同济大学
TONGJI UNIVERSITY

同济大学《图像处理与机器视觉》课程实验报告

实验课程环境搭建

任课老师：范睿、蒋磊

班级：10059903

2024 年 3 月

目 录

第一章　实验内容简述

1.1　问题描述

本次实验的主要目的是为了搭建本学期课程《图像处理与机器视觉》的实验部分的基础环境，包括在 Vmware 虚拟机上搭建 linux 系统以及搭建 C++ 环境下的 OpenCV 环境，以及 eigen、sophus、geographiclib 等包。

其中 Vmware 的版本使用的是 Vmware17 Player，linux 操作系统选择的是 ubuntu22.04 的版本，OpenCV 选取的是比较新的 4.9 版本。

1.2　实验环境

下面是本次实验的主机环境：

项目	内容
CPU	Intel(R) Core(TM)i5-1035G1 CPU @ 1.00GHz
内存	16GB
OS	Windows10
内核	4

虚拟机的基本配置如下：

项目	内容
内存	6GB
OS	Ubuntu22.04.4
内核	2
磁盘空间	80GB

第二章　环境搭建

2.1　在虚拟机上安装 linux 操作系统（ubuntu）

首先是要选择虚拟机，这里选择了 Vmware 17 Player，然后创建虚拟机后导入提前下载好的 Ubuntu22.04 的镜像文件，然后进行相关的配置操作，这里由于电脑本机只有 4 个核，所以给虚拟机分配 2 个核，除此之外，内存由于本机有 16G 分配给虚拟机 6G，而且经过验证，如果分配给虚拟机 8G 后如果主机和虚拟机启动的进程都比较多，会使

得主机出现内存不够死机的问题，改成 6G 后情况有所好转；而本机磁盘空间较大所以分配给虚拟机磁盘空间分配 80GB。

2.2　安装基本的一些工具和包

安装虚拟机以及 Ubuntu 成功后，需要安装用到的一些工具，采用的是 sudo apt install 的方式进行安装，安装了包括 git、g++、cmake、gcc、vim、wget、vim、make 等包，其中 g++ 是 C++ 编译器,gcc 是 C 编译器，git 是版本控制工具，wget 可以用来现在资源，vim 是一个高级的文本查看及编辑器，cmake 和 make 是用来配置编译的工具。

除此之外，还安装了 VScode, 这里没有采用使用安装包安装的方式，而是在 Linux 的软件商店中直接搜索 code 进行了安装。

2.3　安装 opencv、eigen 等相关的包

首先是安装 OpenCV 的包，在安装 OpenCV 前需要安装相关的一些依赖库如下：libgtk2.0-dev、pkg-config、libavcodec-dev、libavformat-dev、libswscale-dev、build-essential、libcanberra-gtk-module 等

依赖库完成后就可以安装 OpenCV 了，OpenCV 的包从学校的 gitlab 下获取，这里安装的是比较新的 4.9 版本。这里需要注意的是即便不安装这些依赖库，也能正常编译安装，但是可能会使得装好之后某些函数不能使用，而如果在装后之后再安装这些依赖包可能会由于版本冲突的问题导致依然发生失败，同样在第二节实验课提到了，如果不安装 libgtk2.0-dev 和 libcanberra-gtk-module 库的话会导致 cv2.imshow 函数无法正常使用，没办法打开图片。

安装完成后在资源包内创建 build 文件夹，然后进入 build 文件夹后采用 cmake .. 的方式来配置编译的各项属性等，然后采用 make -j4 的方式进行编译，这里 j4 的意思是采用 4 个线程编译代码，最后采用 sudo make install 来安装以及编译好的程序，相关的代码如下：

```
cd opencv
mkdir -p build
cd build
cmake ..
make -j4
sudo make install
```

