南工骁鹰视觉组代码编程规范

一、前言

本代码规范参考Google开源项目风格指南

作者：sylybimike

邮箱：sylybimike@gmail.com/sylybimike@163.com

二、头文件

2.1头文件保护

所有头文件都必须使用`#define`来进行保护，防止头文件被多重包含

命名格式：`<PROJECT>_<PATH>_<FILE>_H_`

实例：项目`HITSZ_RoboMaster`中的`HITSZ_RoboMaster/armor/include/armorfinder.h`

```
#ifndef HITSZ_INCLUDE_ARMORFINDER_H_
#define HITSZ_INCLUDE_ARMORFINDER_H_
...
#endif
```

2.2内联函数(inline)

将小于10行的工具函数定义为内联函数，内联函数规模不可超过十行，且应尽量使用lambda表达式等技巧构造函数

请将内联函数放置在头文件中，当你发现工程中频繁用到一些简短函数（如custom的swap）时，应该考虑使用内联函数提高效率

定义:

> 当函数被声明为内联函数之后, 编译器会将其内联展开, 而不是按通常的函数调用机制进行调用.

优点:

> 只要内联的函数体较小, 内联该函数可以令目标代码更加高效. 对于存取函数以及其它函数体比较短, 性能关键的函数, 鼓励使用内联.

缺点:

> 滥用内联将导致程序变得更慢. 内联可能使目标代码量或增或减, 这取决于内联函数的大小. 内联非常短小的存取函数通常会减少代码大小, 但内联一个相当大的函数将戏剧性的增加代码大小. 现代处理器由于更好的利用了指令缓存, 小巧的代码往往执行更快。

2.3 `#include`的路径和顺序

避免使用linux路径标示符号`..`和`.`来指明头文件的相对路径

文件中应该使用如下的顺序进行头文件的包含：

1. 自己书写的头文件，本项目中的其他头文件，使用`“ ”`包含
2. 来自于C/C++的系统库函数
3. 其他外置库

避免重复包含:如某类型`T`在`A.h`中被定义，在`B.h`中包含了`A.h`，而我们想要在`C.h`中使用这一个类型`T`，我们应该直接包含`A.h`，而不是通过包含`B.h`来引入这一类型`T`

2.4头文件命名规则

综述

使用`.h`结尾，全部采用小写和下划线的组合，不应过为简略

正确示例

- 定义装甲板搜索的类:`armor_finder.h`

注：定义类时头文件和类实现应该成对出现，使用相同的命名规则，如`armor_finder.h` `armor_finder.cpp`

- 定义对矿石进行形态学操作的类:`mineral_morph.h`
- 内联函数放在头文件内定义

三、函数

3.1函数参数顺序

参数顺序应该按照：输入参数、其他参数、输出参数的顺序书写

在加入新参数时不要因为它们是新参数就置于参数列表最后, 而是仍然要按照前述的规则, 即将新的输入参数也置于输出参数之前

3.2传递引用作为参数

一个函数接受的参数只能是：值或`const`引用

实例：

```
void ArmorFinder(const Mat& src,float position);
```

输入参数可以是 `const` 指针, 但尽量不使用非 `const` 的引用参数, 除非特殊要求, 比如 `swap()`.

优点

> 避免了构造临时对象的构造函数、析构函数开销

缺点

> 容易对传进来的引用对象造成期望之外的更改，这也是我们传递`const`引用的原因

3.3函数重载

当需要对同一个功能的函数实现不同版本时，若版本较少，可以使用函数重载，若版本较多，且仅根据参数类型不同进行重载，建议使用不同名称的函数

实例：

```
class ArmorFinder{
public:
    //类的构造函数一般采用重载+初始化列表方式实现
    void ArmorFinder(string name);
    void ArmorFinder(const ArmorFinder &a);
}
```

如果打算重载一个函数, 可以试试改在函数名里加上参数信息. 例如, 用 `AppendString()` 和 `AppendInt()` 等, 而不是一口气重载多个 `Append()`.

总结：在函数版本小于等于3个时使用函数重载

3.4函数命名

说明

一般来说, 函数名的每个单词首字母大写 (即 “驼峰变量名” 或 “帕斯卡变量名”), 没有下划线. 对于首字母缩写的单词, 更倾向于将它们视作一个单词进行首字母大写 (例如, 写作 `StartRpc()` 而非 `StartRPC()`)

```
AddTableEntry();
DeleteUrl();
OpenFileOrDie();
```

取值和设值函数的命名与变量一致. 一般来说它们的名称与实际的成员变量对应, 但并不强制要求. 例如 `int count()` 与 `void set_count(int count)`.

