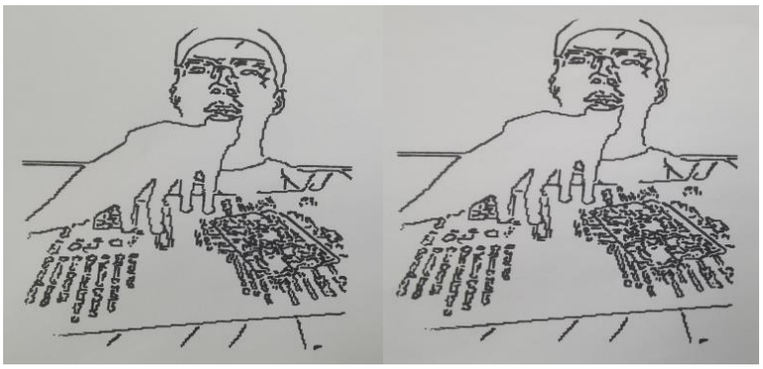


图 4-7 测试组 A2

从前到后的加速度分别为 200，500，800，1000(mm/s²),也可以得出相似的结论。最后，我们在综合比较之后，将加速度定在了 800mm/s²

4.4 结果分析

在系统测试中，我们通过现场对拍摄人员进行实时摄像的方式分别对基于 OpenCV 的 AR 增强现实算法、基于 YoloV5 和 OpenVINO 的目标检测算法、基于 Canny 的图像边缘提取算法和机械臂点路径规划绘制图像进行了准确度、精度和鲁棒性的分析。

下面是我们基于 AR 的图像自动绘画机的生成示意图：

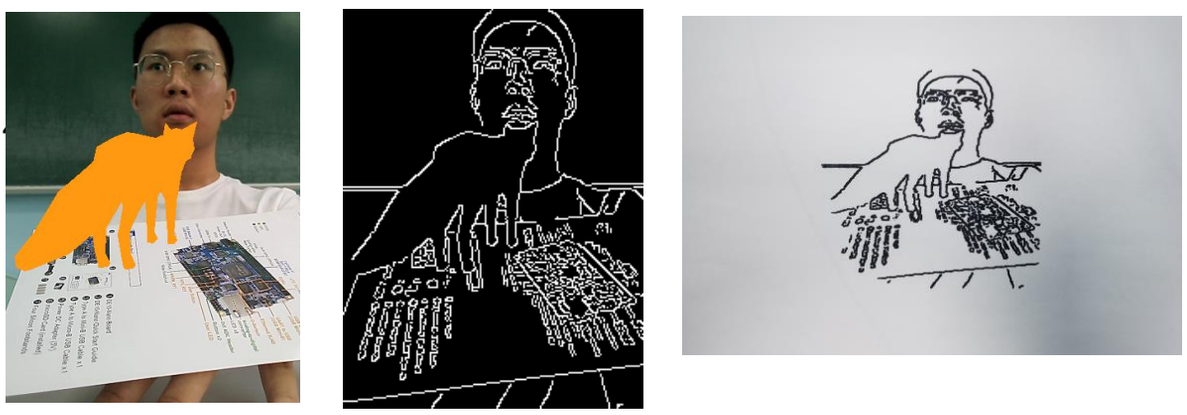


图 4-8 生成示意图

从结果可以看出，Canny 算法完成十分迅速，人物主体和 3D 模型的轮廓特点都有所保留，但细节仍然有待进一步提高和改进

4.5 实现功能

本系统基本实现了通过本地摄像头在拍摄人员的参考平面中添加 AR 增强现实的 3D 物体和添加 YoloV5 算法的人物锚框完成了对视频流的实时处理，显示器上将呈现拍摄人员被锚框包围的含有 AR 增强现实 3D 物体的画面，当拍摄人员认为当前视频帧的画面满意时，可以通过敲击键盘的'S'键实现对视频画面的剪裁，仅含有拍摄主体的图像将存入边缘主机中。系统通过边缘主机使用 Canny 算法完成图像的边缘轮廓线的提取，最后将边缘轮廓线的二维点的坐标传送给机械臂。机械臂获取二维坐标点后将通过路径规划算法完成最后的绘制图像的任务。所有的算法应用与数据处理都在携带有 Intel 处理的信步边缘 AI 计算平台 GNS-V40 上进行的，实现了真正的设备边缘计算，具有着速度快、设备安全、可扩展性、与可靠性，为基于 AR 的自动绘画机提供了一种可行的解决方案。

