文章编号:1671-4598(2025)05-0239-08 DOI:10.16526/j.cnki.11-4762/tp.2025.05.028 中图分类号:TP311.1 文献标识码:A

一种数据流驱动的 Python 视觉实验程序开发方法

胡 杰,何岭松,丁 依
(华中科技大学 机械科学与工程学院,武汉 430000)

摘要:Python+OpenCV 可以满足各种常见的机器视觉实验程序的设计需求,但与 MATLAB 等商业软件相比,缺乏一个好用的可视化编程环境;为此,提出一种数据流驱动的视觉实验程序开发架构;将按钮、文本框等 GUI 交互控件和图像显示、图像处理功能封装为带输入、输出的双端口组件;通过组件间输入、输出端口的直连就可以快速形成可运行的视觉实验程序框架;将双端口组件和组件之间的连接导出就可得到框架程序的 Python 脚本,供用户进一步完善和修改,其功能就类似与 MATLAB 中的 App Designer;为便于理解,最后给出了一个手势检测视觉实验的框架程序快速生成和修改完善样例。

关键词:Python;数据流;可视化编程;机器视觉;实验开发

A Development Method for Data Flow-driven Python Visual Experiment Program

HU Jie, HE Lingsong, DING Yi
(School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430000, China)

Abstract: Python+OpenCV can meet the design needs of various common machine vision experimental programs, but compared with commercial software such as MATLAB, it lacks a user-friendly visual programming environment. For this reason, a development architecture of data flow-driven visual experimental program is proposed; Encapsulate GUI interactive controls such as buttons and text boxes, as well as image display and image processing, into dual-port components with input and output; By directly connecting the input and output ports between components, it can quickly form a running visual experiment program framework; Export the dual-port components and connections between the components to achieve the Python script for the framework program, which can be further improve and modify by users. Its function is similar to App Designer in MATLAB. Finally, a framework program for gesture detection visual experiment is provided, which can quickly generate and modify perfect samples.

Keywords: Python; data flow; visual programming; machine vision; experimental development

0 引言

在现行的机器视觉课程教学中,一般借助 HALCON、MATLAB 这类商用大型软件平台进行机器视觉实验设计[1-2]。其中 HALCON 提供了近 2 000 个算子,可通过 C/C++和. NET 等多种编程语言调用[3],支持 100 多种工业相机和采集卡,性能高效,运行稳定,被公认为是功能最强的机器视觉软件之一;MATLAB 提供 Image Processing Toolbox 功能包[4],包含图像滤波、图像增强、深度学习和三维体图像处理等功能,搭配 App Designer 能进行机器视觉程序的开发[5]。商用软件平台内置稳定可靠的机器视觉算法,学生可以调用这些算法进行视觉实验,认知图像处理的过程,但其中仍

收稿日期:2024-03-14; **修回日期**:2024-05-06。
基金项目:国家重点研发计划(2019YFB1310703)。
作者简介:胡 杰(1999-),男,硕士研究生。
通讯作者:何岭松(1962-),男,教授,博士生导师。
引用格式:胡 杰,何岭松,丁 依.一种数据流驱动的 Python 视觉实验程序开发方法[J].计算机测量与控制,2025,33(5):239-248,254.

存在一系列的问题。一是这些商用软件是闭源软件[6]，学生可能无法深入了解其内部工作原理，限制了其对机器视觉算法代码如何实现的了解；二是软件庞大复杂，文档和支持不足，学习曲线陡峭，需要花费很大精力理解其功能和使用方法；三是平台受限，嵌入式设备由于系统或者性能原因无法运行这些软件[7]，面对一些与硬件相结合的实验无能为力；最后是这些软件缺乏一些有关 AI 的“与时俱进”的新功能[8]。

或者借助 OpenCV 等开源视觉计算库进行视觉实验程序开发[9]，涉及到外部图像数据的读取、图像数据的处理、具体功能的实现、GUI（图形用户界面）库的引入和连接等内容，繁琐的过程中真正与机器视觉算法相关的不多，背离了实验教学的初衷，同时对学生的编程能力是一个很大的挑战。上述原因造成了无论是商业视觉软件还是开源视觉框架在实验教学中的表现都没有达到预期，视觉实验程序开发缺乏一个开放的方便的可视化编程环境。

Python 是现在使用最广泛的解释型语言，拥有众多的优质开源机器视觉计算库[10-11]，例如 OpenCV、Dlib、YOLO 等；也拥有众多优秀的 GUI 框架，例如 PySide6、Tkinter 等[12]。可以设计一个基于 Python 的程序开发框架，将 Python 上的这些资源整合起来，能够通过图形化编程的方式迅速生成可运行的视觉实验程序的框架代码，以 Python 脚本的形式向学生开放，让学生专注于机器视觉算法的调试和完善，同时兼顾功能性、扩展性和跨平台性，Python 脚本可以轻易进行平台移植[13]，基于 Python 上的各类通信协议库与硬件平台通信，实现软硬件协同，这种框架能够很好地满足机器视觉类课程的视觉实验程序开发需求。图形化加低代码的混合编程思路并不鲜见[14]，例如 MATLAB 上 App Designer 工具结合 m 代码开发程序的方式就是该思路的应用[15]。但针对机器视觉实验的程序开发，基于 Python 并采用混合编程思路实现快速开发的方法在国内外公开资料上未见先例。同时，本文方法开发得到的程序为数据流驱动，相较于传统混合编程方法省去了在代码中声明回调函数的环节，更进一步。

