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： 计算机与科学技术专业

年级班别： 2019级 JAVA 1 班

姓 名： 李世豪

指导教师： 李名

2023年3月

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。本文使用了OpenCV和MediaPipe两个工具库来实现该方法，并通过实验验证了其有效性和鲁棒性。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。在测试过程中，使用了多种不同的视频源进行验证，证明了该方法的有效性和鲁棒性。

关键词：手势识别；数字画板；高斯滤波；姿态估计；图像二值化。

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. OpenCV and MediaPipe are used in this article to implement this method, and its effectiveness and robustness are verified through experiments. 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. During testing, we used various video sources to verify the effectiveness and robustness of this method.

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

目录

[摘要 1](#_Toc130481068)

[Abstract 2](#_Toc130481069)

[目录 3](#_Toc130481070)

[**第一章 绪论** 5](#_Toc130481071)

[1.1 研究背景和意义 5](#_Toc130481072)

[1.2 国内外研究现状 6](#_Toc130481073)

[1.3 论文组织结构 7](#_Toc130481074)

[**第二章 相关技术** 8](#_Toc130481075)

[2.1 OpenCV与MediaPipe 8](#_Toc130481076)

[2.2 图像预处理 8](#_Toc130481077)

[2.2.1 噪声来源 9](#_Toc130481078)

[2.2.2 高斯滤波 9](#_Toc130481079)

[2.2.3 图像二值化 13](#_Toc130481080)

[**第三章 手势识别** 16](#_Toc130481081)

[3.1 手势识别的难点 16](#_Toc130481082)

[3.1.1 手势区域检测的影响 16](#_Toc130481083)

[3.1.2 手势识别环境的影响 17](#_Toc130481084)

[3.2 基于深度学习的手势识别算法设计 17](#_Toc130481085)

[3.2.1 预处理 17](#_Toc130481086)

[3.2.2 HandLandmark模块 18](#_Toc130481087)

[3.2.3 手部区域提取 18](#_Toc130481088)

[3.2.4 识别手势特征 19](#_Toc130481089)

[3.3 实验算法的确定 20](#_Toc130481090)

[3.3.1 原理与步骤 20](#_Toc130481091)

[3.3.2 实现细节 20](#_Toc130481092)

[3.3.3 优缺点 21](#_Toc130481093)

[**第四章 基于手势识别的数字画板实现** 22](#_Toc130481094)

[4.1 界面与主要功能 22](#_Toc130481095)

[4.2 指尖检测 23](#_Toc130481096)

[4.3 手势控制逻辑 24](#_Toc130481097)

[4.4 利用蒙版优化多媒体画面 24](#_Toc130481098)

[**第五章 总结与展望** 26](#_Toc130481099)

[致谢 27](#_Toc130481100)

[参考文献 28](#_Toc130481101)

[附录 29](#_Toc130481102)

1. **绪论**
2. 研究背景和意义

手势识别是一种基于计算机视觉和机器学习技术的人机交互技术，其主要目的是将人类语言中的手势姿势转化为计算机可识别的信号，并根据这些信号执行相应的任务。随着智能手机、平板电脑、智能电视等智能设备的普及，手势识别技术已成为人机交互领域中备受关注的热点之一。

手势识别技术的背景可以追溯到二十世纪初期，当时人们已经开始探索如何将手势姿势转化为计算机可识别的信号。随着计算机技术和机器学习技术的发展，手势识别技术得到了快速发展和广泛应用，涵盖了诸多领域。

随着计算机视觉和人机交互技术的发展，手势识别技术逐渐成为一个重要的研究领域。手势识别技术可以通过对人 类手部姿态的检测和分析，将手部运动转化为计算机可以识别和理解的指令。数字画板是一种常见的交互应用，其可以提供一个直观的绘画工具，帮助人们将创意转化为艺术作品。

数字画板已经成为数字媒体创作和设计的重要工具，其使用越来越广泛，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，研究开发一款基于机器学习技术的数字画板，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。

开发基于机器学习技术的数字画板，可以使得数字媒体的创作更加普及和简单。相较于传统数字画板，机器学习数字画板可以自动检测和处理手绘线条，优化用户体验，提高作品的美观度和准确性。同时，数字画板的应用也可以帮助开发更加智能和便利的数字笔记、漫画制作和数字绘画等应用程序，从而促进数字媒体的发展和应用。此外，数字画板也为机器学习算法的应用提供了一个重要的实验场所，有助于推动机器学习技术的进步和发展。

基于手势识别的数字画板研究的意义在于提高数字画板的交互性和可用性，使得人们可以通过手势来直接控制数字画板，而不需要使用鼠标或触摸屏等外设。这种方法可以让用户更加自然地进行操作，提高用户的使用体验。另外，基于手势识别的数字画板也可以拓展数字画板的应用领域，例如在教育、设计和娱乐等方面都有很大的潜力。因此，研究基于手势识别的数字画板具有重要的现实意义和应用价值。

1. 国内外研究现状

当今社会中，人机交互技术的发展迅速，其中手势识别技术作为其中的一种重要技术，得到了越来越广泛的应用和研究。手势识别技术的分类方式有很多种，分为触摸手势识别和非触控手势识别两类。其中，触摸手势识别技术主要是通过对触摸屏幕的触摸动作进行捕捉和识别的技术，例如对触摸屏幕的点击、滑动、缩放等动作进行识别，来实现各种功能。非触控手势识别技术则是通过传感器或摄像头等设备对手部运动进行捕捉和识别的技术。例如通过摄像头捕捉手部的运动轨迹并进行分析，来判断手势的类型和意图。另外，根据手势的识别方式，手势识别技术也可以分为基于传感器和基于图像的两类。

基于传感器的手势识别技术是指通过传感器对手部运动进行捕捉和识别的技术[14]。传感器可以通过加速度计、陀螺仪等设备对手部的加速度、角速度等运动参数进行测量和分析，从而判断手势的类型和意图。基于图像的手势识别技术则是指通过对手部的图像进行处理和分析来识别手势的类型和意图的技术。这种技术可以通过对手部的二维或三维图像进行特征提取和分类，从而实现手势识别。

手势识别技术在诸多领域都有广泛的应用，例如在虚拟现实、智能家居、智能车辆等领域中。随着人机交互技术的不断发展和普及，手势识别技术也将得到越来越广泛的应用和研究。因此，研究和发展手势识别技术对于推动人机交互技术的发展和提高人类生活质量具有重要意义。

传统的数字画板是指那些不依赖于计算机软件的数字画板，通常需要使用专门的数字笔或者触控笔进行操作，可以直接在画板上绘制图像，并将图像传输到计算机中进行后续的编辑和处理。这样虽然便于数字艺术家进行数字绘画和设计，但由于空间限制、价格昂贵、需要外部设备或软件、传统绘画方式的限制以及笔触的精细度限制等缺点，其在一些应用场景下存在不足。因此，需要寻求新的技术和方法来解决这些问题。

目前，数字画板方面的研究主要集中在国外学者中。例如，Liu等人（2021）基于深度学习开展了智能设计的研究，Li等人（2019）探讨了虚拟现实技术在数字画板中的应用，Sun等人（2017）研究了数字画板的在线协作技术，Wang等人（2020）提出了一种基于机器学习的自适应笔画技术，Li等人（2018）研究了数字画板的移动应用技术。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. 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，介绍研究背景以及意义、手势识别的研究内容与发展研究现状；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

1. **相关技术**
2. OpenCV与MediaPipe

Python-OpenCV和MediaPipe[12]都是计算机视觉领域常用的开源工具库，用于处理图像、视频和多媒体数据等任务。下面将对它们进行简单介绍：

Python-OpenCV是一个基于C++的开源计算机视觉库，由Intel开发并维护。它提供了许多用于图像处理、计算机视觉和机器学习等领域的函数和算法。Python-OpenCV支持多种操作系统平台，包括Windows、Linux和macOS等，也提供了Python语言的接口，方便Python开发者使用。Python-OpenCV常用于处理和分析图像、视频和深度图像等数据，也被广泛应用于图像处理、模式识别、物体检测和跟踪等领域。Python-OpenCV还支持GPU加速，可以提高计算效率。

