河南师范大学

本科毕业论文

学号：1928424157

基于手势识别的数字画板设计与实现

学院名称：软件学院

专业名称：计算机科学与技术（软件开发 Java 方向）

年级班别：2019 级 JAVA 1 班

姓　　名：李世豪

指导教师：李名

2023 年 5 月

声　明

本论文所含内容是在指导教师的指导下，由学生本人通过科学的研究方法获得的。论文所使用数据等无抄袭他人成果情况，学生本人对论文的内容和方法的真实性负全部责任。

论文经导师审核，同意参加学院组织的本科生毕业论文答辩，并按照答辩意见对论文进行了修改和完善。

本论文撰写格式符合相关规定，现经导师同意提交学校。论文所提交电子版和打印版一致。

学生签名：　　指导教师签名：

年　　月　　日　　年　　月　　日

摘要

本文介绍了手势识别技术在数字画板应用中的应用。通过检测和分析人类手部姿态，将手部运动转化为计算机可以识别和理解的指令，从而实现使用人体手部姿态来控制数字画板。本文提出了一种基于手势识别技术的数字画板应用方法，该方法利用了 OpenCV 和 MediaPipe 两个强大的工具库。首先使用 MediaPipe 的手部姿态估计模型来检测手指的位置和运动轨迹，然后使用 OpenCV 的绘图功能来绘制数字画板。在 Python 语言下编写了一个应用程序，利用该程序实现了手势识别控制数字画板的功能。

关键词：手势识别；数字画板；高斯滤波；姿态估计；图像二值化。

Design and implementation of digital artboard based on gesture recognition

Abstract

This article introduces the application of gesture recognition technology in digital drawing board applications. By detecting and analyzing human hand posture, hand movements can be converted into computer-readable commands, allowing the digital drawing board to be controlled using hand gestures. A method based on gesture recognition technology for digital drawing board applications is proposed in this article, which utilizes the powerful toolkits of OpenCV and MediaPipe. Firstly, MediaPipe's hand pose estimation model is used to detect the position and movement trajectory of the fingers, and then OpenCV's drawing functions are used to create the digital drawing board. We developed an application program in Python that realizes the functionality of controlling the digital drawing board using hand gestures.

Keywords: Gesture recognition; digital drawing board; Gaussian filtering; pose estimation; image binarization.

目 录

1. 绪论

1.1 研究背景和意义

手势识别的主要目的是将手势转换成可被计算机识别的信号，然后由计算机根据这些信号执行相应的任务。随着智能手机、平板电脑和智能电视等智能设备的日益普及，手势识别技术已成为人机交互领域最有趣的话题之一。

数字画板已经成为数字媒体创作和设计的重要工具，使用越来越广泛，例如数字绘画、手写笔记、图形设计、漫画制作等等。然而，传统数字画板虽然能够满足一般需求，但是其往往需要使用外部设备或者软件，对于一些非专业用户来说使用门槛较高。因此，研究开发一款针对非专业用户的数字画板，既能够方便用户使用，又能够提高绘画体验和效果，成为一个重要的研究方向。

开发一款非专业的数字画板，可以让数字媒体创作更加简单。相较于传统的画板，一方面，数字画板在优化用户使用体验的同时提高作品线条的美观度；另一方面，数字画板的开发出的应用也可以在后续研究与开发中变成更加智能和便利的数字产品，例如数字笔记、和数字绘画等应用程序，从而促进数字媒体的快速发展发展。

数字绘图板的开发为数字媒体的创作提供了便利，优化了用户体验，提高了作品的美感。还有一些智能实用的数字产品，如数字记事本、数字绘图和数字绘画。在未来，数字媒体将变得越来越智能。

1.2 国内外研究现状

手势识别技术已成为人机交互技术中的重要技术，市场应用广泛。手势识别分为触摸手势识别和非触控手势识别两类。触摸手势识别技术是通过触摸屏捕捉用户的手指动作，实现特定功能。非触控手势识别技术更加方便，可以直接使用传感器或摄像头捕捉手部动作，并通过分析确定动作代表的意思。

传统的画板不依赖软件，而是需要专门的数字笔，可以直接在板上绘制图像，并实施传输到计算机上进行编辑。虽然这样的设备为绘画者提供了便利，但在一些应用场景中也有缺点，如空间限制、成本高、需要外部硬件或软件。因此，需要探索新的技术和方法来解决这些问题。

手势识别技术的历史可以追溯到二十世纪初，目前，数字画板方面的研究主

要集中在国外学者中。例如，James M. Rehg 和 Takeo Kanade（2005）研究了高自由度铰链结构的视觉跟踪：人手跟踪的应用[1]；Cristina Manresa,，Javier Varona 等人（2005）提出了用于人机交互的手部跟踪和手势识别[2]；Robert Y. Wang 和 Jovan Popović（2009）实现了带彩色手套的实时手部跟踪[3]；Chen Qian，Xiao Sun 等人（2014）从深度进行实时和稳健的手部追踪[4]；Fan Zhang, Valentin Bazarevsky, Andrey Vakunov 等人（2020）提出的在实时设备上实现的手部跟踪[5]。然而，国内目前尚未涉足该领域的研究。这些国外研究为数字画板的发展提供了新的思路和方法，对数字艺术的推广和应用也起到了积极的促进作用。

1.3 论文组织结构

本文主要研究为基于手势识别的数字画板设计与实现，共分为五部分。论文结构如下：第一章为绪论，介绍手势识别以及数字画板研究背景以及意义、研究内容与国内外发展研究现状；第二章为手势识别技术的介绍以及相关组件；第三章为手势识别的算法设计，对手势识别的相关算法进行比较，确定适合实验的算法；第四章为结合手势识别对数字画板实现的过程；第五章为论文总结与展望