上述操作完成后 OpenCV 将安装到 /usr/local 目录，相关的文件都会被复制到该目录下，如下：

```
/usr/local/bin - executable files
/usr/local/lib - libraries (.so)
/usr/local/cmake/opencv4 - cmake package
/usr/local/include/opencv4 - headers
/usr/local/share/opencv4 - other files (e.g. trained cascades in XML
   format)
```

接下来需要在 Linux 系统中添加 OpenCV 的路径，具体的操作方式如下：

```
sudo gedit /etc/ld.so.conf
/usr/loacal/lib #在打开的文件中添加
sudo ldconfig  # 保存关闭后运行
```

添加完路径后配置环境变量

```
sudo gedit /etc/bash.bashrc # 打开该文件
# 在末端添加下面的代码
PKG_CONFIG_PATH=$PKG_CONFIG_PATH:/usr/local/lib/pkgconfig
export PKG_CONFIG_PATH
# 保存退出后，在终端输入：
source /etc/bash.bashrc
# 最后用pkg-config opencv4 --modversion命令来查看是否配置成功
```

结果如下：

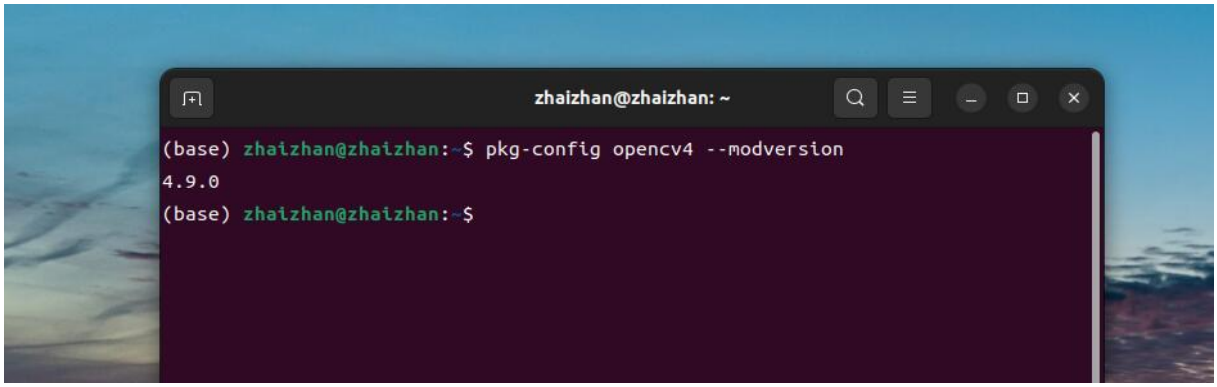

图 2.1: 检验 opencv 是否配置成功

安装完 OpenCV 后，eigen，sophus，geographiclib 等包的安装方式基本同 OpenCV。

2.4　在 VScode 中搭建基本 C/C++ 基本环境

VScode 是一个轻量型文本编辑器，但是支持非常多的插件，也支持配置 C/C++ 的环境，配置步骤如下：

1. 打开 VsCode 搜索安装 C/C++ 和 Code Runner 插件，安装好着两个插件后，就可以直接运行 C++ 的代码了，但是还不能调试，运行一个简单的 helloworld 程序代码如下所示：

```
sudo gedit /etc/bash.bashrc # 打开该文件
#include <stdio.h>
#include <iostream>
int main()
{
printf("hello␣world!\n");
std::cout<<"真不错"<<std::endl;
return 0;
}
```

