《MRPT 指导教程》

《MRPT Tutorial》

MRPT_V1.0.2

汪若博※译

keyearth@gmail.com

Mobile Robot Programming Toolkit
移动 机器人 编程 工具箱

版本	历史
V1.0.2	建立文档

... 说明 ...

Warning:本文内容仅供参考，为方便 mrpt 开发者学习而整理撰写，不保证内容准确无误，以 mrpt 官网内容为准: www.mrpt.org

Tips:编程语言为 C++语言，适合具有一定 C++/STL 基础的开发者。

... 知行 ...

MRPT 版本 1.0.2

http://www.mrpt.org/tutorials/programming/

目　录

第一章 入门

1.1 编译 MRPT

这部分描述了如何从源代码编译安装 MRPT。如果你想要安装二进制包，尽快开始开发 MRPT 应用程序，请进入下载页面, 为您的系统获取二进制安装包。请报告关于 MRPT 编辑错误或疑问到 mrpt 用户论坛。

1.1-1 预备条件

访问这里查看 mrpt 需要哪些外部库以及它们的作用，决定为你的应用添加哪些库。

（1.1） Windows

(1.1.1) CMake（必需）

根据以下网址的指导，在系统中安装 CMake：

http://www.cmake.org/cmake/resources/software.html

(1.1.2) wxWidgets（选配，推荐）

除了以下的介绍，用户可以查阅维基百科中的条目 http://wiki.wxwidgets.org/MSVC。从下载页面下载最新的 wxWidgets 源码包(.zip 或 .7z)。解压缩到任意目录，在 cmake 时需要此目录（比如 C:\wxWidgets)。

注意 1：对于 wxWidgets 2.9 之前的版本，必须手动修改 flag,以便能编译支持 OpenGL 的 wxWidgets。通过设置 wxUSE_GLCANVAS 为 1（默认为 0）实现。下面是文档 wxwidgets\include\wx\msw\setup.h 中需要被修改的部分：

```
// Setting wxUSE_GLCANVAS to 1 enables OpenGL support. You need to have OpenGL
// headers and libraries to be able to compile the library with wxUSE_GLCANVAS
// set to 1. Note that for some compilers (notably Microsoft Visual C++) you
// will need to manually add opengl32.lib and glu32.lib to the list of
// libraries linked with your program if you use OpenGL.
// // Default is 0.
// // Recommended setting: 1 if you intend to use OpenGL, 0 otherwise
#define wxUSE_GLCANVAS 1
```

注意 2：对某些版本的编译器，你可能会遇到一些错误提示“在 “\src\msw\window.cpp” 文件中找不到<pbt.h>头文件”。你可以小心编辑 window.cpp 文件，注释掉 include 行，如下所示：

```
#if !defined __WXWINCE__ && !defined NEED_PBT_H
//      #include <pbt.h>
#endif
```

32 位编译：

打开 MSVC 32 位 命令终端（开始菜单->MSVC->Visual Studio tools），执行“cd”进入目录 WXWIDGETS/build/msw 中，执行下面的命令：

```
$ nmake -f makefile.vc BUILD=release SHARED=1 RUNTIME_LIBS=dynamic DEBUG_INFO=0 VENDOR=mrpt USE_OPENGL=1
$ nmake -f makefile.vc BUILD=debug SHARED=1 RUNTIME_LIBS=dynamic DEBUG_INFO=1 VENDOR=mrpt USE_OPENGL=1
```

64 bit 编译：

打开 MSVC64 位 命令终端（开始菜单->MSVC->Visual Studio tools），执行“cd”进入目录 WXWIDGETS/build/msw 中，执行下面的命令：

```
$ nmake -f makefile.vc BUILD=release SHARED=1 RUNTIME_LIBS=dynamic DEBUG_INFO=0 VENDOR=mrpt USE_OPENGL=1 TARGET_CPU=amd64
$ nmake -f makefile.vc BUILD=debug SHARED=1 RUNTIME_LIBS=dynamic DEBUG_INFO=1 VENDOR=mrpt USE_OPENGL=1 TARGET_CPU=amd64
```

(1.1.3) OpenCV（选配，强烈推荐）

建议使用 OpenCV2.4.0（或更新版本）。

最新版本可以访问 http://opencv.org/。

用 CMake 编译完成 OpenCV 后，在 MRPT 的 CMake 工程中运行“configure”命令，会自动探测到 OpenCV 的 build 目录并使用它。如果没有自动探测到，在 CMake 中手动设置 OpenCV_DIR 变量到 OpenCV 的 build 目录。

(1.1.4) FFmpeg for Win32（选配）

FFmpeg 库是可选配的，仅仅当你需要使用 CFFMPEG_InputStream,这个类时需要。主要是用于支持 IP 摄像头。直接下载并解压缩到任意目录：

- http://ffmpeg.arrozcru.org/builds/shared/ffmpeg-r16537-gpl-lshared-win32.tar.bz2
- 或 http://www.mrpt.org/downloads/ffmpeg-r16537-gpl-lshared-win32.tar.bz2
- 或下载最新的“dev”“shared”Win32 build : http://ffmpeg.zeranoe.com/builds/

接着运行 CMake（cmake-gui），使能 MRPT_HAS_FFMPEG_WIN32, 按下‘Configure’并且设置 FFMPEG_WIN32_ROOT_DIR 为解压后的 FFmpeg 库的路径。

在 Windows 下，FFmpeg DLLs 在用 MRPT 编写的程序编译后运行时被调用。要确保 /ffmpeg/bin 路径在系统环境变量 PATH 中。

如果你在 Visual Studio 2008（或更高版本）工程中使用 ffmpeg 比较新的版本，你可能在某些地方遇到下面的错误：

```
e:\code\ffmpeg-git-5501afa-win32-dev\include\libavutil\mathematics.h(24) : fatal error C1083: Cannot open include file: 'stdint.h': No such file or directory

... : fatal error C1083: Cannot open include file: 'inttypes.h': No such file or directory
```

一个快捷的解决方式是把出错行替换为下面这句：

```
#include <mrpt/utils/mrpt_stdint.h>
```

或者

```
#include <mrpt/utils/mrpt_inttypes.h>
```

（1.1.5）OpenKinect’s freenect（选配，仅使用 Kinect 需要）

详见此页。

（1.1.6) PCL 点云库（选配）

遵照官网上提示下载、编译、安装：http://pointclouds.org/

目前仅仅实现了一小部分 MRPT 和 PCL 交互的功能（详情查看 changelog 记录）。

（1.2） GNU/Linux

这些是建议应该存在于你的 Linux 系统中的库（开发包）（见下文 Ubuntu/ Debian 的/ Fedora 简要说明）。所有这些都是可选配的，但大部分都强烈推荐安装。

- OpenCV
- wxWidgets
- ffmpeg libraries (libavcodec, libswscale, etc.)
- Freeglut or glut
- zlib
- libjpeg
- libftdi (optional)
- libdc1394-22
- libusb-1.0 (This one is for Kinect support in MRPT 0.9.3+)
- PCL, the Point Cloud Library (Optional, and only for MRPT 0.9.5+)

（1.3） Ubuntu/Debian

运行命令：

```
$ sudo apt-get install build-essential pkg-config cmake \
    libwxgtk2.8-dev libftdi-dev freeglut3-dev \
    zlib1g-dev libusb-1.0-0-dev    \
    libdc1394-22-dev libavformat-dev libswscale-dev \
    lib3ds-dev libjpeg-dev libopencv-dev libgtest-dev libeigen3-dev
```

如果你想使用自定义编译的一些依赖库（比如，你从源码编译的 OpenCV），你需要将上面命令中重复的包删掉以免重叠。

（1.4） Fedora

运行命令：

```
$ su -c 'yum install gcc gcc-c++ make cmake wxGTK-devel opencv-devel freeglut-devel'
```

（1.5） OpenSUSE

运行命令：

```
$ sudo zypper install make gcc gcc-c++ cmake cmake-gui pkg-config \
      zlib-devel wxGTK-devel wxGTK-gl libusb-devel freeglut-devel
```

（1.6） Mac

目前在 Mac 上还不能 100%运行。解决所有问题后再说。

1.1-2 CMake build 选项

（2.1） 使用 cmake-gui

打开 cmake-gui (可用于 Windows/Linux) 并设置"source dir" 到你下载的 MRPT 源码路径。设置 "binary directory"到一个新建的空文件夹，用于存放生成的工程文件。点 "configure", 检查错误, 调整所需的选项(阅读下面关于选项的介绍) 并点击 "Generate"。

（2.2） 在控制台运行 cmake

这个选择在 Windows/Linux/MacOS 系统上都有效。新建一个空文件夹用于存放生成的工程/Makefile 文件。然后运行：

```
$ cmake    /home/.../MRPT
```

将“/home/…/MRPT”替换为你下载的 MRPT 源码包的真实路径。命令使用所有默认选项。运行下面命令，进行选项配置：

```
$ ccmake .
```

（2.3） 感兴趣的 build 选项

适用所有平台/编译器：

- BUILD_APPLICATIONS：默认为 ON, 如果未选上，则不能生成应用软件。如果你只想生成 MRPT 库，可以选择取消此选项。你也可以使用 MRPT_BUILD_DIR/libs/MRPT_ALL_LIB.* solution (或 Makefile) 来实现同样的功能。
- BUILD_ARIA：是否生成用于连接 Activemedia 机器人的 ARIA 库。默认为 ON。如果你不打算使用这些机器人，选择取消此选项。
- BUILD_xSENS：是否使用 CMT 库用于连接 xSens 惯性传感器， 默认为 ON。
- BUILD_EXAMPLES：是否编译"/samples" 中的所有示例， 默认为 OFF。
- BUILD_KINECT：默认为 ON。 如果你不需要使用 Kinect，则取消选择。
- BUILD_SHARED_LIBS：如果设定为 OFF，则创建静态链接库，否则创建动态链接库(.so/.dll)。 默认为 ON。**强烈推荐总是使用 shared_libs**，除非你特别需要用静态链接库。
- EIGEN_USE_EMBEDDED_VERSION :默认为 ON。使用/otherlibs/eigen3/中的 Eigen 头文件构造 MRPT。如果你已经在你的系统中安装 Eigen，且对 pkg-config 是可见的，则建议你取消此选项(目前, Eigen3 还没有任何发行版的资源库，这也是为什么默认为 ON）。
- MRPT_ALWAYS_CHECKS_DEBUG：如果设置为 ON, 将会在很多类的运行时进行额外的安全性检查。默认为 OFF。
- MRPT_ALWAYS_CHECKS_DEBUG_MATRICES：如果设置 ON, 将会在一些 Matrix 矩阵操作运行时进行额外安全性检查。默认设置为 ON。
- MRPT_ENABLE_EMBEDDED_ENABLED_PROFILER :如果使能,在 "MRPT_BEGIN/MRPT_END" 宏中的所有代码块将会被统计概况，任意程序运行结束时将会有一些统计数据输出到控制台。默认为 OFF。
- MRPT_HAS_ASIAN_FONTS :在 mrpt::utils::CCanvas（查看这里） 中可使用亚洲字体, 库大小会增加约 1.5Mb 。默认为 ON。
- MRPT_HAS_SVS :为了启动 Videre SVS 库用于驱动他们公司的立体摄像头，在启动此项之前你需要在你的系统中安装供应商的库。设置该选项为"ON"后, 新的配置字段

"SVS_ROOT_DIR" 将会出现，并指向 SVS 库的路径。（2010 年 8 月起, 此项只能在 GNU/Linux 上工作)。

- MRPT_OCCUPANCY_GRID_CELLSIZE：可以是 8 或 16（位）。设定 mrpt 类 mrpt::slam::COccupancyGridMap2D 中每个单元格的大小。默认为 8 位。
- PCL_DIR：如果你已经在你的系统中安装了 PCL 点云库, 设置此选项为 PCLConfig.cmake 文件的路径，以便与 MRPT 集成。
- USER_EXTRA_CPP_FLAGS：在这里你可以添加任何要传递给编译器的选项。

仅适用于 Windows：

- MRPT_HAS_FFMPEG_WIN32：如果 FFmpeg 已经下载解压了，可启动此项(运行 "configure"后) 并设置 FFMPEG_WIN32_ROOT_DIR 为此 FFmpeg 解压文件夹的路径 (如 "c:\ffmpeg-r16537-gpl-lshared-win32")。
- MRPT_HAS_BUMBLEBEE：用于使用 Bumblebee 立体相机 SDK 。你需要供应商提供的"Triclops" 和 "Digiclops" 库。设置此项为"ON"后, 新的配置字段 "BUMBLEBEE_DIGICLOPS_ROOT_DIR" 和"BUMBLEBEE_TRICLOPS_ROOT_DIR" 会出现，必须输入相应正确的路径。

仅适用于 GNU/GCC 编译器:

- MRPT_ENABLE_LIBSTD_PARALLEL_MODE：使能 GNU libstdc++并行模式 (见 http://gcc.gnu.org/onlinedocs/libstdc++/manual/parallel_mode.html)。 默认为 OFF。
- MRPT_ENABLE_PROFILING：启动生成信息用于概要分析，默认为 OFF。
- MRPT_OPTIMIZE_NATIVE :启动对当前架构的优化 (-mtune=native)。对旧的 GCC 版本默认为 OFF，对 GCC4.2+版本默认为 ON。如果你使用旧版 GCC (<4.2)，该选项不能被编译器识别，此项不能设置。不过，可以设置 USER_EXTRA_CPP_FLAGS 为针对你平台的优化选项，比如: -march=pentium4。

1.1-3 生成 Makefiles/IDE 工程

选用你最熟悉的 Makefile/IDE 系统。支持 32/64 位 Visual Studio 工程、Unix Makefiles、基于 MinGW 的 Codeblocks 工程、MinGW Makefiles 等等。对于 Eclipse IDE 有几种集成 CMake 的方式，请见此网页。

对于 Netbeans:

- 首先, 确保已经安装 C/C++ plugin!
- 主菜单 -> 新建工程: C/C++ -> 使用已有源码的 C/C++工程。
- 点击“下一步”并选择 MRPT root dir，内有 CMakeList.txt 文件。
- 在"Select Configuration Mode"，选择自动，然后“cmake”会出现。

1.1-4 编译

正常编译：在 Visual Studio 工程中为你想编译的目标点击 "build all" (debug 或 release)。

对于 Unix Makefiles 则执行:

```
$ make -j2
```

所有编译完后，一个好办法是运行一个测试程序。在 Visual Studio 内编译一个“test”工程，或在 Unix/MacOS 上运行命令“make test”。

1.1-5 其它编译器介绍

（5.1） Windows 下使用 MinGW

- 安装 MinGW: 推荐: http://tdm-gcc.tdragon.net/
- 在用 MinGW 编译 MRPT 之前, 强烈推荐用 MinGW 先编译 wxWidgets 和 OpenCV
- 编译 wxWidgets：打开一个命令行终端并切换到 build/msw 目录下，接着执行以下命令重新编译 Release 和 Debug 版本（如 shared libs），这样 CMake 就可以正确检测到 wxWidgets。

```
# mingw32-make -f makefile.gcc SHARED=1 USE_OPENGL=1 BUILD=release DEBUG_INFO=0 VENDOR=mrpt
# mingw32-make -f makefile.gcc SHARED=1 USE_OPENGL=1 BUILD=debug DEBUG_INFO=1 VENDOR=mrpt
```

就平常使用 make 一样，如果你的电脑是多核处理器，可以加-j4 标志使用并行快速编译。

在 64 位系统下使用 MinGW 编译 wxWidgets 时，需要在上面的参数里增加参数 TARGET_CPU=amd64，否则即使 MinGW64 也只能生成 32 位结果。

- 编译 OpenCV：使用官网介绍的流程，用 CMake 创建，并选择 MinGW 编译器，根据 OpenCV 创建指导进行编译。
- 打开 cmake-gui，选择源代码路径和存放生成的工程文件的文件夹。
- 点击 configure，并在编译器选择框中选择 MinGW Makefiles。如果出现以下错误：

```
CMake Error: CMake was unable to find a build program corresponding to "MinGW Makefiles". CMAKE_MAKE_PROGRAM is not set.  You probably need to select a different build tool.
```

这表明 MinGW 没有正确安装，回顾以上的安装过程.