2. 相关技术

2.1 OpenCV 与 MediaPipe

Python-OpenCV 和 MediaPipe 是开源的工具库，在计算机视觉领域常用于图像、视频和多媒体处理等任务。下面简要介绍一下它们：

Python-OpenCV 支持多种操作系统平台，包括 Windows、Linux 和 macOS，并为 Python 开发者提供了 Python 语言的接口；Python-OpenCV 处理和分析图像、视频和深度图像等数据，广泛用于图像处理、模式识别等；它还支持 GPU 加速以提高计算效率。

MediaPipe 是谷歌开发的一个跨平台机器学习框架，主要用于处理由各种类型数据（视频、音频、各种传感器数据、时间序列数据等）组成的数据源。它支持多种平台，如安卓、iOS、Windows 和 macOS，还提供了一个 Python 编程语言的接口。MediaPipe 的主要特点是，它支持实时处理和使用。此外，MediaPipe 还提供各种预建的处理模块，如人脸识别、手势估计和手部跟踪，以帮助用户快速创建多媒体应用。

2.2 图像预处理

图像的预处理关键在于滤波和二值化处理[6]。图像的预处理可以使用多种技术，但滤波和二值化处理是预处理中非常重要的两个步骤，它们对于手势识别算法的准确性和稳定性起着至关重要的作用。

滤波处理可以帮助去除图像中的椒盐噪声，从而使手势的特征更加突出，有利于算法对手势进行识别；可以使得图像变得更加平滑，降低识别误差。

二值化处理可以将原始图像转换为黑白二值图像，消除颜色和灰度变化对手势识别的影响，通过提高图像的对比度，更好的突出手势的特征，使得算法更加容易和准确地识别手势。

2.2.1 噪声来源

在手势识别中，有几个干扰源会影响手势识别的准确性。本文对手势识别中常见的干扰源进行了总结和分类。

首先，环境噪声是影响手势识别的主要因素之一。这些因素会造成图像的噪声、亮度和颜色失真，从而影响手势识别的准确性。

第二，手势图像的旋转、大小调整和失真可能是由手部姿势的变化引起的，这可能会影响手势识别的准确性。

第三，低分辨率是另一个会影响手势识别准确性的因素。低分辨率的图像会造成图像的模糊和失真等问题。

第四，手部遮挡也会影响手势识别的准确性。如果手被遮挡，一些手势图像信息就会丢失。

为了解决这些噪声问题，可以使用各种图像处理技术，如降噪、图像增强、归一化和过滤，以提高手势识别的准确性和鲁棒性。

2.2.2 高斯滤波

高斯滤波是一种常用的图像处理滤波器，它可以用来平滑图像并去除噪声[7]。该滤波器利用高斯函数对图像进行卷积操作，通过减少像素之间的差异，可以使图像变得更加平滑[8]。

高斯滤波器的具体操作是，将滤波器模板应用于每一个像素点，计算滤波器内邻域像素的加权平均值，并将该值作为中心像素点的新灰度值，用以代替原有的值。

算法步骤：

在图像处理中，最常见的就是种滑窗实现，只有当离散化的窗口非常大，用滑窗计算量非常大（即使用可分离滤波器的实现）的情况下，可能会考虑基于傅里叶变化的实现方法。

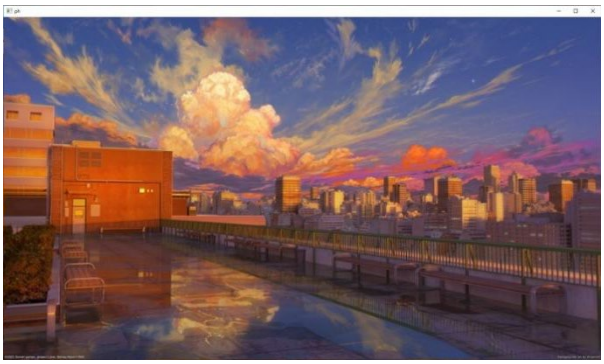
在进行高斯滤波时，需要指定一个卷积核大小和一个标准差值。卷积核大小决定了滤波器的范围，标准差值则决定了高斯函数的形状。

$$I_\sigma = I * G_\sigma \tag{2.1}$$

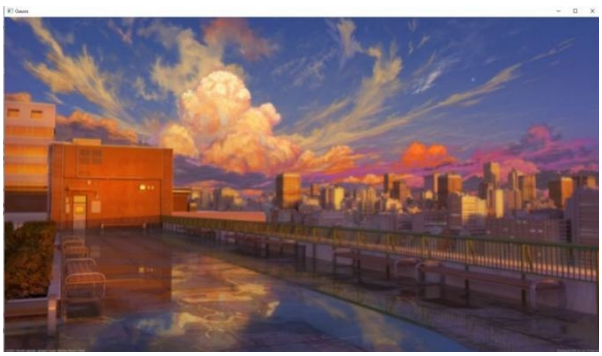
其中 * 表示卷积操作; G_σ 是标准差为σ 的二维高斯核,定义为:

$$G_\sigma = \frac{1}{2\pi\sigma} e^{-\left(x^2+y^2/2\sigma^2\right)} \tag{2.2}$$