第五章 产品特色与项目总结

5.1 产品特色

5.1.1 AR 增强现实算法的实现

本项目的增强现实以 OpenCV 作为基础工具包，创新性地将 ORB 算法获取的二进制字符串作为参考平面的特征描述符，特征匹配通过设定一个阈值，利用汉明距离作为度量距离来实现特征匹配，使得 AR 投影的稳定性大幅度提高，计算成本大幅度降低。在平面姿态估计中，利用针孔模型对计算进行了简化，使用了 RANSAC 算法完成了单应矩阵的估计，将一个复杂的 AR 增强现实问题转化为了一个矩阵变换的问题，由于矩阵变换使用 numpy，使得 AR 算法能够完成实时的投影，大幅度减小了计算量，缩短了延时带来的危害。

5.1.2 OpenVINO 和 Intel Core 软硬件协同加速 YoloV5 目标检测算法

本项目考虑到背景信息可能对 Canny 算法和机械臂绘制图像造成不必要的干扰，综合考虑模型的可行性和准确度需求，基于 Intel 的 openVINO 工具套件，创新性地提出了将 YoloV5 目标检测模型部署在了 Intel 处理器的信步边缘 AI 计算平台 GNS-V40 上，实现了和 Intel Core 处理器的软硬件协同，提高了目标检测主体图像的效率，提高了深度学习网络的推理性能，满足了系统对实时性的要求，提高了拍摄人员的用户体验。

5.1.3 Canny 算法实现了图像轮廓线的快速提取

我们采用 Canny 算法快速得到了一张彩色的数字图像的的边缘轮廓线，在保留图像的基本精度的前提下，提高了算法运行的速度，舍弃了通过少量数据集训练一个 pix2pix 对抗生成网络的方案，减少了不必要的推理时间成本，便于在短时间内快速得到图像线稿的二维坐标点，便于迅速将点的坐标提供给机械臂进行绘制图像。绘制的结果表明传统的数字图像处理方法在精度上丢失并不明显。

5.1.4 点的序列规划在机械臂绘图的应用

本项目在点的矢量化以及深度优先搜索算法思想的基础上，创新性地提出了点的序列规划算法，能够将无序的物体轮廓点经过排序处理之后输出符合机械臂绘图笔顺的点的序列，并且能够识别出线条的末端，给出相应的提笔标志。这不仅能够解决笔顺问题，还能够加快机械臂的运行速度，减少不必要的提笔操作，增加了绘画速度。

5.2 项目总结

参赛作品《基于 AR 图画处理的自动绘画机》历时三个多月，作品基本完成，进入收尾阶段。回想起过去研究与完成作品的过程，寻找主题，查阅相关资料文献，确定作品方向，实施方案与寻找创新点，并制定详细的研究方案与步骤，在指导老师和实验室同学们的帮助下，最终完成了整个系统。完整地实现这个项目，对我们来讲既是一种挑战，也是一种收获：

（1）迎难而上的勇气与坚不可摧的信念

在这个项目中，对于我们来说印象最深刻的困难，就是在获取图像边缘之后，输出的数据并非是已经按照机械臂绘画时的笔顺排列完毕的点，而是根据 x 轴坐标的大小排列的一列列无序的点。这就让绘图变得无从下手：从哪一个点开始绘画？绘画完这一个点之后接下来绘画哪一个？什么时候需要提笔，而什么时候又不需要提笔？我们先后试用了多个算法，查

阅了诸多文献以及资料，虽然每次都较上一次有了一定的进步，解决了一个或两个上一个算法中所存在的问题，但都没有找到能够完美解决这个问题的办法。最后，在一次与老师的交谈中提到了这个问题，老师在经过思考之后也给了我们提示：点的矢量化。于是我们顺着这条线索，查阅了诸多相关资料之后，将点的矢量化与深搜的思想结合在一起，最终终于解决了这个问题，不仅加快了机械臂绘画的速度，并且也让机械臂的绘画变得更加流畅以及连贯。虽然我们在这个问题上耽搁了较长的时间，但我们明白，成功是离不开这一次又一次的失败。这样的失败对我们而言，不仅是一种成长，更是一种收获。我们的进步离不开这一次次的成长，而最终的成功，又鼓舞了我们心中那份迎难而上的勇气与坚不可摧的信念。