- 如果一切正常, 你会看到用红色标记的新的 CMake 变量，根据你自己的需求配置 MRPT，满意后点击 Generate。
- 打开命令行终端，在生成的工程文件夹中执行：

```
mingw32-make
```

注意：要添加全路径(如 c:\MinGW\bin\mingw32-make) ，或者把 MinGW "bin"文件夹的路径添加到系统环境变量 PATH 中，才能开始正常编译过程。

（5.2） GNU/Linux 下使用 clang

- 安装 clang 和 LLVM。在 Debian/Ubuntu 系统：sudo apt-get install clang llvm
- 建立一个空文件夹，并运行下面 CMake 命令：

```
CC=/usr/bin/clang CXX=/usr/bin/clang++ cmake [PATH_TO_MRPT_SOURCES_ROOT]
```

- 最后，就像 GNU/gcc 一样编译生成。

1.2 你的第一个 C++/MRPT 程序

以下是说明如何使用 MRPT C++库创建新的应用程序，假设 MRPT 已经安装在用户的系统中（比如安装在系统目录/usr/include）。查看指导创建 MRPT 库。

Windowns 系统下，你可以选择下载已经编译好的 MRPT 安装包（直接安装），但还是强烈建议重新编译整个 MRPT 库，以适配你的实际系统和编译器。

这个完整的示例可以在 MRPT 包中找到，在{MRPT_DIR}/doc/mrpt_example*.tar.gz，或者到网上下载。

1.2-1 源码文件

查看 MRPT 中存在的库，根据你代码的依赖库添加适当的头文件。

```
#include <mrpt/slam.h>              // Include all classes in mrpt-slam and its dependencies
#include <mrpt/base.h>              // Include all classes in mrpt-base and its dependencies

using namespace mrpt;               // Global methods, and data types.
using namespace mrpt::utils;        // Select namespace for serialization, utilities, etc...
using namespace mrpt::poses;      // Select namespace for 2D & 3D geometry classes.
using namespace mrpt::slam;       // Select namespace for maps, observations, etc...
using namespace std;
```

如果你喜欢在代码里使用带有命名空间的完整类名（比如喜欢用 mrpt::math::CMatrixFloat 而不是 CMatrixFloat），那么你可以去掉上面代码中的命名空间声明。

这个源码在： {MRPT_DIR}/doc/mrpt_example1/test.cpp 文件中。

1.2-2 使用 CMake

推荐使用 CMake，这是一种用 MRPT 库生成跨平台应用的最简单的方式。创建有下面内容的 CMakeLists.txt 文件，保存在你的程序同目录下（在本例中，目录下只有 test.cpp 文件）。

```
PROJECT(mrpt_example1)

CMAKE_MINIMUM_REQUIRED(VERSION 2.4)
if(COMMAND cmake_policy)
      cmake_policy(SET CMP0003 NEW)   # Required by CMake 2.7+
endif(COMMAND cmake_policy)

# ----------------------------------------------------------------------
#    The list of "libs" which can be included can be found in:
#       http://www.mrpt.org/Libraries
#
#    The dependencies of a library are automatically added, so you only
#      need to specify the top-most libraries your code depends on.
# ----------------------------------------------------------------------
FIND_PACKAGE( MRPT REQUIRED base)      # WARNING: Add all the MRPT libs used by your
program: "gui", "obs", "slam",etc.

# Declare the target (an executable)
ADD_EXECUTABLE(mrpt_example1
```

```
    test.cpp
    )
TARGET_LINK_LIBRARIES(mrpt_example1 ${MRPT_LIBS})

# Set optimized building:
IF(CMAKE_COMPILER_IS_GNUCXX AND NOT CMAKE_BUILD_TYPE MATCHES "Debug")
    SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -O3 -mtune=native")
ENDIF(CMAKE_COMPILER_IS_GNUCXX AND NOT CMAKE_BUILD_TYPE MATCHES "Debug")
```

如果你不能确定添加哪个库到“FIND_PACKAGE(MRPT REQUIRED …)”命令中，可以阅读相关教程。

使用下面命令调用 CMake：

```
ccmake .
```

在 Linux/Mac 系统，或：

```
cmake-gui .
```

在 Linux/Windows 系统，需要用户提供 MRPTConfig.cmake 文件路径（在创建 MRPT 库的过程中已经生成），有时候 CMake 会自动找到 MRPT 目录不需要用户提供路径。

在 Windows 中，（对于旧版 CMake）如果 CMake 不能自动找到 wxWidgets 路径。你也必须设定变量 wxWidgets_ROOT_DIR 为 wxWidgets 根目录路径。

在 Linux 中，如果 MRPT 已经安装在系统中（比如/usr 或/usr/local 或），cmake 会自动找到 MRPT 配置文件，不要额外操作。

1.2-3 接下来？

你可以尝试编译这些示例。

1.3 简介 MRPT 库

MRPT 包含一系列 C++库以及很多 ready-to-use 应用。这部分介绍对于移动机器人开发者最感兴趣的部分：库。

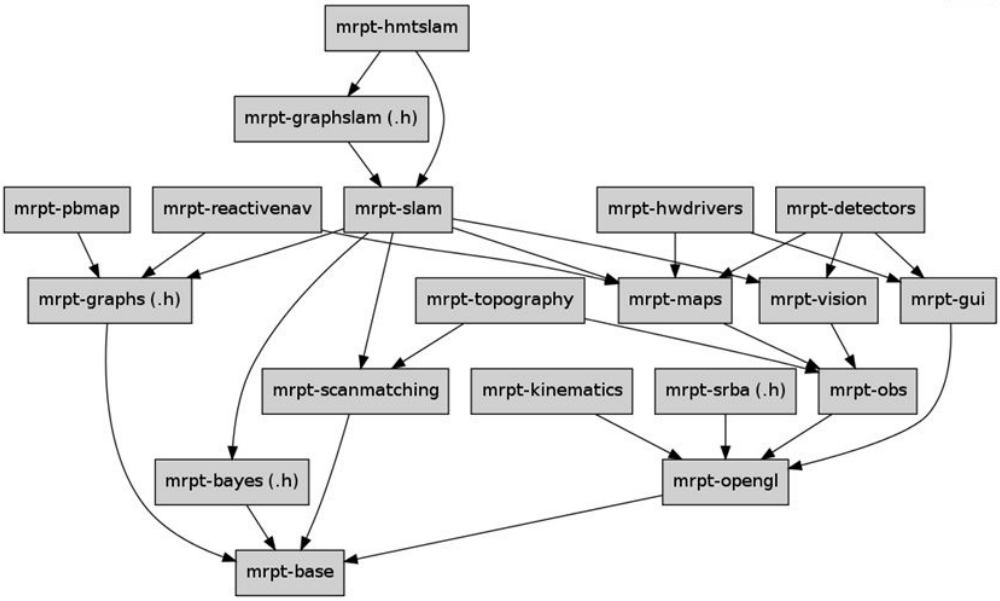
1.3-1 已有库

MRPT 中有大量的 C++模板和类，将它们分割成一系列的库和模块是个好主意，用户可以仅选择依赖的部分库，减少编译时间和以后的依赖问题。

下图是 MRPT 中已有库的依赖图谱。箭头 "A -> B" 意味着 "A 依赖于 B"。

点击任意库可以进入该库的 Doxygen 文档页面，有该库的简介。

这些库还被进一步分割成一个或多个模块：查看所有模块清单。

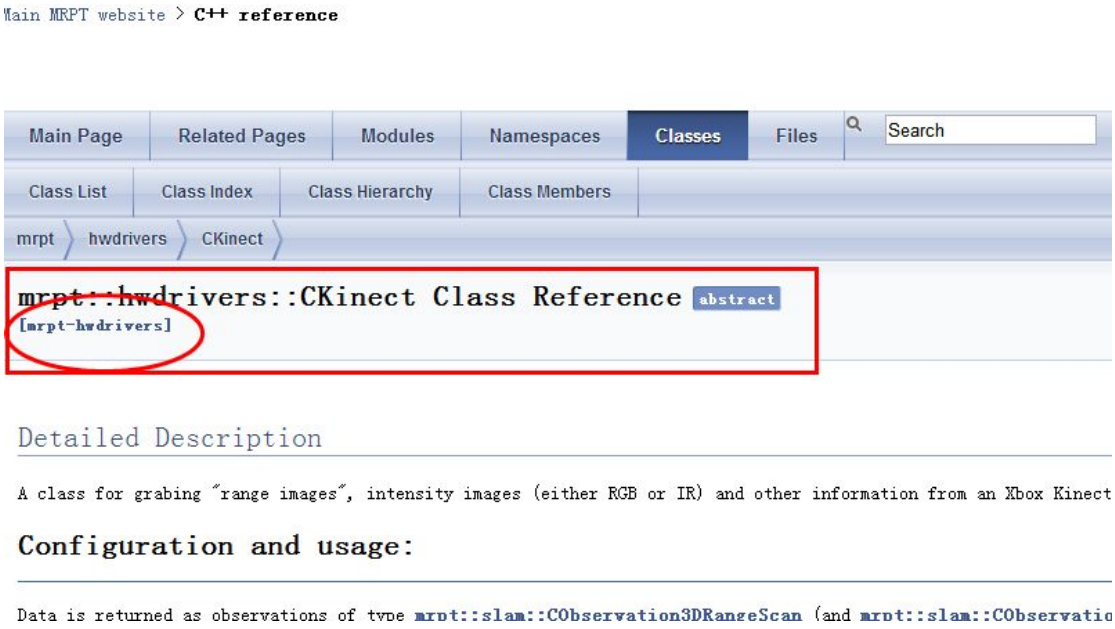


1.3-2 创建库

为了允许任何人都能提交自己的库到 MRPT，我们制定了一个简单的 CMake 规则，比如目录、结构等等。查看网页。

1.4 你需要那个 MRPT 库？

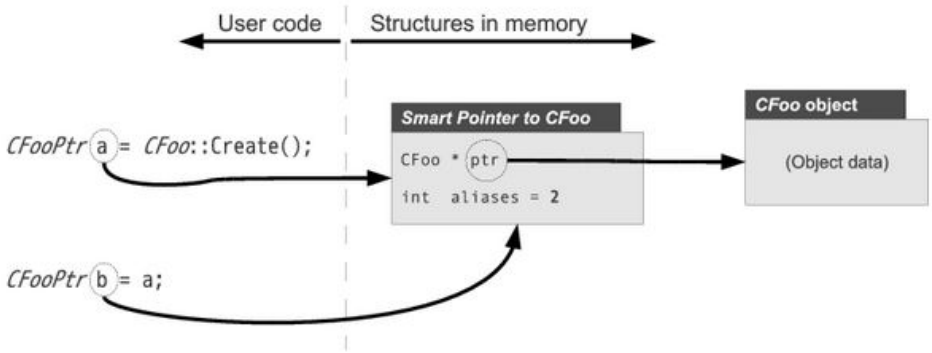
这里有个简单的方法：对于每个你想用的 MRPT C++类，你都可以在 Doxygen API 参考（全部类清单）页面搜索到。就在这个类页面的左上角，在类名下面有个小注解（方括号内），这个方括号内就是这个类所在的库名。



有的类属于一个库下的其中一个“模块”（全部模块清单），左上角注解就是该模块名，再点击才进入该模块所属的库页面。很多 MRPT 库只集成了一个“模块”，有些 MRPT 库内有很多“模块”，比如 mrpt-base 库。

1.5 智能指针

智能指针是一种特殊的类，像普通的指针一样操作，但是有非常智能的性质。智能指针可以自动维护一个内部计数器，用于指示对象存在多少个引用(贯穿整个程序)，当没有有效的引用时会自动删除对象回收内存。以下图表说明了一对智能指针的内部存储结构，它们实际上是同一个对象的别名:



更多智能指针的细节以及方法，请查看《MRPT book》中的智能指针一章。

1.6 常见问题和错误

1.6-1 一般问题

（1.1） 我能否在 Ardunio 上运行 MRPT？

答：MRPT 能在任何可以使用 GCC 编译的 32 或 64 位平台上运行。Arduino 使用 Atmel 8 位 AVR 微控制器，所以它不能运行 MRPT。

（1.2） 我能否在其它嵌入式系统上运行 MRPT？

答：已经成功的在 ARM，MIPS，S390，SPARC,……下编译和运行（至少通过了单元测试）。请查看对于这些平台的 Debian 预编译包列表。

1.6-2 常见错误

（2.1） I have errors like: “*FATAL*: Signal SIGSEGV caught!“

这可能是一个真正的 Bug 或者是兼容问题。首先判断是不是兼容问题，以下面的命令运行你的程序：

```
gdb --args [PROGRAM_NAME [OPTIONAL ARGS] ]
```

然后输入命令“run”并等待错误发生，gdb 会停在出错点。如果你看到 SIGILL：

```
Program received signal SIGILL, Illegal instruction.
 0x005bf52b in .....
   from /usr/lib/libmrpt-....
```

那么这个问题可能是 MRPT 编译时被过度优化，你的 CPU 不支持。

解决方案：

a) 如果 MRPT 从一个 Ubuntu 资源库安装的,首先在资源搜索源中删除当前的 MRPT

PPA 源(例如,从 synaptics - >软件源),然后换成替代安装源,针对支持 SSE/SSE2 的处理器(多年前的)进行优化。如果你的处理器仍然太老旧,你只能使用源码编译(见下一点)。

b) 如果 MRPT 是手动编译的,可以在 cmake 中重新配置它。注意选中在 cmake 高级视图中的选项 DISABLE_SSE3 和 DISABLE_SSE4。然后,点击"configure"、"generate",重新编译整个 MRPT 或你需要的部分。

(2.2) I have errors like "*FATAL*: Signal SIGSEGV caught!" and it's not a CPU issue!

首先,以下面的命令运行你的程序:

```
gdb --args [PROGRAM_NAME [OPTIONAL ARGS] ]
```

然后输入"run",如果程序崩溃,并提示很多不是 SIGILL 的错误,那么很可能存在一个大 Bug。 仍然在"gdb"中,执行命令"bt"(backtrace)去判断到底哪里发生的错误。如果错误发生在 MRPT 代码内部,请考虑提交 bug 报告。

(2.3) CMake complains: "could not find module FindMRPT.cmake or a configuration file for package MRPT"

如果,当你使用 MRPT 编写你自己的程序时,CMake 报告不能找到 MRPT 时:

[1] 确认 MRPT 是:(a)已经以二进制安装包方式安装在系统中,或(b)你成功的使用源代码编译。

[2] 从 CMake 窗口中,寻找 MRPT_DIR 变量并且确保它指向(a) /usr/share/mrpt/或 "c:/Program Files/mrpt" 或(b)你编译 MRPT 时指定的文件夹路径(编译时在 CMake 中的 BINARY_DIR)。

在这两种情况中,CMake 真正寻找的是一个 MRPTConfig.cmake 配置文件,确保在这些路径中一定能够找到。

(2.4) I found the error: "fatal error C1083: Cannot open include file: 'unordered_set': No such file or directory"

如果你是在 Visual Studio 2008 下编译,这可能意味着你需要安装 Visual Studio 2008 FeaturePack 用于提供 C++11 新类支持。

1.7 在 Linux 中用 Makefile 和 pkg-config 编译定制应用

提示: 在前面章节查看如何在不同系统平台下使用 cmake 创建用户程序。

除了用 cmake 的方法,MRPT 也可以允许用户使用 GNU make 及 pkg-config 创建自己的程序。

请到 MRPT/doc/mrpt_example1-with-Makefile/ 文件夹里查看 README 和 Makefile 文件注释。

使用 pkg-config,你还需要:

- 在你的系统中安装 MRPT。选择一个 Ubuntu 资源库(libmrpt-dev),或使用"sudo make install"
- 或者,在你的~/.basrc 文件中设定环境变量 PKG_CONFIG_PATH 为 pkgconfig-no-install 的目录。

```
export PKG_CONFIG_PATH =
[YOUR_MRPT_BUILD_DIR]/pkgconfig-no-install:$PKG_CONFIG_PATH
```

以下是 Makefile 的示例,说明 pkg-config 的用法(或者在线查看)。

```
# Example Makefile script
# Purpose: Demonstrate usage of pkg-config with MRPT libraries
# By: Jose Luis Blanco, 2010.
#
#     ============ *IMPORTANT* ======================
#     For this method to work MRPT must be installed in your
#     system in a path accesible to pkg-config. To check if pkg-config
#     sees MRPT config files, execute:
#          pkg-config --list-all | grep mrpt
#     If no package appears, MRPT is not installed or something else is wrong.
#     ================================================
#

# Set up basic variables:
CC              = g++
CFLAGS          = -c -Wall
LDFLAGS         =

# List of sources:
SOURCES         = test.cpp
OBJECTS         = $(SOURCES:.cpp=.o)

# Name of executable target:
EXECUTABLE = mrpt-example1

# MRPT specific flags:
#    Here we invoke "pkg-config" passing it as argument the list of the
#    MRPT libraries needed by our program (see available libs
#     with "pkg-config --list-all | grep mrpt").
#
CFLAGS          += `pkg-config --cflags mrpt-base`
LDFLAGS         += `pkg-config --libs mrpt-base`

all: $(SOURCES) $(EXECUTABLE)

$(EXECUTABLE): $(OBJECTS)
        $(CC) $(OBJECTS) -o $@ $(LDFLAGS)

.cpp.o:
        $(CC) $(CFLAGS) $< -o $@

clean:
        rm $(OBJECTS) $(EXECUTABLE)
```

1.7 怎样在 ROS 节点中使用 MRPT？

1.7-1 小动作

安装 mrpt_common 包。

在 manifest.xml 文件中声明 mrpt_libs 作为你的 ROS 节点依赖。

当创建一个新的包时，仅需要在依赖列表中追加 mrpt_libs，比如：

```
$ roscreate-pkg  my_package  roscpp  mrpt_libs
```

不需要别的修改：不用修改 CMakeLists.txt 文件。

1.7-2 Ready-to-use 示例

在 MRPT 根目录下查看/sample-ros 文件夹里的内容。

跟随下面的一步步介绍，来创建并测试一个简单的基于 MRPT 类的 ROS(C++)节点。（也可以在 README 文件中找到）

设置 ROS 环境以便能找到教程包：

```
$ cd [MRPT_DIR]/samples-ros
$ export ROS_PACKAGE_PATH=$ROS_PACKAGE_PATH:$(pwd)/mrpt_ros_tutorial
```

生成示例节点：

```
$ rosmake  mrpt_ros_tutorial
```

执行并测试：在一个命令行终端窗口，加载“roscore”。在另一个窗口运行下面的命令：

```
$ rosrun  mrpt_ros_tutorial  example1
```

可选的，在另一个窗口运行下面命令：

```
$ rostopic  echo  /chatter
```

1.8 怎样与 PCL 交互使用 MRPT？

1）确定你知道 CMake 如何工作，你能够用 CMake 创建一个正确设置#include 并链接正确的简单程序。

2）在你的 CMakeLists.txt 中添加 FIND_PACKAGE(PCL ...)命令。到 MRPT 安装目录 MRPT/doc/mrpt-pcl-example 下参考一个可以运行的 MRPT-PCL 集成示例。

另外，到这个网页查看在 MRPT 中有哪些类和函数可以支持 PCL。

第二章 GUI 窗口和 3D OpenGL 图形

2.1 3D 场景简介

2.1-1 示例

- MRPT 源码目录下的/sample/opengl_objects_demo
- MRPT 源码目录下的/sample/display3D

2.1-2 类

3D 场景主要基于类 mrpt::opengl::COpenGLScene。这个类允许用户使用 OpenGL 图元去创建、加载、保存和渲染 3D 场景。这个类可以被理解为一个运行在 OpenGL 系统上的程序，包含一系列的视角定义、图元渲染 等...

