

保密等级：□绝密 ■机密 □敏感 □公开

**移动机器人开发代码规范**

**说明书**

*2021-10-08-001*

|  |  |  |  |
| --- | --- | --- | --- |
| 拟 制 |  | 日 期 |  |
| 审 核 |  | 日 期 |  |
| 批 准 |  | 日 期 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **版本履历** | | | |
| 历史版本 | 作者 | 日期 | 修改说明 |
| 1.0 | 张炎高 | 2021-10-08 | 初稿 |
| 1.0 | 李思琦、陈戗 | 2021-10-12 | 修改格式添加新的ROS C++规范 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

*注：该章节主要用于记录本文档（除最后一个章节外）所有内容、格式的变更。*

目录

[第一章 文档介绍 2](#_Toc12349)

[1.1. 文档目的 2](#_Toc16017)

[1.2. 文档范围 2](#_Toc4062)

[1.3. 读者对象 2](#_Toc10189)

[第二章 项目介绍 2](#_Toc31339)

[第三章 代码规范总体要求 2](#_Toc14717)

[3.1. 代码规范的重要性 2](#_Toc1266)

[3.2. ROS代码格式自动化工具 3](#_Toc16454)

[3.3. clang\_format指南 3](#_Toc20005)

[3.3.1. 设置环境 3](#_Toc23078)

[3.3.2. 运行clang\_format 3](#_Toc30138)

[第四章 ROS C++代码具体规范 5](#_Toc742)

[4.1. 良好的命名 5](#_Toc8089)

[4.2. 版权声明 6](#_Toc4363)

[4.3. 代码风格 7](#_Toc27015)

[4.3.1. 编辑器自动格式化 7](#_Toc3917)

[4.3.2. 代码风格规范 7](#_Toc12089)

[4.4. 控制台输出 8](#_Toc32488)

[4.5. 宏定义 8](#_Toc6132)

[4.6. 预处理命令（#if与#ifdef） 8](#_Toc12601)

[4.7. 输出参数 9](#_Toc541)

[4.8. 命名空间 9](#_Toc7050)

[4.9. 继承 9](#_Toc17516)

[4.10. 异常处理 10](#_Toc24058)

[4.10.1. 编写抛出异常时安全的代码 10](#_Toc14754)

[4.11. 枚举 10](#_Toc16097)

[4.12. 全局变量 11](#_Toc29420)

[4.13. Static class variables 12](#_Toc9766)

[4.14. 调用exit() 12](#_Toc3672)

[4.15. 断言 12](#_Toc16471)

[4.16. 可移植性 13](#_Toc23050)

[4.17. 弃用 Deprecation 13](#_Toc28557)

**缩略语**

|  |  |  |
| --- | --- | --- |
| 缩略语 | 英文全名 | 中文解释 |
| ROS | Robot Operating System | 机器人操作系统，提供一系列程序库和工具以帮助软件开发者创建机器人应用软件。 |

# 文档介绍

## **文档目的**

本文档记录移动机器人开发代码规范，包括编码规范、提交规范。目的在于规范代码风格、提交日志。

## **文档范围**

本文档范围主要包括移动机器人开发c++编码规范、git提交规范。

## **读者对象**

本文档的读者对象为移动机器人相关开发人员。

# 项目介绍

移动机器人由计算控制单元、双舵轮、四万向轮、激光雷达、双目相机等组成，使用ROS系统，实现自动跟随、自主移动、自动导航、网络交互等功能，在实际生产中最主要的用途是搬运。

# **代码规范总体要求**

## 代码规范的重要性

代码风格很重要。干净、一致的代码风格可以使代码更容易阅读、调试和维护。我们努力编写优雅的代码，不仅仅是为了简单地完成当下功能需求，还为了让这份代码持续存在，并在未来很多年内被其他开发人员重复使用、阅读和改进。