运行结果如下图所示：

图 2.2: helloworld **运行代码**

2. 想要正常运行 C++ 程序，还需要使得其可以正常进行调试，配置调试的方法是选定一个文件夹作为放置 C++ 程序的文件夹，并在里面创建.vscode 文件夹，创建三个 json 文件，文件名分别为 launch.json，tasks.json 和 c_cpp_properties.json。这几个文件是和调试编译等相关的配置文件，这里列出 helloworld 程序所需的 json 文件如下（OpenCV 的程序需要在配置文件中添加依赖包等信息）：

   launch.json:

```
    {
"version": "0.2.0",
"configurations": [
    {
        "name": "C/C++",
        "type": "cppdbg",
        "request": "launch",
        "program": "${fileDirname}/${fileBasenameNoExtension}",
        "args": [],
        "stopAtEntry": false,
        "cwd": "${workspaceFolder}",
        "environment": [],
        "externalConsole": false,
        "MIMode": "gdb",
        "preLaunchTask": "compile",
        "setupCommands": [
            {
                "description": "Enable␣pretty-printing␣for␣gdb",
                "text": "-enable-pretty-printing",
                "ignoreFailures": true
            }
        ]
    }
]
```

```
}
```

tasks.json:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "label": "compile",
            "command": "g++",
            "args": [
                "-g",
                "${file}",
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}"
            ],
            "problemMatcher": {
                "owner": "cpp",
                "fileLocation": [
                    "relative",
                    "${workspaceRoot}"
                ],
                "pattern": {
                    "regexp": "^(.*):(\\d+):(\\d+):\\s+(warning|
                       error):\\s+(.*)$",
                    "file": 1,
                    "line": 2,
                    "column": 3,
                    "severity": 4,
                    "message": 5
                }
            },
            "group": "build"
        },
        {
            "type": "cppbuild",
            "label": "C/C++:␣g++␣生成活动文件",
            "command": "/usr/bin/g++",
            "args": [
                "-fdiagnostics-color=always",
                "-g",
                "${file}",
                "-o",
                "${fileDirname}/${fileBasenameNoExtension}"
            ],
            "options": {
                "cwd": "${fileDirname}"
            },
            "problemMatcher": [
                "$gcc"
            ],
            "group": {
                "kind": "build",
                "isDefault": true
            },
```

```
            "detail": "调试器生成的任务。"
        }
    ]
}
```

c_cpp_properties.json:

```
{
    "configurations": [
        {
            "name": "Linux",
            "includePath": [
                "${workspaceFolder}/**"
            ],
            "defines": [],
            "compilerPath": "/usr/bin/gcc",
            "cStandard": "c17",
            "cppStandard": "gnu++17",
            "intelliSenseMode": "linux-gcc-x64"
        }
    ],
    "version": 4
}
```

调试上面的“helloworld”代码的截图如下所示

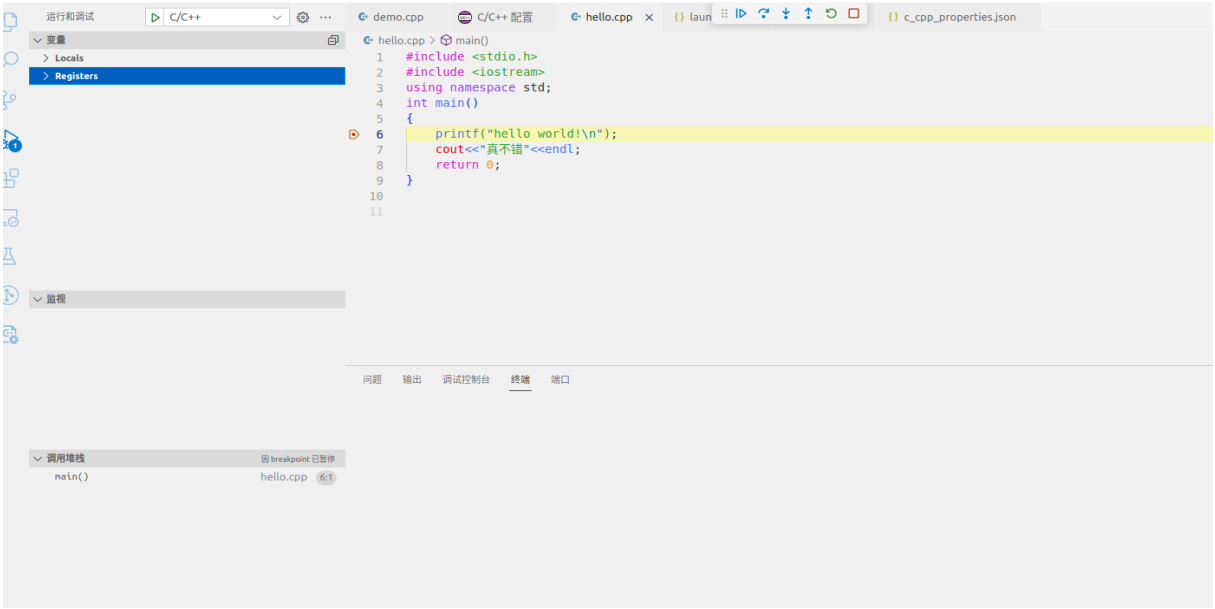

图 2.3: helloworld 调试代码

2.5　验证 OpenCV 安装并进行简单实践

接下来是验证 OpenCV 是否安装成功：写了一个简单的用于打开图片和保存为别的格式的图片的代码：