一个 "OpenGL 场景"包含一到任意数量的视窗，每一个视窗关联着一组 OpenGL 对象，可选的，一个合适的摄像头位置。正交(2d / 3d)和投影摄像头模型都可以独立用于每个视窗，极大增强了场景渲染的灵活度。每个 OpenGL 场景的对象总是包含至少一个视窗(mrpt::opengl::COpenGLViewport)，这个视窗被命名为 "main" (如果你不设定视窗名称，默认为"main"，因此不用担心忘记名字)。可选地,可以存在任意数量的其它视窗。

视窗是通过它们的名字（区分大小写的字符串）被引用的。每个视窗包含一个不同的 3d 场景(即它们渲染不同的对象)，虽然有多个视窗共享同一个 3D 场景的机制（复制同一个智能指针以避免内存浪费）(见 COpenGLViewport::setCloneView)。

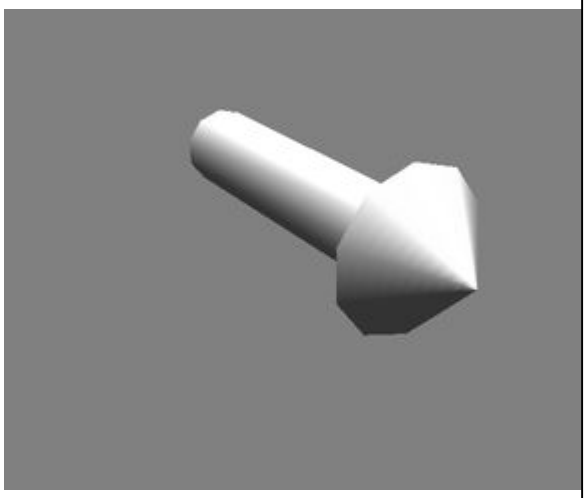
主要的渲染方法：COpenGLScene::render( )，假设已经为整个目标窗口建立了一个视窗。创建别的视窗时，该方法会在内部发送请求到 OpenGL。注意,为允许透明穿透，默认情况下只有各个视窗(非主视窗)的深度缓存会被清除，这个功能可以对被 COpenGLViewport 的相应成员取消。

一般情况下用户不需要手动调用 COpenGLScene::render()，除非是使用 standalone 3D viewer(SceneViewer)或者是运行时 3D 显示窗口(mrpt::gui::CDisplayWindow3D)。

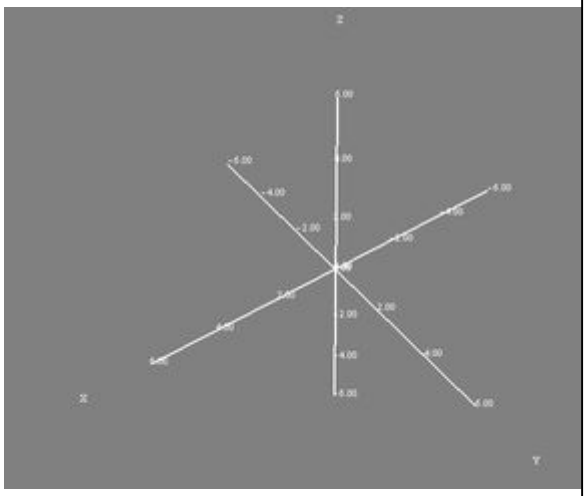
一个 COpenGLScene 对象可以使用 COpenGLScene::saveToFile()保存到 ".3Dscene"文件，以便可以用独立应用程序 3D SceneViewer 可视化查看。

在下面表格中是一些图元类渲染的预览(大多数最新的版本可以在网页找到):

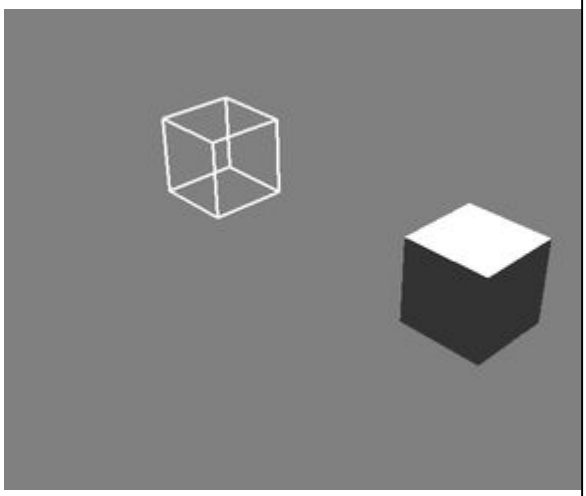
mrpt::opengl::CArrow



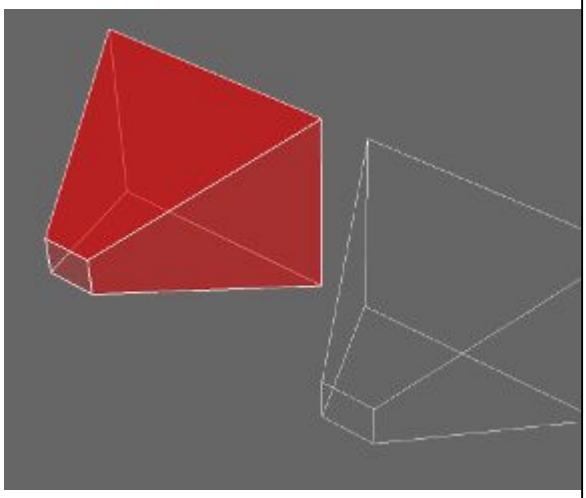
mrpt::opengl::CAxis



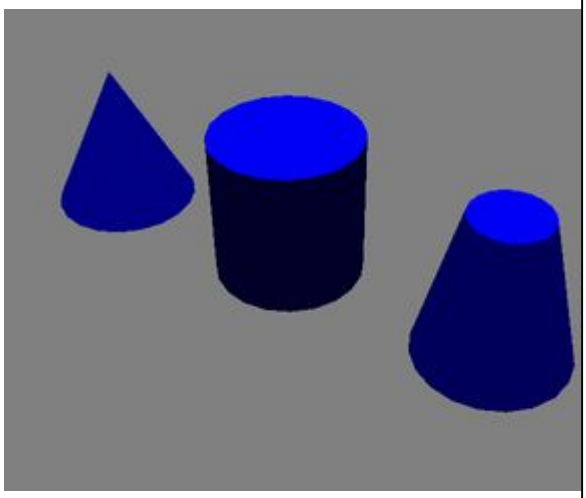
mrpt::opengl::CBox



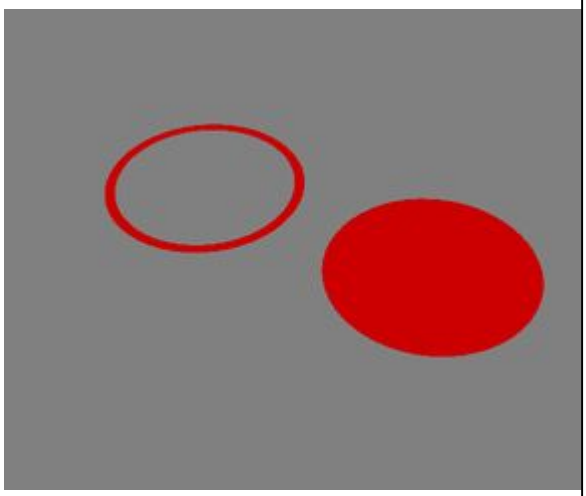
mrpt::opengl::CFrustum

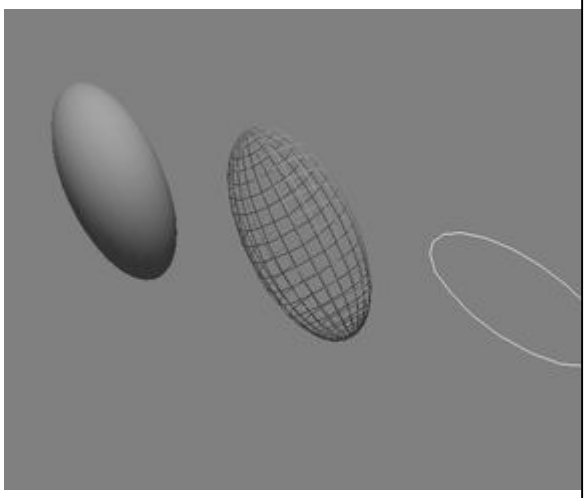
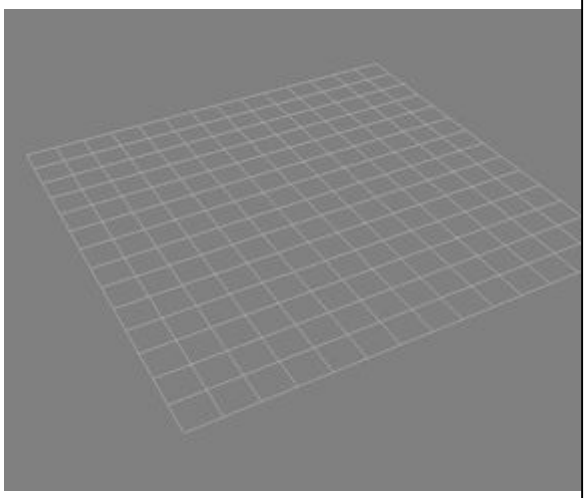
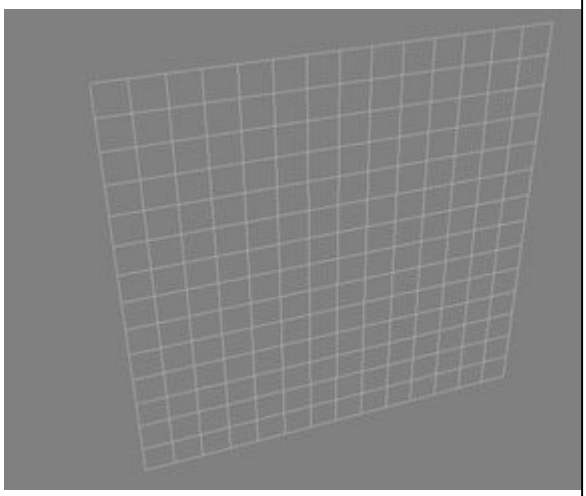