以下内容是编码准则，除了极少数例外，在独立开发过程中尽可能遵循本指南。由于是基于开源的代码，在遵循开源现有规范的同时，需要比较明确的区分开开源和私有的部分。

**总体要求：**

1. 所有新增加的文件、函数、单行代码都需要加注释，每个注释都必须包含“hqauto:”的字段。
2. 如新增一个文件，则在文件说明中需添加“hqauto:”关键字；如果新增函数则在函数声明中添加“hqauto:”；如果新增几行代码则在代码前添加“hqauto:”关键字。
3. 不随意删除开源的源码，不需要的注释掉，同时添加注释的理由和“hqauto:”关键字。
4. 其他编码规范遵守第四章所述要求。

## ROS代码格式自动化工具

当我们致力于构建性能出色的机器人时，为什么要浪费您大量的宝贵开发时间来格式化代码呢？这里介绍一款出色的工具—— clang-format，

参考链接:https://github.com/davetcoleman/roscpp\_code\_format

clang\_format工具已在2020年更新，并支持ROS Melodic系统。在"使用方法"内容中，选择了Linux命令行以及VS Code插件来进行说明。更多其他工具请参考相关readme.md文件

## clang\_format指南

### 设置环境

\* 检索clang-format: sudo apt-cache search clang-format

\* 安装clang\_format: sudo apt install clang-format-3.9

\* 复制.clang-format文件到机器人代码工程的根目录中，如: ~/catkin\_ws/.clang-format如果您有兴趣改进此配置文件，建议您检查git repo和symlink: ln -s ~/roscpp\_code\_format/.clang-format ~/catkin\_ws/.clang-format

\* 现在，您的catkin\_workspace工程文件夹中的任何文件都将使用此配置文件中所述的ROS编程规范进行格式化。

### 运行clang\_format

**命令行**

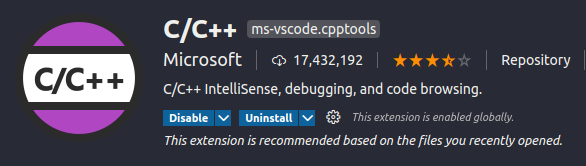
\* 格式化单个文件：clang-format-3.9 -i -style=file MY\_ROS\_NODE.cpp

\* 递归格式化整个目录，包括子文件夹：find . -name '\*.h' -or -name '\*.hpp' -or -name '\*.cpp' | xargs clang-format-3.9 -i -style=file $1