（2）理论结合实际

评价一个作品成功与否，不仅要看这个作品是否有着一定的现实意义，还要看他的使用价值。我们在设计项目的同时，应当将自己的眼光放长远一些，去学习最新的研究方向以及热点，不应该固步自封，闭门造车。自从元宇宙的概念近年来逐渐火热起来，AR 增强现实和 VR 虚拟现实都是十分热门的研究方向，我们充分结合深度学习、数字图像处理和计算机图形学所学的理论知识，查阅了许多资料，最终实现了机械臂绘图和 AR 增强现实的结合，很好地诠释了理论结合实践，从现实生活中学习。

（3）团队的力量

在整个项目的制作中，团队的合作素养，交流与同进退的协作工作能力也不断让我们意识到团队合作的重要性。从开始到结尾，每个人都时刻在提出自己的意见与学习心得，工作的分担以及相互的交流让我们少走了很多弯路。

当然，项目之中仍存有一些可以拓展的地方，如可以增加更加丰富的 3D 物体的增强现实，使 AR 的效果更加逼真；绘画不仅可以绘制单一线条的线稿，还可以绘制其他风格的彩色绘画如油画，水墨画等来使作品更加完善。如若今后再次有机会能够接手相关任务，希望我们可以做出一个更优秀的基于 AR 的自动绘画机。

最后，一个项目的完成，离不开团队成员的努力，更离不开导师的指导，学校的支持，以及实验室同学的鼓励与帮助，在项目的最后对他们表以最衷心的感谢！

参考文献

[1]靳艳红. 基于改进 canny 算法的图像边缘检测的研究[D].重庆师范大学,2011.
[2]戴燕. 图像边缘检测与应用[D].西安科技大学,2010.
[3]金刚. 自适应 Canny 算法研究及其在图像边缘检测中的应用[D].浙江大学,2009.
[4]吉冬玉. 基于 ORB 算法的图像匹配方法研究[D].陕西师范大学,2016.
[5]李磊. 增强现实系统中的实时稳定姿态跟踪[D].华中科技大学,2017.
[6]曾晶晶. 基于点特征的图像配准算法研究[D].西安电子科技大学,2021.DOI:10.27389/d.cnki.gxadu.2021.001672.
[7]刘彦清. 基于 YOLO 系列的目标检测改进算法[D].吉林大学,2021.DOI:10.27162/d.cnki.gjlin.2021.005205.
[8]陈扬. 复杂场景下的 YOLOv5 目标检测算法的改进方法研究[D].湖北师范大学,2022.DOI:10.27796/d.cnki.ghbsf.2022.000248.
[9]刘晓蓉. 基于视频的运动目标检测与跟踪方法研究[D].西南科技大学,2022.DOI:10.27415/d.cnki.gxngc.2022.000120.
[10]王光耀. 基于深度强化学习的目标检测算法与应用研究[D].吉林大学,2022.DOI:10.27162/d.cnki.gjlin.2022.000210.
[11]武剑峰.简单的位图矢量化[J].科技展望,2016,26(01):196.
[12]张鹏. 大面积皮革轮廓的视觉检测技术与应用研究[D].南京理工大学,2020.DOI:10.27241/d.cnki.gnjgu.2020.001967.