本文提出的视觉实验程序快速开发方法示意图如图 1 所示。其设计优势在于：

1）强大功能性和跨平台性。基于 Python 语言，程序功能性强，能够方便地引入 Python 强大、丰富、前沿的视觉计算功能包；不受平台限制，能够进行近乎无成本的平台移植，包括支持 Python 的嵌入式设备[16]，相较于目前依赖大型商业软件的做法，留给学生的发挥空间更大；

2）开发迅速高效。能够迅速开发视觉实验程序，且所得程序基于数据流驱动，用户可以专注于输入的图像数据和具体功能算法，而无需关注数据的传递和流动；

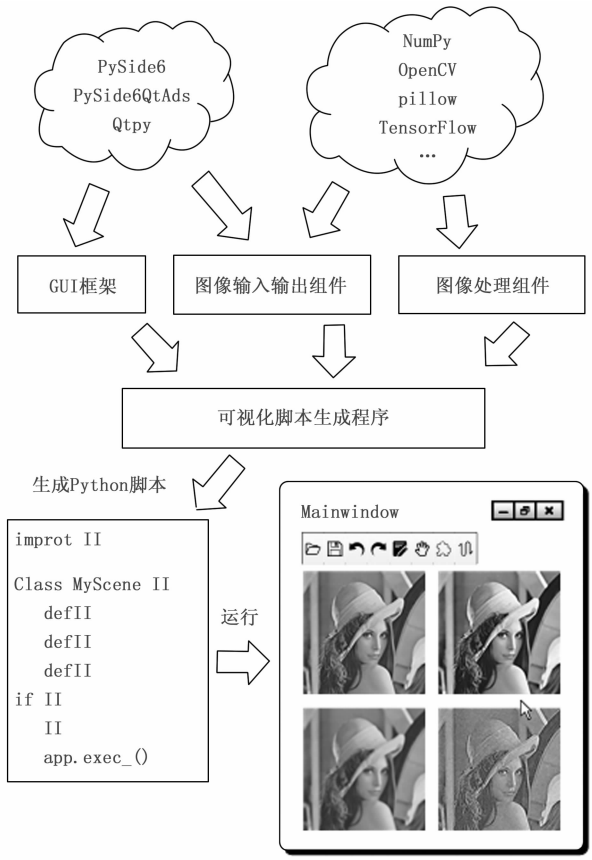

图 1 Python 实现的视觉实验程序开发方法示意图

3）扩展性极强。开发所得的程序本质上是一段 Python 脚本，可以方便地引入其它模块进行扩展。包括但不限于通过 RTMP 协议获取网络视频流进行图像处理[17]，通过串口通信模块[18]，结合图像中的视觉信息编写指令发送给与计算机连接的机械臂，做出一些动作反馈，实现软硬件联动。

本文将对数据流驱动的视觉实验程序架构、程序 UI 框架和机器视觉实验组件的 Python 类结构实现方法、Python 脚本的结构和装配以及动态修改进行介绍，并于最后给出图形化的脚本生成程序，使用其快速设计一个手势检测视觉实验作为样例。

1 数据流驱动的机器视觉实验程序架构

1.1 数据流驱动的双端口组件设计

涉及到带有 GUI 的桌面应用程序开发时，常借助 Qt、Tkinter 等图形界面库提供的控件来创建窗口、按钮、文本框等用户界面元素[19]，并通过图形控件的触发事件和特定的函数或方法进行事件绑定，来构建交互式的应用程序。例如在 Qt 中，通过信号与槽机制在对象之间进行通信[20]。但是这类基于事件绑定的程序内组件通信方案，在视觉实验程序开发中有一定的缺陷：

1）从机器视觉实验程序的易用性出发，程序的触发事件应当尽量少，较为理想情况是用户只需要关心图像数据的输入，输入后图像数据能够在程序的组件之间自如地流动和传递，实现对图像数据的处理和输出。换言之，视觉实验程序的组件之间需要的是数据流绑定，而非事件绑定。

2）触发事件接口不统一，通常需要在 GUI 布局后，在代码中实现具体的事件绑定。本文希望在视觉实验程序开发中，在 GUI 布局的同时能够轻易地设置组件之间的数据流绑定关系，以便直接生成可运行的框架代码，因此组件需要具备统一的数据端口。

基于事件绑定在视觉实验程序开发上的两点不足，本文提出一种支持数据流的双端口组件设计，其结构如图 2 所示。组件具备统一的输入输出接口，有相同数量的数据通道，以便于在 GUI 布局时能够完成组件之间的数据流绑定。组件的核心功能在数据处理模块实现，通过订阅管理模块记录与该组件绑定的下游组件，并通过数据发送模块实现数据流的传递。

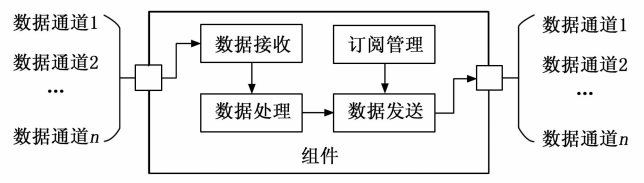

图 2　支持数据流的双端口组件

1.2　视觉实验程序架构设计

对于一个机器视觉实验，其中涉及的要素通常有：原始图像、若干个图像处理过程、若干个阶段性处理所得图像和最终处理完成的图像。对应到视觉实验程序，表现为一个图像输入组件，若干个图像处理组件和若干个图像显示组件的有序连接。组件均为支持数据流的双端口组件，但对于图像输入组件，其输入端口废弃，采用事件绑定的方式，通过用户的 GUI 操作从系统中获取图像数据；对于图像显示组件，其输出端口废弃，无数据输出的需求。假设某个机器视觉实验有 3 个图像处理步骤，则其视觉实验程序的核心组件及数据流绑定关系如图 3 所示。