MediaPipe是一个由Google开发的跨平台机器学习框架，MediaPipe 主要用于处理构建使用了多种形式的数据源(如视频、音频和各种其它传感器数据和时间序列数据)[2]。它提供了一组预先训练好的机器学习模型和工具，方便用户快速搭建自己的多媒体应用。MediaPipe支持多种平台，包括Android、iOS、Windows和macOS等，也提供了Python语言的接口。MediaPipe的一个重要特点是支持实时处理，可以在实时应用场景中运行，比如人脸识别、手势识别和姿态估计等。MediaPipe还提供了一些预先构建的处理模块，如人脸检测、姿态估计和手部跟踪等，方便用户快速搭建多媒体应用。

总的来说，Python-OpenCV和MediaPipe都是计算机视觉领域常用的工具库，它们提供了丰富的函数和算法，方便用户处理和分析图像、视频和深度图像等多媒体数据。Python-OpenCV常用于图像处理、计算机视觉和机器学习等领域，而MediaPipe则专注于实时多媒体处理应用程序，如人脸识别、手势识别和姿态估计等

1. 图像预处理

图像的预处理关键在于滤波和二值化处理[4]。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噪声和不必要的细节，从而使手势的特征更加突出，有利于算法对手势进行分析和识别。 滤波可以使得图像变得更加平滑，提高后续处理的效率，同时也可以降低识别误差。

在手势识别中，手势通常被表示为黑白二值图像，其中手势部分为白色，背景部分为黑色。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，使得算法更加简单和高效。同时，二值化可以通过提高图像的对比度和减少不必要的细节，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。

1. 噪声来源

在手势识别过程中，存在多种噪声来源，这些因素可能会对手势识别的准确性产生不良影响。本文对手势识别中常见的噪声来源进行了总结和归纳。

首先，环境噪声是影响手势识别的主要因素之一。包括光线、阴影、背景等因素对图像的影响。这些因素可能会导致图像中出现噪点、亮度失真、颜色失真等问题，从而影响手势识别的准确性。

其次，姿态噪声也是影响手势识别准确性的重要因素。由于手部的姿态变化可能会导致手势图像的旋转、缩放、畸变等问题，从而影响手势识别的准确性。

第三，低分辨率也是影响手势识别准确性的因素之一。低分辨率图像可能会导致图像中出现模糊、失真等问题，从而影响手势识别的准确性。

第四，手部遮挡也会影响手势识别的准确性。手部遮挡可能会导致手势图像的部分信息丢失，从而影响手势识别的准确性。

最后，噪声干扰也是影响手势识别准确性的因素之一。噪声干扰包括图像中的随机噪声、椒盐噪声等，这些噪声可能会影响手势图像的清晰度和准确性，从而影响手势识别的准确性。

为了解决这些噪声问题，可以采取一系列的图像处理技术，例如：去噪、增强、归一化、滤波等技术，以提高手势识别的准确性和鲁棒性。

1. 高斯滤波

高斯滤波是一种图像处理中常用的线性滤波方法，用于平滑和降噪图像。它的基本思想是将每个像素点的灰度值，按照与其距离越远的像素点给予越小的权重，然后计算加权平均值来代替该像素点的灰度值。这种权重函数通常被称为高斯函数[5]。

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。高斯滤波器的优点是能够有效地消除服从正态分布的噪声，并能够在一定程度上保留图像的细节信息。因此，在计算机视觉和图像处理领域，高斯滤波器被广泛应用于图像降噪和平滑处理等任务。

**算法步骤：**

在图像处理中，高斯滤波一般有两种实现方式，一是用离散化窗口滑窗卷积，另一种通过傅里叶变换。最常见的就是第一种滑窗实现，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[6]。

由于高斯函数可以写成可分离的形式，因此可以采用可分离滤波器实现来加速。所谓的可分离滤波器，就是可以把多维的卷积化成多个一维卷积。具体到二维的高斯滤波，就是指先对行做一维卷积，再对列做一维卷积。这样就可以将计算复杂度从O(M×M×N×N)降到O(2×M×M×N)，M，N分别是图像和滤波器的窗口大小。

高斯模糊是一个非常典型的图像卷积例子，本质上，高斯模糊就是将(灰度)图像 I 和一个高斯核进行卷积操作:

 （2.1）

其中 \* 表示卷积操作;  是标准差为 的二维高斯核,定义为:

 （2.2）

卷积是一个单纯的定义，本身没有什么意义可言，但是其在各个领域的应用是十分广泛的，在滤波中可以理解为一个加权平均过程，每一个像素点的值，都由其本身和邻域内的其他像素值经过加权平均后得到,而如何加权则是依据核函数高斯函数。

**平均的过程:**

对于图像来说，进行平滑和模糊，就是利用周边像素的平均值。

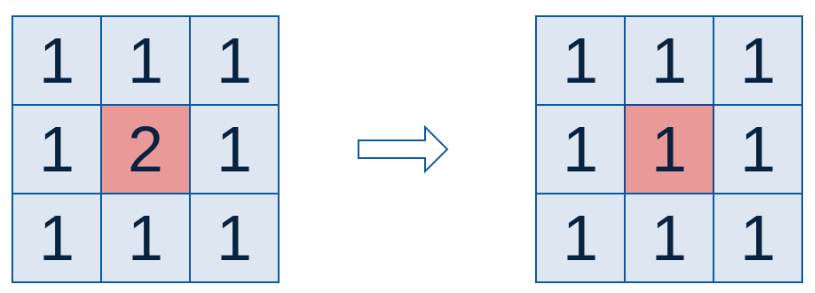


图 2. 1 像素矩阵

“中间点”取“周围点”的平均值，就会变成1。在数值上，这是一种“平滑化”。在图形上，就相当于产生“模糊”效果，“中间点”失去细节。显然，计算平均值时，取值范围越大，“模糊效果”越强烈。

计算平均值的时候，只需要将“中心点”作为原点，其他点按照其在正态曲线上的位置，分配权重，就可以得到一个加权平均值，而这就是上述的与二维高斯核进行卷积的过程（图2.1）。

**计算权重矩阵的过程:**

假定中心点的坐标是（0,0），那么距离它最近的8个点的坐标如下（图2.2）：

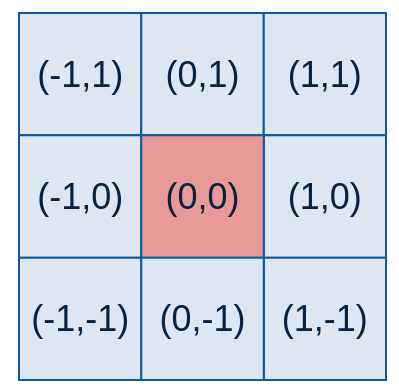


图 2. 2 权重矩阵坐标

假定=1.5，则模糊半径为1的权重矩阵如下（图2.3）：

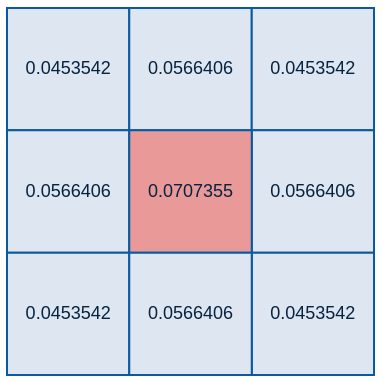


图 2. 3 模糊半径为1的权重矩阵

这9个点的权重总和等于0.4787147，如果只计算这9个点的加权平均，还必须让它们的权重之和等于1，因此上面9个值还要分别除以0.4787147，得到最终的权重矩阵（图2.4）：