附录

（1）AR 投影代码

```
import os

import numpy as np
import pygame
from OpenGL.GL import *


def MTL(dir_name, filename):
    contents = {}
    mtl = None
    for line in open(dir_name + filename, "r"):
        if line.startswith('#'): continue
        values = line.split()
        if not values: continue
        if values[0] == 'newmtl':
            mtl = contents[values[1]] = {}
        elif mtl is None:
            raise (ValueError, "mtl file doesn't start with newmtl stmt")
        elif values[0] == 'map_Kd':
            # load the texture referred to by this declaration
            mtl[values[0]] = values[1]
            surf = pygame.image.load(dir_name + mtl['map_Kd'])
            image = pygame.image.tostring(surf, 'RGBX', 1)
            ix, iy = surf.get_rect().size
            texid = mtl['texture_Kd'] = glGenTextures(1)
            glBindTexture(GL_TEXTURE_2D, texid)
            #Texture Filtering
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
                            GL_LINEAR)
            glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
                            GL_LINEAR)
            glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA, ix, iy, 0, GL_RGBA,
                         GL_UNSIGNED_BYTE, image)
        else:
            mtl[values[0]] = [float(x) for x in values[1:4]]

    return contents
```

```
class OBJ:
    def __init__(self, filename, swapyz=False):
        """Loads a Wavefront OBJ file. """
        self.vertices = []
        self.normals = []
        self.texcoords = []
        self.faces = []
        material = None
        dir_name = os.getcwd() + '/src/models/'
        for line in open(filename, "r"):
            if line.startswith('#'): continue
            values = line.split()
            if not values: continue
            if values[0] == 'v':
                v = [float(x) for x in values[1:4]]
                if swapyz:
                    v = v[0], v[2], v[1]
                self.vertices.append(v)
            elif values[0] == 'vn':
                v = [float(x) for x in values[1:4]]
                if swapyz:
                    v = v[0], v[2], v[1]
                self.normals.append(v)
            elif values[0] == 'vt':
                self.texcoords.append([float(x) for x in values[1:3]])
            elif values[0] in ('usemtl', 'usemat'):
                material = values[1]
            elif values[0] == 'mtllib':
                self.mtl = MTL(dir_name, values[1])
            elif values[0] == 'f':
                face = []
                texcoords = []
                norms = []
                for v in values[1:]:
                    w = v.split('/')
                    face.append(int(w[0]))
                    if len(w) >= 2 and len(w[1]) > 0:
                        texcoords.append(int(w[1]))
                    else:
                        texcoords.append(0)
                    if len(w) >= 3 and len(w[2]) > 0:
                        norms.append(int(w[2]))
                    else:
```

```
                                norms.append(0)
                        self.faces.append((face, norms, texcoords, material))


                        # self.faces.append((face, norms, texcoords))
                self.create_gl_list()

        def create_gl_list(self):
                self.gl_list = glGenLists(1)
                glNewList(self.gl_list, GL_COMPILE)
                glEnable(GL_TEXTURE_2D)
                glFrontFace(GL_CCW)

                for face in self.faces:
                        vertices, normals, texture_coords, material = face

                        mtl = self.mtl[material]
                        if 'texture_Kd' in mtl:
                                # use diffuse texmap
                                glBindTexture(GL_TEXTURE_2D, mtl['texture_Kd'])
                        else:
                                # just use diffuse colour
                                glColor(*mtl['Kd'])
                                glColor(*mtl['Ka'])
                                glColor(*mtl['Ks'])

                        glBegin(GL_POLYGON)
                        for i in range(len(vertices)):
                                if normals[i] > 0:
                                        glNormal3fv(self.normals[normals[i] - 1])
                                if texture_coords[i] > 0:
                                        glTexCoord2fv(self.texcoords[texture_coords[i] - 1])
                                glVertex3fv(self.vertices[vertices[i] - 1])
                        glEnd()
                glDisable(GL_TEXTURE_2D)
                glEndList()
import math
import cv2
import numpy as np

MIN_MATCHES = 120
DEFAULT_COLOR = (18, 153, 255)
rectangle = False
target_name = '3.jpg'
```

```
model_name = 'low-poly-fox-by-pixelmannen.obj'


def projection_matrix(camera_parameters, homography):
    """
    From the camera calibration matrix and the estimated homography
    compute the 3D projection matrix
    """
    # Compute rotation along the x and y axis as well as the translation
    homography = homography * (-1)
    rot_and_transl = np.dot(np.linalg.inv(camera_parameters), homography)
    col_1 = rot_and_transl[:, 0]
    col_2 = rot_and_transl[:, 1]
    col_3 = rot_and_transl[:, 2]
    # normalise vectors
    l = math.sqrt(np.linalg.norm(col_1, 2) * np.linalg.norm(col_2, 2))
    rot_1 = col_1 / l
    rot_2 = col_2 / l
    translation = col_3 / l
    # compute the orthonormal basis
    c = rot_1 + rot_2
    p = np.cross(rot_1, rot_2)
    d = np.cross(c, p)
    rot_1 = np.dot(c / np.linalg.norm(c, 2) + d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
    rot_2 = np.dot(c / np.linalg.norm(c, 2) - d / np.linalg.norm(d, 2), 1 / math.sqrt(2))
    rot_3 = np.cross(rot_1, rot_2)
    # finally, compute the 3D projection matrix from the model to the current frame
    projection = np.stack((rot_1, rot_2, rot_3, translation)).T
    return np.dot(camera_parameters, projection)


def hex_to_rgb(hex_color):
    """
    Helper function to convert hex strings to RGB
    """
    hex_color = hex_color.lstrip('#')
    h_len = len(hex_color)
    return tuple(int(hex_color[i:i + h_len // 3], 16) for i in range(0, h_len, h_len // 3))


def render(img, obj, projection, model, color=False):
    """
    Render a loaded obj model into the current video frame
    """
```

```
        vertices = obj.vertices
        scale_matrix = np.eye(3) * 3
        h, w = model.shape

        for face in obj.faces:
            face_vertices = face[0]
            points = np.array([vertices[vertex - 1] for vertex in face_vertices])
            points = np.dot(points, scale_matrix)
            # render model in the middle of the reference surface. To do so,
            # model points must be displaced
            points = np.array([[p[0] + w / 2, p[1] + h / 2, p[2]] for p in points])
            dst = cv2.perspectiveTransform(points.reshape(-1, 1, 3), projection)
            imgpts = np.int32(dst)
            if color is False:
                cv2.fillConvexPoly(img, imgpts, DEFAULT_COLOR)
                pass
            else:
                color = hex_to_rgb(face[-2])
                print(color)
                color = color[::-1]    # reverse
                cv2.fillConvexPoly(img, imgpts, color)

        return img
```