图像输入组件主要是对一些常规图像源的数据流封装，例如摄像头、图片、视频和网络视频流等，依赖 PySide6、OpenCV 和 NumPy 实现；图像处理组件是对各类视觉算法和功能的封装，是视觉实验程序的核心，依赖 OpenCV、Pillow、TensorFlow 等开源视觉库实现；图像显示组件负责接收图像数据并显示，依赖 PySide6 和 OpenCV 实现；除去上述组件外，视觉实验程序还包括 UI 框架，其负责输入组件和显示组件的界面布局和

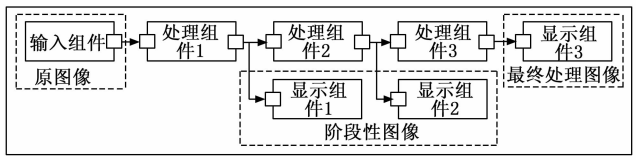

图 3　程序核心组件及数据流绑定关系

显示，同时提供 GUI 界面与用户交互，依赖 PySide6、PySide6QtAds 实现。单个视觉实验程序对应单个机器视觉实验场景，用户在图形化的脚本生成程序中选择和布局组件、设置组件之间的数据流关系，脚本生成程序根据用户操作生成 Python 脚本，该脚本能够一键运行，得到视觉实验程序，视觉实验程序架构设计如图 4 所示。脚本生成程序生成的 Python 脚本是框架代码，需要根据视觉实验的具体场景进一步调整图像处理组件的相关代码，才能得到理想的实验效果。

在该架构下，视觉实验程序被分解为程序 UI 框架和若干个组件的耦合，组件之间以及组件和 UI 框架之间的连接通过统一的数据流接口实现，清晰的程序结构和简单高效的数据传递方式，使得 Python 脚本的自动化装配易于实现。将函数或者功能组件化的设计，让程序的功能界限划分明显，便于用户进行调试，同时通过提供组件封装接口，用户可自行扩充组件库，丰富视觉实验程序的功能。

2　UI 框架和组件的 Python 类结构实现方法

2.1　UI 框架的实现

程序 UI 框架负责界面的显示、交互、图像输入/输出组件的布局、图像处理组件的注册和编辑页面加载等内容，其继承自 PySide6 中的 QMainWindow 类，并依赖 PySide6 中的 QWidget、QAction、QGridLayout 等控件实现 UI 功能，同时依赖 PySide6QtAds 包进行窗口布局管理。本文将 Python 脚本程序 UI 框架对象命名

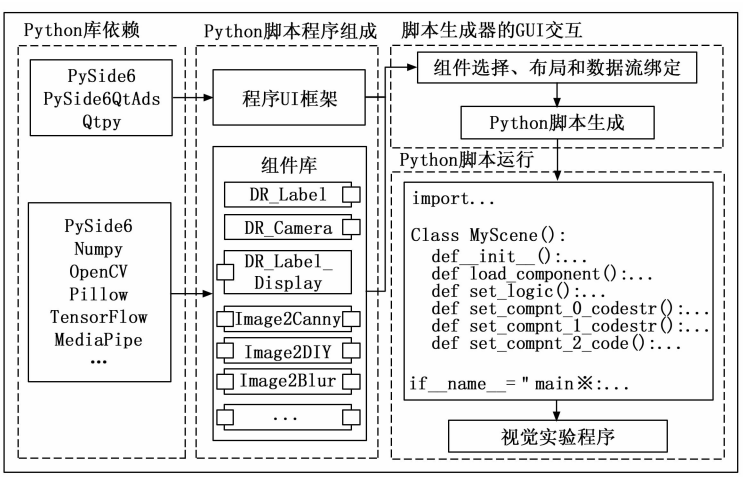

图 4　视觉实验程序架构设计

为 DRVision，意为动态可重构视觉（Dynamic Reconfigurable Vision）对象。每个 Python 脚本程序对应一个 Scene（视觉实验场景）对象，Scene 继承 DRVision，加载各类图像组件、添加图像处理组件的特定函数，组成一个功能化的可运行的 Python 脚本。图 5 是 Python 脚本程序 UI 框架核心部分的 UML 类图。