下面是进行高斯模糊前后的图片对比：



2. 1 原图



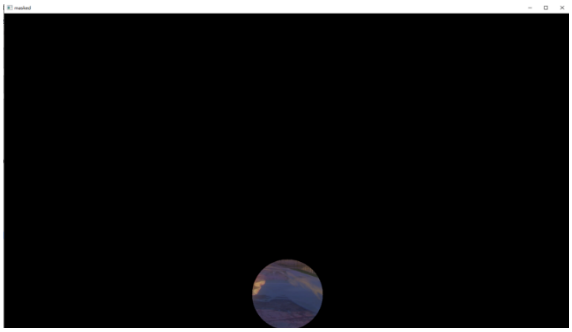
2. 2 高斯模糊

2.2.3 图像二值化

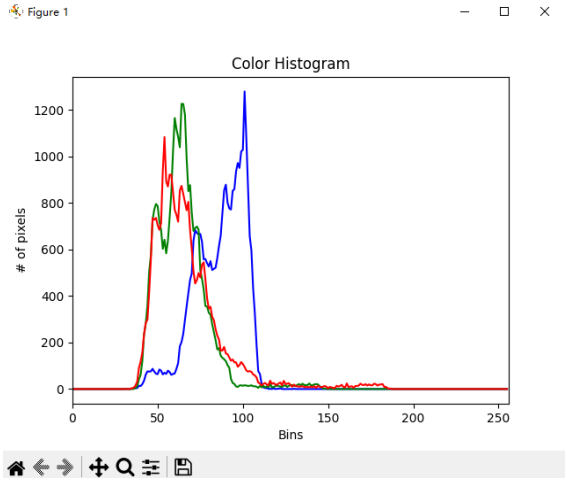
计算机视觉一般使用矩阵来表示图像。在矩阵中的每个像素都是矩阵的一个元素：对于三通道彩色图像，三个数字为一个元组；对于单通道灰度图像，一个数字就是一个元组（0 表示全黑，255 表示全白）。

图像二值化是将一幅图像转化为只有两种颜色的图像的过程[9]。通常情况下，这两种颜色是白色和黑色，因此二值化图像也被称为黑白图像或二值图像。

在进行图像二值化时，需要设置一个阈值，对于像素值高于阈值的像素，被赋予白色，而对于像素值低于阈值的像素，则被赋予黑色。因此，阈值的选择对最终的二值化效果有着重要的影响。根据阈值选取方式的不同，可以分为全局阈值和局部阈值[10]，二值图像也常常用作原始图像的掩模（图 2.3）。



2.3 加入蒙版遮罩后的图



2.4 未被遮罩部分的直方图

3. 手势识别

3.1 手势识别的难点

3.1.1 手势区域检测的影响

手势识别是一种基于人机交互的技术，可能具有挑战性。在这篇文章中将详细讨论手势识别技术的挑战。

首先，多样性和可变性是手势识别的主要问题之一。有许多不同的手势，甚至同一手势也因人而异，因此手势识别系统必须足够强大，以准确识别不同类型的手势。目前，一些基于深度学习的手势识别方法在处理手势多样性方面已经显示出良好的效果。

其次，实时性和效率是手势识别的另一个挑战。由于手势识别系统经常被用于游戏中的实时交互、手势控制和虚拟现实等场景中，因此必须满足对实时性和效率的高要求。近年来，并行 GPU 计算和嵌入式系统等技术的出现，大大改善了手势识别系统的实时性和效率。

第三，噪声和干扰也是手势识别系统的一个挑战[11]。手势识别系统可能会因为各种噪音和干扰而无法识别手势或识别错误，如光线的变化、手的抖动等。为了解决这些困难，研究人员提出了一些方法，包括光线不变性、手势估计和运动模型。

第四，数据的数量和质量也是手势识别系统的一个挑战。手势识别系统的性能一般受数据的数量和质量的影响。大型数据集和高质量的数据收集可以提高手势识别系统的准确性和通用性。

3.1.2 手势识别环境的影响

进行手势识别的环境对识别结果有很大影响。在实践中，手势识别系统必须在各种具有挑战性的条件下运行，如不同的照明条件、噪音和干扰、背景噪音、工作距离和角度的差异、手势的多样性和可变性等等。

光线的强度、方向和颜色等因素会影响手势图像的质量和特征，从而影响识别的准确性；手势识别是基于对相机或传感器收集的数据的分析和识别，噪音和干扰会对数据的准确性产生负面影响；复杂的纹理和颜色、人、物体和其他背景元素也会混入手势图像，导致手势识别不正确；户和数据采集设备之间的距离和

角度也会影响手势图像的大小、形状和方向，从而影响识别的准确性。研究人员需要开发更灵活的算法，考虑到不同的工作距离和角度，以提高手势识别系统的适应性和准确性。

手势识别系统需要考虑用户习惯，设计出易于使用的手势识别系统，尽量减少用户的学习成本。因此，设计一个合适的界面并提供良好的反馈和互动是提高手势识别系统易用性的一个重要因素。