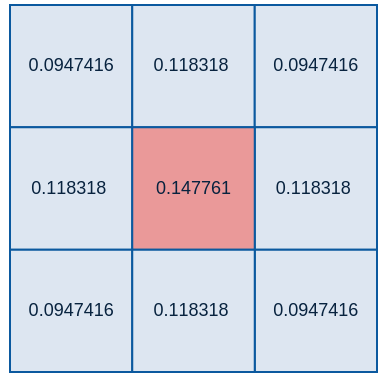


图 2. 4 最终权重矩阵

有了权重矩阵，就可以计算高斯模糊的值了。中心点以及周边n个点，每个点乘以自己的权重值并将这些值相加，就是中心点的高斯模糊的值。对所有点重复这个过程，就得到了高斯模糊后的图像。如果原图是彩色图片，可以对RGB三个通道分别做高斯模糊。



图 2. 5 原图



图 2. 6 高斯模糊

对于边界点来说，周边没有足够的点，一个变通方法是把已有的点拷贝到另一面的对应位置，模拟出完整的矩阵

1. 图像二值化

最简单的图像分割方法是二值化（Binarization）。图像二值化（Image Binarization）就是将图像上的像素点的灰度值设置为0或255，也就是将整个图像呈现出明显的黑白效果的过程。二值图像每个像素只有两种取值：要么纯黑，要么纯白。其中最常用的就是采用阈值法（Thresholding）进行二值化，（根据阈值选取方式的不同，可以分为全局阈值和局部阈值）[7]。

由于二值图像数据足够简单，许多视觉算法都依赖二值图像。通过二值图像，能更好地分析物体的形状和轮廓。二值图像也常常用作原始图像的掩模（图2.8）（又称遮罩、蒙版，Mask）：它就像一张部分镂空的纸，把不感兴趣的区域遮掉。进行二值化有多种方式，其中最常用的就是采用阈值法（Thresholding）进行二值化。



图 2.7 未被遮罩的原图

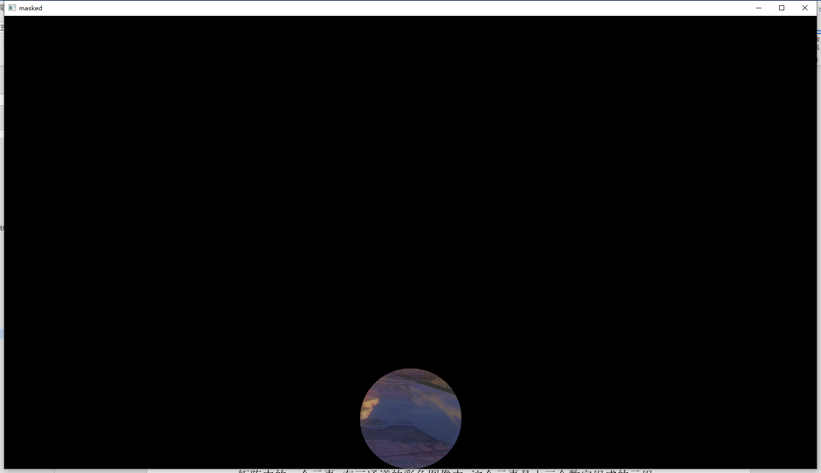


图 2.8 加入蒙版遮罩后的图

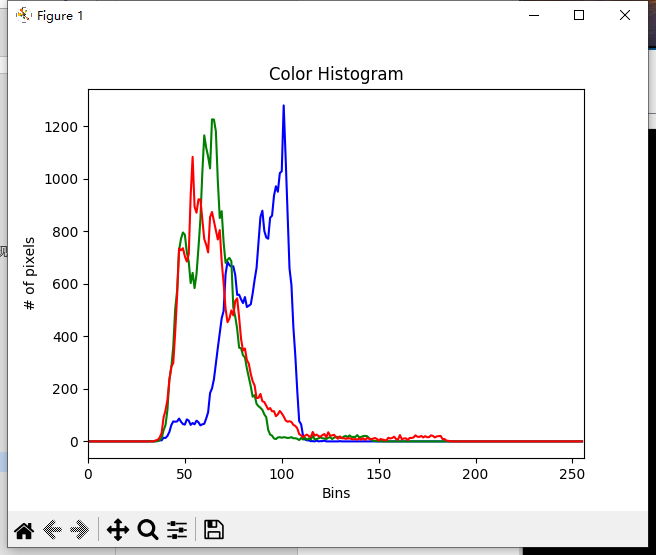


图 2.9 未被遮罩部分的直方图

在计算机视觉里，一般用矩阵来表示图像。也就是说，无论你的图片看上去多么好看，对计算机来说都不过是个矩阵而已。在这个矩阵里，每一个像素就是矩阵中的一个元素。在三通道的彩色图像中，这个元素是由三个数字组成的元组。而对于单通道的灰度图像来说，这个元素就是一个数字。这个数字代表了图像在这个点的亮度，数字越大像素点也就越亮，在常见的八位单通道色彩空间中，0代表全黑，255代表全白。

阈值法是指选取一个数字，大于它就视为全白，小于它就视为全黑。就像教室里的灯管开关，轻轻地推动它，如果突然间超过了某个阈值，灯就啪的一声亮了。根据阈值选取方式的不同，可以分为全局阈值和局部阈值。

全局阈值：将整个图像分为两个部分，即黑色和白色。在全局阈值中，使用一个固定的阈值，将图像中的每个像素与该阈值进行比较。如果像素值大于阈值，则像素被认为是白色，否则被认为是黑色。全局阈值适用于整个图像的亮度和对比度相似的情况。

局部阈值：根据每个像素周围的像素值来确定该像素的阈值。在局部阈值中，图像被分割成许多小块，每个小块都有自己的阈值。这些小块可以是固定大小的，也可以是不同大小的。这种方法适用于图像的亮度和对比度变化很大的情况，因为它可以根据像素周围的情况来确定阈值。

1. **手势识别**
2. 手势识别的难点
3. 手势区域检测的影响

手势识别作为一种基于人机交互的技术，具有一些挑战性的难点。本文将对手势识别技术中的难点进行深入探讨。

首先，多样性和变异性是手势识别的主要难点之一。人类的手势多种多样，而且同一种手势在不同的人之间也可能存在差异。因此，手势识别系统需要具备很强的鲁棒性，能够准确识别不同形态的手势。目前，一些基于深度学习的手势识别方法在解决手势多样性和变异性方面表现出了较好的效果。

其次，实时性和高效性也是手势识别技术中需要解决的难点之一。手势识别系统需要在实时性和高效性方面具备较强的能力，因为它通常会被应用在实时交互等场景中，如游戏、手势控制、虚拟现实等。近年来，基于GPU的并行计算和嵌入式系统等技术的出现，使得手势识别系统在实时性和高效性方面有了较大的提升。

第三，噪声和干扰也是手势识别系统需要解决的难点之一。手势识别系统在实际应用中可能会受到各种噪声和干扰，如光照变化、遮挡、手部晃动等，这些因素都可能导致手势的失效或误识别。针对这一难点，研究者们提出了一系列方法，如光照不变性、手部姿态估计、运动模型等。

第四，数据量和质量也是手势识别系统需要解决的难点之一。手势识别系统的性能通常受到数据量和数据质量的影响。大规模数据集和高质量的数据采集可以提高手势识别系统的准确性和泛化能力。近年来，一些研究者提出了一些数据增强技术和深度学习算法来解决数据量和质量的问题。

最后，实用性和易用性也是手势识别系统需要解决的难点之一。一个成功的手势识别系统需要考虑用户的需求和习惯，尽量设计出简单易用、易学易记的手势，并尽可能降低用户学习成本。因此，设计合适的用户界面、提供良好的用户反馈和交互方式，是提高手势识别系统实用性和易用性的关键因素。

综上所述，手势识别技术面临着多个挑战性的难点，如多样性和变异性、实时性和高效性、噪声和干扰、数据量和质量以及实用性和易用性等。这些难点需要研究者们从不同角度进行研究和解决，才能进一步推动手势识别技术的发展和应用。

1. 手势识别环境的影响

手势识别的环境对于识别结果具有重要影响。在实际应用中，手势识别系统需要在各种复杂的环境中运行，包括不同的光照条件、存在噪声和干扰的情况、背景干扰、操作距离和角度的不同、手势多样性和变异性等因素。