```
#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;// 省去函数前面加cv::的必要性
int main()
{
    char imageName[] = "test.jpg";
    Mat M=imread(imageName,IMREAD_COLOR);    // 读入图片

    if(M.empty())      // 判断文件是否正常打开
    {
         fprintf(stderr, "Can not load image %s\n", imageName);
         waitKey(6000);  // 等待6000 ms后窗口自动关闭
         return -1;
    }
    else
    {
        fprintf(stdout,"heloo");
}
    imshow("image",M);  // 显示图片
    waitKey();
    imwrite("pic.bmp",M); // 存为bmp格式图片
    return 0;
}
```

这里即便在 json 文件中配置好了 opencv 所需要链接的库，但是使用 vscode 运行或者调试时候还是会出现以下的问题：

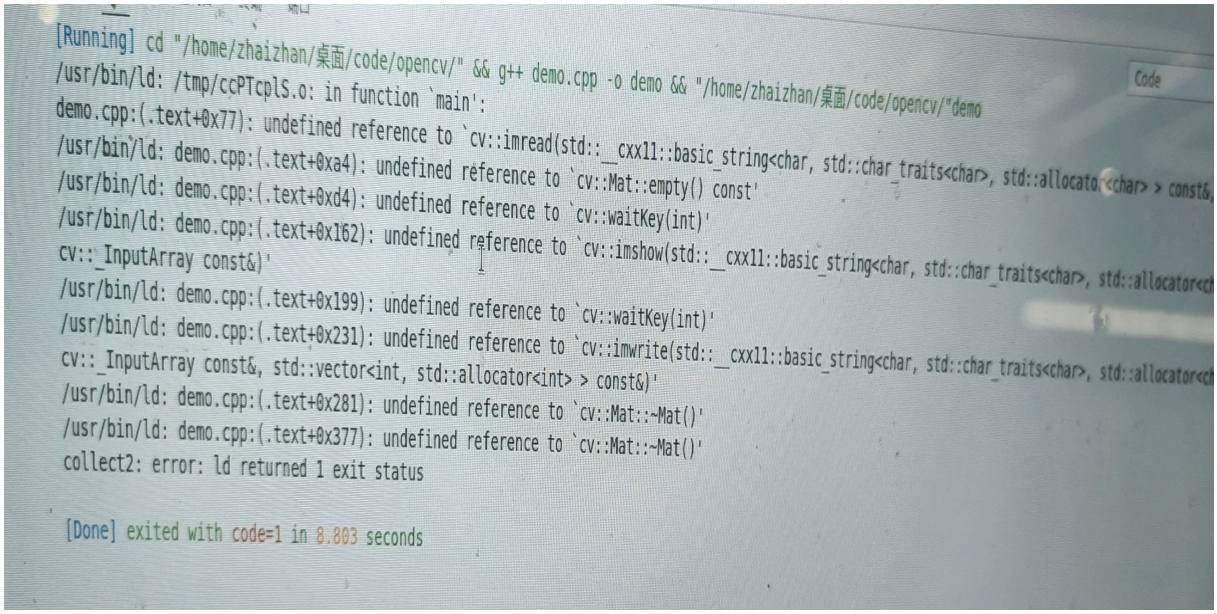

图 2.4: 报错信息

分析过后认为即便是配置好了相关的配置文件但是 vscode 还是没有链接到头文件所以发生了报错，采取的方式是在终端中使用 g++ 进行编译（这里写了脚本进行运行），然后手动将头文件所在的位置链接：