mrpt::opengl::CCylinder

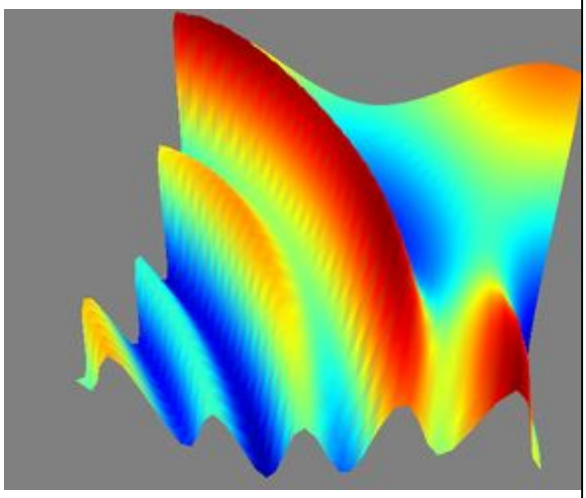


mrpt::opengl::CDisk

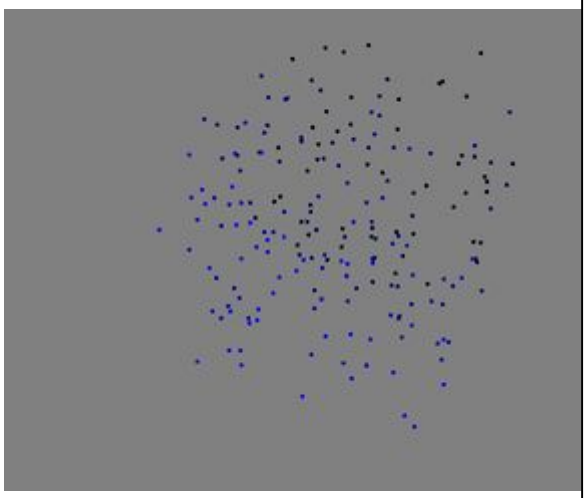


mrpt::opengl::CEllipsoid	
mrpt::opengl::CGridPlaneXY	
mrpt::opengl::CGridPlaneXZ	

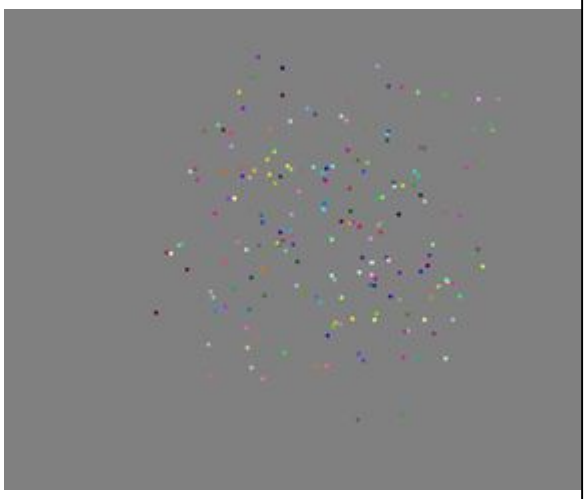
mrpt::opengl::CMesh



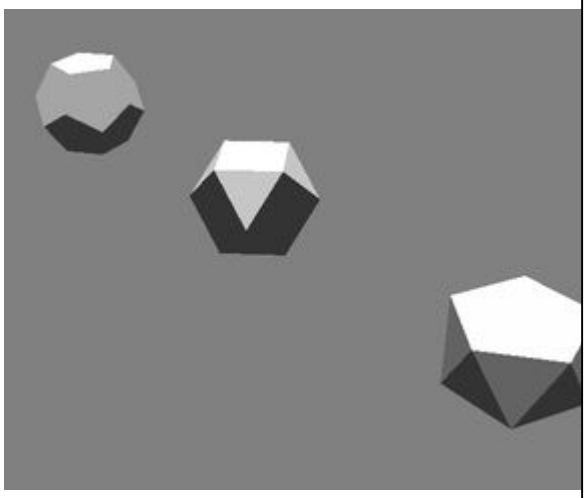
mrpt::opengl::CPointCloud



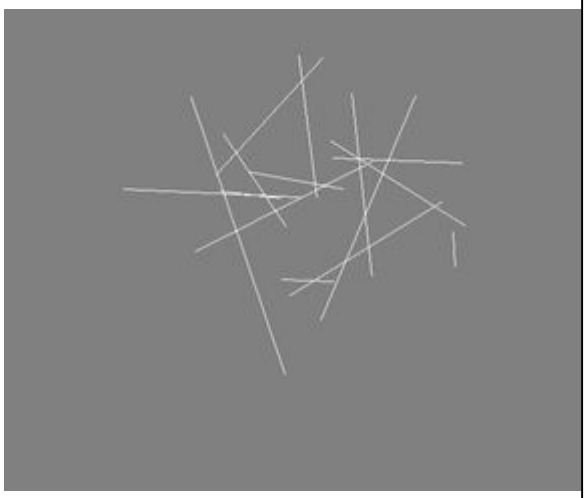
mrpt::opengl::CPointCloudColoured



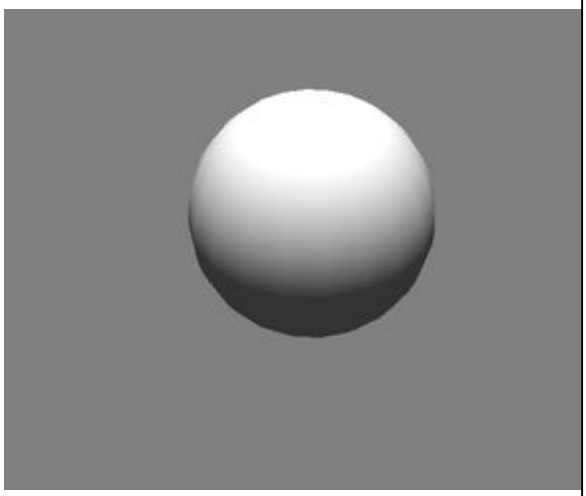
mrpt::opengl::CPolyhedron



mrpt::opengl::CSetOfLines

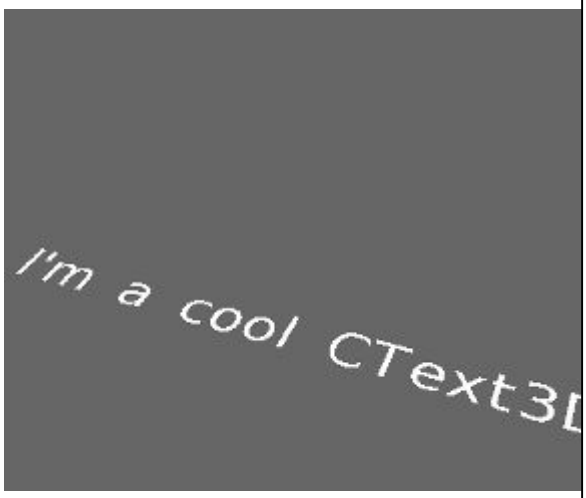


mrpt::opengl::CSphere

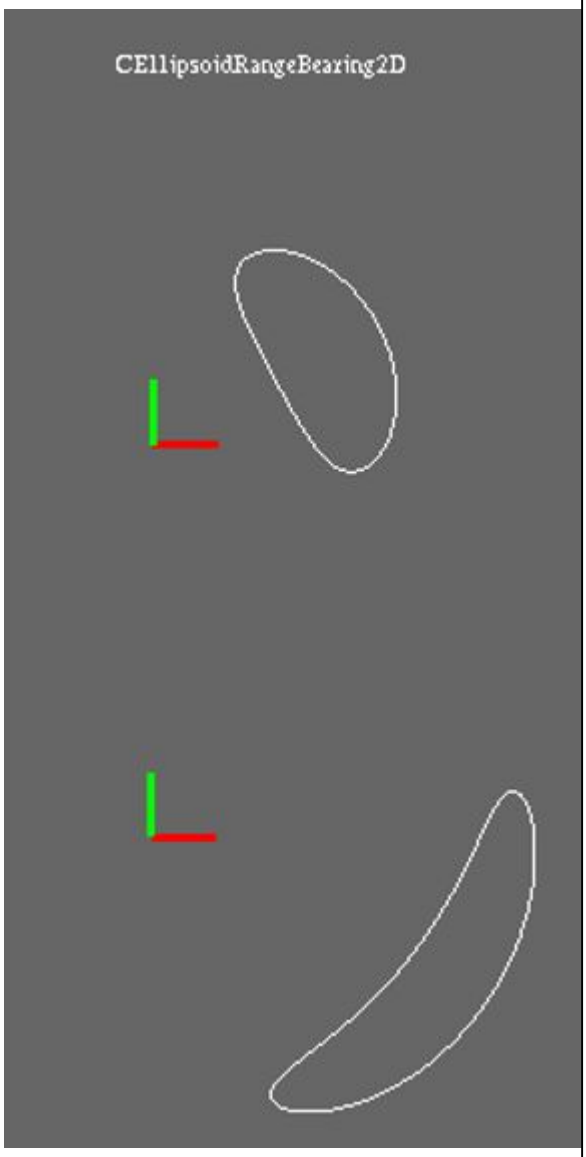


mrpt::opengl::CText

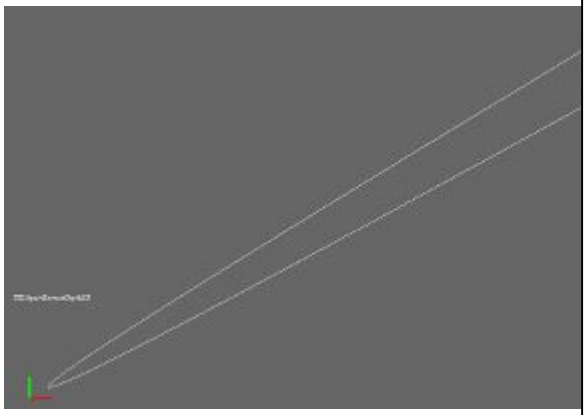
mrpt::opengl::CText3D



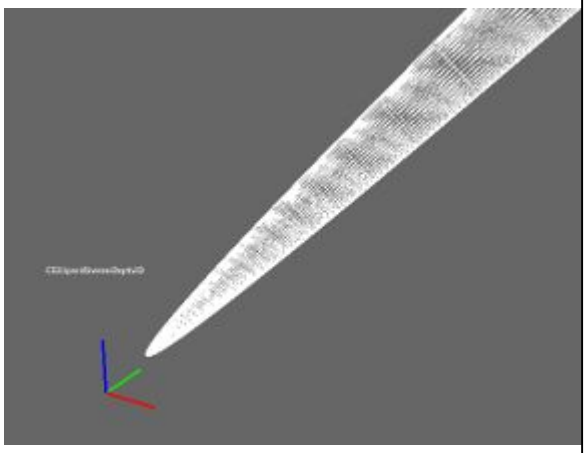
mrpt::opengl::CEllipsoidRangeBearing2D



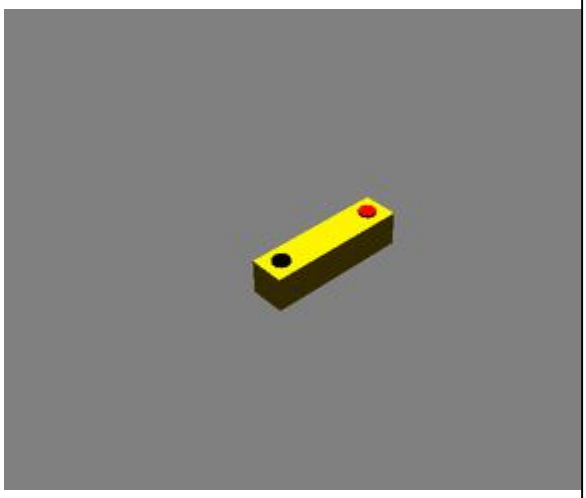
mrpt::opengl::CEllipsoidInverseDepth2D



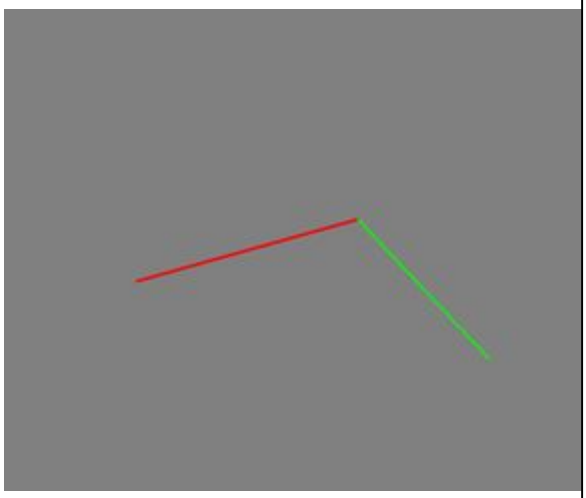
mrpt::opengl::CEllipsoidInverseDepth3D

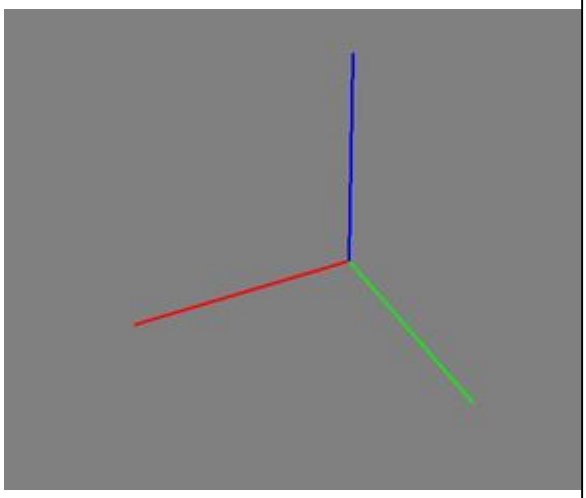
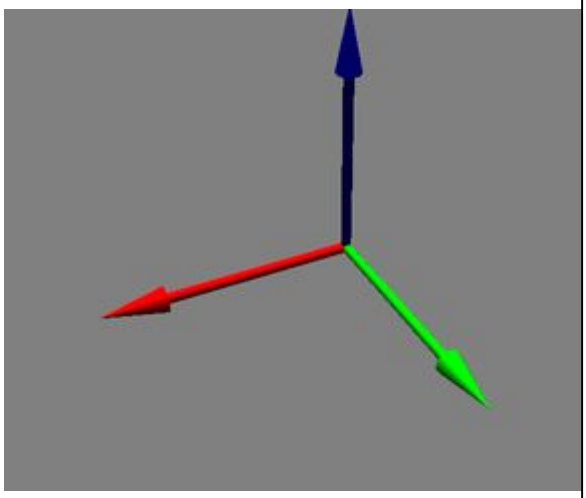


mrpt::opengl::stock_objects::BumblebeeCamera()



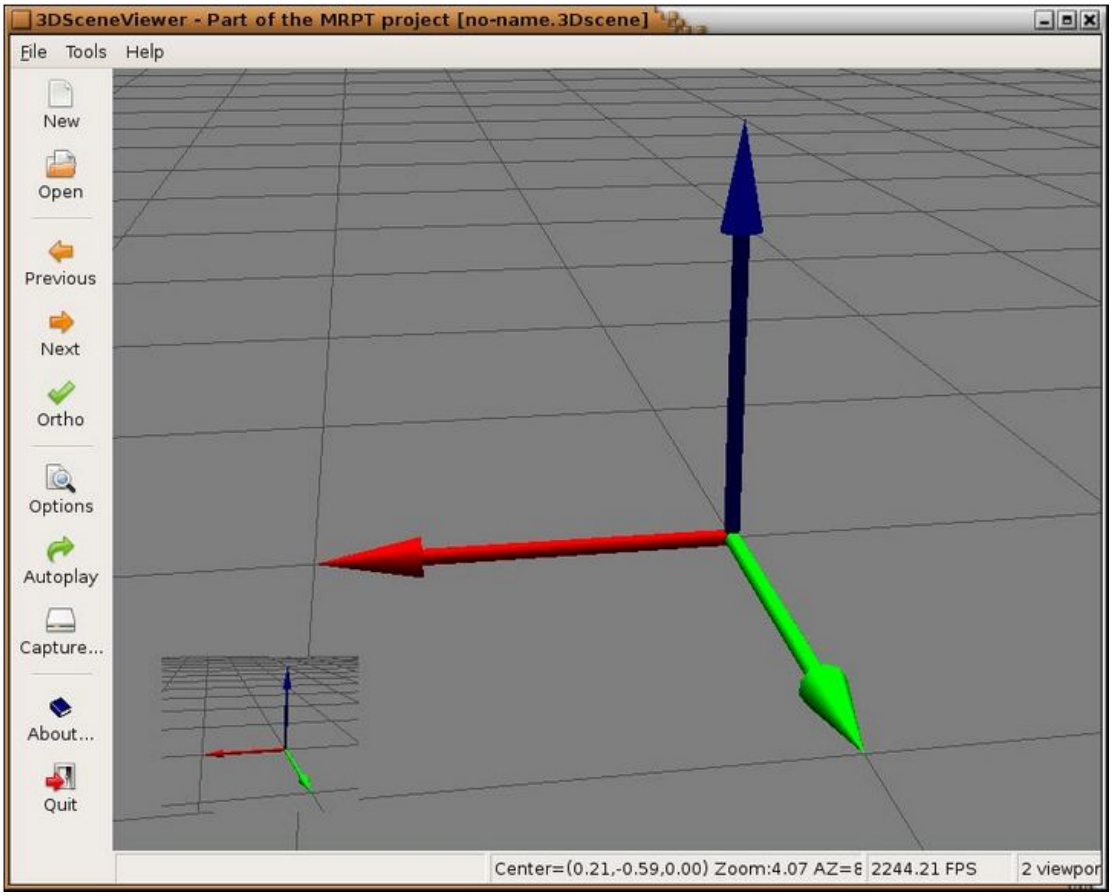
mrpt::opengl::stock_objects::CornerXYSimple()



mrpt::opengl::stock_objects::CornerXYZSimple()	
mrpt::opengl::stock_objects::CornerXYZ()	
mrpt::opengl::stock_objects::RobotPioneer()	

2.1-3 视窗

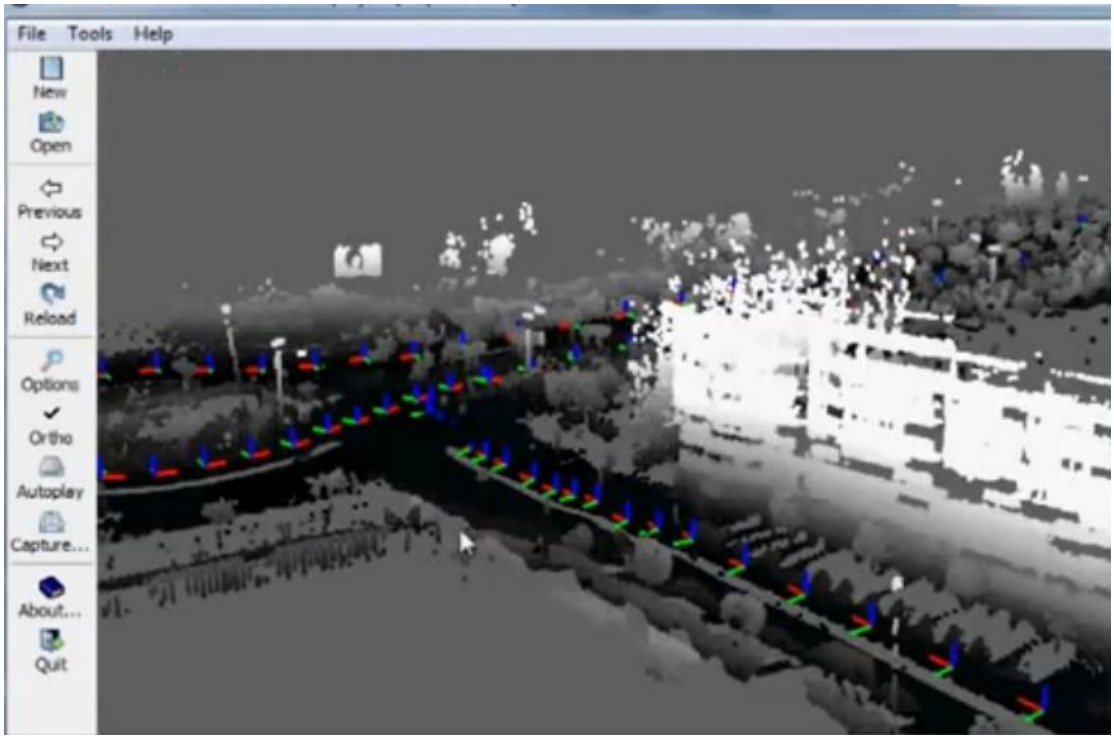
如上所述，每个场景可以包含任意数量的视窗。下面的快照显示了一个有两个视窗的场景: 主视窗扩展到整个窗口，另一个小一点的视窗在左下角。在这个示例中，我们选择让小视窗复制主视窗的内容。



2.2 高效的百万级点云渲染

渲染引擎中已经实现下面三种优化，允许绘制百万级的点云：

- 显示列表。
- 使用八叉树（octrees）以避免绘制超出屏幕的点云。
- 对每个可见的八叉树节点(一个 3d 立方体)，如果每平方像素的点密度高于给定的阈值，就进行样本抽取。（具体见：mrpt::global_settings::OCTREE_RENDER_MAX_DENSITY_POINTS_PER_SQPIXEL)。



2.3 对 Stanford 3D 模型文件格式（PLY）的支持

MRPT 应用软件 SceneViewer3D 已经内置了支持 PLY 格式文件的导入导出功能。在软件菜单“File->Import/Export”中。

类 API 中的 PLY 支持：

- mrpt::utils::PLY_Importer 和 mrpt::utils::PLY_Exporter: 查看这些虚基类的派生类，看哪个类允许加载/保存 PLY 文件。
- 注意：MRPT 使用来自 Diego Nehab 的 C++美化版和 STL 美化版的 RPly 库。

其它有用链接：

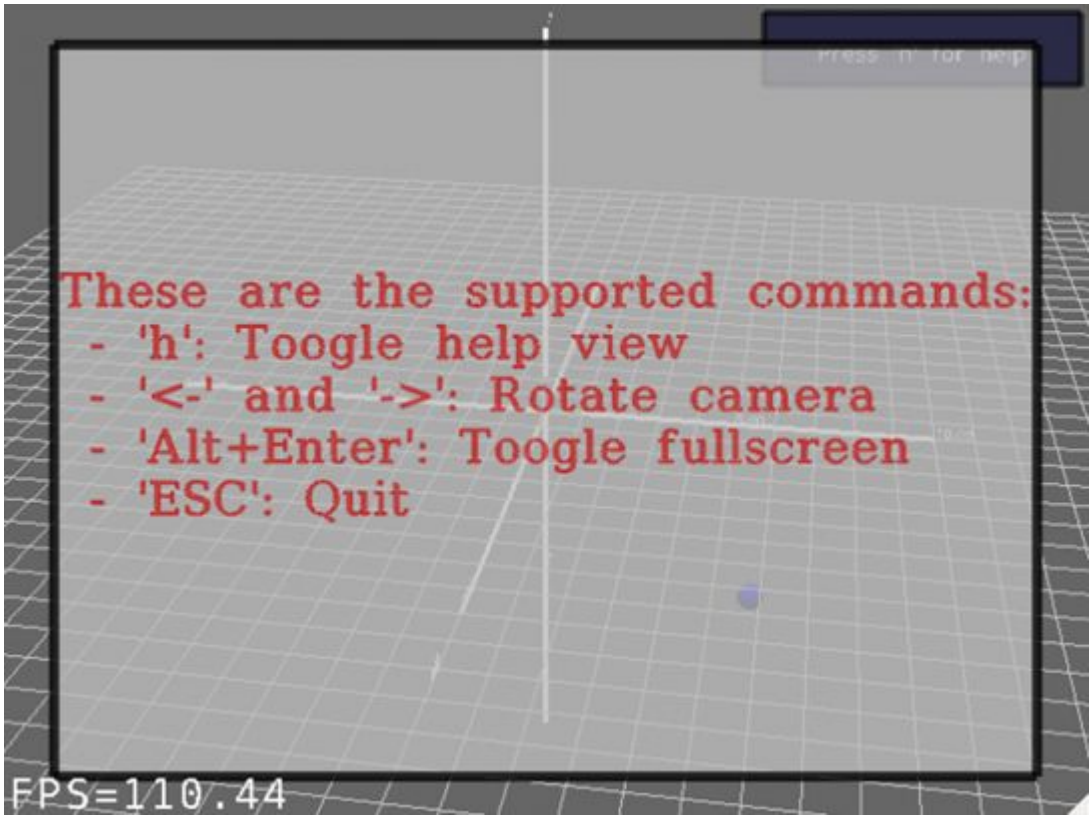
The Stanford 3D Scanning Repository

2.4 消息框（渲染为矢量 OpenGL 图形）

命名空间 mrpt::opengl::gl_utils 内部的函数 renderMessageBox()，用于在 OpenGL 渲染上下文中绘制消息框。

这个方法是高可配置的，并可以包含透明度、一行或多行文本，以及根据提供的字体自动调整大小，比如使文本不超出框界。

对于更多细节，参考示例: {MRPT_DIR}/samples/display3D_custom_render。



2.5 在 3D（OpenGL）窗口渲染视频

在 OpenGL 窗口(比如 MRPT's mrpt::gui::CDisplayWindow3D)中渲染图像的最有效的方式就是将视频帧传入到视频内存区并作为 texture of quads 渲染它们。