在光照条件方面，光线的强弱、方向和颜色等因素都会影响手势图像的质量和特征，从而影响识别的准确率。研究者们需要考虑不同的光照条件对手势识别系统的影响，提出更加稳健的算法，以保证在不同的光照环境下都能够准确地识别手势。

在噪声和干扰方面，由于手势识别是基于摄像头或传感器采集的数据进行分析和识别，因此噪声和干扰都会对数据的准确性产生负面影响。研究者们需要开发出更加鲁棒的算法，以抑制和降低噪声和干扰的影响，提高手势识别系统的准确性和稳定性。

在背景干扰方面，背景中存在的复杂纹理和颜色、人物、物体等元素，也可能与手势图像发生混淆，导致手势的误识别。研究者们需要开发出更加智能的算法，以区分手势和背景中的干扰因素，提高手势识别系统的准确性和鲁棒性。

在操作距离和角度方面，操作者和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。研究者们需要开发出更加灵活的算法，以适应不同的操作距离和角度，提高手势识别系统的适应性和准确性。

在手势多样性和变异性方面，不同人的手势习惯、手部特征、手势姿势等因素都可能导致手势的多样性和变异性，从而增加了识别的难度。研究者们需要开发出更加智能、自适应的算法，以应对不同用户的手势特征和习惯，提高手势识别系统的识别率和实用性。

综上所述，手势识别的环境因素对于识别结果具有重要影响，需要针对不同的环境因素进行相应的算法优化和改进。对于复杂的环境，需要开发出更加鲁棒、灵活和智能的手势识别算法，以实现高效、准确、稳定的手势识别系统的应用。

1. 基于深度学习的手势识别算法设计
2. 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。

1. HandLandmark模块

利用MediaPipe的HandLandmark[11]模块进行手部检测，得到手部关键点位置信息。该模块基于深度学习模型，能够快速、准确地检测手部[8]。

1. 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。

首先，通过MediaPipe库提供的HandLandmark模型，可以获取手部区域的21个关键点的三维坐标信息（图3.1）。可以通过这些关键点确定手部区域的坐标范围。

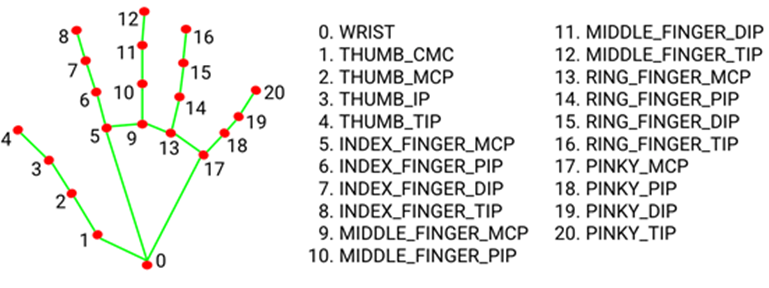


图 3. 1 手部关键点坐标

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）：

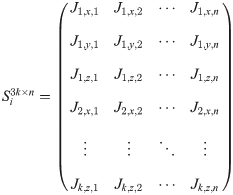


图 3. 2 信号组

其中k是关节特征的数量，n是帧数，Ju,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[13]。

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。

1. 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。这些特征被输入到支持向量机（SVM）分类器中进行分类，得到手势类别。