3.2 基于深度学习的手势识别算法设计

3.2.1 预处理

对输入图像进行预处理，包括图像增强、降噪、二值化等操作。其中，降噪使用高斯滤波器，二值化采用 Otsu 自适应阈值法。

3.2.2 HandLandmark 模块

利用 MediaPipe 的 HandLandmark 模块进行手部检测，得到手部关键点位置信息。

3.2.3 手部区域提取

基于手势识别结果，实现数字画板的手势控制。

手部区域的提取是实现手势识别的重要步骤。本研究结合了 MediaPipe 和 OpenCV，实现了手部区域的提取和绘制。

在图像中检测手部：使用手部检测模型输出手部的 21 个关键点的坐标（图 4.10）。

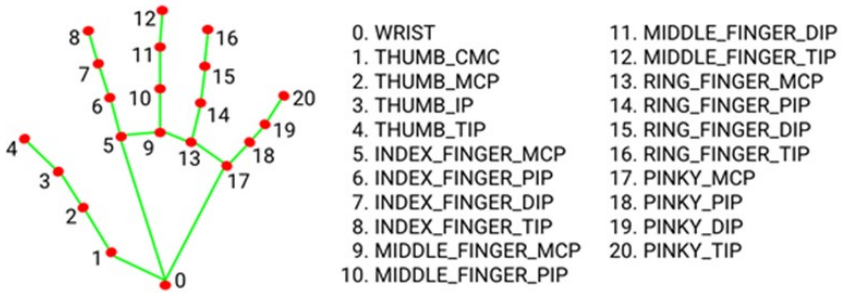


图 3.1 手部关键点坐标

一旦获得了标志性值，就为数据库中的每个视频创建一组信号。关节的坐标值用于为视频 i 创建信号组 S，其定义如下（图 3.2）：

$$S_i^{3k\times n}=\begin{pmatrix} j_{1,x,1} & j_{1,x,2} & \cdots & j_{1,x,n} \\ j_{1,y,1} & j_{1,y,2} & \cdots & j_{1,y,n} \\ j_{1,z,1} & j_{1,z,2} & \cdots & j_{1,z,n} \\ j_{2,x,1} & j_{2,x,2} & \cdots & j_{2,x,n} \\ \vdots & \vdots & \ddots & \vdots \\ j_{k,z,1} & j_{k,z,2} & \cdots & j_{k,z,n} \end{pmatrix}$$

图 3.2 信号组

其中 k 是关节特征的数量，n 是帧数，J_u,c,v 是关节 u 的地标值，坐标 c：x，y，z 和帧 v。为每个帧提取的关节数是 21 (k = 21)，并且由于每个地标由 (x, y, z) 值组成，因此信号矩阵的行数为 63：21 个关节中的每一个都有 3 个值 (x, y, z) （3 × 21 = 63）。 由于 z 坐标与手腕相关，因此在执行分类时可能无关紧要。

其次，将手部区域坐标转换为像素坐标。

最后，结合 OpenCV 库提供的函数，可以将手部区域绘制在图像上，并通过窗口显示出来。

本研究的实验结果表明，所提出的算法能够准确地提取手部区域，并将其绘制在图像上。这为后续的手势识别等应用奠定了基础。

3.2.4 识别手势特征

基于手部关键点位置信息，计算手势的特征。使用了两种特征：手势方向和手指数量。手势方向基于手部关键点的坐标差值计算，手指数量则基于手部关键点之间的角度差值计算。

如果中指坐标在食指坐标位置之上，那么为选择模式（图 3.3），用户可以在控制面板区域选择画笔以及橡皮擦；反之为绘画模式（图 3.4），用户可以在绘画区域作画。

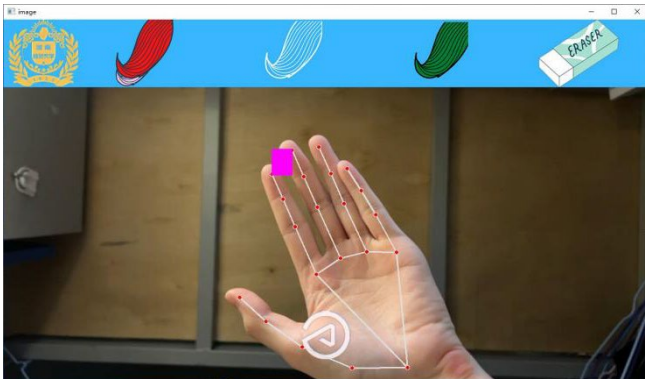


图 3.3 选择模式

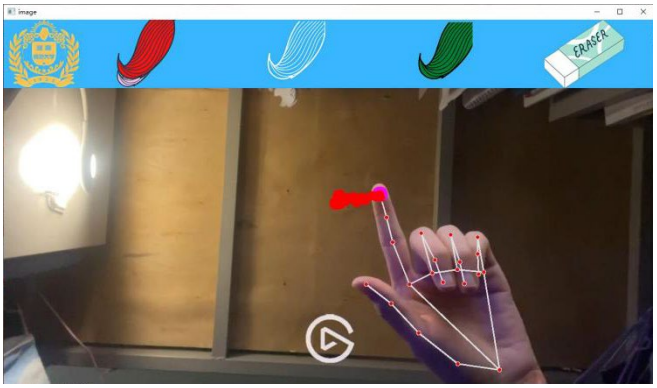


图 3.4 绘画模式

3.3 实验算法的确定

手部姿态估计算法是本文实现基于手势识别的数字画板功能的关键技术之一。本节将介绍手部姿态估计算法的原理、步骤、实现细节以及优缺点。

MediaPipe 内置的算法可以实时进行手势识别。该算法使用了一种称为“手部关键点检测”的技术，通过对这些关键点的检测和跟踪，可以实现对手部姿态和手势的识别。本文调整了手势识别算法的参数，以便于提高识别的准确性，能够更好地识别手势信息；优化手势检测的速度和响应时间，使其可以在实时绘图过程中保持流畅和稳定。

3.3.1 原理与步骤

本文使用的是 MediaPipe 中内置的基于深度学习的手部姿态估计算法。该算法的主要步骤如下：

1) 使用嵌入式深度学习网络检测出手部区域；

2) 对手部区域进行图像预处理，如缩放、裁剪等；

3) 确定关键点的位置；

4) 对估计结果进行处理，得到最终的结果。

3.3.2 实现细节

为了实现手部姿态估计算法，本文使用了 OpenCV 和 MediaPipe 提供的 API。具体实现细节如下：

1) 调用 MediaPipe 提供的手部姿态估计模型，并加载模型参数；

```
mp_hands = mp.solutions.hands.Hands(static_image_mode=False,
 max_num_hands=1, min_detection_confidence=0.5, min_tracking
_confidence=0.5)
```