可以在 3D 场景中用类 mrpt::opengl::CTexturedPlane 的对象方法手动实现。

从 MRPT 0.9.4 开始有更简单的方式：视窗“图片模式”（"image mode"）。参考 mrpt::opengl::COpenGLViewport 文档 获取更多的信息。

你可以查看示例/samples/opengl_video_viewport_demo 的源代码。示例中生成两个窗口: 一个在整个窗口渲染视频，另一个是在 3D 场景中加一个小视窗用于显示视频。

源码文件：

http://mrpt.googlecode.com/svn/trunk/samples/opengl_video_viewport_demo/test.cpp

屏幕截图:

2.6 2D 字体实现

选择 2D 字体：

任何实现 mrpt::utils::CCanvas 的类中，都可以使用 CCanvas::selectTextFont 选择文本字体，然后，使用 Canvas::textOut 绘制文本。最常用的 CCanvas 实现是 CImage。

字体列表：

- "6x13"
- "6x13B" (bold)
- "6x13O" (italic)
- "9x15"
- "9x15B" (bold)
- "10x20"
- "18x18ja" (Japanese, UNICODE character values)

下面的截图来自示例：{MRPT_DIR}/samples/textFonts，参考源代码查看更详细的信息。


Hello World! with font "6x13"
Hello World! with font "6x13B"
Hello World! with font "6x13O"
Hello World! with font "9x15"
Hello World! with font "9x15B"
ＭＲＰＴのフォントは易しいです！
Hello World! with font "10x20"

第三章 图像，图像处理，摄像头模型

3.1 特征探测和跟踪

3.1-1 简介

在 MRPT 中，使用 mrpt::vision::CFeatureList 类来表示一组特征（比如带有 patches, descriptors 的特征），也可以使用轻型结构体 mrpt::vision::TSimpleFeatureList 来表示一组没有描述子（descriptors）的特征。

这两个是在特征探测追踪类中的核心数据类型。比如 mrpt::vision::CFeatureList，它包含一系列独立的特征结构体(mrpt::vision::CFeature 类型），每个特征结构体包含一个 (x,y) 像素坐标、一个 ID、和可选的一个或多个描述子。因此说清特征探测器和它的描述子之间的区别是非常重要的。

查看(mrpt-vision 库中的) 'Features'模块获得更多信息。

3.1-2 探测器&描述子

使用 mrpt::vision::CFeatureExtraction 类实现从一幅图像中探测特征、提取描述子。

用户可以查看 MRPT 应用 track-video-features 了解更多。

3.1-3 跟踪

高效的特征跟踪，参见 mrpt::vision::CGenericFeatureTracker 类。这是一个实现跟踪框架的虚基类，独立于特定的跟踪算法，实现了共用的部分，比如用 FAST 在旧特征丢失的图像区域进行新特征检测、重新计算 KLT 评分、识别误追踪的特征等。

3.1-4 加载/保存为文本文件

一系列 CFeatureList 类型的特征对象可以被保存/加载（到/从）文本文件，用户容易阅读的格式。这些文件也可以从 MATLAB/octave 中加载。

详见函数 mrpt::vision::CFeatureList::loadFromTextFile() 和 mrpt::vision::CFeatureList::saveToTextFile()。

这种文件的内容示例如下:

```
% Dump of mrpt::vision::CFeatureList. Each line format is:
% ID TYPE X Y ORIENTATION SCALE TRACK_STATUS RESPONSE HAS_SIFT [SIFT] HAS_SURF [SURF]
% \---------------------- feature ------------------/ \--------- descriptors -------/
% with:
%   TYPE   : The used detector: 0:KLT, 1: Harris, 2: BCD, 3: SIFT, 4: SURF, 5: Beacon, 6: FAST
%   HAS_* : 1 if a descriptor of that type is associated to the feature.
%   SIFT   : Present if HAS_SIFT=1: N DESC_0 ... DESC_N-1
%   SURF   : Present if HAS_SURF=1: N DESC_0 ... DESC_N-1
%-------------------------------------------------------------------------------------
    0   0     8.816 202.564     1.42    25.60    5    0.000    1    128       0       0       0       0       0       0
```

```
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0  26   1   0   0   0   0   0   3  52   5   0
0   2   0   0   0   1   0   0   0  12   1   0   0   0   0   0   0   3   0
  0   0 182   3   0   0   0   0   0 182 182  27  16  12   6   0   0 149
10   6  23  30  46   3   0   0   0   0   0   2  15   0   0   0 182  11  10
   0   0   0   0 182 182  57 105  19   0   0   0 102  39  19  36  17   3
5   5  23   0   0   0   1   7   7   2   1   0
        1   0  21.781 205.529   1.41  25.60   5   0.000   1 128   0   0   0   0   0   0
0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0  45   1   0   0   0   0   0   8  81   8   0
0   2   0   0   2   1   0   0   0  17   2   0   0   0   0   0   0   4   0
  0   0 180   3   0   0   0   0   0 180 180  29  20  15   5   0   0 153
9   6  28  34  40   3   0   1   0   0   0   3  14   0   0   0 180  13  13
  1   0   0   0 180 173  65 116  19   0   0   0  79  49  21  31  12   3
6   6  25   0   0   0   0   7   8   2   1   0
    ...
```

3.2 高效的非畸变图像序列

问题

正如此处网页所解释，所有摄像头都会在图像中引入某种径向和切向畸变。为了去掉畸变并希望以"理想的投影模型"来恢复图像，必须对图像的每个像素执行畸变校正。

MRPT 大部分计算机视觉算法基于 OpenCV。在 OpenCV 中畸变校正包括两步：

- 准备一个像素坐标映射图，即“转换函数”。
- 使用映射图对像素进行重映射(cvRemap)。

在大多数情况下，如果我们希望校正一个图像序列，比如，来自一个视频文件或者一个摄像头。比较幼稚的办法是对每一帧图像都调用 cvUndistort2( ) （相当于 MRPT'sCImage::rectifyImage），因为对每一帧都需要重新计算导致这种办法效率非常低。

一个更高效的办法是调用 OpenCV 的 cvInitUndistortRectifyMap( )函数计算一次映射图，然后对每一帧执行重映射 cvRemap( )操作。

然而，这些函数要求一些临时的矩阵变量，因为 OpenCV 的矩阵（尤其 2.1.0 之前的版本）对于 C++开发者不是很直观的数据结构，所以在 MRPT 中提供了一个帮助类（helper class）让用户更容易使用：mrpt::vision::CUndistortMap。

见下面的使用示例：

```
mrpt::vision::CUndistortMap undistortMap;
mrpt::utils::TCamera cam_params;
while (true)
{
 CImage new_img = ... // grab image from wherever.
```

```
  // Only the first time, prepare undistort map:
  if (!undistortMap.isSet())
     undistortMap.setFromCamParams(cam_params);

  undistortMap.undistort(new_img); // In place
  // now you can work on the undistorted image: ...
}
```

3.3 使用 MRPT 校准多种摄像头

MRPT 提供以下工具用于校准摄像头：

- camera-calib 应用: 用于校准独立的摄像头：USB 摄像头、IP 摄像头或 1394 摄像头、Kinect RGB 或 IR 通道 (独立地), 等。
- 完整的 Kinect 校准应用: 用于准确地修复深度+RGB 信息。
- stereo-calib-gui 应用: 用于校准立体摄像头。

3.4 立体图像校正

理论

相关理论请查看 Willow Garage's OpenCV 文档 cv::stereoRectify()部分。

实践

查看 mrpt::vision::CStereoRectifyMap 类的相关文档。

示例代码：

```
mrpt::vision::CStereoRectifyMap      rectify_map;
// Set options as desired:
// rectify_map.setAlpha(...);
// rectify_map.enableBothCentersCoincide(...);

while (true) {
   // Grab stereo observation from wherever
   mrpt::slam::CObservationStereoImagesPtr obs_stereo = ...

   // Only once, construct the rectification maps:
   if (!rectify_map.isSet())
      rectify_map.setFromCamParams(*obs_stereo);

   // Rectify in place:
   unmap.rectify(*obs_stereo);
   // Rectified images are now in: obs_stereo->imageLeft & obs_stereo->imageRight
}
```

完整的示例在目录<MRPT>/samples/stereoRectify 。

第四章 地图(定位，SLAM，建图)

4.1 Graph-SLAM 地图

4.1-1 文本文件.graph 格式

这种文件格式（据我所知）曾首次用于公开软件 TORO 上，并自从那以后开始出现在 OpenSLAM.org 发布的程序和其它库中。

Graphs 在 2D(x,y,phi)或 3D(x,y,z,yaw,pitch,roll)都是支持的。文件包含逐行解释的纯文本，每一行第一个字决定了该条目的类型。

已有下列条目类型：

- **VERTEX2: A 2D node**

```
VERTEX2 id x y phi
```

id 是每个节点的唯一识别符。Phi 单位是弧度。

- **VERTEX3: A 3D node**

```
VERTEX3 id x y z roll pitch yaw
```

id 是每个节点的唯一识别符。yaw,pitch 和 roll 单位是弧度。

- **EDGE2: A 2D pose constraint**

```
EDGE2 from_id to_id Ax Ay Aphi inf_xx inf_xy inf_yy inf_pp inf_xp inf_yp
```

- **EDGE3: A 3D pose constraint**

```
EDGE3 from_id to_id Ax Ay Az Aroll Apitch Ayaw inf_11 inf_12 .. inf_16 inf_22 .. inf_66
```

- **EQUIV: A "topological loop closure"，强制合并两个节点**

```
EQUIV id1 id2
```

4.1-2 二进制文件.graphbin 格式

如果磁盘读写性能有限，则可以使用这种格式。它存储了相应对象的直接二进制串行化数据。即：

```
// Write:
CNetworkOfPoses2D    graph;
CFileGZOutputStream("my_graph.graphbin") << graph;
```

```
// Read:
CNetworkOfPoses2D   graph;
CFileGZInputStream("my_graph.graphbin") >> graph;
```

4.1-3 相关 C++代码

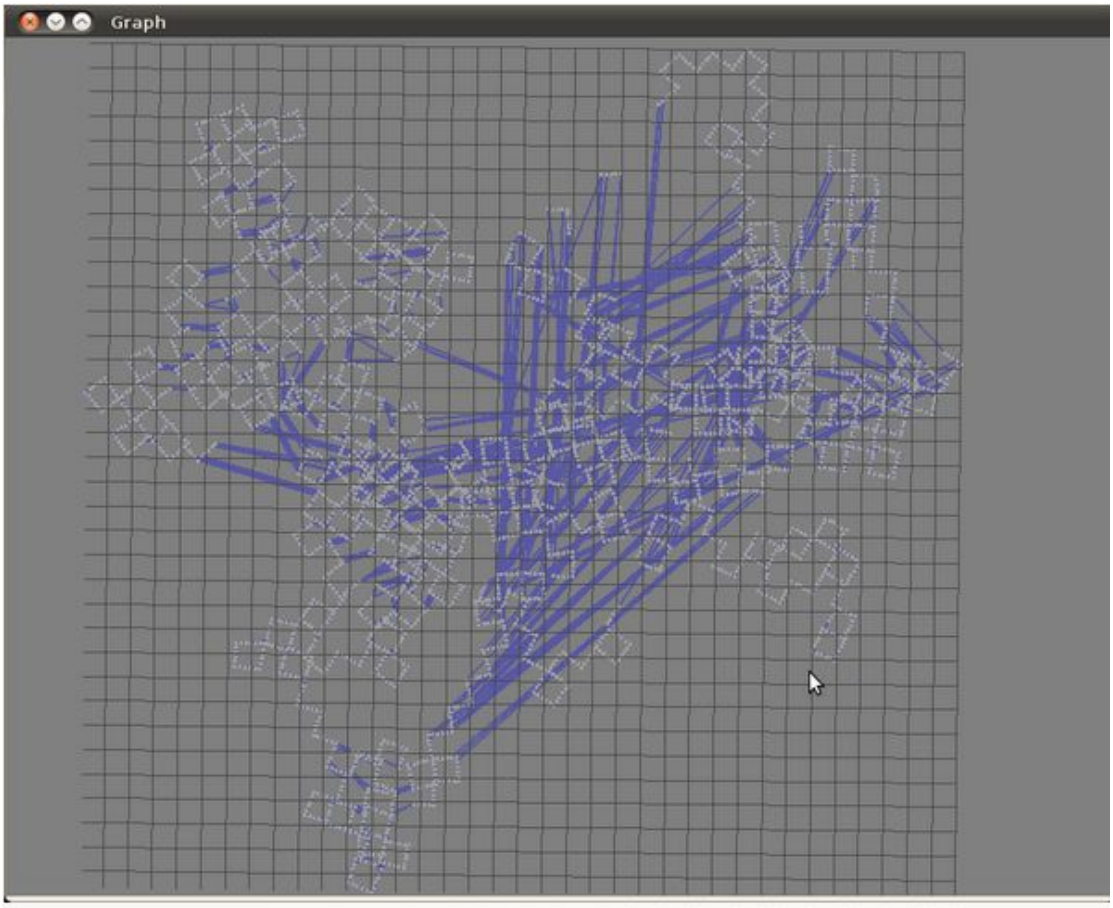
- 应用程序 graph-slam.
- 应用示例 graph_slam_demo
- 通常模板类（mrpt::graphs::CNetworkOfPoses<>）提供了加载/保存的功能，支持上面提到的文本和二进制文件格式。有信息矩阵的 2D 和 3D Graph 这种最常见情况一般都有 Typedefs。

相关库：

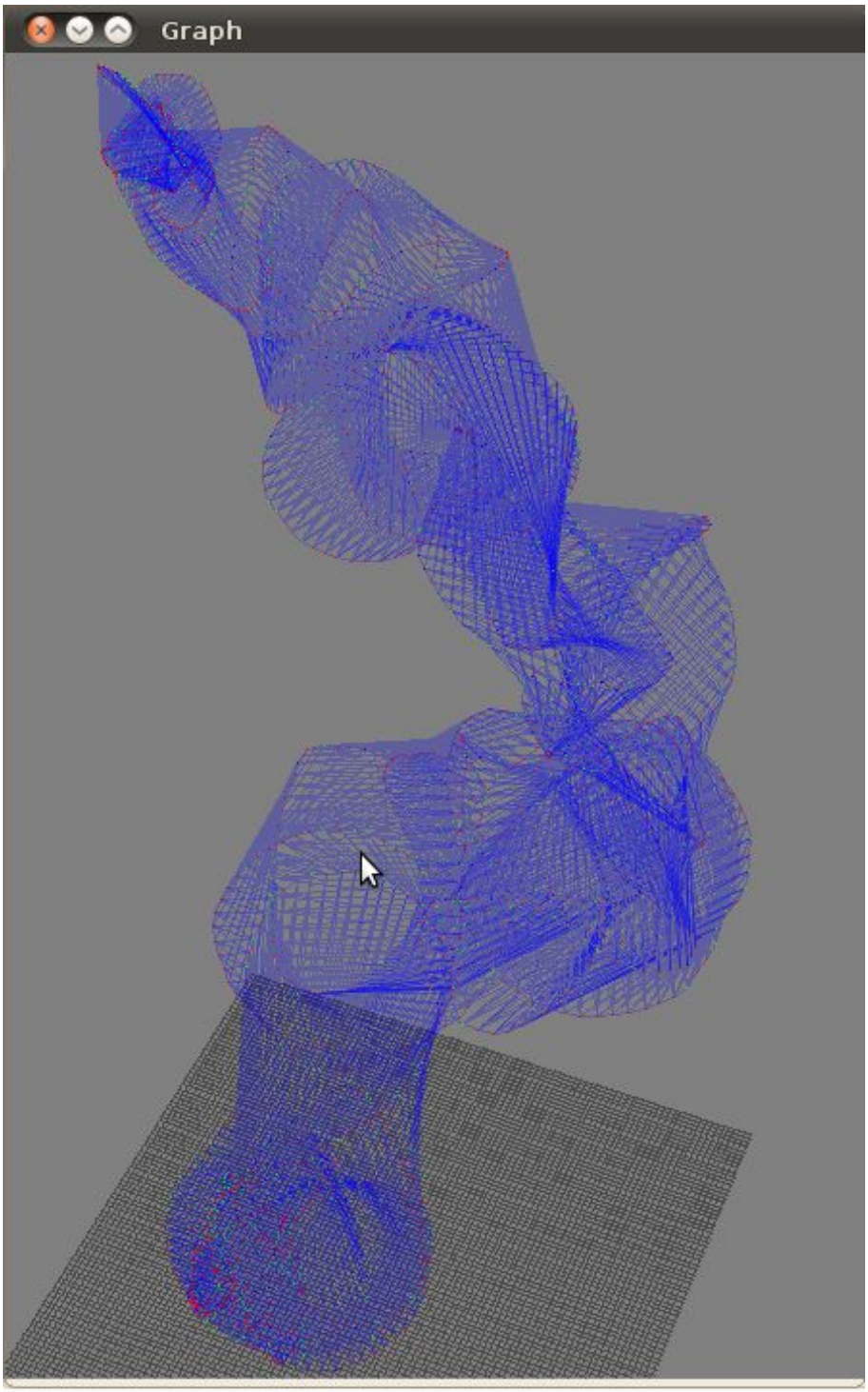
- mrpt-graphs 包含抽象图形数据类型。
- mrpt-graphslam 包含 graph-slam 算法（Levenberg-Marquardt 有限优化图，等）