（2）Yolov5 代码

```
import cv2
import numpy as np


model_path = "./yolov5/yolov5s.xml"
class_names = ['person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'train', 'truck', 'boat', 'traffic
light',
                'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse',
'sheep', 'cow',
                'elephant', 'bear', 'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase',
'frisbee',
                'skis', 'snowboard', 'sports ball', 'kite', 'baseball bat', 'baseball glove', 'skateboard',
'surfboard',
                'tennis racket', 'bottle',
                'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich',
'orange',
                'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', 'cake', 'chair', 'couch', 'potted plant',
'bed',
                'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone',
'microwave',
```

```
                'oven', 'toaster',
                'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear', 'hair drier',
'toothbrush']


def letterbox(img, new_shape=(640, 640), color=(114, 114, 114), auto=True, scaleFill=False,
scaleup=True):
    shape = img.shape[:2]  # current shape [height, width]
    if isinstance(new_shape, int):
        new_shape = (new_shape, new_shape)

    # Scale ratio (new / old)
    r = min(new_shape[0] / shape[0], new_shape[1] / shape[1])
    if not scaleup:  # only scale down, do not scale up (for better test mAP)
        r = min(r, 1.0)

    # Compute padding
    ratio = r, r  # width, height ratios
    new_unpad = int(round(shape[1] * r)), int(round(shape[0] * r))
    dw, dh = new_shape[1] - new_unpad[0], new_shape[0] - new_unpad[1]  # wh padding
    if auto:  # minimum rectangle
        dw, dh = np.mod(dw, 64), np.mod(dh, 64)  # wh padding
    elif scaleFill:  # stretch
        dw, dh = 0.0, 0.0
        new_unpad = (new_shape[1], new_shape[0])
        ratio = new_shape[1] / shape[1], new_shape[0] / shape[0]  # width, height ratios
    dw /= 2
    dh /= 2

    if shape[::-1] != new_unpad:  # resize
        img = cv2.resize(img, new_unpad, interpolation=cv2.INTER_LINEAR)
    top, bottom = int(round(dh - 0.1)), int(round(dh + 0.1))
    left, right = int(round(dw - 0.1)), int(round(dw + 0.1))
    img = cv2.copyMakeBorder(img, top, bottom, left, right, cv2.BORDER_CONSTANT,
value=color)  # add border
    return img, ratio, (dw, dh)
```