2) 读取输入图像并将其转换为 RGB 格式

```
image = cv2.imread("hand_image.jpg")
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

3) 使用 Hands 模型进行手部区域检测

```
results = mp_hands.process(image_rgb)
```

4) 对手部区域进行预处理，如缩放、裁剪等；

```
if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            # 进行关键点估计
            for id, lm in enumerate(hand_landmarks.land
mark):
                # 处理关键点坐标
                x = int(lm.x * image.shape[1])
                y = int(lm.y * image.shape[0])
                z = lm.z  # 如果需要深度信息
                # 可以根据需要对关键点进行进一步处理
```

5) 使用 MediaPipe 提供的手部姿态估计 API 进行关键点绘制；

```
if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                image, hand_landmarks, mp_hands.HAND_CONNECTIO
NS)
```

需要注意的是，上述步骤仅提供了基本的手部姿态估计流程，具体的应用场景和需求可能需要进一步的算法和处理步骤。本文根据实际情况，进行模型的调参、处理、优化等操作，达到了更好的结果[A]。

3.3.3 优缺点

优点：

1) 实现手部姿态的准确估计；

2) 可以适应不同的手势姿态，如手指伸展、握拳等；

3) 可以应用于手势识别、手写输入等领域。

缺点：

1) 对输入图像的质量要求较高，如光照、角度等；

2) 算法的计算复杂度较高，需要较高的计算资源支持。

4. 基于手势识别的数字画板实现

4.1 界面与主要功能

在介绍界面与主要功能之前，先说明本文所写代码的文件架构，项目文件树如图所示（图 4.1），其中 Header 文件夹内图片为数字画板控制面板区域所显示内容，HandTrackingModel.py[A] 为手部识别模块，VirtualPainter.py[B] 为数字画板项目主要代码文件，requirements.txt[C] 为项目所需库。

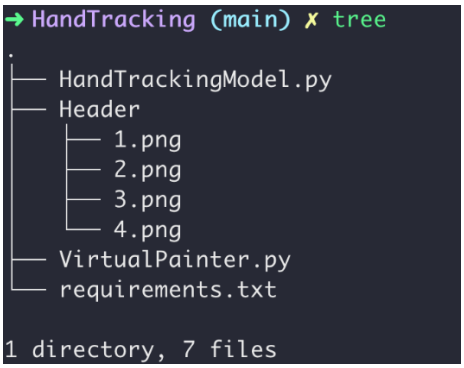


图 4. 1 文件树

本文设计了一个简洁的界面，包括绘图区域和控制面板两部分。绘图区域用于显示绘图内容，控制面板用于设置画笔颜色、和橡皮擦功能。其中黑色区域为绘图区域，黑色区域上方为控制面板（图 4.2）。数字画板控制面板区域所显示内容，按照图片顺序依次为红色画笔（图 4.3）、蓝色画笔（图 4.4）、绿色画笔（图 4.5）以及橡皮擦（图 4.6）。

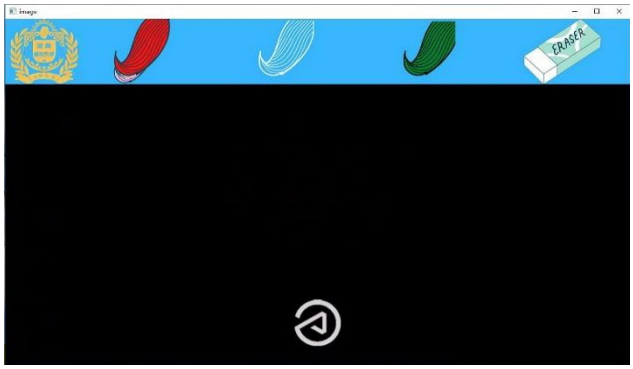


图 4.2 界面设计分区

图 4.3 1.png

图 4.4 2.png

图 4. 5 3.png

图 4. 6 4.png

本文实现了基本的绘图功能—画笔和橡皮擦操作。用户可以通过手势控制来选择不同的绘图工具和属性，从而实现自由绘图和创作。用户还可以通过控制区域切换画笔颜色（图 4.8）进行绘画（图 4.7）以及采用橡皮擦进行墨迹擦除（图 4.9）。

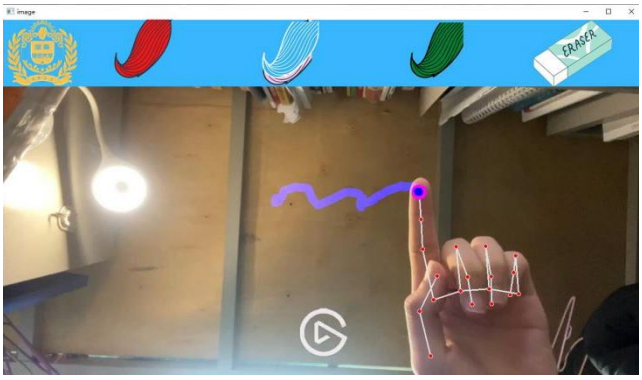


图 4.7 绘画

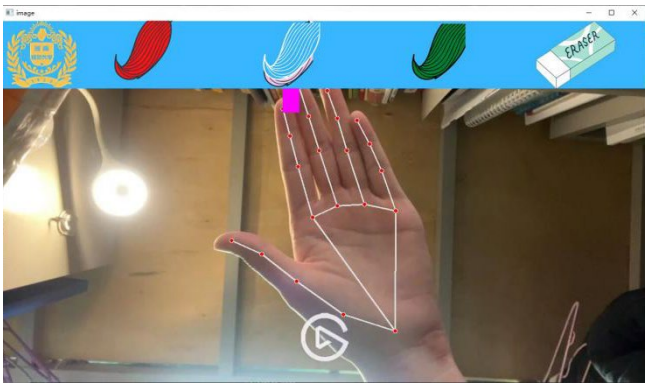


图 4. 8 切换画笔

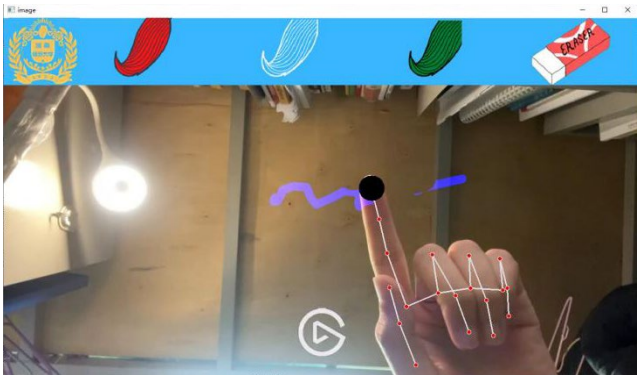


图 4. 9 擦除墨迹

4.2 指尖检测

手势识别算法基于神经网络模型，通过对手部关键点的分析和计算来识别不同的手势动作。

在图像中检测手部：使用 MediaPipe 的 HandTracking 模型可以检测到图像中的手部。这个模型可以输出手部的 21 个关键点的坐标（图 4.10），其中包括手掌、手指和指尖。

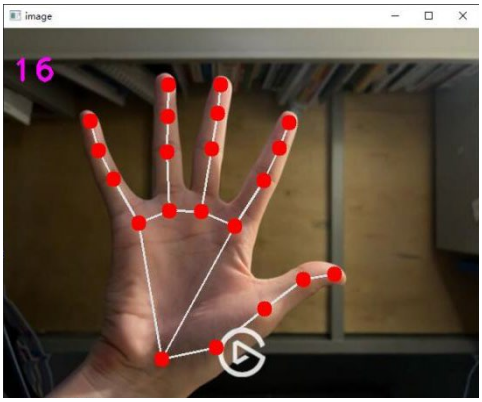


图 4.10 HandTracking 模型手指关节地标

4.3 手势控制逻辑

手势控制逻辑是数字画板的核心功能之一，通过手势识别算法实现用户手势的检测和识别。本文采用了 Mediapipe 内置的手势识别算法，可以实现对不同手势的识别和跟踪。具体而言，用户可以通过不同的手势来控制画笔的移动和颜色等属性。

为了提高画板的稳定性和精度，调整了手势识别算法的参数，使其能够更好地识别手势信息。

4.4 利用蒙版优化多媒体画面

在使用 Mediapipe 进行手势识别和画画时，因为图层叠加会影响成像效果，导致画面不清晰（图 4.11），所以可以使用蒙版来优化多媒体画面和画笔笔迹。

蒙版是在图像上添加一个透明图层来实现画面的显示与隐藏，在此情况下，可以在获取的图像上添加一个透明图层，然后在该透明图层上进行操作，以便将多媒体画面和绘图图层隔离出来。

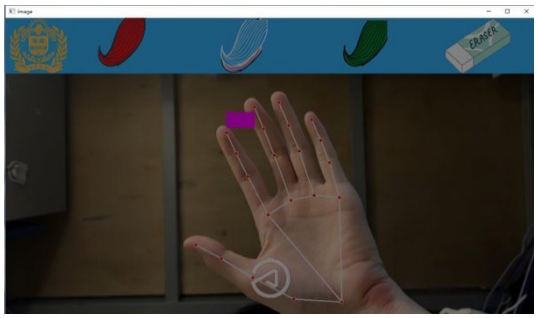


图 4.11 未优化前画面

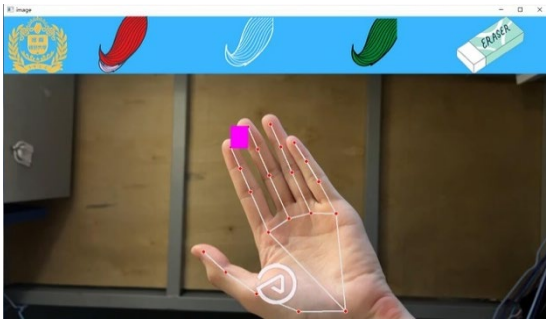


图 4.12 优化后图像

在将两个图像合并时，使用 alpha 混合技术来控制透明度[12]，从而使多媒体

画面和画笔笔记能够正确地显示在一起（图 4.12）。通过调整 alpha 值，可以控制画笔笔迹的不透明度，以便它们不会完全覆盖获取的图像。