\* 可以将 "格式化整个目录"功能设置快捷指令ros\_format,将下面代码添加到.bashrc或者.zshrc文件中。

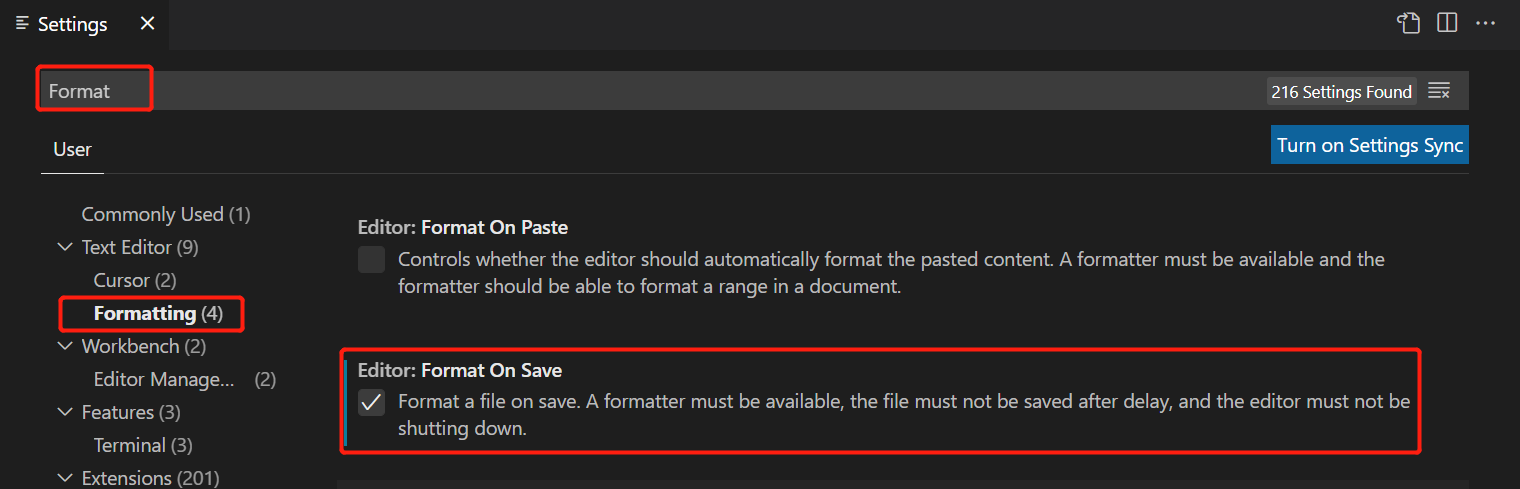
**VS Code插件**

\* 安装 C/C++ 插件。

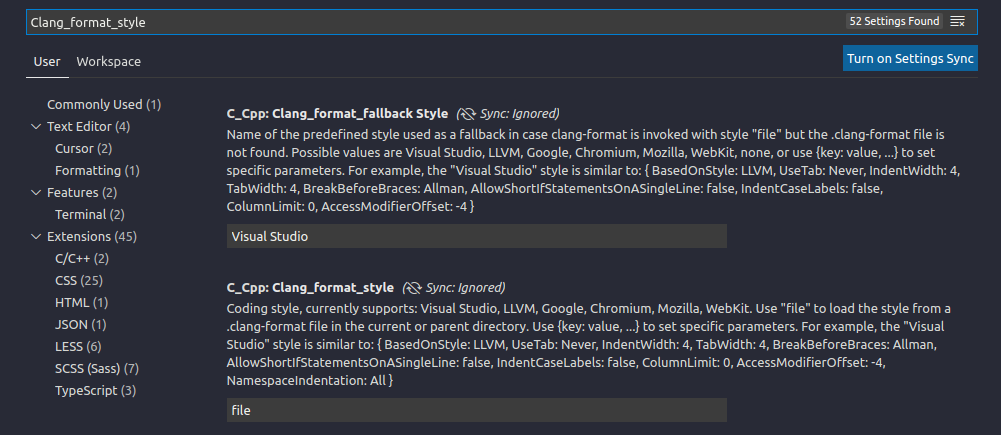


\* 将.clang-format文件添加到工程中，如: ~/catkin\_ws/.clang-format。

\* 打开首选项设置（ctrl + ,），搜索format，可勾选format on save 自动保存。



\* 在Seetings页面重新输入Clang\_format\_style，使得以下两个选项如图配置。



配置好后，在vscode中编辑代码，保存代码(ctrl + s)时编辑器会自动按照脚本规则检查和修改代码，使其满足ROS代码规范。

附: vscode插件工具推荐

\* Highlight Matching Tag 突出显示匹配的开始或者结束标签

\* Image Preview 悬停时显示图像预览

\* Indent Rainbow 使文本的缩进着色，在每个步骤上交替使用四种不同的颜色。

\* TODO Highlight 在代码中突出显示TODO

\* Better Comments BetterComments可以帮助你编写便于阅读的注释。

# ROS C++代码具体规范

## 良好的命名

以下例子表示ROS的命名体系：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 范例 | 命名规范名称 | 规则 | 应用场景 | 示例 |
| **CamelCased** | 大驼峰(匈牙利命名法) | 首字母大写，其后每个单词首字母大写 | 用于表示类名、类型。 | class ExampleClass;(类名) class HokuyoURGLaser;（带缩写单词的类名，缩写字母URG全大写） |
| **camelCased** | 小驼峰(匈牙利命名法) | 首字母小写，之后单词首字母大写 | 方法、函数名 | int exampleMethod(int example\_arg); |
| **under\_scored** | 小写+下划线 | 名称仅使用小写字母，单词之间用下划线分隔。 | ROS packages名称； Topics、Services名； 文件名(.cpp、.h)； 库名(注意格式是libxxx\_yyy,而不是lib\_xxx\_yyy) ； 命名空间 | ros\_openvino\_toolkit (功能包名) action\_server.h(文件名) libmy\_great\_thing(库名) pid\_list; (变量名) int example\_int\_; (成员变量以下划线\_结尾) int g\_shutdown; (全局变量以g\_开头) |
| **ALL\_CAPITALS** | 全部大写 | 全部字母大写，单词之间用下划线分隔。 | 常量 | PI |
| **\_\_XXXX** | 前置下划线 | 前置下划线 (\_\_)，在命名中不要使用前置下划线 | 系统保留 | \_\_builtin\_expect (一般开发者不需要修改这方面内容) |

## 版权声明

* 每个源文件和头文件必须在文件开头包含版权声明。
* 在ros-pkg和wg-ros-pkg存储库中，LICENSE目录包含许可证模板，以注释形式包含在C / C ++代码中。

文件开头加入版权公告，然后是文件内容描述。文件包含以下项:

1. 版权(Copyright statement): 如Copyright (c) 2018 Intel Corporation