4.1-4 Graph 地图示例

下面的 graph-slam 图已使用上面提到的 C++类生成，且.graph 文件使用 TORO 发布到 OpenSLAM.org：

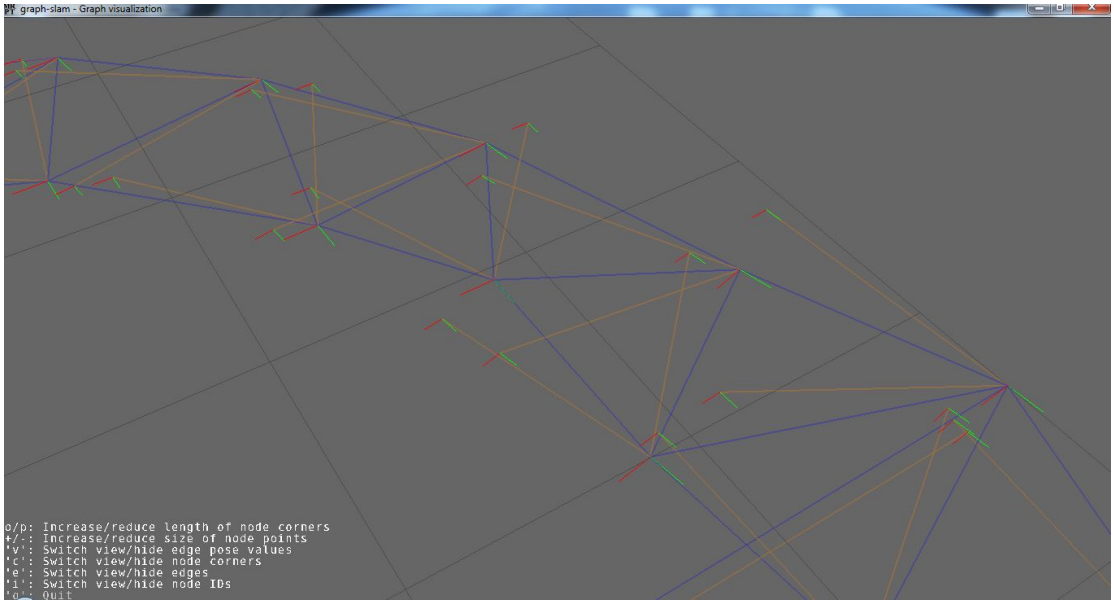


2D graph (raw odometry poses)



3D graph (sphere, raw odometry poses)

下面这张图展示用 graph-slam 如何编程(用 mrpt::opengl::graph_tools 内的平均函数) 展示节点位姿和边缘约束的不匹配。



4.2 占据栅格地图

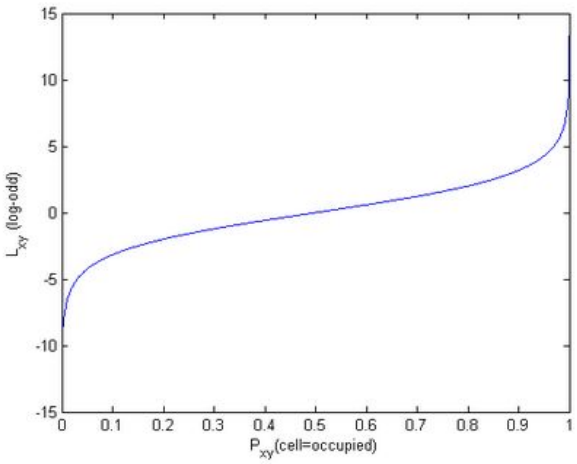
4.2-1 理论基础

。。。待写。。。

4.2-2 一种高效实现

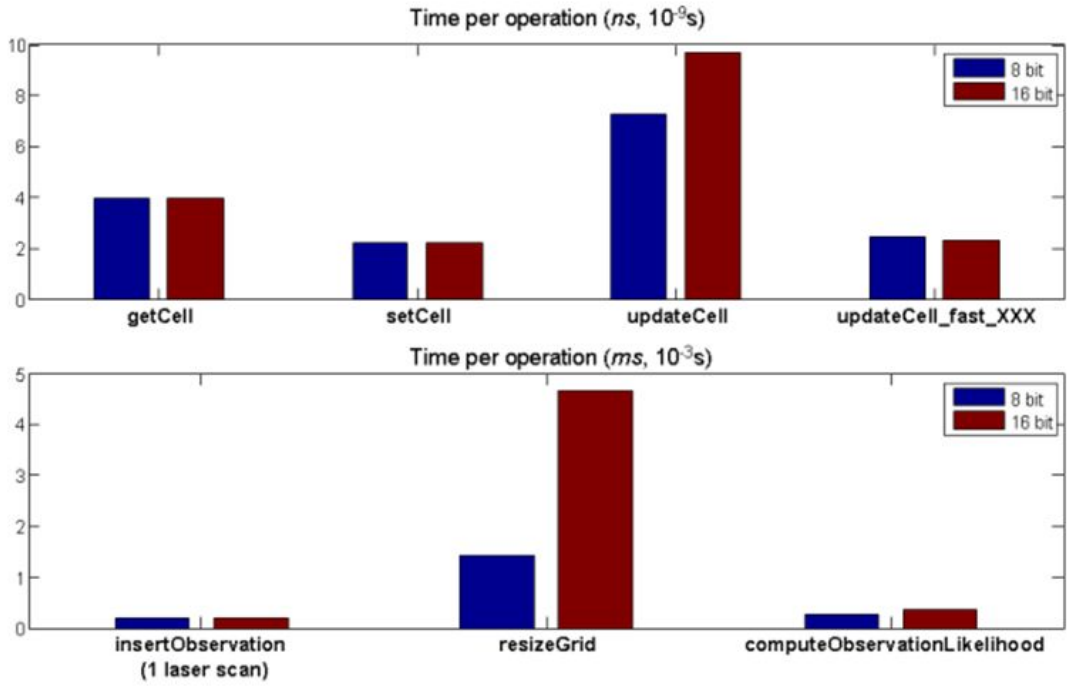
地图表示：

用于贝叶斯整合的 Log-odds 理论方法是通过将每个单元离散为 8 位实现的。



基准：

下一张图表总结了在栅格地图中最常用的操作。使用"samples/benchmarkGridmaps"程序生成的这个结果，基于 Intel Core 2 Duo 2.2Ghz, MRPT_V0.5.5。千次均值。



关于次数的一些解释:

- 激光每秒嵌入:5680/5160(8 bit/ 16 bit)。
- 每秒似然计算:3770/2810(8 bit/ 16 bit), method=LF_thrun,无抽取。
- 这些结果基于单元尺寸为 R=0.04m (4cm)。大部分操作复杂度为 O(1/R^2)。
- 8-bit 和 16-bit 的表示在所有的操作上有非常相似的表现，除了调整大小操作。这是占用更多内存的效果，将导致更多高速内存丢失。
- 除非你的应用对于每个栅格单元都需要相当精确的概率值，一般使用 8-bit。
- 如果一些方法中调用到 updateCell，可以用使用 updateCell_fast_XXX 的方法代替。

4.2-3 已实现传感器模型（似然观测模型）

通过 COccupancyGridMap2D::TLikelihoodOptions 的成员变量可以选择似然函数。查看 doxygen 文档获得更多详情。下面仅提供每个方法的基本描述：

Beam model (lmRayTracing)

。。。待写。。。

Likelihood Field (lmLikelihoodField_Thrun)

。。。待写。。。

Consensus (lmConsensus)

。。。待写。。。

4.2-4 已实现操作

（4.1） 插入激光扫描数据

插入激光扫描数据有两种方式：有扩展光束、无扩展光束。

（4.2） 似然观测

。。。。。

（4.3） 保存/载入/转换

。。。。。。

（4.4） 从图片文件载入

使用 COccupancyGridMap2D::loadFromBitmapFile 可以从任何图片格式(png,bmp)中加载 bitmap。你需要设定每个像素的尺寸（单位：米），可选设定地图中心在图片上的(x,y)坐标(默认为 0,0)。

。。。。。。

（4.5） 保存为图片文件

使用 COccupancyGridMap2D::saveMetricMapRepresentationToFile，至少会生成两个文件：一个是表示栅格地图的图片，另一个是带限制坐标（x,y）的文档文件。通过这种方式，很容易在 Matlab 导入栅格地图。

。。。。。。

（4.6） 载入/保存 为 MRPT 二进制文件

可以使用 MRPT 标准的 CSerializable 接口实现载入或保存一个栅格地图为二进制文件。

```
COccupancyGridMap2D gridmap;
CFileInputStream in_s("file.gridmap");
in_s >> gridmap;
```

```
CFileOutputStream out_s("another_file.gridmap");
out_s << gridmap;
```

（4.7） 熵

。。。。。。

（4.8） Voronoi 图

。。。。。。

4.3 度量地图层次模型

4.3-1 关于 MRPT 中的地图

所有度量地图都有一个通用接口以便易于多态性和泛型编程。参考 C++类 CMetricMap 查看这些通用接口。所有地图类都在命名空间 mrpt::slam 中，这里就不再多说了。

也可查看库 mrpt-maps 的文档。

甚至为了易于实现通用算法，MRPT 有一个非常重要的地图类型：**multi-metric map**。

这个类提供了一种标准度量地图的接口，它内部可以管理任意数量的派生度量地图。为了实现这种方法的强大功能及易用性，可以想象，编程实现将 3D 激光测距仪的扫描数据插入到 3D 点云地图(因此，点云地图被递增建立)。

只需把点云地图换为多度量地图，我们现在就可以在创建点云地图的同时，选择创建三个不同高度的占据栅格地图，而原始代码不需要任何修改。

这就是 MRPT 地图被称为“分层”模型的原因，意味着一个地图("多度量地图") 将调用传递到所有“子地图”。

4.3-2 地图们

（2.1） 多度量地图 Multi-metric map

See the reference for CMultiMetricMap.

（2.2） 信标地图 Beacon maps

一种基于有 ID 的 3D 信标的地图，用于范围定位和 SLAM。

See the reference for CBeaconMap.

（2.3） 2D 气体浓度地图

一种平面的气体浓度点阵，用于气体浓度建图。

See the reference for CGasConcentrationGridMap2D.

（2.4） 2D 高度（海拔）地图

一种点阵，每个单元记录了平方区域内探测点的海拔均值（Z 坐标）。

See the reference for CHeightGridMap2D.

（2.5） 路标地图 Landmark maps

一组有 ID 和位置高斯分布的 3D 路标，主要用于 visual SLAM。

See the reference for CLandmarksMap.

（2.6） 占据栅格地图 Occupancy grid maps

一种平面占据栅格地图。查看网页获取更多关于它的实现和操作的详情。它被用在很多 SLAM 以及基于粒子滤波的定位程序中。

See the reference for COccupancyGridMap2D。

（2.7） 点云地图 Point maps

用于 2D 或 3D 点地图的虚类。它实现了高效的基于 KD 树的查表法。（见 CPointsMap::kdTreeClosestPoint3D）。

点云地图使用 mrpt::math::KDTreeCapable 提供高效的基于 KD 树的 2D 或 3D 点搜索，并计划实现四叉树（QuadTrees）和八叉树（OctTrees）。

See the reference for CPointsMap。

（2.7.1） 简单点云地图

每个点只有（x,y,z)坐标的一种点云地图。

See the reference for CSimplePointsMap

（2.7.2） 有色点云地图

每个点有（x,y,z）坐标和 RGB 颜色数据的一种点云地图。

See the reference for CColouredPointsMap

4.3-3 多度量地图的配置

通常情况下，所有配置多度量地图的参数都可以从类似 ini 的配置文件中加载(或者其他任何文本输入，例如在 GUI 输入框中输入)。当然，也可以写死在代码中。

核心结构体是 TSetOfMetricMapInitializers.

配置文件的格式在 TSetOfMetricMapInitializers::loadFromConfigFile 中有解释并及时更新。

配置文件的示例可参考 MRPT/share/mrpt/config_files/* 目录下的.ini 文件。[用于 icp-slam, rbpf-slam 的配置文件示例]

```
/** Loads the configuration for the set of internal maps from a textual definition in an INI-like file.
* The format of the ini file is defined in utils::CConfigFile. The list of maps and their options
* will be loaded from a handle of sections:
*
* \code
* []
* ; Creation of maps:
* occupancyGrid_count=
* gasGrid_count=
* landmarksMap_count=
* beaconMap_count=
* pointsMap_count=
* heightMap_count=
* colourPointsMap_count=
*
* ; Selection of map for likelihood: (fuseAll=-1, occGrid=0, points=1,landmarks=2,gasGrid=3,4=landmarks SOG, 5=beacon map, 6=height map)
* likelihoodMapSelection=[-1, 6]
*
* ; Enables (1) / Disables (0) insertion into specific maps:
* enableInsertion_pointsMap=
* enableInsertion_landmarksMap=
* enableInsertion_gridMaps=
* enableInsertion_gasGridMaps=
* enableInsertion_beaconMap=
* enableInsertion_heightMap=
* enableInsertion_colourPointsMap=
.........
.........
```

第五章 数学和几何

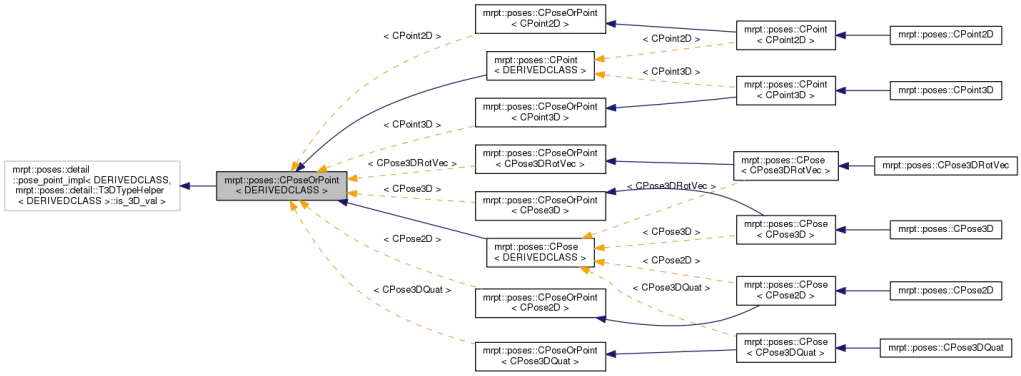
5.1 2D_3D_几何

5.1-1 2D/3D Point、Pose、齐次坐标

使用齐次坐标是构成 2D 或 3D 平移旋转的最简单的一种方式。本页描述了在 MRPT 库中的一组用于 2D/3D 几何计算的 C++类，它们内部依赖这些矩阵。

5.1-2 MRPT 中的空间几何类

所有 2D/3D point 类和 pose 类的基类是 poses::CPoseOrPoint:



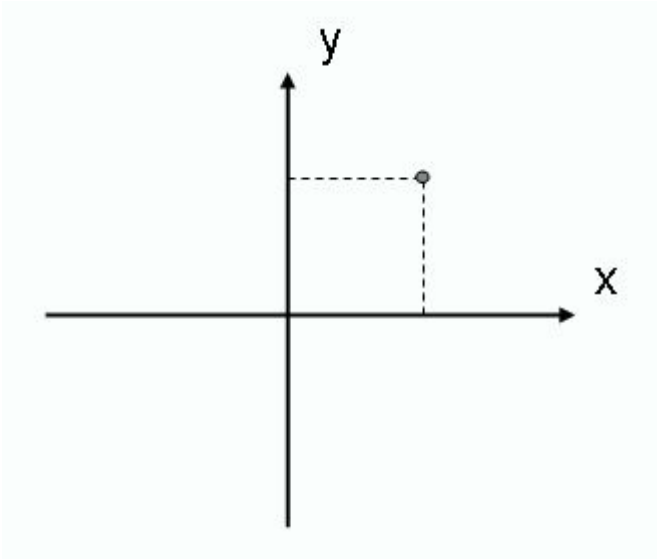
这个类和那些直接继承于它的类（CPose and CPoint）都是抽象类，所以你不能直接创建它们的实例。反而，用户应当经常使用的是 CPoint2D, CPoint3D, CPose2D, CPose3D 和 CPose3DQuat,这些类。下面分别介绍。

注意这些类只能表示数值，而另一组类（网页）则可以描述几何关系的概率密度函数

（2.1） CPoint2D: (x,y)

这个类用于表示 2D 点（即默认 z 坐标为 0）：

齐次变换矩阵	空间表示

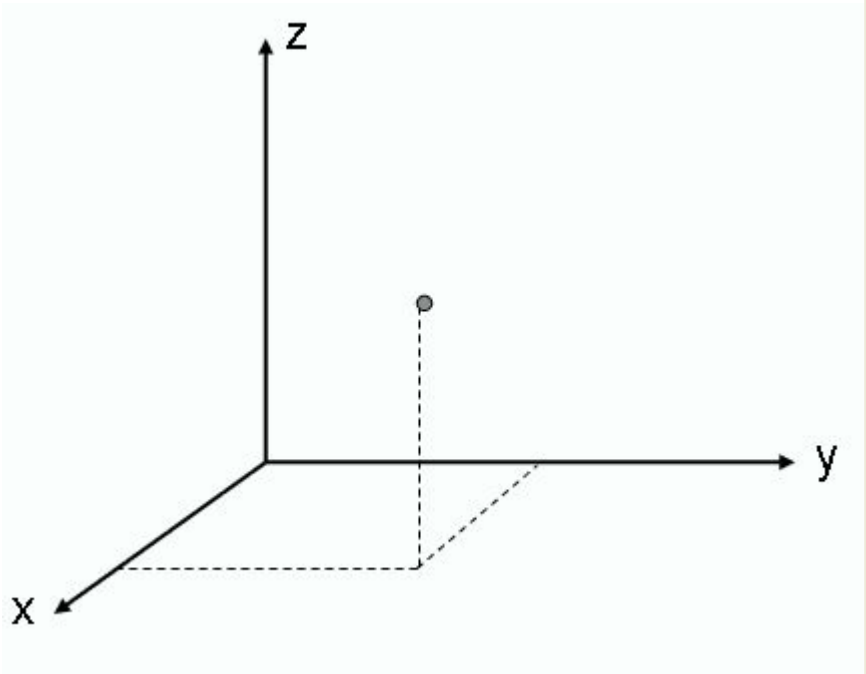
$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