（3）Canny 算法获取边缘

```
import os.path
import cv2
import numpy as np
```

```
def rgbtogray(filename, path):
    img0 = cv2.imread(path + filename)
    img1 = cv2.resize(img0, dsize=None, fx=0.5, fy=0.5)
    img2 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    cv2.imwrite(path + 'gray_' + filename, img2)
    h, w = img1.shape[:2]
    return img2


def imageFilter(img, filter):
    h = len(img)
    w = len(img[0])
    for i in range(h):
        for j in range(w):
            if img[i, j] >= filter:
                img[i, j] = 255
            else:
                img[i, j] = 0
    return img


def getLineOfImage(img, filename, path, filter):
    if not os.path.exists(path):
        os.mkdir(path)
    # Sobel 算子
    x_grad = cv2.Sobel(img, cv2.CV_32F, 1, 0)
    y_grad = cv2.Sobel(img, cv2.CV_32F, 0, 1)
    x_grad = cv2.convertScaleAbs(x_grad)
    y_grad = cv2.convertScaleAbs(y_grad)
    img_sobel = cv2.add(x_grad, y_grad, dtype=cv2.CV_16S)
    img_sobel = cv2.convertScaleAbs(img_sobel)

    img_sobel = imageFilter(img_sobel, filter)
    cv2.imwrite(path + "Sobel_" + filename, img_sobel)
    # Laplacian 算子
    dst = cv2.Laplacian(img, cv2.CV_16S, ksize=3)
    img_laplacian = cv2.convertScaleAbs(dst)

    img_laplacian = imageFilter(img_laplacian, filter)
    cv2.imwrite(path + "Laplacian_" + filename, img_laplacian)
    # Scharr 算子
    x_grad = cv2.Scharr(img, cv2.CV_32F, 1, 0)
    y_grad = cv2.Scharr(img, cv2.CV_32F, 0, 1)
    x_grad = cv2.convertScaleAbs(x_grad)
```

```
    y_grad = cv2.convertScaleAbs(y_grad)
    img_scharr = cv2.add(x_grad, y_grad, dtype=cv2.CV_16S)
    img_scharr = cv2.convertScaleAbs(img_scharr)
    img_scharr = imageFilter(img_scharr, filter)
    cv2.imwrite(path + "Scharr_" + filename, img_scharr)

    # Canny 算子
    img_canny = cv2.Canny(img, 50, 150)
    cv2.imwrite(path + "Canny_" + filename, img_canny)
    return img_sobel, img_laplacian, img_scharr, img_canny


def image2line(filename, path1, path2, path3, filter):
    img2 = rgbtogray(filename=filename, path=path1)
    img_sobel, img_laplacian, img_scharr, img_canny = getLineOfImage(img2,
filename=filename, path=path2, filter=filter)
    getXYLocation(img_sobel, path3, filename[:-4], 'Sobel')
    getXYLocation(img_laplacian, path3, filename[:-4], 'Laplacian')
    getXYLocation(img_scharr, path3, filename[:-4], 'Scharr')
    getXYLocation(img_canny, path3, filename[:-4], 'Canny')


def getXYLocation(edges, path, name, algo_name):
    if not os.path.exists(path):
        os.mkdir(path)
    points = []
    h = len(edges)
    w = len(edges[0])
    for i in range(h):
        for j in range(w):
            tmp = []
            if edges[i, j] == 255:
                tmp.append(i)
                tmp.append(j)
                points.append(tmp)
    np.savetxt(path + name + '_' + algo_name + '-' + "edges.txt", points, fmt="%d")

# 拍摄照片
def get_photo(hx, frame, savePath):
    if hx is False:
        print('read video error')
    cv2.imwrite(savePath, frame)
```