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。

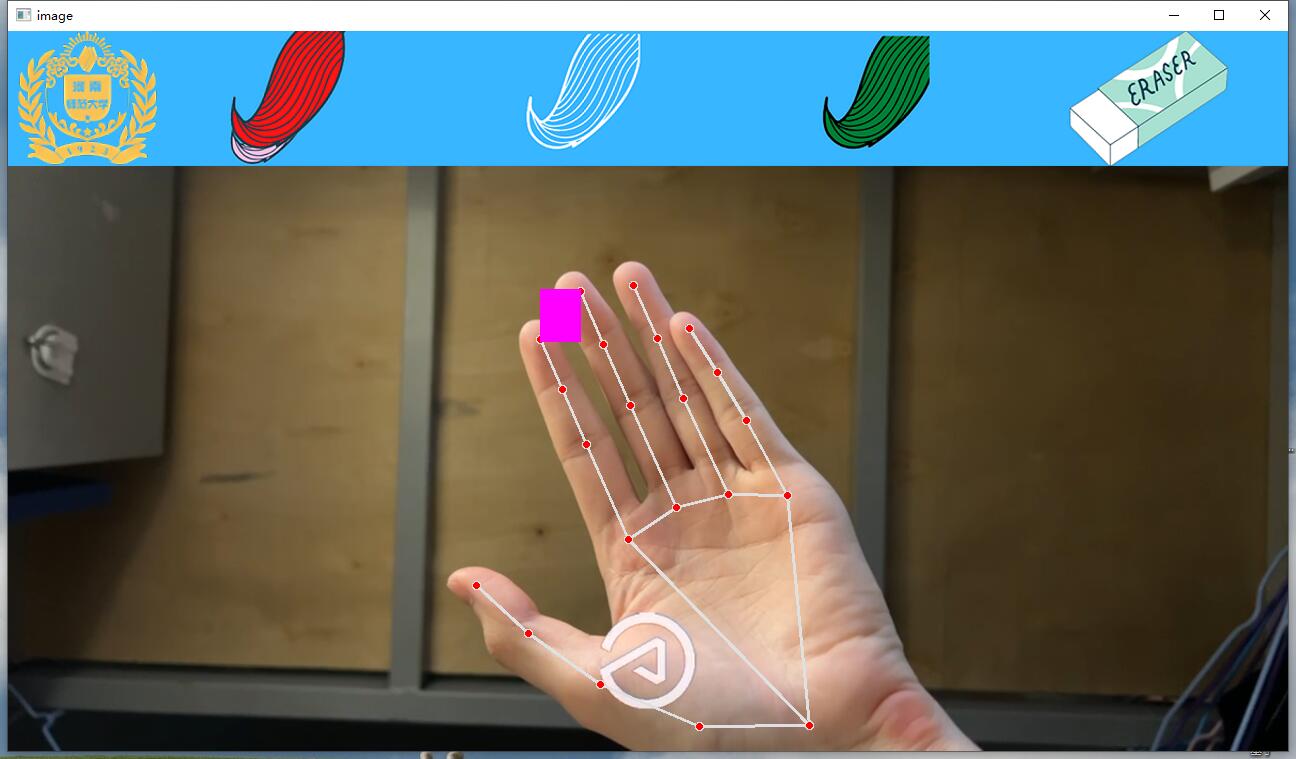


图 3. 3 选择模式

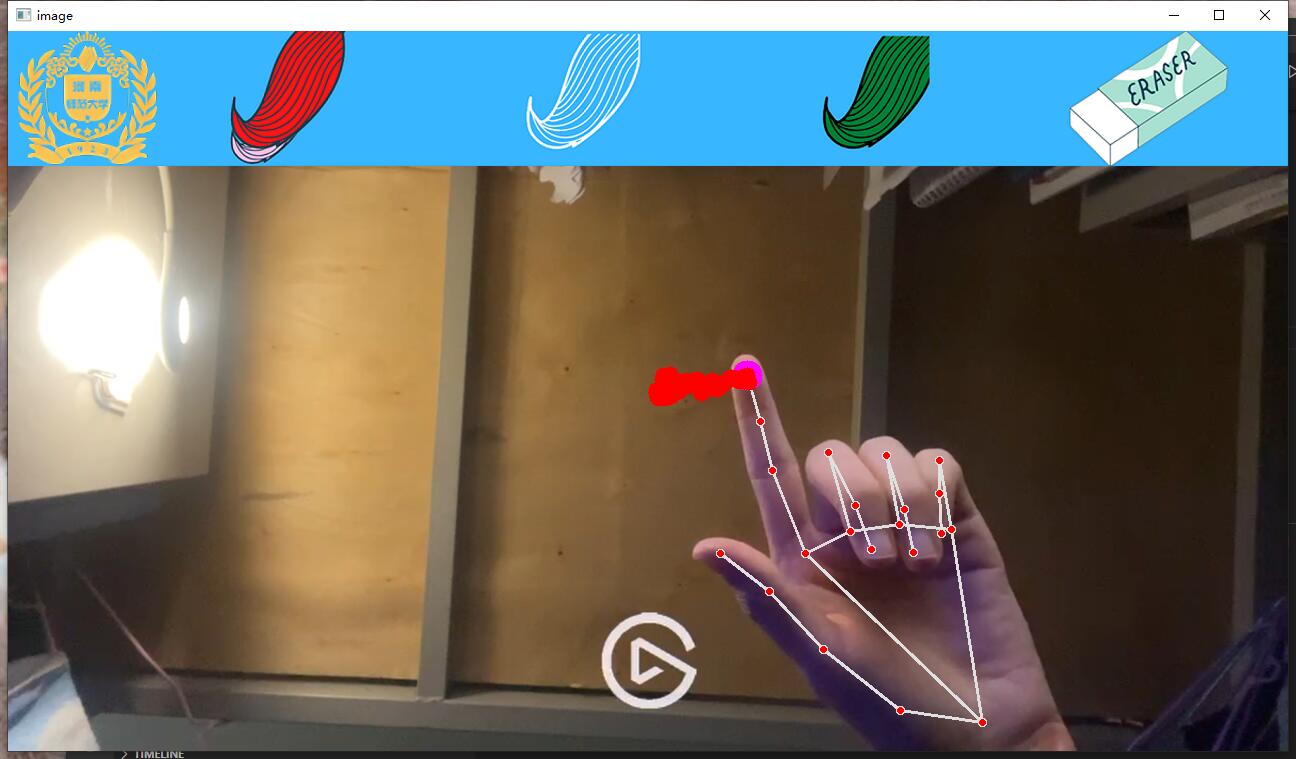


图 3. 4 绘画模式

1. 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 内置的手部识别算法是基于深度学习的卷积神经网络（Convolutional Neural Network, CNN）模型，可以实现实时的手部姿态估计和手势识别。该算法使用了一种称为“手部关键点检测”的技术，即通过检测手部的关键点来获取手部的姿态信息。这些关键点包括手腕、手掌中心、五指关节、指尖等等，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。

* 1. 原理与步骤

手部姿态估计算法的目标是对手部的姿态进行估计，即确定手部关键点的位置，如手指关节、掌心等。常见的手部姿态估计算法包括基于深度学习的方法和基于传统机器学习的方法[9]。

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. 使用基于深度学习的姿态估计算法估计手部关键点的位置；
4. 对估计结果进行后处理，如滤波、插值等，得到最终的手部姿态估计结果。
   1. 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；
2. 使用 OpenCV 读取输入图像，调用 MediaPipe 提供的 API 进行手部区域检测；
3. 对手部区域进行预处理，如缩放、裁剪等；
4. 使用 MediaPipe 提供的手部姿态估计 API 进行关键点估计；
5. 对估计结果进行后处理，如滤波、插值等，得到最终的手部姿态估计结果。
   1. 优缺点

**优点：**

1. 可以实现对手部姿态的快速准确估计；
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. 算法在复杂背景下的鲁棒性有待提高；
    3. 算法的计算复杂度较高，需要较高的计算资源支持。

1. **基于手势识别的数字画板实现**
2. 界面与主要功能

数字画板的界面设计应该简单、直观、易于操作。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4.1）。

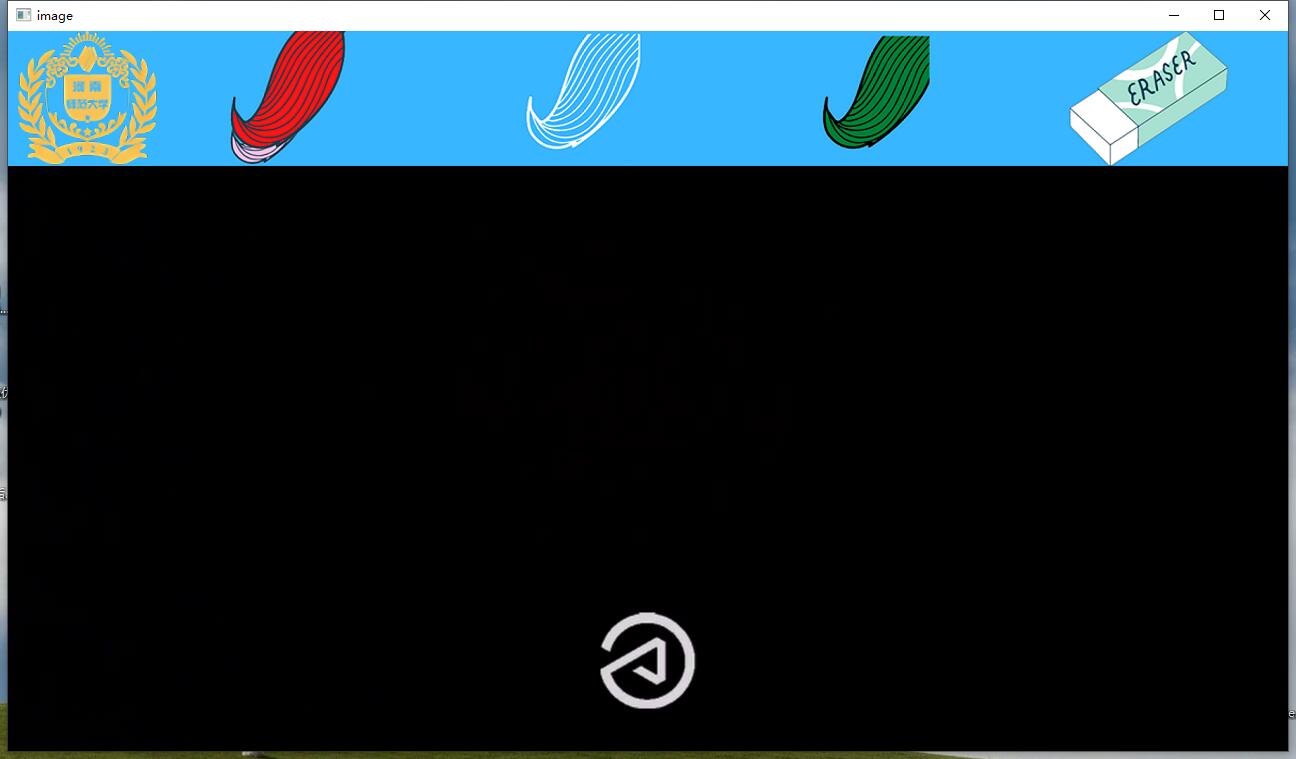


图 4. 1 界面设计分区

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（图4.2）。用户还可以通过控制区域切换画笔颜色（图4.3）进行绘画以及采用橡皮擦进行墨迹擦除（图4.4）。