```
g++ test.cpp -o test `pkg-config --cflags --libs opencv4`&&./test
```

运行结果如下：

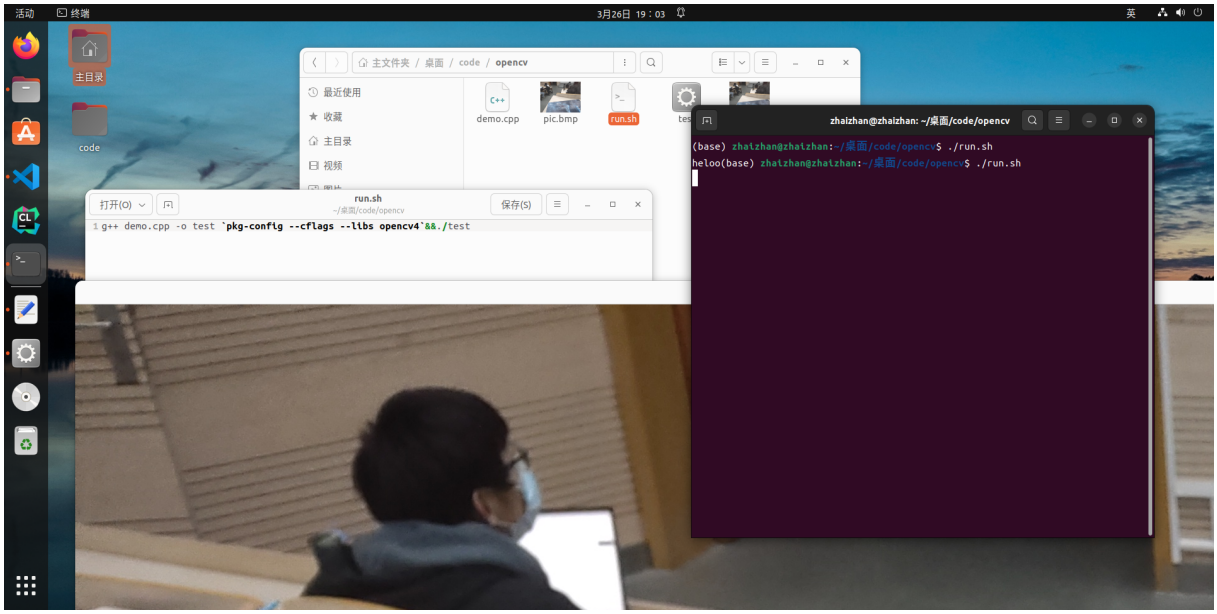

图 2.5: opencv 简单代码运行结果

这里可以看到成功打开了代码中写的 test.jpg(由于没有限定大小导致以原图大小打开所以比较大，只截图了一半)，同时也生产了新的图片 pic.bmp, 证明 opencv 是可以正常运行的。

第三章　问题与解决

在本次实验过程中我遇到了以下的一些问题，我自己的思考与最终解决的方法如下：

1. 遇到的第一个问题是在通过 make -j4 编译几个包的过程中发现编译过程总会卡在进度的某个位置，然后报错弹出，经过分析我认为是由于主机的核数本来就很少，再加上给虚拟机分配的内存也有限，所以在多线程编译的时候很容易造成内存不够，对此我的解决办法是取消 j4 命令，直接用 make 命令进行编译，这样虽然编译过程变长了但是没有出现卡住的问题。

2. 第二个问题是最开始装好了 OpenCV 发现有问题卸载重装的时候，发现会和已经下载好的 eigen 库发生版本冲突，我当时的解决办法是先把 eigen 卸载掉再进行重新安装，然后发现还是会提示冲突，这时候我经过查阅资料后采取的方法是把 linux 系统上所有带 eigen 名字的文件夹也删掉（包括从 tongji 的 gitlab 拉取的源文件）之后才编译通过，后续通过第二节实验课学习到可以通过修改 OpenCV 的 CMakeLists.txt 来改变编译选项，只要在文件中加入