（4）**main 函数：AR 绘制和 YoloV5 模型推理**

```
import os
```

```
import warnings

import cv2
import numpy as np
from cv2 import waitKey

from Yolo import model_path, letterbox, class_names
from getImageLine.utils import image2line, get_photo
from myAR.OBJ import OBJ
from myAR.utils import target_name, model_name, MIN_MATCHES, rectangle, render,
projection_matrix
from openvino.runtime import Core, Layout, Type
from openvino.preprocess import PrePostProcessor, ColorFormat

if __name__ == '__main__':
    warnings.filterwarnings('ignore')
    """AR"""
    homography = None
    camera_parameters = np.array([[400, 0, 320], [0, 400, 120], [0, 0, 1]])
    orb = cv2.ORB_create()
    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    dir_name = os.getcwd()
    model_obj = cv2.imread(dir_name + '/src/reference/' + target_name, 0)
    kp_model, des_model = orb.detectAndCompute(model_obj, None)
    obj = OBJ(dir_name + '/src/models/' + model_name, swapyz=True)
    obj.create_gl_list()
    """yolov5"""
    core = Core()
    model = core.read_model(model_path)
    ppp = PrePostProcessor(model)
    ppp.input().tensor() \
        .set_color_format(ColorFormat.BGR) \
        .set_element_type(Type.u8) \
        .set_layout(Layout('NHWC'))
    ppp.input().model().set_layout(Layout('NCHW'))
    ppp.output().tensor().set_element_type(Type.f32)
    ppp.input().preprocess() \
        .convert_element_type(Type.f32) \
        .convert_color(ColorFormat.RGB) \
        .mean([0.0, 0.0, 0.0]) \
        .scale([255.0, 255.0, 255.0])
    model = ppp.build()
    net = core.compile_model(model, "AUTO")
```

```
    cap = cv2.VideoCapture(0)
    cnt = 0
    while True:
        success, frame = cap.read()
        if not success:
            print("Unable to capture video")
            exit(0)
        kp_frame, des_frame = orb.detectAndCompute(frame, None)
        matches = bf.match(des_model, des_frame)
        matches = sorted(matches, key=lambda x: x.distance)

        print(len(matches))

        if len(matches) > MIN_MATCHES:
            src_pts = np.float32([kp_model[m.queryIdx].pt for m in matches]).reshape(-1, 1, 2)
            dst_pts = np.float32([kp_frame[m.trainIdx].pt for m in matches]).reshape(-1, 1, 2)

            homography, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC, 5.0)
            if rectangle:
                h, w = model_obj.shape
                pts = np.float32([[0, 0], [0, h - 1], [w - 1, h - 1], [w - 1, 0]]).reshape(-1, 1, 2)
                dst = cv2.perspectiveTransform(pts, homography)
                frame = cv2.polylines(frame, [np.int32(dst)], True, 255, 3, cv2.LINE_AA)
            if homography is not None:
                try:
                    # print(len(matches))
                    # obtain 3D projection matrix from homography matrix and camera
parameters
                    projection = projection_matrix(camera_parameters, homography)
                    # project cube or model
                    frame = render(frame, obj, projection, model_obj, False)
                    # frame = render(frame, model, projection)
                except:
                    pass

        letterbox_img, ratio, (dw, dh) = letterbox(frame, auto=False)
        input_tensor = np.expand_dims(letterbox_img, axis=0)
        predictions = net([input_tensor])[net.outputs[0]]
        class_ids = []
        confidences = []
        boxes = []
        for pred in predictions:
            for i, det in enumerate(pred):
                confidence = det[4]
```

```
                scores = det[5:]
                class_id = np.argmax(scores)
                if scores[class_id] > 0.25:
                    confidences.append(confidence)
                    class_ids.append(class_id)
                    x, y, w, h = det[0].item(), det[1].item(), det[2].item(), det[3].item()
                    left = int((x - 0.5 * w - dw) / ratio[0])
                    top = int((y - 0.5 * h - dh) / ratio[1])
                    width = int(w / ratio[0])
                    height = int(h / ratio[1])
                    box = np.array([left, top, width, height])
                    boxes.append(box)
        indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.25, 0.45)
        filtered_ids = []
        filered_confidences = []
        filtered_boxes = []
        for i in indexes:
            filtered_ids.append(class_ids[i])
            filered_confidences.append(confidences[i])
            filtered_boxes.append(boxes[i])
        colors = [(255, 255, 0), (0, 255, 0), (0, 255, 255), (255, 0, 0)]
        class_id_filter = [0]
        first_flag = 0
        for (class_id, confidence, box) in zip(filtered_ids, filered_confidences, filtered_boxes):
            if class_id not in class_id_filter:
                continue
            if not first_flag:
                Box = box
                color = colors[int(class_id) % len(colors)]
                cv2.rectangle(frame, box, color, 2)
                cv2.rectangle(frame, (box[0], box[1] - 20), (box[0] + box[2], box[1]), color,
-1)
            first_flag = 1

            # cv2.putText(frame, class_names[class_id], (box[0], box[1] + 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5,
            #                 (255, 255, 255), 1)

        # if matches:
        #     frame = cv2.drawMatches(model_obj, kp_model, frame, kp_frame, matches[:10],
0, flags=2)

        cv2.imshow("Welcome to our AR Magic Draw Image", frame)
```

```
        if cv2.waitKey(1) == ord('s'):
            """getImagePoints"""
            cnt += 1
            for i in range(4):
                if Box[i] < 0:
                    Box[i] = 0

            get_photo(success, frame[Box[1] + 2:Box[3] - 2, Box[0] + 2:Box[2] - 5],
                      savePath='./image/' + str(cnt) + '.png')
            image2line(filename=str(cnt) + '.png', path1='./image/', path2='./output_image/' +
str(cnt) + '/',
                       path3='./output_edges/' + str(cnt) + '/',
                       filter=240)
            print('save success')
        elif cv2.waitKey(1) == ord('q'):
            break
```