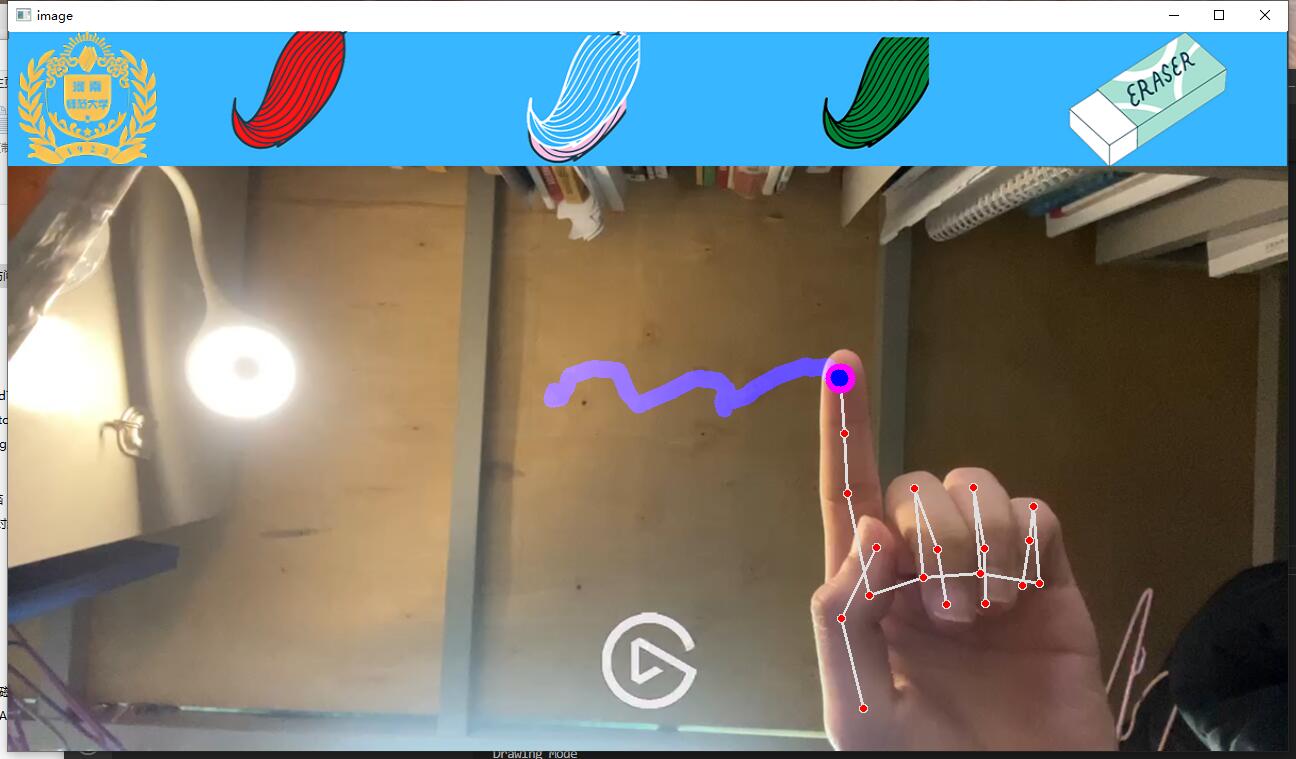


图 4.2 绘画

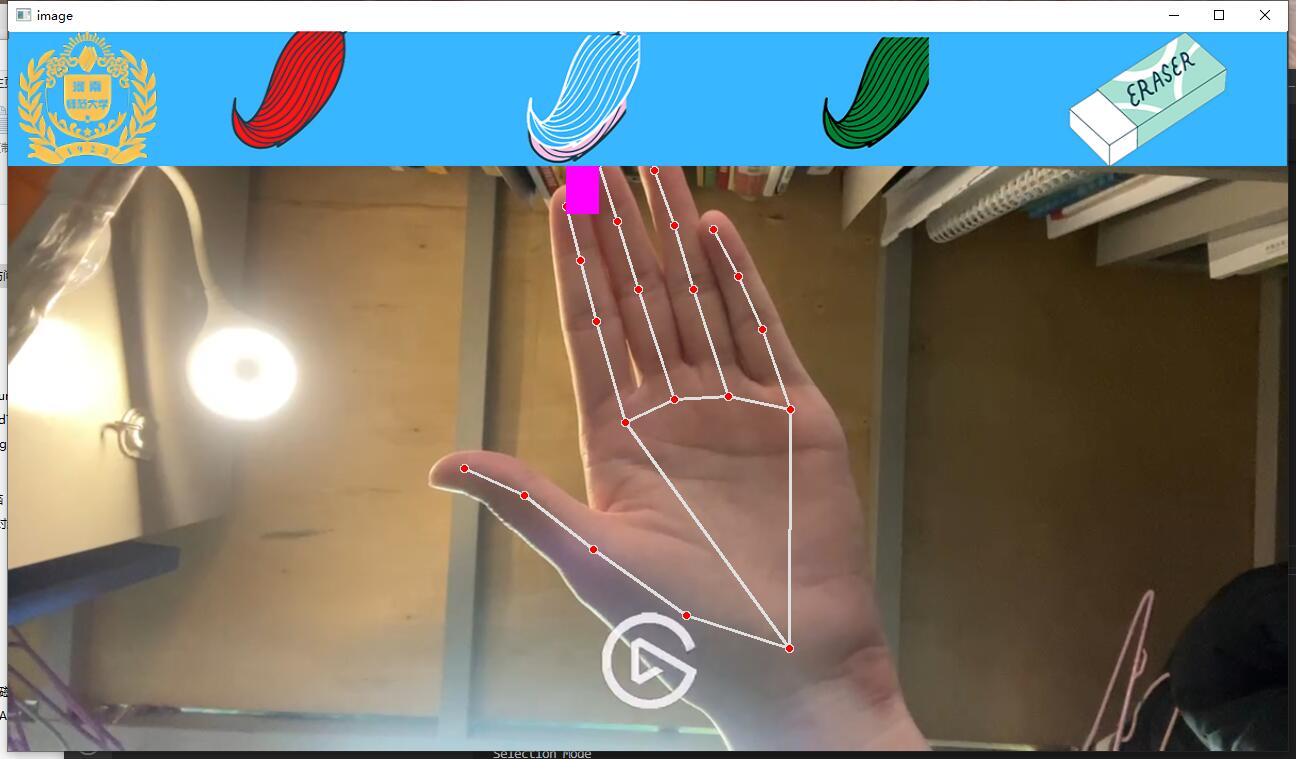


图 4. 3 切换画笔

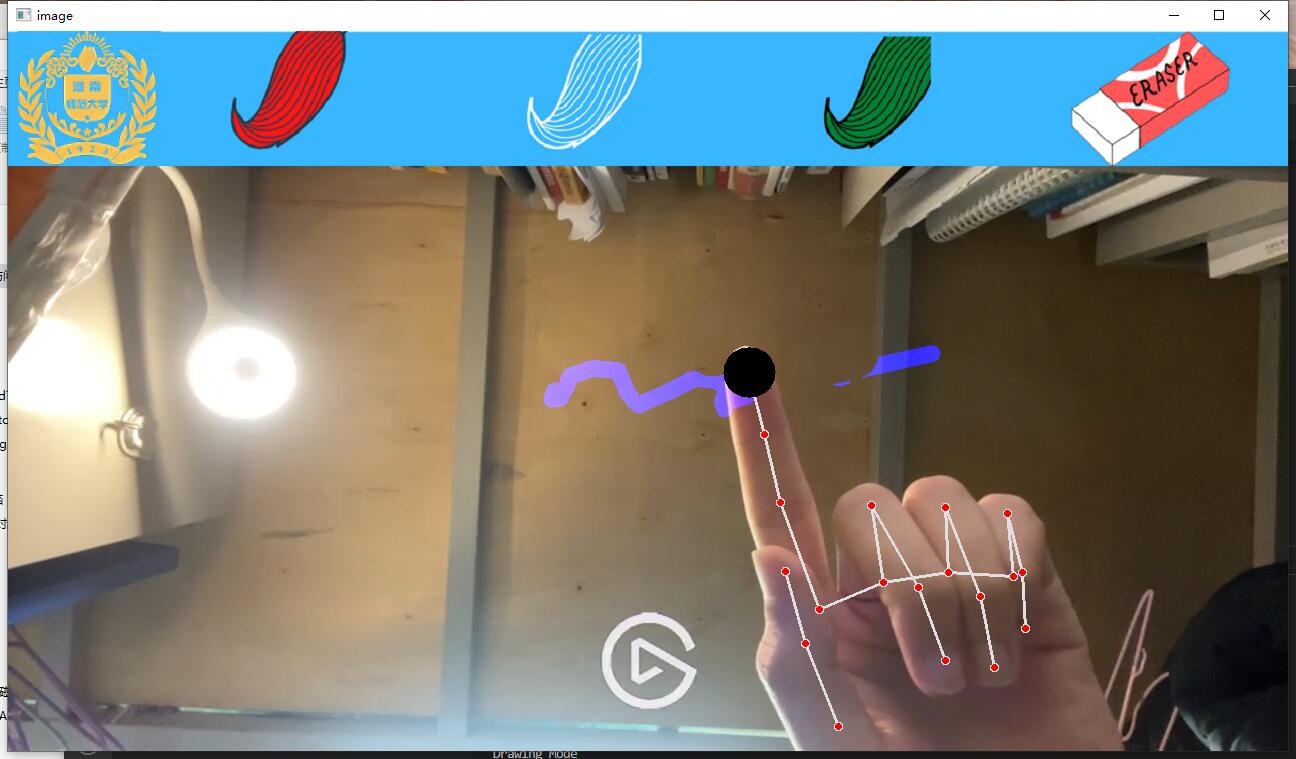


图 4. 4 擦除墨迹

1. 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。具体而言，算法首先将手部图像输入到神经网络中，然后通过卷积神经网络（CNN）对图像进行特征提取和分类，最终输出手势的类型和位置信息。