```
include_directories("/usr/include/eigen3")
set(CMAKE_CXX_STANDARD 17)
```

即可。

3. 同样是在重新安装 opencv 过程中遇到的问题，在编译过程中发现会有以下报错：

```
untime library [libssl.so.1.1] in /usr/lib/x86_64-linux-gnu may be
   hidden by files in:/home/anaconda3/lib
runtime library [libcrypto.so.1.1] in /usr/lib/x86_64-linux-gnu may
   be hidden by files in:/home/anaconda3/lib
```

查阅资料后发现是由于我之前装了 anaconda，导致在编译 opencv 的时 Anaconda 将系统路径屏蔽掉了。将指令变为

```
sudo cmake -D CMAKE_CXX_COMPILER:FILEPATH=/usr/bin/g++ -D
   CMAKE_C_COMPILER:FILEPATH=/usr/bin/gcc -D CMAKE_BUILD_TYPE=
   RELEASE -D  CMAKE_INSTALL_PREFIX=/usr/local -D CUDA_GENERATION=
   Auto OPENCV_EXTRA_MODULES_PATH=../opencv/opencv_contrib/modules/
    ..
```

这样可以防止出现报错。

4. 在第一次安装好 OpenCV 后，在根据网上的教程配置 pkg-config 的环境时候发现在/usr/local/lib/pkgconfig/路径下并没有找到 Opencv4.pc 文件，查阅资料后发现是因为 OpenCV4 在编译的时候默认不生成该 pc 文件，采用的解决办法是在编译的时候，使用 cmake 配置 OpenCV 的时候进行配置，代码如下：

```
cmake -D CMAKE_BUILD_TYPE=Release -D
OPENCV_GENERATE_PKGCONFIG=YES ..
```

而不是默认的 cmake .., 通过这样的方式可以生存 opencv4.pc 文件，方便进行 pkg-config 的配置。

5. 还遇到的问题是配置好了环境，但是在 vscode 里编辑 OpenCV 项目的时候仍然提醒找不到包含 opencv 的头文件，这里经过分析后认为可能是通常是因为默认安装的头文件路径与搜索的头文件路径不匹配，opencv 默认安装的头文件路径是/usr/local/include，而我们搜索的默认路径在/usr/include，所以导致没有找到头文件，这里给出的解决方法是进行软链接：

```
sudo ln -s /usr/local/include/opencv4/opencv2 /usr/include/
```

通过这样的方式将两个位置链接起来，编译器就可以找到了。

6. 由于电脑物理主机没电导致虚拟机也非正常关机，然后再次启动虚拟机时候出现了以下的问题：

图 3.1: 报错信息

查阅资料后了解到是虚拟机为了防止有多虚拟机共用一个虚拟磁盘(就是后缀为.vmdk那个文件）造成数据的丢失和性能的削弱，每次启动虚拟机时会给每个虚拟磁盘加一个磁盘锁（也就是后缀为.lck 的那个文件夹）对虚拟磁盘文件进行锁定保护在关掉虚拟机时又会自动删除那个磁盘锁文件。当虚拟机非正常关闭，就会出现一个文件夹带有缀.lck，不会自动删除，才会引起这样报错。删除掉后再次启动发现可以正常开机。