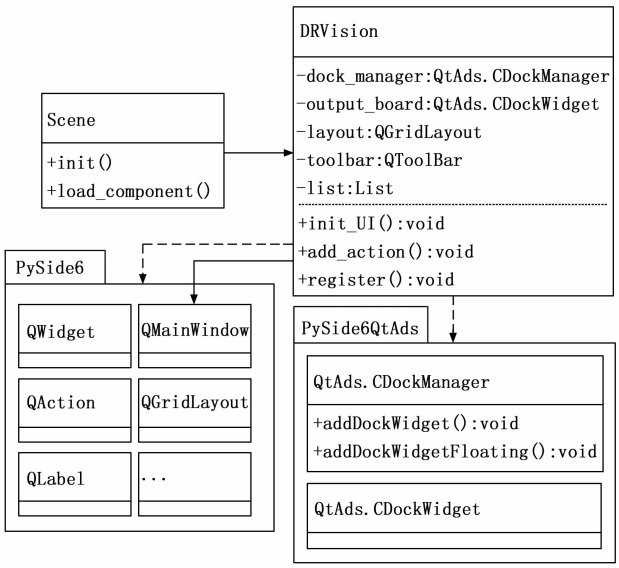

图 5 UI 框架核心部分 UML 类图

在 UI 框架的基础上，Python 脚本仅需在 Scene 对象中声明图像输入/输出组件和图像处理组件，并在 load _ component（）方法中进行注册和加载，便可以实现 Python 脚本程序的界面初始化。在组件规范命名的基础上，UI 相关 Python 脚本能够由图形化的脚本生成程序基于用户操作自动生成，成为最终可运行的 Python 脚本的一部分。

2.2 图像输入和输出组件的实现

图像输入组件从系统中读取图像数据，显示原图像，并通过统一的数据流端口传递给图像处理组件；图像输出组件接收图像处理组件的数据流，并显示处理后的图像；二者均使用了 PySide6 中的 QLabel 控件。视觉实验中涉及的图像输入方式主要有图片、摄像头和视频，分别对应图片输入组件、摄像头输入组件和视频输入组件；对于这 3 种图像数据输入，均可由数据流驱动的图片显示组件显示图像处理结果。其中，图片输入组件的实现比较简单，本文将介绍摄像头输入组件、视频输入组件和图片显示组件的 Python 类实现，并说明组件之间统一的数据流格式。

1）摄像头输入组件：

摄像头输入组件主要成员的 UML 类图如图 6 所示，本文将其命名为 DR _ Camera。DR _ Camera 继承自 PySide6 中的 QLabel 控件，实现摄像头图像的显示；

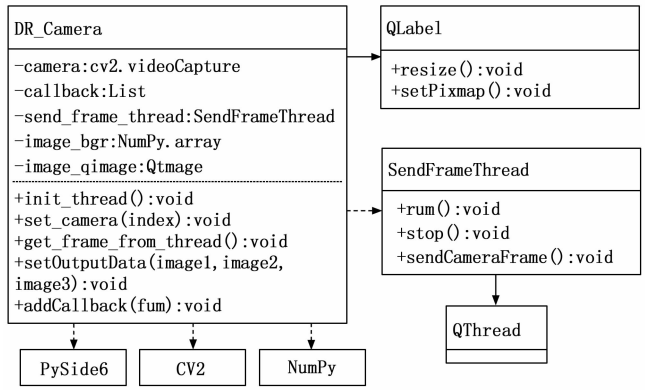

图 6 摄像头输入组件主要成员的 UML 类图

通过 OpenCV 调取摄像头，并依赖 QThread 为摄像头图像读取任务开辟单独线程，通过 get _ frame _ from _ thread（）方法从线程中持续获取图像帧；addCallback（）方法用来记录其它视觉实验组件的数据输入端口地址，保存在 callback 链表对象中，用于建立组件之间的数据流关系；setOutputData（）方法用来向其它组件发送图像数据，有 3 个图像通道，本文设计时将其对应每帧图像的灰度图像、原图像和 QImage 图像，DR _ Camera 向外发送数据流时仅有原图像和 QImage 图像，对应 image _ bgr 和 image _ qimage，灰度通道将被置空，发送时遍历 callback 成员中的函数地址，并将 image _ bgr 和 image _ qimage 数据传入。

2）视频输入组件：

视频输入组件的数据流封装与摄像头组件类似，不同的是其图像帧依赖 PySide6 中的 QMediaPlayer 和 QVideoSink 控件，从本地视频文件中获取，本文将其命名为 DR _ Video。DR _ Video 中创建 QMediaPlayer 对象并指定本地视频文件路径，创建 QVideoSink 对象并通过 QMediaPlayer 对象的 setVideoSink（）方法将其绑定，基于 QVideoSink 的 videoFrameChanged（）信号获取图像帧，处理后通过 DR _ Video 的 setOutputData（）方法对外发送图像数据流。

3）图片显示组件：

图片显示组件同样继承自 QLabel，本文将其命名为 DR _ Label _ Display。其接收数据的函数为 setInputData（），其它组件需要向其传递数据时，调用 addCallback（）方法将 setInputData（）的函数地址记录即可，setInputData（）函数同样有 3 个形参，对应 3 个图像通道。对于图片显示组件，其一般取第 3 个通道的 QImage 数据，使用 QLabel 的 setPixmap（）方法显示图片。