5. 总结与展望

这个项目结合了两个计算机视觉库，OpenCV 和 MediaPipe，以开发一个基于手势的数字画板应用。它跟随用户的手并识别手势，允许用户使用数字笔在多媒体屏幕上作画。

在这个项目中，计算机视觉和机器学习的基础知识可以很好地应用，比如如何编写属于自己的手部跟踪模块。项目中还有一些可以的改进的方面：可以通过优化手势识别的算法来提高手部跟踪和手势识别的准确性和可靠性，以便于用户更容易使用开发出的软件；在功能设计上可以增加额外的功能，以提高应用程序的可用性；通过根据用户的特殊需求定制应用程序来提高其效率。

这个项目是一个很好的计算机视觉和机器学习的实践项目，通过这个项目，不仅学习了相关的知识和技能，还实现了一个实用的应用程序。在未来，可以继续改进和扩展这个应用程序，同时也可以探索更多有趣的计算机视觉和机器学习应用的开发。

致谢

昔日的冥冥中，我迷失在思考的深渊之中。仿佛走了千年，依旧找不到前行的道路。在我最黑暗的时刻，有一群人向我伸出了援手，让我重新看到了光明，寻找到了人生的价值。

首先，我要感谢我的导师，他为我提供了无私的指导和帮助，让我在研究中不断取得进步。在这里，我想感谢我的导师李名。他不仅为我提供了无私的指导和支持，还鼓励我不断尝试新的研究方向和方法，让我成为一名更好的科研人员。

首先，我要感谢那些为我提供指引和帮助的人。他们不仅在学术研究中给我提供了无私的指导，还在生活中关心、照顾我。他们如同指南针，让我明白了人生的方向，并为我照亮前行的路途。在此，我要感谢他们，感谢他们为我付出的一切。

其次，我要感谢我的同窗好友。他们的友谊如同灵魂般温暖，让我在孤独的学术道路上不再感到孤单。他们与我并肩前行，分享着学术研究的乐趣和挑战，是我学术生涯中最坚实的后盾。在此，我要感谢他们，感谢他们一直以来的陪伴和支持。

最后，我要感谢我的家人。他们是我生命中最亲密的人，无论我何时何地，他们永远支持我、鼓励我。他们的爱如同温暖的阳光，给予我前行的勇气和力量。在这里，我要感谢冯明洋，感谢她一直以来的支持和爱。

人生充满了挑战和坎坷，但我知道，有那么一群人，他们在我人生的每一个节点都给予了我支持和鼓励。他们让我重新看到了人生的意义，让我充满了前行的勇气。在此，我要感谢所有支持我、帮助我和鼓励我的人们，感谢他们为我点亮前行的路途，让我永远铭记在心

李世豪

2023 年 5 月于河南师范大学

参考文献

[1] Rehg J M, Kanade T. Visual tracking of high dof articulated structures: an application to human hand tracking[C].Computer Vision—ECCV'94: Third European Conference on Computer Vision Stockholm, Sweden, May 2－6 1994 Proceedings, Volume II 3. Springer Berlin Heidelberg, 1994: 35-46.
[2] Manresa C, Varona J, Mas R, et al. Hand tracking and gesture recognition for human-computer interaction[J]. ELCVIA Electronic Letters on Computer Vision and Image Analysis, 2005, 5(3): 96-104.
[3] Wang R Y, Popović J. Real-time hand-tracking with a color glove[J]. ACM transactions on graphics (TOG), 2009, 28(3): 1-8.
[4] Qian C, Sun X, Wei Y, et al. Realtime and robust hand tracking from depth [C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2014: 1106-1113.
[5] Zhang F, Bazarevsky V, Vakunov A, Tkachenka A, Sung G, Chang CL, et al. MediaPipe Hands: On-device Real-time Hand Tracking. arXiv preprint arXiv:200610214. 2020.
[6] 张良安,陈洋,谢胜龙等.基于机器视觉与深度学习的飞机防护栅裂纹检测系统[J].兵工学报,2023,44(02):507-516.
[7] 郭运冲,李孟军,刘名果等.基于 Canny 算子的建筑裂缝边缘检测改进算法[J].计算机仿真,2022,39(11):360-365+410.
[8] 于浩. 基于 Qt 的增强现实可视化仪表界面设计[D].青岛理工大学,2020.DOI:10.27263/d.cnki.gqudc.2020.000120.
[9] 马跃峰. 基于数字图像处理的交警手势识别算法研究[D].吉林大学,2016.
[10] 樊诚昊. 基于异构平台的 IC 芯片印刷字符嵌入式识别系统实现[D].上海大学,2021.DOI:10.27300/d.cnki.gshau.2021.000352.
[11] 李静慧,赵恩良,孙丽华等. 基于噪声检测和统计特性的图像去噪算法研究[C]//中共沈阳市委,沈阳市人民政府.第十九届沈阳科学学术年会论文集.[出版者不详],2022:6.DOI:10.26914/c.cnkihy.2022.016472.
[12] 程文涛,任冬伟,王旗龙.基于循环神经网络的散焦图像去模糊算法[J].计算机应用研究,2022,39(07):2203-2209.DOI:10.19734/j.issn.1001-3695.2021.11.0635.

附录

A. HandTrackingModel.py