四、变量

4.1 局部变量

Tip:将函数变量尽可能置于最小作用域内, 并在变量声明时进行初始化.

C++ 允许在函数的任何位置声明变量. 我们提倡在尽可能小的作用域中声明变量, 离第一次使用越近越好. 这使得代码浏览者更容易定位变量声明的位置, 了解变量的类型和初始值. 特别是，应使用初始化的方式替代声明再赋值, 比如:

```
int i;
i = f();         // 坏——初始化和声明分离

int j = g();     // 好——初始化时声明

vector<int> v;
v.push_back(1); // 坏——用花括号初始化更好
v.push_back(2);


vector<int> v = {1, 2}; // 好—— 一开始就初始化
```

属于 `if`, `while` 和 `for` 语句的变量应当在这些语句中正常地声明，这样子这些变量的作用域就被限制在这些语句中了，举例而言:

```
while (const char* p = strchr(str, '/')) str = p + 1;
```

有一个例外, 如果变量是一个对象, 每次进入作用域都要调用其构造函数, 每次退出作用域都要调用其析构函数. 这会导致效率降低.

```
// Foo是一个类，下面是低效的实现
for (int i = 0; i < 1000000; ++i) {
    Foo f;                  // 构造函数和析构函数分别调用 1000000 次!
    f.DoSomething(i);
}
```

在循环作用域外面声明这类变量要高效的多:

```
Foo f;                      // 构造函数和析构函数只调用 1 次
for (int i = 0; i < 1000000; ++i) {
    f.DoSomething(i);
}
```

4.2变量命名规则

- 对于普通变量，使用小写+下划线的组合，简短的可以使用全部小写的形式

```
int mineral_detect_threshold = 300; //好
int boxlength = 0;//好，使用全小写
int testNum  = 0;//坏
```

- 对于类内成员变量，变量名后加下划线，且应使用小写，避免混淆

```
class ArmorBox{
private:
    cv::Rect box_;
    cv::Point anchor_;
}
```

- 对于结构体内部变量，严格约束使用小写+下划线的组合

说明：由于RM工程中存在很多信息收发的环节，若该结构体属于信息收发包，在结构体结尾处使用`_MSG`进行命名

```
struct ROBOT_CONTROL_MSG{
  uint_8 head;
  uint_8 yaw_angle;      //好
  uint_8 pitch_angle_;   //禁止
  uint_8 end;
}
```

五、类

5.1构造函数

应至少为类提供默认构造函数和拷贝构造函数

> 一个完整的类应该提供默认构造函数、拷贝构造函数、移动构造函数等，但在RM工程中很少被使用，一般是接受不同参数的构造函数最常用

注：若不打算为你的类提供复制构造函数和`=`重载运算符，应当禁止他们

实例：

```
class MineralFinder{
pubilc:
  MineralFinder() = default;
  MineralFinder(int mineral_type);              //函数实现略
private:
  MineralFinder(const MineralFinder& other);  //如果没有提供拷贝构造函数，应当禁用它
                                              //可以将它放在private中
}
```

或者使用

```
// MyClass is neither copyable nor movable.
MyClass(const MyClass&) = delete;
MyClass& operator=(const MyClass&) = delete;
```

5.2类型转换

使用`explicit`定义显示的构造函数

```
class Foo {
  explicit Foo(int x, double y);
  ...
};
void Func(Foo f);
```

此时下面的代码是不允许的:

```
Func({42, 3.14});  // Error
```

这一代码从技术上说并非隐式类型转换, 但是语言标准认为这是 `explicit` 应当限制的行为.

进行类型转换时，应该显式调用`static_cast`和`dynamic_cast`等，应该尽量少使用编译器默许的隐式类型转换，以求代码阅读性更强

5.3类的命名

应该遵循简介的原则，首字母大写，驼峰命名法

```
class MyClass{
}
class ArmorDetector{
}
```

六、其他命名规则

常量命名

总述

声明为 `constexpr` 或 `const` 的变量, 或在程序运行期间其值始终保持不变的, 命名时以 “k” 开头, 大小写混合. 例如:

```
const int kDaysInAWeek = 7;
```

说明

所有具有静态存储类型的变量 (例如静态变量或全局变量, 参见 存储类型) 都应当以此方式命名.

七、注释风格

7.1头文件注释

每个类的定义都要附带一份注释, 描述类的功能和用法, 除非它的功能相当明显.