2.3 图像处理组件的实现

Python 脚本程序的图像处理组件依赖 OpenCV、Pillow 等开源视觉库实现，本文提出的视觉实验程序开

发方法理论上支持所有的 Python 开源视觉库。本文以对 OpenCV 库函数的封装为例，介绍图像处理组件的实现方法。

图像处理组件继承 Component 类实现，如图 7 所示，其命名参照自身功能。在 Python 脚本程序中，图像处理组件注册时会被赋予一个变量名用于识别，变量名存储在 object _ name 中，其父对象为 Python 脚本中的 Scene 对象，Scene 继承自 DR _ Vision。图像处理组件具备 3 个用于存储图像数据的成员变量，分别是 image _ gray、image _ bgr 和 image _ qimage；同时输入接口 setInputData（）和输出接口 setOutputData（）均具备 3 个图像通道，与各类输入组件和图像显示组件保持一致；addCallback（）方法添加其它组件的输入接口地址，即 setInputData（）方法的地址，用于建立组件之间的数据流关系；imageProcess（）方法为组件对图像数据的处理函数，具体实现需要根据组件的功能，在继承 Component 类时重写 code _ str；setCodeStr（）方法提供组件核心代码修改的接口；initQtAdsWidget（）方法用于在 UI 框架上添加组件的交互功能，并将组件的图标（icon）载入 UI 框架。

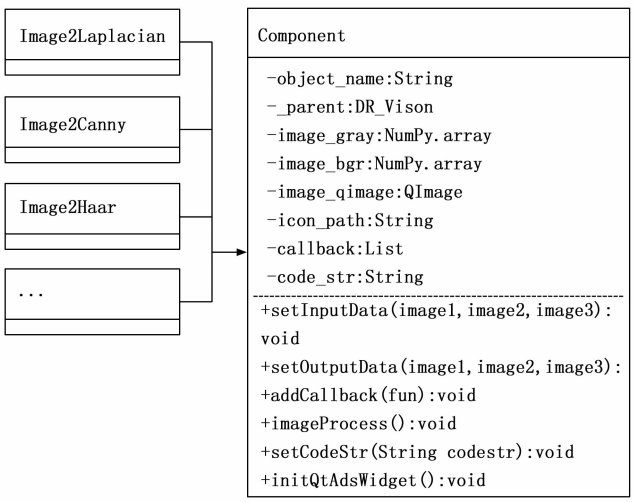

图 7　图像处理组件的 UML 类图

图像处理组件的工作流程如图 8 所示。除去数据的输入和处理后数据的输出外，组件的核心功能体现在 imageProcess（）方法，其执行对图像数据进行处理的 Python 脚本，而被执行的 Python 脚本通过 setCodeStr（）方法管理，并存储在组件的 code _ str 中。图像处理组件的数据流模型如图 9 所示，在 imageProcess（）方法中，除去数据深拷贝与浅拷贝，格式转换等内容外，通过 exec（）方法执行 code _ str 中的 Python 语句，exec（）是 Python 内置的一个函数。通过组件的 3 个图像变量 image _ gray、image _ bgr 和 image _ qimage，exec（）函数能够获取输入数据，调用 OpenCV、Pillow 等开源视觉算法库进行处理，生出新的图像数据并输出。

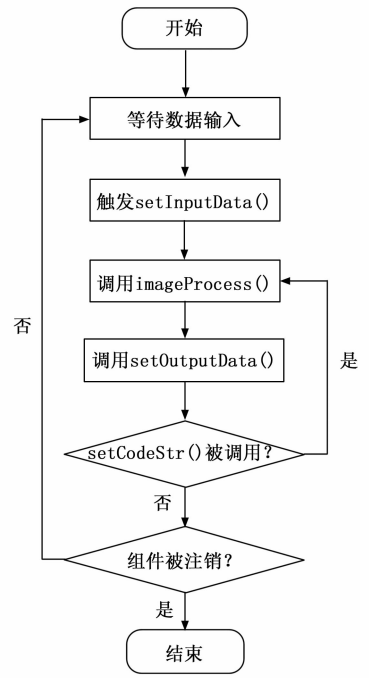

图 8　图像处理组件主要工作流程图

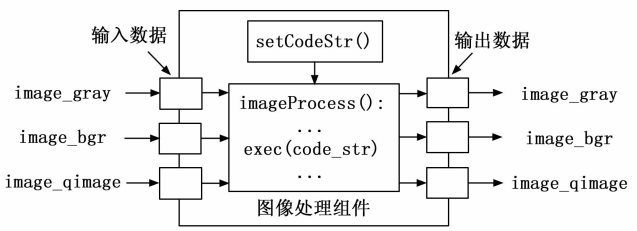

图 9　图像处理组件的数据流模型

通过 exec（）方法动态执行 code _ str 中的 Python 语句，使得在视觉实验程序中，能够通过调节 Python 语句参数或者修改 Python 语句对组件的功能进行调整，甚至可以通过替换 Python 语句，完全改变组件的功能，具备一定的自由度，但 code _ str 中的 Python 语句仍然受到限制。由于所有的图像处理组件均声明在组件库中，code _ str 中的 Python 语句必须在组件库已导入的功能包中被定义，否则 exec（）函数无法执行，在最终生成的 Python 程序脚本中导入其它功能包对组件无效；同时，也无法执行用户自定义的函数，因其未在组件库中声明；这种局限限制了图像处理组件功能的扩展。

针对上述情况，本文提出一种自定义图像处理组件的实现方式，其同样继承 Component 基类，但对 imageProcess（）方法进行了重写。由于组件在 Python 脚本的 Scene 对象中声明，且组件是 Scene 的子成员，那么在 Scene 中声明组件的功能函数，在执行组件的 im-

ageProcess（）方法时跳转到父对象 Scene 中的该函数，再将处理后的数据返回。这样便可以解决上述问题，同时这种方式在 Python 脚本的拼接上易于实现。