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4.2），其中包括手掌、手指和指尖。

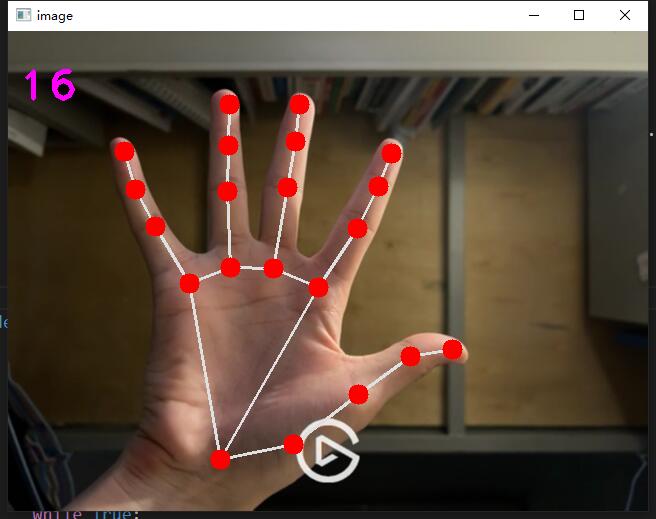


图 4.5 HandTracking模型手指关节地标

1. 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，本文还对手势控制逻辑进行了优化和调整。例如，对手势识别算法进行了参数调整，使其能够更好地适应不同的手势形态和动作。此外，还对手势检测的速度和响应时间进行了优化，使其可以在实时绘图过程中保持流畅和稳定。

1. 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4.6），所以可以使用蒙版来优化多媒体画面和画笔笔迹。

蒙版是一种用于控制图像显示和隐藏的技术，可以通过在图像上添加一个透明图层来实现。在此情况下，可以在获取的图像上添加一个透明图层，然后在该图层上进行手势识别和画画操作，以便将多媒体画面和画笔笔记分开，并使它们不会相互干扰。

具体来说，可以创建一个与获取的图像相同大小的空白图像，并将其用作蒙版。在蒙版图像上进行手势识别和画画操作，然后将其与获取的图像合并。这将使多媒体画面和画笔笔记分开，并使它们分别在蒙版和获取的图像中显示。

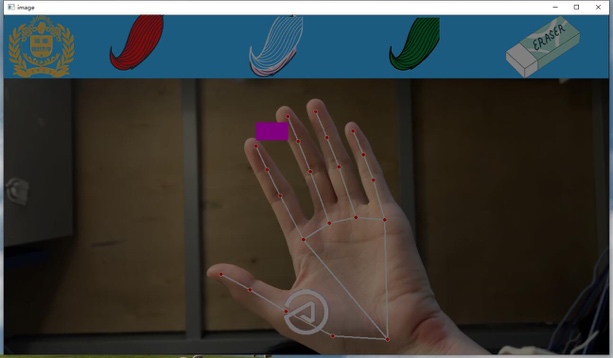


图 4.6 未优化前画面

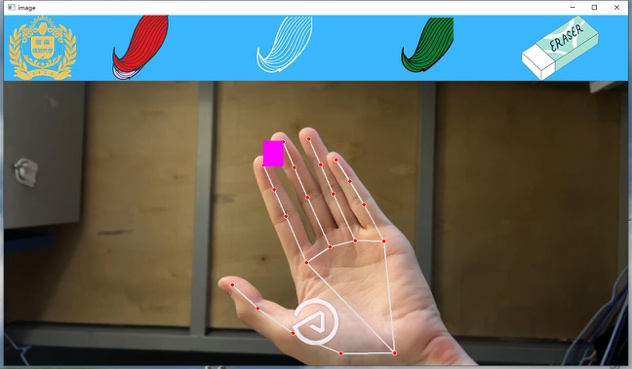


图 4.7 优化后图像

在将两个图像合并时，使用alpha混合技术来控制透明度，从而使多媒体画面和画笔笔记能够正确地显示在一起（图4.7）。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[10]。

1. **总结与展望**

在这个项目中，结合OpenCV和MediaPipe两个计算机视觉库开发了一个基于手势识别的数字画板应用程序。通过对用户手部的跟踪和手势的识别，用户可以使用数字笔在计算机屏幕上进行绘图。

在项目的实现过程中，学习了许多计算机视觉和机器学习的基础知识，例如图像预处理、特征提取、分类器训练等。还掌握了OpenCV和MediaPipe两个计算机视觉库的使用方法，学会了如何实现手部跟踪和手势识别等功能。

通过完成这个项目，也发现了一些潜在的改进和扩展方向。首先，可以进一步提高手部跟踪和手势识别的准确度和稳定性，以便用户可以更加流畅地进行绘画。其次，可以增加更多的功能，例如撤销和重做、保存和导出绘图等，以增强应用程序的实用性和用户体验。

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

参考文献

1. 庄会伟. 基于视觉的静态手势识别中关键技术的研究[D]. 济南: 山东大学, 2017.
2. 徐林尉. 手势识别研究及在教学系统中的应用[D].杭州电子科技大学,2017.
3. 王强宇. 基于深度神经网络的动态手势识别技术研究[D]. 中国矿业大学 (北京), 2019.
4. 陈忠辉,王等准,万广,方洪波,黄以卫,谢本亮.基于特征融合的手势识别[J].智能计算机与应用,2021,11(07):212-215+221.
5. 刘进锋. 几种 CUDA 加速高斯滤波算法的比较[J]. 计算机工程与应用, 2013, 49(23): 14-18.
6. 黄凯奇, 陈晓棠, 康运锋, 等. 智能视频监控技术综述[J]. 计算机学报, 2015, 38(6): 1093-1118.
7. 江明, 刘辉, 黄欢. 图像二值化技术的研究[J]. 教育技术导刊, 2009 (4): 175-177.
8. Zhang F, Bazarevsky V, Vakunov A, et al. Mediapipe hands: On-device real-time hand tracking[J]. arXiv preprint arXiv:2006.10214, 2020.
9. 孙浩翔. 基于深度学习的手部姿态估计[D]. 西北师范大学, 2020.
10. Magpayo J I D, Bacong J, Combras J J, et al. Basic Virtual Painting in Digital Canvas Using Mediapipe Algorithms: An Aid for Children with Amputated Fingers[J]. Available at SSRN 4357007.
11. 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.
12. Lugaresi C, Tang J, Nash H, McClanahan C, Uboweja E, Hays M, et al. MediaPipe: A framework for building perception pipelines. arXiv preprint arXiv:190608172. 2019.
13. Rodríguez-Moreno I, Martínez-Otzeta J M, Goienetxea I, et al. Sign language recognition by means of common spatial patterns[C]//2021 The 5th International Conference on Machine Learning and Soft Computing. 2021: 96-102.
14. 施列昱. 基于轻量级深度网络算法的CSI手势识别研究[D].南京邮电大学,2022.DOI:10.27251/d.cnki.gnjdc.2022.001497.