```
import cv2 as cv
import mediapipe as mp
import time

mpHands = mp.solutions.hands
hands = mpHands.Hands()
mpDraw = mp.solutions.drawing_utils

class handDetector():
    def __init__(self, mode=False, maxHand=2, detectionCon=1, trackCon=0.5):
        self.mode = mode
        self.maxHands = maxHand
        self.detectionCon = detectionCon
        self.trackCon = trackCon

        self.mpHands = mp.solutions.hands
        self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.detectionCon, self.trackCon)
        self.mpDraw = mp.solutions.drawing_utils

        self.tipsIds = [4, 8, 12, 16, 20]
    def findHands(self, img, draw=True):
        imgRGB = cv.cvtColor(img, cv.COLOR_BGR2RGB)
        self.results = hands.process(imgRGB)

        if self.results.multi_hand_landmarks:
            for handLms in self.results.multi_hand_landmarks:
                # 编号和地标
                if draw:
                    self.mpDraw.draw_landmarks(img, handLms, self.mpHands.HAND_CONNECTIONS)
        return img
    def findPosition(self, img, handNo=0, draw=True):
        self.lmList = []
        if self.results.multi_hand_landmarks:
            myHand = self.results.multi_hand_landmarks[handNo]
            for id, lm in enumerate(myHand.landmark):
                # print(id, lm)
                h, w, c = img.shape
```