3 Python 脚本的结构、装配和动态修改

3.1 Python 脚本的结构和装配

Python 脚本合理的分区设计能够提高代码的模块化水平和可维护性[21]，使得代码更易于理解、调试和扩展。每个区域的功能独立，可根据需要添加、修改或删除，而不影响其他部分，方便新功能的引入。本文将 Python 脚本分为 8 个区域，各个区域的功能独立，且每个区域内部 Python 语句的顺序不影响功能。图形化的脚本生成程序通过响应用户的 UI 操作，在相应的区域添加语句，导出脚本时将 8 个区域按照顺序装配合并，即可生成视觉实验程序的框架代码。Python 脚本结构如图 10 所示，8 个区域依次如下。

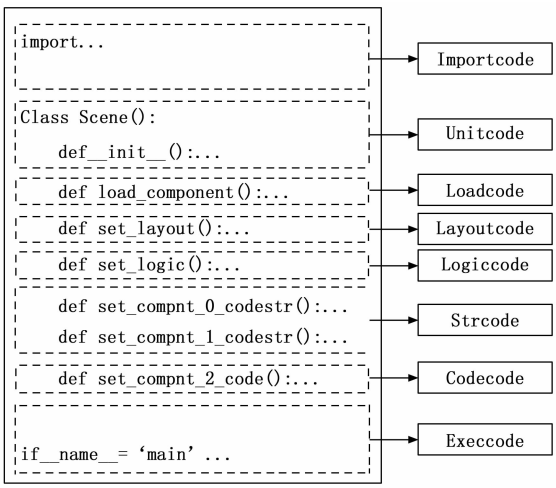

图 10 Python 脚本结构

1）Importcode：功能包导入区域；包括但不限于 DRVision 包，视觉实验组件库包，操作系统包等。生成时只会引入必要的功能包，后期扩展所需功能包可以手动引入；

2）Unitcode：Scene 类及其成员声明区域；Scene 继承自 DRVision，在 Scene 中声明此 Python 脚本中用到的组件，例如“self. camera _ 0 = None”、“self. faceDetector _ 0 = None”等，并添加后续调用的函数，例如“self. load _ component（）”、“self. set _ logic（）”等，需要注意语句前端的缩进；

3）Loadcode：组件实例化区域；UI 框架加载注册组件，例如“self. camera _ 0 = DR _ Camera（）”、“self. faceDetector _ 0 = Image2FaceDetector（self，" faceDetector _ 0")”等；

4）Layoutcode：UI 界面布局区域；设置图像输入/输出组件的布局，例如“self. layout. addWidget（self. camera _ 0，0，0)”；

5）Logiccode：数据流关系设置区域；设置组件之间的数据流关系，例如“self. camera _ 0. addCallback（self. FaceDetector _ 0. setInputData)”；

6）Strcode：图像处理组件的核心 Python 语句的函数设置区域；根据函数名称对应组件，每个函数内部通过对相关组件 code _ str 的赋值实现；

7）Codecode：自定义图像处理组件的处理函数设置区域；根据函数名称与自定义组件对应，每个函数具备 3 个形参，对应数据流的 3 个图像通道，函数末尾将 3 个图像数据返回；

8）Execcode：Python 脚本的启动区域；通过 PySide6 的 QApplication 类启动。

上述的 Python 脚本的分区设计，让脚本的各个区域语句功能独立，且每个区域内部语句的顺序不影响功能。用户仅需要关心 Strcode 和 Codecode 区域，即图像处理组件和自定义组件的功能实现，而无需考虑其他区域的代码。

3.2 Python 脚本的动态修改

在开发带有用户界面的 Python 脚本程序时，交互式开发的程度较低，通常需要重新运行 Python 脚本才能看到修改效果，在 MATLAB 的 App Designer 也是如此，本文将其称为静态修改。然而对于视觉实验程序的开发，一般涉及到频繁的 OpenCV 函数或者其他视觉功能包的函数参数调节，这种静态修改的方式效率低下。本文提出的视觉实验程序开发采用数据流驱动的架构，同时图像处理组件的核心 Python 语句由 exec（）函数动态执行，因此所开发的 Python 脚本程序支持动态修改，能够在程序运行时在 UI 界面上修改组件的核心语句，并立即得到程序响应。同时，程序运行时对组件核心语句的修改，也会反映到程序本身的 Python 脚本上，Python 脚本会将修改保存。其实现方式如图 11 所示。

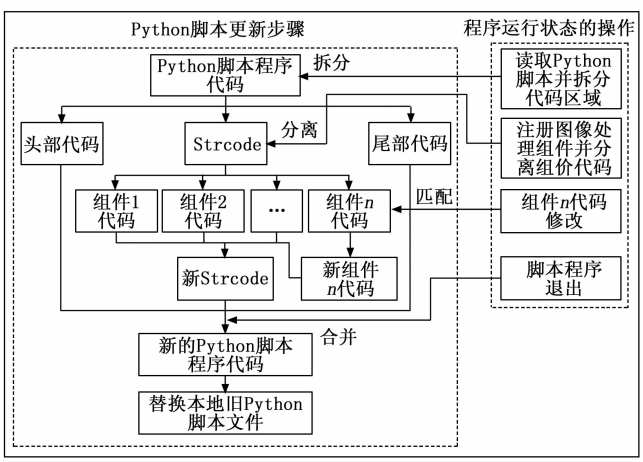

图 11 Python 脚本动态修改实现方法

Python 脚本需要动态修改的内容是图像处理组件的核心 Python 语句，即脚本中的 Strcode 区域，因此脚本程序在启动时读取自身的 Python 代码，并通过字符串分割的方法将 Strcode 区域拆分出来。根据已注册的图像处理组件的名称，在 Strcode 中通过字符串匹配的方法分离不同组件的核心 Python 语句设置函数。当在程序 UI 界面上修改某一组件的 Python 语句，匹配该组件的核心 Python 语句设置函数并将其修改。当脚本程序退出时，合并所有代码部分，组成新的 Python 脚本代码并覆盖本地旧代码，即将运行状态对脚本的修改记录到了 Python 脚本本身。