如果所编写的类是作为工具类使用，请提供简易的使用实例如：

```
// Iterates over the contents of a GargantuanTable.
// Example:
//    GargantuanTableIterator* iter = table->NewIterator();
//    for (iter->Seek("foo"); !iter->done(); iter->Next()) {
//      process(iter->key(), iter->value());
//    }
//    delete iter;
class GargantuanTableIterator {
  ...
};
```

7.2函数注释

对于重要的核心函数，应该提供函数参数和返回值的说明，使用`///`格式进行注释，部分IDE会帮助你自动补全

```
///
/// \param dst:image after pre-process
/// \param BackGround:use for DEBUG
/// \return number of possible_rect
int Mineral::getFitContours(Mat &dst,Mat &BackGround) {
...
  return possible_rect; //此处使用了变量名比较明显的方式，可以省略对于返回值的注释
}
```

7.3块注释和多行注释

尽量成块地对代码进行注释，并且应该在代码块上方一行注释

```
//fliter possible rect with ratio and area,draw the result on background image
for (int i = 0; i < contours.size(); i++)
    {
        Rect fit = boundingRect(contours[i]);

        bool scale_fit =
                float(fit.width) / fit.height < g_para.mineral_threshold_scale_high &&

                float(fit.width) / fit.height > g_para.mineral_threshold_scale_low;
        bool area_fit = float(fit.width) * float(fit.height) >
g_para.mineral_threshold_area_low &&
                        float(fit.width) * float(fit.height) <
                        g_para.mineral_threshold_area_high;
        if (scale_fit && area_fit && hierarchy[i][3] != -1)
```

```
        {
            all_fit_rects.push_back(fit);
#ifdef DEBUG
            rectangle(BackGround, fit, Scalar(255, 0, 0), 2);
            drawContours(BackGround, contours, i, Scalar(0, 255, 0), 2);
#endif
        }
    }
```

如果要使用多行注释，应该让他们对齐

```
    template<class T>                       //T is msg_type,commonly struct
    class MessagePipe{
        friend class Subscriber<T>;         //allow subscriber to edit message pipe
        friend class Publisher<T>;          //allow publisher to edit message pipe
    public:
        using MsgType = T;
    private:
        std::mutex subs_mtx;                //lock this thread,make it possible for
adding or moving elements in subs.
        std::list<Subscriber<T> *> subs;    //contain all subscriber bind to this
message.(ptr)
        std::mutex pubs_mtx;                //lock this thread,make it possible for
adding or moving elements in pubs.
        std::list<Publisher<T> *> pubs;     //contain all publisher bind to this
message.(ptr)
    };
```

八、一些格式细节

8.1运算符

涉及到公式表达或者数学运算的部分，要在运算符两端加入空格

```
int a=c+d;  //错误
int a = c+d;  //错误
int a = c + d;//正确
```

涉及到类型强转的计算，应该显式地给读者展示强转过程或者添加注释

如果一行公式过长，另起一行，应该在运算符后断句并使得下一行对齐

```
 center.x = all_fit_rects[k].x+all_fit_rects[k].width/2+
            all_fit_rects[k].y+all_fit_rects[k].height/2;
```

8.2返回值

如果返回值是一个运算表达式，应该用括号括起来

```
return (apple_tree_seed && lemon_tree_seed)
```

8.3循环

`for`循环标准：应该使用`++i`而不是`i++`，括号间用空格隔开

```
for (int i=1;i<100;++i) { //正确，注意此处在每一对括号间，使用空格隔开，且应该放在for语句这一行

}
for(int j=1;j<100;j++)  //错误，没有空格，而且使用j++
{                       //应该放在上一行

}
```

单句循环可以省略大括号，但不可在for这一行写循环体内容

```
for (int i=1;i<100;++i) { //正确
  test_function(i);
}

for (int i=1;i<100;++i) test_function(i); //错误

for (int i=1;i<100;++i)   //正确
  test_function(i);
```

8.4缩进

统一使用空格缩进，禁止使用制表符(tab)

8.5初始化列表

尽量多使用初始化列表的方式书写构造函数，每行对齐

```
Mineral::Mineral() :locate_complete(false),
                    Number_white(0),
                    Number_yellow(0),
                    src(Mat::ones(720,480,CV_8UC3)){
}
```

九、结束语

欢迎队内成员共同维护这个编程规则，如果有任何补充或觉得不妥的地方，欢迎联系QQ:793182082