附录

项目文件树如图1所示，其中HandTrackingModel.py为手部识别模块，Header内图片为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2）、蓝色画笔（图3）、绿色画笔（图4）以及橡皮擦（图5），VirtualPainter.py为主程序，requirements.txt为项目所需文件库。

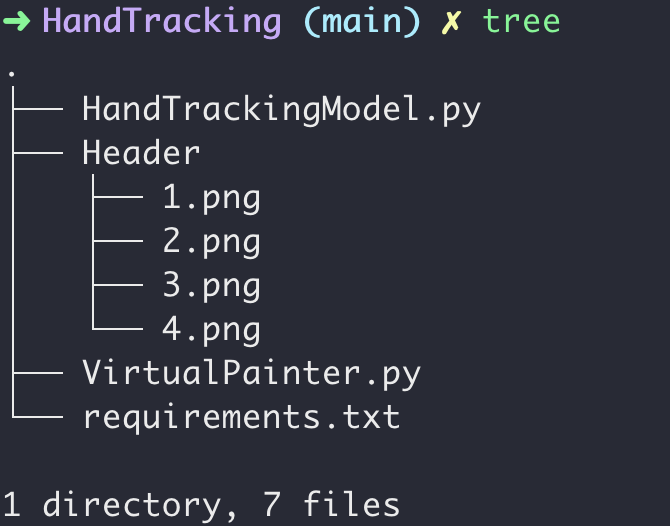


图 1 项目文件树



图 3 1.png



图 4 2.png



图 5 3.png



图 6 4.png

相关代码

HandTrackingModel.py

1. **import** cv2 as cv
2. **import** mediapipe as mp
3. **import** time

6. mpHands = mp.solutions.hands
7. hands = mpHands.Hands()
8. mpDraw = mp.solutions.drawing\_utils
10. **class** handDetector():
11. **def** \_\_init\_\_(self, mode=False, maxHand=2, detectionCon=1, trackCon=0.5):
12. self.mode = mode
13. self.maxHands = maxHand
14. self.detectionCon = detectionCon
15. self.trackCon = trackCon
17. self.mpHands = mp.solutions.hands
18. self.hands = self.mpHands.Hands(self.mode, self.maxHands, self.detectionCon, self.trackCon)
19. self.mpDraw = mp.solutions.drawing\_utils
21. self.tipsIds = [4, 8, 12, 16, 20]
22. **def** findHands(self, img, draw=True):
23. imgRGB = cv.cvtColor(img, cv.COLOR\_BGR2RGB)
24. self.results = hands.process(imgRGB)
26. **if** self.results.multi\_hand\_landmarks:
27. **for** handLms **in** self.results.multi\_hand\_landmarks:
28. # 编号和地标
29. **if** draw:
30. self.mpDraw.draw\_landmarks(img, handLms, self.mpHands.HAND\_CONNECTIONS)
31. **return** img
32. **def** findPosition(self, img, handNo=0, draw=True):
33. self.lmList = []
34. **if** self.results.multi\_hand\_landmarks:
35. myHand = self.results.multi\_hand\_landmarks[handNo]
36. **for** id, lm **in** enumerate(myHand.landmark):
37. # print(id, lm)
38. h, w, c = img.shape
39. cx, cy = int(lm.x \* w), int(lm.y \* h)
40. # print(id, cx, cy)
41. self.lmList.append([id, cx, cy])
42. # if id == 0:
43. **if** draw:
44. cv.circle(img, (cx, cy), 10, (0, 0, 255), cv.FILLED)
45. **return** self.lmList
47. **def** fingersUp(self):
48. fingers = []
49. # print(lmList)
50. # 大拇指 左手
51. **if** self.lmList[self.tipsIds[0]][1] > self.lmList[self.tipsIds[0] - 1][1]:
52. fingers.append(1)
53. **else**:
54. fingers.append(0)
55. # 其他四指
56. **for** id **in** range(1, 5):
57. **if** self.lmList[self.tipsIds[id]][2] < self.lmList[self.tipsIds[id] - 2][2]:
58. fingers.append(1)
59. **else**:
60. fingers.append(0)
61. **return** fingers


65. **def** main():
66. # 摄像头
67. cap = cv.VideoCapture(0)
69. pTime = 0
70. cTime = 0
72. detector = handDetector()
73. **while** True:
74. success, img = cap.read()
75. img = detector.findHands(img)
77. lmList = detector.findPosition(img)
78. **if** len(lmList) != 0:
79. **print**(lmList[8])
80. # fps
81. cTime = time.time()
82. fps = 1 / (cTime - pTime)
83. pTime = cTime
85. # 显示fps到屏幕
86. cv.putText(img, str(int(fps)), (10, 70), cv.FONT\_HERSHEY\_PLAIN, 3, (255, 0, 255), 3)

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

93. **if** \_\_name\_\_ == '\_\_main\_\_':
94. main()

VirtualPainter.py

1. **import** cv2 as cv
2. **import** numpy as np
3. **import** time
4. **import** os
5. **import** HandTrackingModel as htm
7. brushTickness = 15
8. eraserTickness = 50
10. folderPath = 'Header'
11. myList = os.listdir(folderPath)
12. # print(myList)
13. overlayList = []
15. **for** imPath **in** myList:
16. image = cv.imread(f'{folderPath}/{imPath}')
17. overlayList.append(image)
19. # print(len(overlayList))
20. header = overlayList[0]
21. drawColor = (0, 0, 255)

24. cap = cv.VideoCapture(0)
25. cap.set(3, 1280)
26. cap.set(4, 720)
28. detector = htm.handDetector()
29. xp, yp = 0, 0
31. imgCanvas = np.zeros((720, 1280, 3), dtype='uint8')
32. **while** True:
33. success, img = cap.read()
34. # img = cv.flip(img, 1)
35. img = detector.findHands(img)
36. lmList = detector.findPosition(img, draw=False)
37. **if** len(lmList) != 0:
38. # print(lmList)
40. x1, y1 = lmList[8][1:]
41. x2, y2 = lmList[12][1:]
43. fingers = detector.fingersUp()
44. # print(fingers)
46. # 模式选择
47. **if** fingers[1] & fingers[2]:
48. cv.rectangle(img, (x1, y1-50), (x2, y2+50), (255, 0, 255), cv.FILLED)
49. **print**("Selection Mode")
50. # 画面高度135
51. **if** y1 < 135:
52. **if** 250< x1 < 450:
53. header = overlayList[0]
54. drawColor = (0, 0, 255)
55. **elif** 550< x1 < 750:
56. header = overlayList[1]
57. drawColor = (255, 0, 0)
58. **elif** 800< x1 < 950:
59. header = overlayList[2]
60. drawColor = (0, 255, 0)
61. **elif** 1050< x1 < 1200:
62. header = overlayList[3]
63. drawColor = (0, 0, 0)
64. cv.circle(img, (x1, y1), 15, drawColor, cv.FILLED)
65. xp, yp = x1, y1
67. **if** fingers[1] & fingers[2]==False:
68. cv.circle(img, (x1, y1), 15, (255, 0, 255), cv.FILLED)
69. **print**("Drawing Mode")
70. **if** xp == 0 **and** yp == 0:
71. xp, yp = x1, y1
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, eraserTickness)
77. **else**:
78. # draw
79. cv.line(img, (xp, yp), (x1, y1), drawColor, brushTickness)
80. cv.line(imgCanvas, (xp, yp), (x1, y1), drawColor, brushTickness)
82. xp, yp = x1, y1
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)
91. h, w, c = overlayList[0].shape
92. img[0:135, 0:1280] = header
93. cv.imshow('image', img)
94. **if** cv.waitKey(10) & 0xFF==ord('q'):
95. **break**
96. cv.waitKey(1)

requirement.txt

absl-py==1.3.0

attrs==22.1.0

contourpy==1.0.6

cycler==0.11.0

fonttools==4.38.0

kiwisolver==1.4.4

matplotlib==3.6.2

mediapipe==0.8.11

numpy==1.23.4

opencv-contrib-python==4.6.0.66

opencv-python==4.6.0.66

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