2. 许可版本(License boilerplate): 为项目选择合适的许可证版本，如Apache2.0，BSD，LGPL，GPL
3. 作者(author line): 标识文件的原始作者

例:

|  |
| --- |
| /\*  \* Copyright (c) 2018 Intel Corporation  \*  \* Licensed under the Apache License, Version 2.0 (the "License");  \* you may not use this file except in compliance with the License.  \* You may obtain a copy of the License at  \*  \* http://www.apache.org/licenses/LICENSE-2.0  \*  \* Unless required by applicable law or agreed to in writing, software  \* distributed under the License is distributed on an "AS IS" BASIS,  \* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.  \* See the License for the specific language governing permissions and  \* limitations under the License.  \*/ |

如果你对其他人创建的文件做了重大修改，将你的信息添加到作者信息中，这样后续开发者有疑问时知道该联系谁。

## 代码风格

### 编辑器自动格式化

编辑器应处理大多数格式化任务，参考3.2节-ROS代码格式自动化工具。

以笔者最常用的编辑器-vim的配置文件为例，设置编辑器的配置文件：

|  |
| --- |
| " 自动缩进  set autoindent  set cindent  " Tab键的宽度  set tabstop=2  " 统一缩进为2  set softtabstop=2  set shiftwidth=2  " 使用空格代替制表符  set expandtab  " 在行和段开始处使用制表符  set smarttab  " 显示行号  set number  " 历史记录数  set history=1000 |

### 代码风格规范

* 每个块缩进2个空格。切勿插入tabs，设定编辑器将tab转为空格，UNIX / Linux下无条件使用空格。
* 命名空间的内容不缩进。
* 括号，无论是左右括号，都独占一列。例：

|  |
| --- |
| if(a < b)  {  // do stuff  }  else  {  // do other stuff  } |

* 每行最长120个字符
* 每个头文件开头都应该包含#ifndef，防止重复包含。例：

|  |
| --- |
| #ifndef PACKAGE\_PATH\_FILE\_H  #define PACKAGE\_PATH\_FILE\_H ...  #endif |

* 尽量不使用非ASCII字符，使用时必须使用UTF-8格式。

## 控制台输出

避免使用C或者C++语言风格的字符串输出(比如printf， cout…)

可以使用rosconsole来满足您所有的输出需求，它提供了带有printf和stream-style的宏参数。不过其与printf不同的地方是：

1. 带颜色的控制台格式化输出
2. 详细的信息级别以及配置文件控制
3. 输出到/rosout话题上，可以在同一网络下的所有用户查看到。
4. 可以选择记录在磁盘上。