（5）控制机械臂的代码

```
import threading
import DobotDllType as dType
import numpy as np

LENGTHX = 120
LENGTHY = 150
BASICX = 172
BASICY = -108
# PAINTZ = -40
PAINTZ = -53

CON_STR = {
    dType.DobotConnect.DobotConnect_NoError: "DobotConnect_NoError",
    dType.DobotConnect.DobotConnect_NotFound: "DobotConnect_NotFound",
    dType.DobotConnect.DobotConnect_Occupied: "DobotConnect_Occupied"}

print(1)
api = dType.load()
# Connect Dobot
state = dType.ConnectDobot(api, "", 115200)[0]
print("Connect status:", CON_STR[state])

if (state == dType.DobotConnect.DobotConnect_NoError):
    # Clean Command Queued
    dType.SetQueuedCmdClear(api)

    # Async Motion Params Setting
```

```
    dType.SetHOMEParams(api, BASICX, BASICY, PAINTZ + 20, 200, isQueued=1)

    # Async Home
    dType.SetHOMECmd(api, temp=0, isQueued=1)
    dType.SetQueuedCmdStartExec(api)

    # Async PTP Motion
    data = np.loadtxt("C:\\Users\\ren\\Desktop\\dobot\\data.txt")
    dType.SetPTPCmd(api, 0, BASICX + ((data[0][0] / 250) * LENGTHX), BASICY +
((data[0][1] / 250) * LENGTHY), PAINTZ,
                    62, 1)
    for i in range(1, len(data)):
        if data[i - 1][2] == 0:
            dType.SetPTPCmd(api, 2, BASICX + ((data[i][0] / 250) * LENGTHX), BASICY +
((data[i][1] / 250) * LENGTHY),
                            PAINTZ, 62, 1)
        else:
            dType.SetPTPCmd(api, 0, BASICX + ((data[i][0] / 250) * LENGTHX), BASICY +
((data[i][1] / 250) * LENGTHY),
                            PAINTZ, 62, 1)
    dType.SetPTPCmd(api, 0, BASICX, BASICY, PAINTZ + 10, 62, 1)
    dType.SetQueuedCmdStartExec(api)

    # Stop to Execute Command Queued
    dType.SetQueuedCmdStopExec(api)

# Disconnect Dobot
dType.DisconnectDobot(api)
```

（6）路径规划代码

```
import numpy as np

def find(x, y):
   round = []
   mov = [[-1, -1], [-1, 1], [1, 1], [1, -1], [1, 0], [0, 1], [-1, 0], [0, -1]]
   for n in range(len(mov)):
      if (x + mov[n][0] <= maxX) & (x + mov[n][0] >= 0) & (y + mov[n][1] <= maxY) & (y +
mov[n][1] >= 0):
         if M[x + mov[n][0]][y + mov[n][1]] == 1:
            round.append((x + mov[n][0], y + mov[n][1]))
   return round


data = np.loadtxt("8_Canny-edges.txt", dtype=int)
```

```
maxX = -1
maxY = -1

for i in range(len(data)):
    if data[i][0] > maxX:
        maxX = data[i][0]
    if data[i][1] > maxY:
        maxY = data[i][1]

M = np.zeros((maxX + 1, maxY + 1))
flag = np.zeros((maxX + 1, maxY + 1))
branch = []
f = open('data.txt', 'w')

for i in range(len(data)):
    M[data[i][0]][data[i][1]] = 1

for x1 in range(maxX):
    for y1 in range(maxY):
        if M[x1][y1] == 1:
            x = x1
            y = y1
            while 1:
                List = find(x, y)
                if len(List) > 1:
                    M[x][y] = 0
                    branch.append((x, y))
                    print(str(x) + ' ' + str(y) + ' 0', file=f)
                    flag[x][y] = 1
                    pos = List.pop()
                    x = pos[0]
                    y = pos[1]
                elif len(List) == 1:
                    M[x][y] = 0
                    print(str(x) + ' ' + str(y) + ' 0', file=f)
                    pos = List.pop()
                    x = pos[0]
                    y = pos[1]
                elif len(List) == 0:
                    if M[x][y] == 1:
                        M[x][y] = 0
                        print(str(x) + ' ' + str(y) + ' 1', file=f)
                    if branch:
```

```
        pos = branch.pop()
        x = pos[0]
        y = pos[1]
    else:
        break
```