支持 Python 脚本的动态修改能够提高视觉实验程序开发的效率，同时单个视觉实验场景的内容全部存储在对应的单个 Python 脚本中，能够给用户带来便利。

4　Python 视觉实验程序开发实例

4.1　图形化的脚本生成程序介绍

脚本生成程序主界面如图 12 所示。其可用于快速生成 Python 脚本，提供 40 余个视觉实验组件以供选择，提供 Input/Output Table 窗口设置组件之间的数据流关系，完成后导出 Python 脚本即可；同时也可用于进行机器视觉实验，程序提供 30 余个视觉实验样例，即搭建好的 Python 视觉实验程序脚本，双击即可运行，且 Python 脚本代码可见，供用户参考。

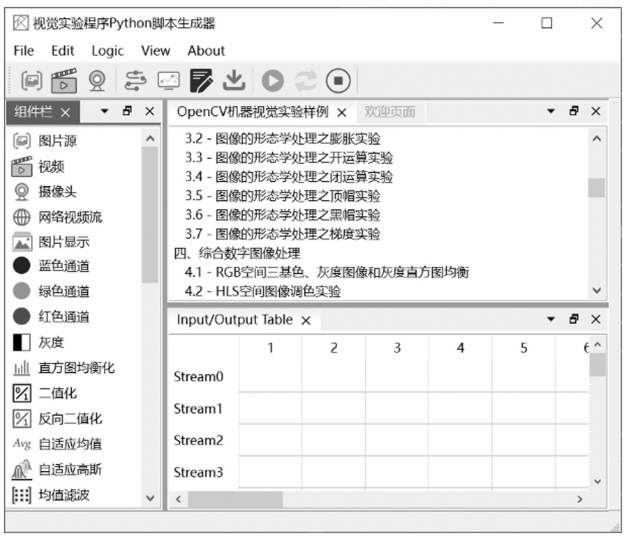

图 12　图形化的脚本生成程序主界面

图 13 是程序提供的部分机器视觉实验样例，从左至右、从上到下依次是米粒图像的边缘提取算法对比实验、图像形态学操作实验、Haar 分类器人脸检测实验和霍夫圆检测实验。

4.2　手势检测的 Python 机器视觉实验程序开发

对于手势检测视觉实验，采用脚本生成程序开发步骤如下：

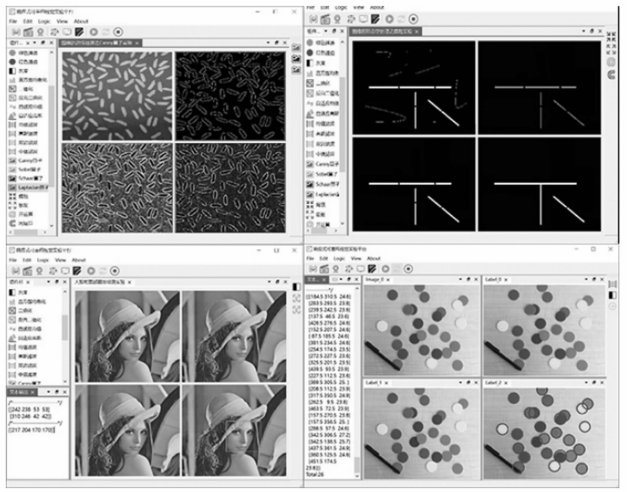

图 13　部分机器视觉实验样例

1）不妨采用本地图片作为数据源，在脚本生成程序中选择图片源组件，即 DR _ Label；

2）选择一个图片显示组件，即 DR _ Label _ Display；

3）脚本生成程序中没有手势检测组件，因此需要采用一个自定义组件实现，即 Image2DIY；

4）设置数据流方向为图片源至自定义组件，再至图片显示组件；

5）导出 Python 脚本；

6）补全 Python 脚本中自定义组件的功能函数，即手势检测的具体实现。

最终的 Python 脚本如图 14 所示，其中虚线框中的代码为自定义组件的功能代码，为手动输入，另外导入了其所需的功能包 cv2 和 mediapipe，其余代码均为程序自动生成。从图 14 中可以清晰看到 Python 脚本的结构，以及函数的执行顺序。运行脚本，在生成的程序中指定本地图片输入，最终运行结果如图 15 所示。从中可见，检测到了手掌和手指，实现了手势检测功能。该实例证明了本文的开发方法能够快速生成机器视觉实验程序的框架代码，简化开发流程，并具备充分的扩展性，能够方便地引入新功能。在本文开发方法中，用户能够专注于机器视觉算法的实现和优化，而无需费心于程序开发中的诸多事项。