## 宏定义

尽可能避免使用宏。与内联函数和const变量不同，宏既没有类型也没有范围。

推荐参阅谷歌cpp代码规范中对于宏的描述。

## 预处理命令（#if与#ifdef）

对于条件编译（上面6.2小节解释的#ifndef头文件保护除外），请始终使用#if，而不是#ifdef。有人可能会编写如下代码：

|  |
| --- |
| #ifdef DEBUG  temporary\_debugger\_break();  #endif |

其他人可能会在关闭调试信息的情况下来编译代码，例如：

|  |
| --- |
| cc -c lurker.cpp -DDEBUG = 0 |

这时候就有风险。如果必须使用预处理器，请始终使用#if。即使根本没有定义DEBUG，它也可以正常工作，并且做正确的事情。

|  |
| --- |
| #if DEBUG  temporary\_debugger\_break();  #endif |

## 输出参数

方法/函数的输出参数（例如：函数可以修改的变量），是通过指针而不是通过引用传递的。

例如：

|  |
| --- |
| int exampleMethod（FooThing输入，BarThing \*输出）; |

相比之下，当通过引用传递输出参数时，调用者（或后续维护人员）被告知参数是否可以在不读取方法原型的情况下被修改。

## 命名空间

推荐使用namespace来限定代码范围，根据package的名称来选择一个描述性强的名称

切勿在头文件中使用using。这样做会污染包括头文件的所有代码的namespace。

在源文件(cpp)中使用using指令是可以接受的。但是最好使用using-declarations，它仅提取您打算使用的内容。

例如：

|  |
| --- |
| using namespace std; // Bad, because it imports all names from std:: |

可以改为：

|  |
| --- |
| using std::list; // I want to refer to std::list as list  using std::vector; // I want to refer to std::vector as vector |

## 继承

使用组合通常比使用继承更适宜(这一点在GOF在《Design Patterns》里是反复强调的)。如果使用继承的话，只是用公共继承。

当子类继承父类时，子类包含了父基类所有数据以及操作的定义。

在C++实践中，继承主要用于两种场合: 实现继承和接口继承。

* 实现继承 (implementation inheitance)，子类继承父类的实现代码。
* 接口继承(interface inheritance)，子类仅继承父类的方法名称。

继承是定义和实现公共接口的合适手段。基类定义接口，子类实现该接口。(Inheritance is the appropriate way to define and implement a common interface. The base class defines the interface, and the subclasses implement it.)

继承还可以用于提供从基类到子类的通用代码。这种情况下不鼓励使用继承。(Inheritance can also be used to provide common code from a base class to subclasses. This use of inheritance is discouraged. )

在大多数情况下，“子类”可以包含“基类”的实例，并以较少的混淆可能性实现相同的结果。(discouraged. In most cases, the “subclass” could instead contain an instance of the “base class” and achieve the same result with less potential for confusion.)

子类重载虚拟(virtual)方法时，始终将其声明为virtual方法，以便读者了解正在发生的事情。(When overriding a virtual method in a subclass, always declare it to be virtual, so that the reader knows what’s going on.)

强烈建议不要多重继承，多重继承允许子类拥有多个父类，它会引起无法容忍的混乱。

## 异常处理

与返回整数error codes相反，异常(Exceptions)是首选的错误报告机制。在测试框架中，异常确实十分好用。

对于现有代码，引入异常会牵连到所有依赖代码，异常会导致程序控制流无法通过查看代码确定——函数有可能在不确定的地方返回。所以有以下需要注意的地方:

* 始终在每个相关函数/方法上，记录您的package可能会抛出哪些异常。
* 不要抛出析构函数的异常。
* 不要从您不直接调用的回调中引发异常。
* 如果您在package中选择使用错误代码代替异常，则仅使用错误代码。 始终如一。