创建一个有初始值的 CPoint2D 对象：

```
// Constructor: CPoint2D (float x=0, float y=0)
CPoint2D        p( 3.0f, 2.0f );
```

（2.2） CPoint3D: (x,y,z)

这个类用于表示 3D 点：

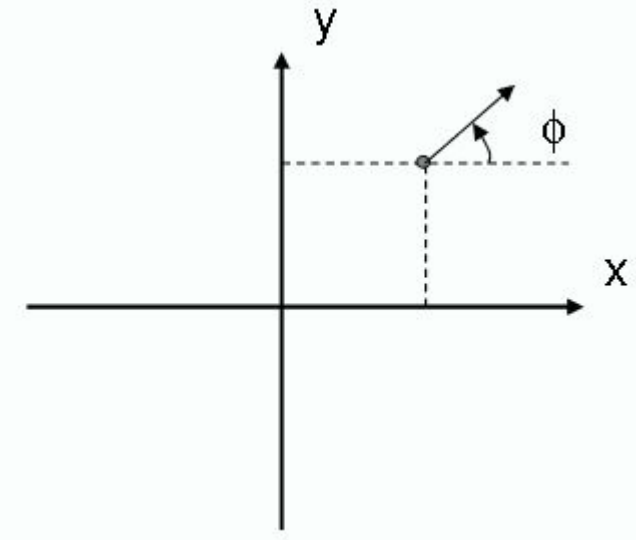
齐次变换矩阵	空间表示
$\begin{bmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

创建一个有初始值的 CPoint3D 对象：

```
// Constructor: CPoint3D (double x=0, double y=0, double z=0)
CPoint3D        p( 3.0, 2.0, 1.0 );
```

（2.3） CPose2D: (x,y,ϕ)

这个类用于表示 2D 位置+航向角（也称为“偏航”或“朝向”）。这是在平坦地面运动的机器人的典型表示。默认 z 坐标为 0。

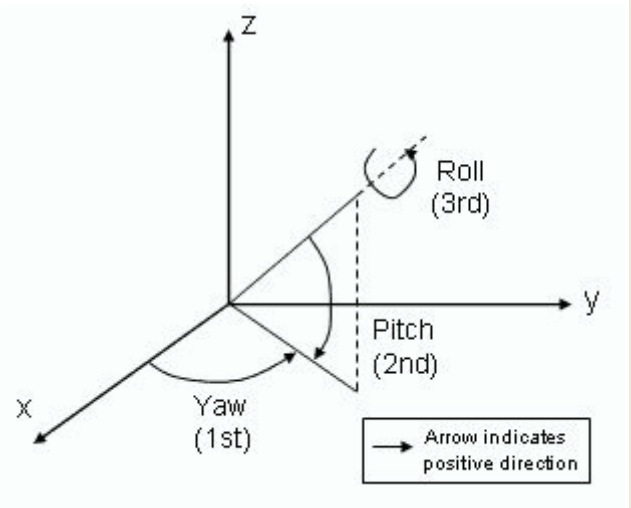
齐次变换矩阵	空间表示
$\begin{bmatrix} \mathrm{Cos}(\mathrm{phi}) & -\mathrm{Sin}(\mathrm{phi}) & 0 & x \\ \mathrm{Sin}(\mathrm{phi}) & \mathrm{Cos}(\mathrm{phi}) & 0 & y \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

创建一个有初始值的 CPose2D 对象：

```
// Constructor:    CPose2D (double x=0,double y=0, double phi=0)
CPose2D         p( 1.0, 0.0, DEG2RAD( 45.0 ) );
```

（2.4） CPose3D: (x,y,z,yaw,pitch,roll)

这个类用于表示一个完整的 6D 位姿：1 个 3D 位置+3 个姿态角：偏航角(yaw)、俯仰角(pitch)、横滚角(roll)。在下图中查看定义。（请注意正俯仰角(pitch)位于 XY 平面下方）

齐次变换矩阵	空间表示
$\begin{bmatrix} CyCp & CySpSr-SyCr & CySpCr+SySr & x \\ SyCp & SySpSr+CyCr & SySpCr-CySr & y \\ -Sp & CpSr & CpCr & z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	

上面矩阵中的缩写解释如下：

Cy 表示 Cos(yaw)； Cp 表示 Cos(pitch)； Cr 表示 Cos(roll)；
Sy 表示 Sin(yaw)； Sp 表示 Sin(pitch)； Sr 表示 Sin(roll)；

创建一个有初始值的 CPose3D 对象：

```
// Constructor:  CPose3D (double x=0, double y=0, double z=0, double yaw=0, float double =0, float double =0)
CPose3D  p( 1.0, 2.0, 3.0, DEG2RAD( 45.0 ), DEG2RAD( -90.0 ), DEG2RAD( 180.0 ) );
```

（2.5） CPose3DQuat: 3D 变换+四元数(x,y,z, qr,qx,qy,qz)

这个类用于表示一个完整的 6D 位姿：1 个 3D 位置(x,y,z)+1 个 3D 旋转四元数(qr,qx,qy,qz)

$$\begin{bmatrix} a^2+b^2-c^2-d^2 & 2bc-2ad & 2bd+2ac \\ 2bc+2ad & a^2-b^2+c^2-d^2 & 2cd-2ab \\ 2bd-2ac & 2cd+2ab & a^2-b^2-c^2+d^2 \end{bmatrix}$$

创建一个有初始值的 CPose3DQuat 对象：

```
// Constructor:
CPose3DQuat  p1 = CPose3D( 1.0, 2.0, 3.0, DEG2RAD( 45.0 ), DEG2RAD( -90.0 ), DEG2RAD( 180.0 ) );
CPose3DQuat  p2(x,y,z, CQuaternionDouble(1,0,0,0)  );
```

5.1-3 3D 旋转矩阵的推导

用于任意 yaw/pitch/roll 旋转的 3x3 变换矩阵（也称为方向余弦矩阵，DCM）可以从独立旋转中推导出来：

$$RzRyRx = \begin{bmatrix} CyCp & CySpSr - SyCr & CySpCr + SySr \\ SyCp & SySpSr + CyCr & SySpCr - CySr \\ -Sp & CpSr & CpCr \end{bmatrix}$$

独立旋转变换：

$$R\_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(roll) & -sin(roll \\ 0 & sin(roll) & cos(roll) \end{bmatrix}$$

$$R\_y = \begin{bmatrix} \cos(pitch) & 0 & \sin(pith) \\ 0 & 1 & 0 \\ -\sin(pitch) & 0 & cos(pitch) \end{bmatrix}$$

$$R\_z = \begin{bmatrix} \cos(yaw) & -\sin(yaw) & 0 \\ \sin(yaw) & \cos(yaw) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

在这里我们使用了一个定理，即相对后继变换轴的旋转可以通过右侧矩阵乘的方式实现。旋转顺序是先 Rz，然后 Ry，最后 Rx。更多关于矩阵乘的顺序的信息，请查看 Wikipedia 的 Euler angles -> DCM 页面。

更多讨论在 http://mathworld.wolfram.com/EulerAngles.html

5.1-4 类之间转换

任何 pose 类或 point 类的对象都可以互相转换，并在转换时尽可能保留所有信息。大多数转换都可以隐式实现，比如：

```
CPose2D      p1;
CPose3D      p2 = p1;   // Ok: p2.x
```

但是，当实施转换的源类信息比目标类信息要多时，用户必须自己实现，比如下面的示例。这个规则（从 MRPT 0.6.5 开始）为了尽量避免信息的意外损失。

```
CPose2D      pose;
CPoint3D     pnt;
CPose3D      pose6D;

pose6D = pnt;        // Ok.
pose6D = pose;       // Ok.
pose   = pose6D;   // **ERROR** Doesn't compile
pose   = CPose2D(pose6D);   // Ok. pitch,roll and z are lost.
```

5.1-5 常用运算

（5.1） 位姿合并/逆合并

位姿合并是指 将(pose or point) B【以(pose or point) A 为参考坐标系】转换到 A 的参考坐标系下，得到一个新的(pose or point) C。使用 MRPT C++库中的类可以将此过程简化为：

```
C = A + B;
```

该操作自然会考虑到所有需要的旋转变换（通过齐次矩阵相乘）。它的逆过程，得到在全局坐标系下的 C 相对于 A 的参考坐标 B 的操作可以使用减号(-)操作符：

```
B = C - A;
```

Points , Poses, 2D and 3D 全部可以自由混合这些操作。下表总结了所有输入组合和预期输出：

Does "a+b" return a Pose or a Point?

a \ b	**Pose**	**Point**
Pose	Pose	Point
Point	Pose	Point

Does "a-b" return a Pose or a Point?

a \ b	**Pose**	**Point**
Pose	Pose	Pose
Point	Point	Point

Does "a+b"(and "a-b") return a 2D or 3D object?

a \ b	2D	3D
2D	2D	3D
3D	2D	3D

（5.2） 计算距离

有一些适用于任何 point or pose 的距离相关的函数：

当前对象到任何其它 point or pose 的距离（如果是计算 2D 对象相对于 3D 对象的距离，函数会考虑到 z 坐标）：

```
float        distanceTo (const CPoseOrPoint &b) const;
```

当前对象到任何其它 point or pose 的距离平方：

```
float        sqrDistanceTo (const CPoseOrPoint &b) const;
```

当前对象到任何用户指定的 2D 或 3D 坐标：

```
float     distance2DTo (float ax, float ay) const;
float     distance3DTo (float ax, float ay, float az) const;
```

还有他们的距离平方：

```
float     distance2DTo (float ax, float ay) const;
float     distance3DTo (float ax, float ay, float az) const;
```

查看 poses::CPoseOrPoint 的 doxygen 参考文档获得更多详情。

（5.3） 计算范数

方法：

```
double    norm ()    const;
```

返回这个点的欧几里得范数：

$$\| p \| = \sqrt{x^2 + y^2 + z^2}$$

（5.4） 输出到控制台

所有 point/pose 类都可以使用操作符 << 输出到控制台。

5.1-6 四元数

3D 旋转的另一种表示形式是四元数。MRPT 通过模板类 mrpt::math::CQuaternion 提供支持。

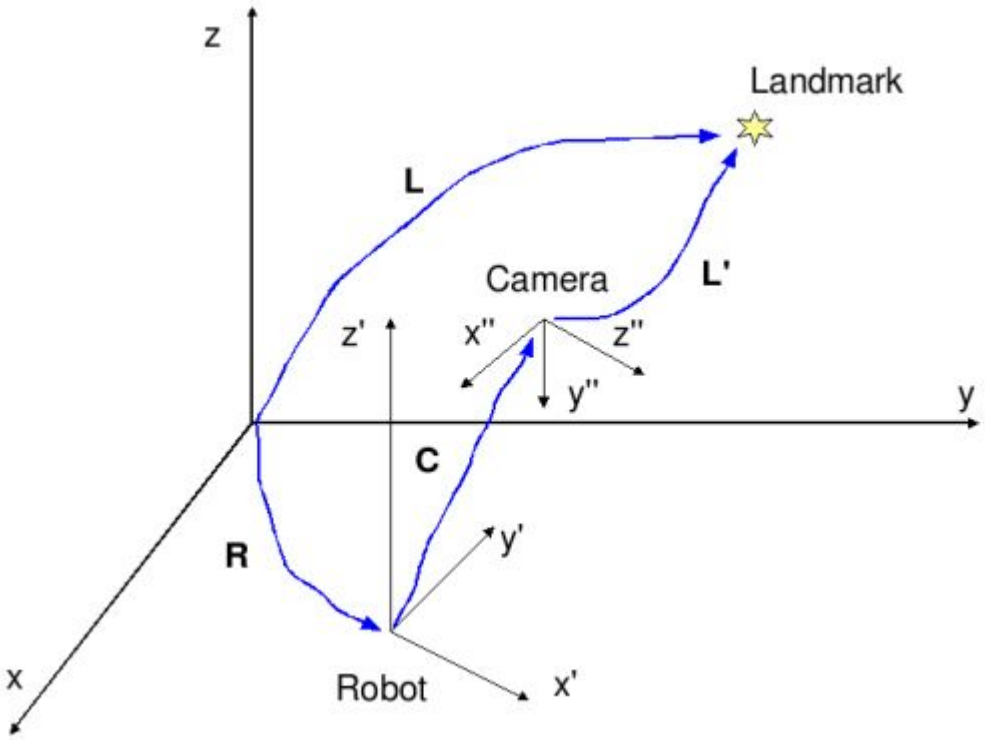
MRPT 中已经有从 CPose3D 转换四元数的方法，但注意这种转换只是纯旋转（没有平移）。

关于完整的有四元数的 6D 位姿，查看类 CPose3DQuat。

```
CPose3D    p(0,0,0, DEG2RAD(20),DEG2RAD(45),DEG2RAD(-80));
CQuaternionDouble q;
p.getAsQuaternion(q);     // Transform a 3D-pose into a quaternion.
```

5.1-7 实例：移动机器人摄像头坐标变换

这个例子设定一个有已知的 2D 位姿(x,y,phi)R 的移动机器人。它内置一个摄像头，摄像头有相对机器人坐标系的 6D 位姿 C。我们要计算摄像头探测到的全局路标的 3D 位置(x,y,z)，其中 L 表示路标的绝对坐标，L’表示路标相对于摄像头的坐标。见下图所示：



此时，摄像头观察到的路标位置是：

$$L' = L - (R + C)$$

然后，我们检查这种计算是否正确：将摄像头的观测结果重映射到绝对坐标系，它必须与原始绝对坐标相等：

$$R + C + L' == L$$

作为一个位姿重建的示例，考虑摄像头相对于机器人的位姿：

```
// Camera pose relative to the robot: 6D (x,y,z,yaw,pitch,roll).
CPose3D    C(0.5,1.0,1.5 ,DEG2RAD(-90.0),DEG2RAD(0),DEG2RAD(-90.0)    );
```

前三个参数是摄像头的 x,y,z 坐标. 其余是 yaw=-90º, pitch=0º, roll=-90º。这个位姿建立了一个顺+Z 轴朝向前方的摄像头(参考上图所示)

整个示例的源代码可以在 MRPT/samples/geometry3D/中找到。

程序输出结果：

```
L: (0,4,2)
 R: (2.000,1.000,45.00deg)
 C: (x,y,z,yaw,pitch,roll)=(0.5,0.5,1.5,-90deg,0deg,-90deg)
 R+C:(x,y,z,yaw,pitch,roll)=(2,1.70711,1.5,-45deg,-0deg,-90deg)
 L': (-3.03553,-0.5,0.207106)
 R(+)C(+)L' = (-1.23207e-07,4,2)
 Should be equal to L = (0,4,2)
 |(R(+)C)-L|= 3.0834
 |L-(R(+)C)|= 3.0834
```

5.2 Levenberg-Marquardt 算法（数字雅可比矩阵）

5.3 矩阵、向量、数组和线性代数：MRPT 和 Eigen 类

5.3-1 线性代数类：Eigen

MRPT 使用功能强大的 Eigen C++ 库 (API 的版本 3)，用于管理数值型的矩阵和向量，因此如果你对 Eigen 熟悉的话，你已经了解了 MRPT 的大部分类、接口和方法！

如果你不知道 Eigen，这些是很好的起点：

- 入门教程
- Eigen 内容参考表

与标准 Eigen API 主要的不同是 MRPT 加入了很多新的类型定义和方法扩展了 Eigen::Matrix<>，扩展了更多可用于任何向量或矩阵的容易使用的操作，并顺便提供了基于旧版的 non-Eigen MRPT 矩阵 APIs 的代码的平滑过渡。

你可以查看 Eigen::MatrixBase 的文档，其中既包含了 Eigen 原始方法，也包含有 MRPT 基于插件机制提供的方法(寻找名称为“MRPT plugin:……”的部分)。

5.3-2 MRPT 特有类

对于新代码，最好是直接使用 Eigen::Matrix<>(或者任何 Eigen typedef: MatrixXd, MatrixXf, 等)而不是用 MRPT 的（矩阵）类型。因为所有的 MRPT（矩阵）类型都是 Eigen::Matrix<>的实例（面向对象编程思想里的“继承于”），只是使用了不同的模板参数。

如下表总结：

MRPT 类型	继承自...	内存布局
mrpt::dynamicsize_vector < T > 是下面这两个类的基类: ◆ mrpt::vector_float ◆ mrpt::vector_double	Eigen::Matrix< T,Dynamic,1 >	Auto (→ ColMajor) (列优先存储)
mrpt::math::CArrayNumeric < T,LEN >	Eigen::Matrix< T,N,1 >	Auto (→ ColMajor) (列优先存储)
mrpt::math::CMatrixTemplateNumeric < T > 类型定义： ◆ mrpt::math::CMatrixDouble ◆ mrpt::math::CMatrixFloat 兼容二进制串行化的继承类： ◆ mrpt::math::CMatrix (float elements) ◆ mrpt::math::CMatrixD (double elements)	Eigen::Matrix T,Dynamic,Dynamic >	RowMajor (行优先存储)

mrpt::math::CMatrixFixedNumeric < T, NROWS, NCOLS >	Eigen::Matrix< T,NROWS, NCOLS >	仅当 NCOLS=1 时，为 ColMajor。其它都为 RowMajor
◆ mrpt::vector_signed_byte ◆ mrpt::vector_signed_word ◆ mrpt::vector_int ◆ mrpt::vector_long ◆ mrpt::vector_size_t ◆ mrpt::vector_byte ◆ mrpt::vector_word ◆ mrpt::vector_uint	这些类不是基于 Eigen，而是继承于 STL std::vector<>	元素顺序存储在内存
mrpt::math::CArray<T,SIZE>	不是基于 Eigen,而是设计用于处理数值型或非数值型元素	元素顺序存储在内存