5　结束语

本文提出一种统一数据流接口的双端口组件概念，将 Python 上优秀的 GUI 框架（例如 Pyside6）中的控件或工具、以及 Python 上前沿且丰富的开源机器视觉计算库（例如 OpenCV、Pillow、YOLO 等）中的算法或功能进行组件化封装，使用户可以通过图形化的脚本生成程序选择组件、GUI 布局和数据流绑定，且通过组件的规范化、体系化命名以及科学合理的脚本分区使得

```
import cv2
import mediapipe as mp

from DRVision import *
from Controls.DR_Label import*
from Controls.DR_ImageProcessing import*

class Scene(DRVison):
  def__init__(self,parent=None):
      super().__init__(parent)
      self.get_python_file(sys.argv[0])
      self.image_0=None
      self.label_0=None
      self.load_component()
      self.set_layout()
      self.set_logic()

  def load_component(self):
    self.image_0=DR_Label()
    self.label_0=DR_Label_Display()
    self.DIY_0=Image2DIY(self, "DIY_0")
    self.DIY_0.setFunpath("set_DIY_0_code")

  def set_layout(self):
      self.layout.addWidget(self.image_0,0,0)
      self.layout.addWidget(self.label_0,0,1)

  def set_logic(self):
      self.image_0.addCallback(self.DIY_0.setInputData)
      self.DIY_0.addCallback(self.label_0,setInputData)

  def set_DIY_0_code(self,image1,image2,image3):
      # write your code here:
      mpHands=mp.solutions.hands
      hands=mpHands.Hands()
      mpDraw=mp.solutions.drawing_utils
      results=hands.process(image2)
      if results.multi_hand_landmarks:
          for handLms in results.multi_hand_landmarks:
              for id,lm in enumerate(handLms.landmark):
                  h,w,c=image2.shape
                  cx,cy=int(lm.x*w),int(lm.y*h)
                  cv2.circle(image2,(cx,cy),5,(255,0,255),cv2.FILLED)
            mpDraw.draw_landmarks(image2,handLms,mpHands.HAND_
            CONNECTIONS)
      return image1,image2,image3

if__name__='__main__':
    app=QApplication(sys.argv)
    myscene=Scene()
    myscene.show()
    app.exec_()
```

图 14 手势检测视觉实验的 Python 脚本

图 15 手势检测 Python 脚本运行结果

Python 脚本的自动化装配得以实现，用户能够导出可一键运行的 Python 脚本，同时也能够修改、扩展 Python 脚本。这种程序开发架构能够充分发挥 Python 语言的优势，简化视觉实验程序开发流程，兼具强大功能性、强扩展性和跨平台性。

本文提出的视觉实验程序开发方法，其开发过程类似于 MATLAB 中的 App designer，都是通过 GUI 操作生成框架代码。但是与后者相比有两点显著优势，一是在组件选择并 GUI 布局的同时，能进行组件之间的数据流绑定，而后者需要通过回调函数的方式在代码中实现；二是在开放性和可扩展性方面都优于后者，在 Python 中能引入最新的视觉运算库，实现前沿的图像处理功能，而后者是商业闭源软件，其功能函数的扩充有一定的迟滞，同时 Python 脚本可以方便引入模块，例如引入通信模块将图像处理结果发送至下位机等，实现丰富的功能。

通过本文的开发方法，能够简化视觉实验程序的开发过程，让用户专注于视觉算法的实现和调整，在机器视觉类课程实验设计方面具备很大的应用价值。同时，随着深度学习技术的进步，新的视觉算法和模型不断涌现，本文提出的开发架构能够很好地适应这种变化，用户可以轻松地将最新的算法集成到他们的程序中，而不需要重新设计整个程序。此外，随着支持 Python 语言的嵌入式设备的进一步发展，这种开发方法也能应用到其中，更好地与硬件进行交互。

参考文献：

[1] CONGP. Design of Machine Vision Teaching Experiment System Based on HALCON [C] // 2020 International Conference on Data Processing Techniques and Applications for Cyber-Physical Systems. Singapore: Springer, 2021: 1245 - 1250.

[2] 祁峥东，阎 妍，杨正颖，等. 基于 MATLAB GUI 的图像处理和机器视觉辅助教学平台设计与实现 [J]. 南京晓庄学院学报，2023，39 (6)：57 - 65.

[3] 刘志毅，杨桂华，唐卫卫. Research on Multi-type Workpiece Measurement System Based on Machine Vision [J]. Machine Tool & Hydraulics，50 (4)：6 - 12.

[4] MOLER C，LITTLE J. A history of MATLAB [J]. Proceedings of the ACM on Programming Languages，2020，4 (HOPL)：1 - 67.

[5] JI H W，YOO S S，LEE B J，et al. Measurement of Wastewater Discharge in Sewer Pipes Using Image Analysis [J]. Water，2020，12 (6)：1771.

[6] 朱 云，凌志刚，张雨强. 机器视觉技术研究进展及展望 [J]. 图学学报，2020，41 (6)：871 - 890.

[7] BATZOLIS E，VROCHIDOU E，PAPAKOSTAS G A. Machine Learning in Embedded Systems：Limitations，Solutions and Future Challenges [C] //2023 IEEE 13th Annual Computing and Communication Workshop and Conference (CCWC). 2023：0345 - 0350.

[8] JONAS OPPENLAENDER. The Creativity of Text-to-Image Generation [C] //Proceedings of the 25th International Academic Mindtrek Conference. 2022：192 - 202.

(下转第 254 页)