CVRA 2019编程规范1.3

孔德优 胡斌雁 编写
张霁寒 施殊 修改整理

1.源文件

1.1命名规范

- 头文件的文件扩展名为.h,源码文件的文件扩展名为.c或.cpp; 例:Project.h、Project.c、Project.cpp。
- 模块化代码,模块的头文件和源码文件的文件名主体部分必须相同; 例:实现AngleSlover算法的模块,头文件名为AngleSolver.h,则源码文件名为AngleSlover.cpp。
- 模块代码的文件名以文件中主体类的名称命名,采用帕斯卡命名法,即每个单词首字母均为大写,每个间可存在数字,但不能存在其他字符,命名以大写字母开头;

例:实现标签检测算法的模块,主体类为CTagDet,则文件命名为CTagDet.h与CTagDet.cpp。 * 必要时,文件名称可以以类似“RM”这样的项目名称前缀缩写开头,避免与其他项目文件名称冲突。

1.2代码块

- 避免层次过多的嵌套,冗长的函数段,单行语句嵌套一般不要超过3层,如无必要,多层循环语句不要超过4层,可以进行优化、拆分成函数或者划分段落并添加相应注释来解决。
- 相同逻辑、功能的代码,需要写在相近的地方。

1.3头文件

- 各个模块的头文件尽量少地包含其他头文件。例如,Pose模块包含 AngleSolver.cpp和AngleSolver.h,在AngleSolver.h中尽量少地包含其他头文件,应只包含声明需要的头文件,在AngleSolver.cpp中需要用到的其它头文件(如string.h, malloc.h等)应包含在Buffer.cpp中。该项规范能使接口头文件包含的其他头文件尽量地少,有助于更直观地查看和使用模块代码。
- 为了避免重复包含头文件,C++的头文件务必增添#pragma once语句,例如,在C++的头文件ArmorDector.h中,需要在开头有#pragma once语句。

2.代码通用命名规范

- 所有表示参数的变量（如：参数类成员、语句块中的参数变量等）使用下划线命名法，命名应以尽可能描述参数含义为目的。 float src_threshold; cv::Size work_size; bool en_tracking; int num_img_len;
- const类型的参数变量、enum、define使用全大写字母下划线命名法，命名应尽可能描述参数含义。 const float ABS_SIN_TOLERANCE = 0.2; enum buffFlag { BUFF_NO = 0, BUFF_PART = 1, BUFF_FULL = 2, }; define SHOW_RESULTS
- 类型名、类和结构体名、枚举体名使用大驼峰（帕斯卡）命名法 struct BuffParam class BuffDetector enum BuffFlag
- 其他变量和函数使用小驼峰命名法，变量含义的根本属性（Img，Rect，Mat，Roi等）置于变量末尾，对变量含义的修饰（avg，old，min，max等）则作为前缀 void loadImg(…) Rect workImgBound = …
- 模块类的成员变量加下划线前缀（参数类为方便读取不必遵循此规则），以便在算法实现中区分 int _flag
- 调试中用到而实战中不用到的函数、变量和语句块应包括在预编译块（#ifdef，#endif）中，以免实战中被误用 -#ifdef SHOW_RESULTS template void coutNumbers(const std::array<T, 9>numbers) -#endif （注意，井号前的-是为了编辑方便，实际没有）
- 名词后加”s“表示数组 std::array<cv::Point2f, 4> buffCorners;

3.注释

原则

- 头文件中需要给外部使用的类定义、类成员、接口函数或方法等必须添加详细注释,c或cpp文件中实现处可以省略,注释格式将在下文软件编程规范的注释一节中给出;
- 适当添加必要的注释,不可以通篇代码没有注释,也不建议写各种无意义的注释内容;
- 注释应尽量保证严谨,语言不要过分口语化和戏剧化;
- 逻辑层次较深、循环层次较多、if-else嵌套较多的地方,以及有需要注意、容易出错的地方,一定要添加注释;

- 未完成的地方或者框架代码,需要添加 TODO 标记,并说明 TODO 的内容;
- 待完善或待修复的一些不影响主要功能的小地方,需要添加FIXME标记,并说明内容;
- 注释中不应该指名道姓,例如不应该出现“XXX说这里有个问题,待会让他修复”这样的内容。
- 根据需要使用//行注释或者/ …… /段注释标记。

4.排版

- 统一缩进风格,缩进为一个TAB键,TAB设定为4个空格宽度;
- 注释与代码间有必要时需要用空行分隔开;
- 相对独立的代码块之间、变量声明等之后必须加空行;
- if、else、while等语句体内即使只有一行代码也必须添加花括号括住;
- 在有==逻辑处理的地方,应将常量前置,避免错误书写成=导致直接赋值产生出错;
- 花括号的首尾统一占一行,首括号切记不可以写在上一行的末尾,示例代码如下:

```
 int main()
{ // 此处首括号独占一行,该括号不可以放上一行的末尾
// ...
return 0;
} // 此处尾括号独占一行
```

- switch和case必须拥有相同的缩进,case语句下需要用花括号括住语句,末尾如果有break;语句则放置在花括号的后一行上,示例代码如下:

```
// ...

switch (iJudge)
{
case 0:
{
// ...
}
break;

case 1:
{
// ...
}
break;

//...
}

// ...
```

5.标识符命名与缩写

标识符的命名要清晰、明了,有明确含义,同时使用完整的单词或大家基本可以理解的缩写尽量不要使用非公认或有歧义的缩写。命名中若使用特殊约定或缩写,则要有注释说明。 常用的缩写标识有:

- parameter ——— param
- buffer ———————— buf、buff
- length ———————— len
- current ———————- crnt
- destination ——- dest
- message ———————— msg
- image ————————— img
- position ———————— pos
- point ————————- pnt
- rectangle ———- rect
- index ————————- idx
- direction ———————— dir
- address ———————— addr

- receive ————- recv
- to ———————— 2 ……

6. 代码可读性

- 避免使用不易理解的数字,应采用有意义的标识名,涉及物理状态或者含有物理意义的常量,不应直接使用数字,用有意义的枚举、宏或常量类型来代替;
- 源程序中关系较为紧密的代码应尽可能相邻;
- 不要使用难以理解的技巧性很高嵌套很深的语句,除非很有必要;
- 对于难以理解的运算或条件判断,可通过封装以增强可读性。

7. 函数和方法

- 一个函数仅完成一项功能,不要杂糅;
- 函数的参数不宜超过6个,当需要传递的参数过多时可以考虑将其包装在结构体中;
- 函数名应力求准确描述函数的功能;
- 注意保证函数所有参数输入的有效性;
- 在同一项目组中,应明确规定对接口函数参数的合法性检查应由函数的调用者负责,还是由接口函数本身负责;
- 尽量减少函数本身或函数间的递归调用,递归算法容易造成资源的大占用和理解上的困难,因此除非在某些算法或功能上的实现简便或有优化作用,否则应减少不必要的递归调用。

8. 模块

- 需要且只需要一个实体的类，应被定义为单体类（Singleton） class BuffDetector:public Singleton { public: friend Singleton;

… }

- 每个模块类用到的参数存储在与之对应的参数类中，该参数类应包含从文件初始化的接口（参数存入xml文件）。模块类可以提供改变其参数的接口。