```
                cx, cy = int(lm.x * w), int(lm.y * h)
                # print(id, cx, cy)
                self.lmList.append([id, cx, cy])
                # if id == 0:
                if draw:
                    cv.circle(img, (cx, cy), 10, (0, 0, 255), cv.FILLED)
        return self.lmList

    def fingersUp(self):
        fingers = []
        # print(lmList)
        # 大拇指 左手
        if self.lmList[self.tipsIds[0]][1] > self.lmList[self.tipsIds[0] - 1
][1]:
            fingers.append(1)
        else:
            fingers.append(0)
        # 其他四指
        for id in range(1, 5):
            if self.lmList[self.tipsIds[id]][2] < self.lmList[self.tipsIds[i
d] - 2][2]:
                fingers.append(1)
            else:
                fingers.append(0)
        return fingers


def main():
    # 摄像头
    cap = cv.VideoCapture(0)

    pTime = 0
    cTime = 0

    detector = handDetector()
    while True:
        success, img = cap.read()
        img = detector.findHands(img)

        lmList = detector.findPosition(img)
        if len(lmList) != 0:
            print(lmList[8])
        # fps
```

```
        cTime = time.time()
        fps = 1 / (cTime - pTime)
        pTime = cTime

        # 显示 fps 到屏幕
        cv.putText(img, str(int(fps)), (10, 70), cv.FONT_HERSHEY_PLAIN, 3, (255, 0, 255), 3)

        cv.imshow('image', img)
        cv.waitKey(1)

  if __name__ == '__main__':
    main()
```

B. VirtualPainter.py

```
import cv2 as cv
import numpy as np
import time
import os
import HandTrackingModel as htm

brushTickness = 15
eraserTickness = 50

folderPath = 'Header'
myList = os.listdir(folderPath)
# print(myList)
overlayList = []

for imPath in myList:
    image = cv.imread(f'{folderPath}/{imPath}')
    overlayList.append(image)

# print(len(overlayList))
header = overlayList[0]
drawColor = (0, 0, 255)


cap = cv.VideoCapture(0)
cap.set(3, 1280)
cap.set(4, 720)

detector = htm.handDetector()
```

```
xp, yp = 0, 0

imgCanvas = np.zeros((720, 1280, 3), dtype='uint8')
while True:
    success, img = cap.read()
    # img = cv.flip(img, 1)
    img = detector.findHands(img)
    lmList = detector.findPosition(img, draw=False)
    if len(lmList) != 0:
        # print(lmList)

        x1, y1 = lmList[8][1:]
        x2, y2 = lmList[12][1:]

        fingers = detector.fingersUp()
        # print(fingers)

        # 模式选择
        if fingers[1] & fingers[2]:
            cv.rectangle(img, (x1, y1-
50), (x2, y2+50), (255, 0, 255), cv.FILLED)
            print("Selection Mode")
            # 画面高度 135
            if y1 < 135:
                if 250< x1 < 450:
                    header = overlayList[0]
                    drawColor = (0, 0, 255)
                elif 550< x1 < 750:
                    header = overlayList[1]
                    drawColor = (255, 0, 0)
                elif 800< x1 < 950:
                    header = overlayList[2]
                    drawColor = (0, 255, 0)
                elif 1050< x1 < 1200:
                    header = overlayList[3]
                    drawColor = (0, 0, 0)
                cv.circle(img, (x1, y1), 15, drawColor, cv.FILLED)
            xp, yp = x1, y1

        if fingers[1] & fingers[2]==False:
            cv.circle(img, (x1, y1), 15, (255, 0, 255), cv.FILLED)
            print("Drawing Mode")
            if xp == 0 and yp == 0:
                xp, yp = x1, y1
```

```
72.
73.             # 橡皮
74.             if drawColor == (0, 0, 0):
75.                 cv.line(img, (xp, yp), (x1, y1), drawColor, eraserTickness)
76.                 cv.line(imgCanvas, (xp, yp), (x1, y1), drawColor, eraserTick
    ness)
77.             else:
78.                 # draw
79.                 cv.line(img, (xp, yp), (x1, y1), drawColor, brushTickness)
80.                 cv.line(imgCanvas, (xp, yp), (x1, y1), drawColor, brushTickn
    ess)
81.
82.             xp, yp = x1, y1
83.
84.     # 合并画板
85.     imgGray = cv.cvtColor(imgCanvas, cv.COLOR_BGR2GRAY)
86.     _, imgInv = cv.threshold(imgGray, 50, 255, cv.THRESH_BINARY_INV)
87.     imgInv = cv.cvtColor(imgInv, cv.COLOR_GRAY2BGR)
88.     img = cv.bitwise_and(img, imgInv)
89.     img = cv.bitwise_or(img, imgCanvas)
90.
91.     h, w, c = overlayList[0].shape
92.     img[0:135, 0:1280] = header
93.
94.     cv.imshow('image', img)
95.     if cv.waitKey(10) & 0xFF==ord('q'):
96.             break
97.     cv.waitKey(1)
```

C. requirement.txt

```
1.  absl-py==1.3.0
2.  attrs==22.1.0
3.  contourpy==1.0.6
4.  cycler==0.11.0
5.  fonttools==4.38.0
6.  kiwisolver==1.4.4
7.  matplotlib==3.6.2
8.  mediapipe==0.8.11
9.  numpy==1.23.4
10. opencv-contrib-python==4.6.0.66
11. opencv-python==4.6.0.66
12. packaging==21.3
```

```
Pillow==9.3.0
protobuf==3.20.3
pyparsing==3.0.9
python-dateutil==2.8.2
python-opencv==1.0.0.14
six==1.16.0
```