### 编写抛出异常时安全的代码

当您的代码可以被异常中断时，您必须确保当堆栈溢出时，相关资源将被释放。特别是，必须释放互斥锁，并且必须释放堆分配的内存。

## 枚举

命名您的枚举，例如：

|  |
| --- |
| namespace Choices  {  enum Choice  {  Choice1,  Choice2,  Choice3  };  }  typedef Choices::Choice Choice; |

这样可以防止枚举污染它们所在的命名空间。

枚举中的单独的item引用：Choices :: Choice1。

typedef仍然允许声明Choice enum而不是命名空间。

如果您使用的是C ++ 11和更高版本，则可以使用范围枚举。例如：

|  |
| --- |
| enum class Choise  {  Choice1,  Choice2,  Choice3  };  Choise c = Choise::Choice1; |

## 全局变量

不建议使用全局变量（无论变量还是函数）。它们会污染namespace，并使代码的可重用性降低，耦合性大大提高，使得维护变得困难。它们阻止代码的多个实例化，并使多线程编程成为一场噩梦。(They prevent multiple instantiations of a piece of code and make multi-threaded programming a nightmare.)

大多数变量和函数应在类内部声明。其余应在namespace中声明。

例外：文件可能包含main()函数和一些全局的小辅助函数。但是请记住，有一天这些辅助功能可能对其他人有用。

## Static class variables

不建议使用静态类变量。它们阻止代码的多个实例化，并使多线程编程成为一场噩梦。

## 调用exit()

仅在应用程序中定义明确的退出点(exit point)时调用exit()。

切勿在库中调用exit()。

## 断言

使用断言检查先决条件，数据结构完整性和内存分配器的返回值。

断言比编写条件语句要好，后者很少会被执行。

不要直接调用assert（）。而是使用在ros / assert.h中声明的以下函数之一（rosconsole软件包的一部分）：

* ROS\_ASSERT(x > y);
* ROS\_ASSERT\_MSG(x > 0, “Uh oh, x went negative. Value = %d”, x);
* ROS\_ASSERT\_CMD(x > 0, handleError(…));
* ROS\_BREADK();

|  |
| --- |
| /\*\* ROS\_ASSERT asserts that the provided expression evaluates to  \* true. If it is false, program execution will abort, with an informative  \* statement about which assertion failed, in what file. Use ROS\_ASSERT  \* instead of assert() itself.  \* Example usage:  \*/  ROS\_ASSERT(x > y); |

不要在断言中做任何工作；仅检查逻辑表达式。取决于编译环境的设置，可能不会执行该断言。通常会开发启用了断言检查的软件，以捕获异常情况(in order to catch violations)。当软件即将完成时，并且在进行大量测试时发现断言始终是正确的时，您将使用一个标志从编译中删除断言，从而使它们不占用任何空间或时间。catkin\_make的以下选项将为所有ROS package定义NDEBUG宏，从而删除断言检查。

|  |
| --- |
| catkin\_make -DCMAKE\_CXX\_FLAGS:STRING="-DNDEBUG" |

注意：当您使用此命令运行cmake时，它将重新全部编译，并且在后续运行catkin\_make时会记住相关设置，直到删除build和devel目录重新编译为止。

## 可移植性

保持C ++代码的可移植性很重要。以下是注意事项：

1. 不要将uint用作类型。而是使用unsigned int。
2. 从std命名空间中调用isnan()，即std :: isnan()

## 弃用 Deprecation

当要弃用package中的头文件时，可以包含相关警告：

|  |
| --- |
| #warning mypkg/my\_header.h has been deprecated |

当要弃用一个函数时，请添加不建议使用的描述：

|  |
| --- |
| ROS\_DEPRECATED int myFunc(); |

当要弃用一个类时，请弃用其构造函数和所有静态函数：

|  |
| --- |
| class MyClass  {  public:  ROS\_DEPRECATED MyClass();    ROS\_DEPRECATED static int myStaticFunc();  }; |