继承于 Eigen::Matrix 的 MRPT 类，与父类有一些差异，在下面一节解释。

5.3-3 MRPT 矩阵类和 Eigen 类之间的差异

- 注意上面表格中的内存布局是很重要的，因为之前的 MRPT 类都是 RowMajor（行优先存储），而 Eigen 的默认是 ColMajor（列优先存储）。通常，用户不会注意到这些差别， 一些特殊情况除外。例如，下面的代码预期是矩阵的构造函数以 RowMajor（行优先存储）顺序从数组中读取数据(因此"2" 在第一行，第二列结尾，等)：

```
const double nums[] = { 1,2,3, 4,5,6 };
mrpt::math::CMatrixFixedNumeric M(nums);
std::cout << M;
```

- MRPT 矩阵的默认构造函数设置所有的参数为 0，在 Eigen 中默认值是未定义的。 在不同的应用中两种方式都是有用的。
- 将一个 MRPT 矩阵输出到 std::cout 会被自动追加一个换行，而 Eigen 类的光标仍将保持在最后。所以下面这两个打印命令将产生相同的输出：

```
mrpt::math::CMatrixTemplateNumeric M1 = ...
Eigen::MatrixXd M2 = M1;
std::cout << M1; // MRPT matrix: automatically append new line char.
std::cout << M2 << std::endl; // Eigen matrix: need to add new line char manually.
```

5.3-4 常见错误（及解决方案）

（4.1）编译时对齐问题

固定大小的向量化 Eigen 对象对内存对齐有特殊要求。为了概括这些类以及任何包含这种矩阵的类或结构体(在任何层次深度!)，我会用“问题类”(problematic classes)这个名字表示它们（“问题”表示“需要特别注意它们的对齐问题”）。

在 MRPT 中有很多受到影响的类(CMatrixFixedNumeric, CPose3D, 等), 你自己的类也很有可能受到影响, 因为一个类中即使只有一个固定大小的矩阵也会被标记为“问题”。

通常，编译器自动处理对齐要求，但也有一些情况下，它不会自动处理，你可能会遇到下面的错误：

STL 容器

如果你有以下错误:

```
error C2719: '_Val': formal parameter with __declspec(align('16')) won't be aligned
```

出现在一个 STL 容器附近，就意味着你必须针对容器(container)使用 Eigen 特殊对齐的内存配置器(allocator)。Eigen 的网页详细的解释了这个问题。

简单的，下面这些是必要的更改：

```
Let "TYPE" be any "problematic" class or struct.

// vector
std::vector<TYPE> v;    // ERROR.
std::vector<TYPE, Eigen::aligned_allocator<TYPE> > v;    // OK, or alternatively
mrpt::aligned_containers<TYPE>::vector_t v;    // this version is a short-cut.

// map
std::map<KEY,VALUE> m;    // ERROR.
std::map<KEY,VALUE,        std::less<KEY>,        Eigen::aligned_allocator<std::pair<const
KEY,VALUE>
> > m;    // OK, or alternatively
mrpt::aligned_containers<KEY,VALUE>::map_t    m;    // this version is a short-cut.
```

提示：MRPT 提供了 mrpt::aligned_containers 以简化这种 STL 容器的声明操作。

如果你有像这样的错误:

```
error C2719: 'NAME': formal parameter with __declspec(align('16')) won't be aligned
```

且它不是出现在 STL 容器附近，是因为你想将“问题”类型作为函数参数值。请阅读 Eigen 的网站解释了这种情况。

简单的解决方案是换为传递“引用”：

```
// CPose3D: Is a problematic type since it contains Eigen matrixes of fixed-size.
void myFunc(CPose3D p) // ERROR!
{...}
void myFunc(const CPose3D p)    // ERROR!
{...}
void myFunc(CPose3D &p)    // OK
{...}
void myFunc(const CPose3D &p) // OK
{...}
```

（4.2）运行时对齐问题

如果你发现类似下面这样的运行时错误：

```
Assertion failed: (reinterpret_cast<size_t>(array) & 0xf) == 0 && "this assertion is explained here: " "http://eigen.tuxfamily.org/dox/UnalignedArrayAssert.html " " **** READ THIS WEB PAGE !!! ****", file ...
```

在访问一个类中包含“问题”类型的数据成员时，且类的实例是动态创建(new)：

```
class CoolStuff {
  MyProblematicType m_damnit;
  ...
};
CoolStuff c1; // OK CoolStuff
c2 = new CoolStuff; // ERROR!
```

那么，通过在你的类声明中添加宏 EIGEN_MAKE_ALIGNED_OPERATOR_NEW 的方法很可能解决这个问题：

```
class CoolStuff {
  MyProblematicType m_damnit;
  ...
  public:
   EIGEN_MAKE_ALIGNED_OPERATOR_NEW // Add this, in the "public" scope
};
CoolStuff c1; // OK
CoolStuff c2 = new CoolStuff; // Now, it's OK too.
```

当你使用 new CoolStuff( )的方式创建一个类实例时，一个自定义的内存分配器将会被调用，并总是返回已对齐的内存。

重要提示：所有从 mrpt::utils::CObject 继承的类已经有了定制的 new 操作符，因此如果你的自定义类也是继承于 CObject 或 CSerializable 类，那么就不需要再添加这个宏了。

（4.3）混合数值类型：显式转换

为了避免从一种类型的矩阵/向量不小心转换成另一种类型（比如 double<->float），以免浪费时间，Eigen 会在运行时发生下面的错误：

```
Eigen/src/Core/Assign.h:504: Derived& Eigen::DenseBase::lazyAssign(const Eigen::DenseBase&)
[with OtherDerived = Eigen::Matrix<float,...>, Derived = Eigen::Matrix<double, ...="">]: Assertion `(SameType) &&
"YOU_MIXED_DIFFERENT_NUMERIC_TYPES__YOU_NEED_TO_USE_THE_CAST_METHOD_OF_MATRIXBASE_TO_CAST_NUMERIC_TYPES_EXPLICITLY"' failed.
```

这种错误会在类似下面这种操作时发生：

```
Eigen::Matrix<double,3,3> Md;
Eigen::Matrix<float,3,3> Mf;
Md = Mf; // This causes the error!
```

解决方案是使用 cast( )显式转换：

```
Eigen::Matrix<double,3,3> Md;
Eigen::Matrix<float,3,3> Mf;
Md = Mf.cast(); // Now this is OK
```

（4.4）resize( )与 setSize( )

注意 resize( )是 Eigen 的原生方法，不保存矩阵的旧内容。而 setSize( )是一种 MRPT 方法，新值填 0，保持旧值不变。

参考 Eigen::MatrixBase 文档(包含来自 MRPT 插件的方法)。

5.3-5 从 MRPT 0.9.2 移植代码的问题

除了上面提到的编译或运行时错误 (适用于新旧 Eigen 和基于 MRPT 的代码)，如果你的代码中密集地使用了旧(非 Eigen)MRPT 的向量和矩阵，那么还有下列这些需要注意的要点。

（5.1）MRPT:: vector_float 和 MRPT:: vector_double 问题

类型 mrpt::vector_float 和 std::vector<float> (或者 double 版本) 不能再互相转换了。

（5.2）从位姿(TPose2D,...)显式构造矩阵

例如：以前的代码“CMatrixDouble31 M = myPose2D;”现在不能成立。应该改为：“CMatrixDouble31 M = CMatrixDouble31（myPose2D）;”

（5.3）矩阵元素的迭代器

使用之前 MRPT 类的统一接口实现的向量和矩阵迭代器不能再用了，因为 Eigen 不提供支持。阅读这个网页了解原因。

（5.4）CVectorFloat 和 CVectorDouble

不建议在新代码中使用这些类型。它们现在的同义词是 mrpt::vector_float 和 mrpt::vector_double.

（5.5）矩阵方法 unit( )

Unit( )方法用于设置一个单位矩阵，对操作固定大小和动态大小的矩阵有不同的标识。现在 Eigen::MatrixBase::unit( )统一起来，当然你总是可以直接使用 Eigen 的 setIdentity( )。

5.3-6 高级主题

（6.1）在你的代码中同时包含 MRPT 和 Eigen 头文件

- 首先包含 MRPT 头文件（因为它使用了插件机制...）
- 包含 "mrpt-base" 以确保 Eigen 头文件被包含在内。

- 不能混用版本 2&3。请使用 MRPT 嵌入式版本或 Eigen 版本 3，但不能同时使用。
- MRPT 中已有基于 CMake 机制的用于扩展 Eigen::MatrixBase 的插件。

5.4 RANSAC C++类

5.4-1 RANSAC 算法

MRPT 包括了这种鲁棒模型拟合算法的一种通用 C++实现。关于算法的理论描述，请参见 Wikipedia 的文章。MRPT 的实现主要基于由 Kovesi 实现的优秀的 MATLAB 工具箱。

当前的实现也可以被用于拟合 RANSAC 变异遗传类模型。请参见模板类：mrpt::math::ModelSearch.

5.4-2 C++示例

MRPT 也提供了一些简单的 RANSAC 应用示例。可以在 mrpt::math::RANSAC_template 类中找到通用实现，核心的方法是：

```
static bool mrpt::math::RANSAC_Template< NUMTYPE >::execute (
    const CMatrixTemplateNumeric< NUMTYPE > & data,
    TRansacFitFunctor                          fit_func,
    TRansacDistanceFunctor                     dist_func,
    TRansacDegenerateFunctor                   degen_func,
    const double                               distanceThreshold,
    const unsigned int                         minimumSizeSamplesToFit,
    mrpt::vector_size_t                      & out_best_inliers,
    CMatrixTemplateNumeric< NUMTYPE >          & out_best_model,
    bool                                       verbose = false,
    const double                               prob_good_sample = 0.999,
    const size_t                               maxIter = 2000
    )
```

关于这个类的完整文档，请参阅 MRPT::math:: RANSAC_Template<>。

关于如何使用这个模板化的版本的完整示例可以在/sample 目录下找到：

https://raw.github.com/jlblancoc/mrpt/master/samples/ransac-demo-plane3D/

也有一些其它的应用特定方法（如 N 平面探测器），它是通用模板的实例，用于针对实际情况提供一个更友好的用户接口。

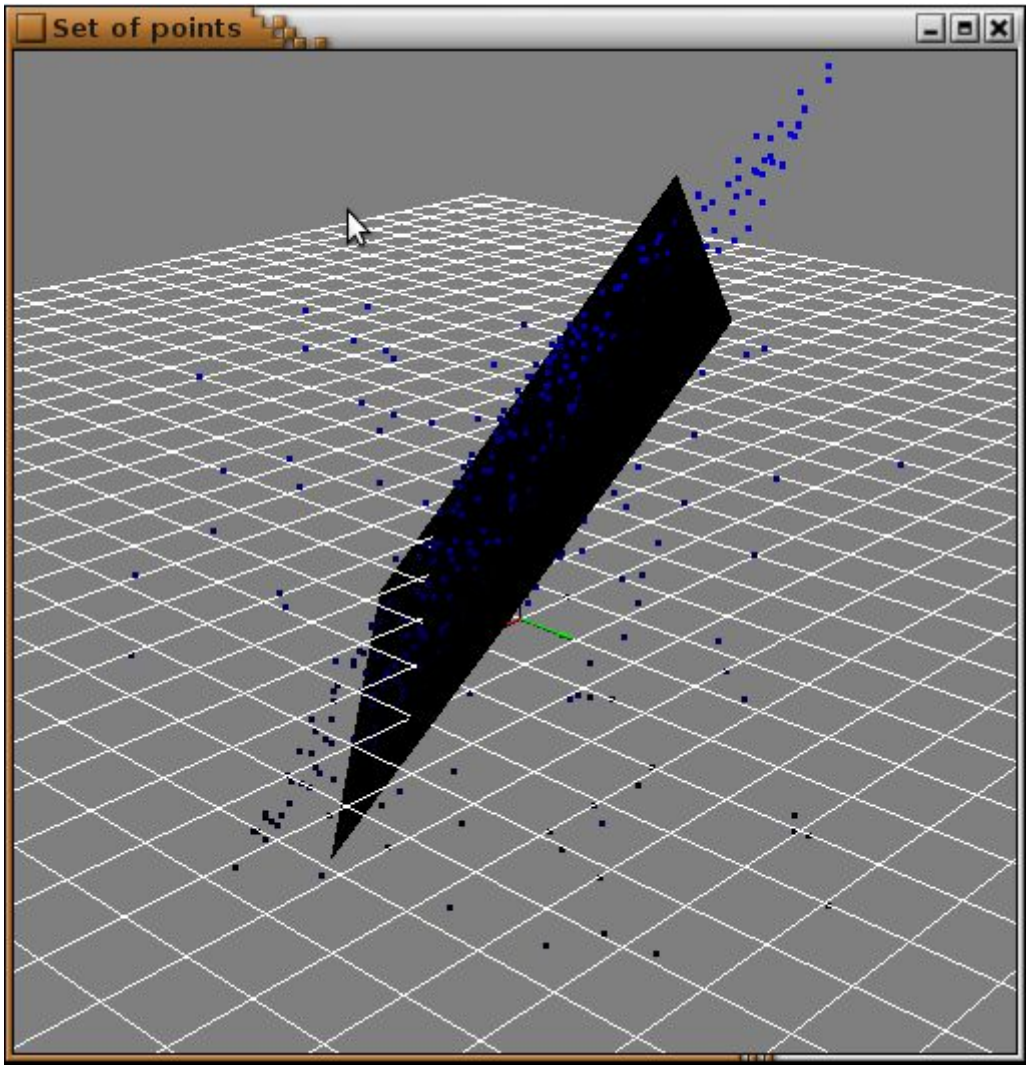
见 mrpt-base 库中的 mrpt::math::ransac_detect_3D_planes 和“RANSAC detectors” 模块部分。

（2.1） 拟合 3D 平面

这个示例的源代码可以在此找到：

https://raw.github.com/jlblancoc/mrpt/master/samples/ransac-demo-plane3D

这是所得到的拟合平面。在 Pentium M @2.0GHz 平台上，对 300 个内点和 100 个外点的平均执行时间是 0.5 毫秒。



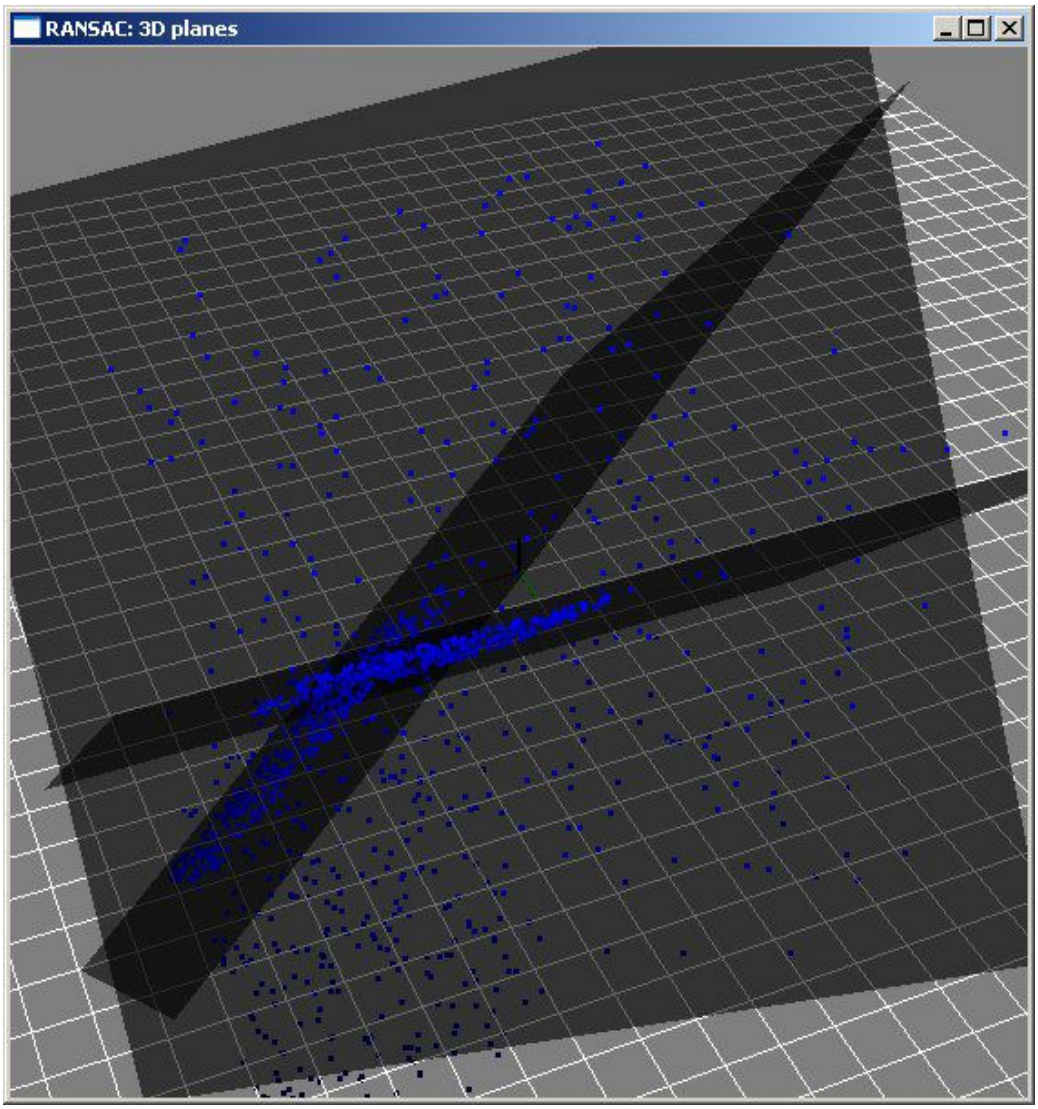
（2.2） 拟合多个 3D 平面

这个示例的源码可以在此找到：

https://raw.github.com/jlblancoc/mrpt/master/samples/ransac-demo-applications.

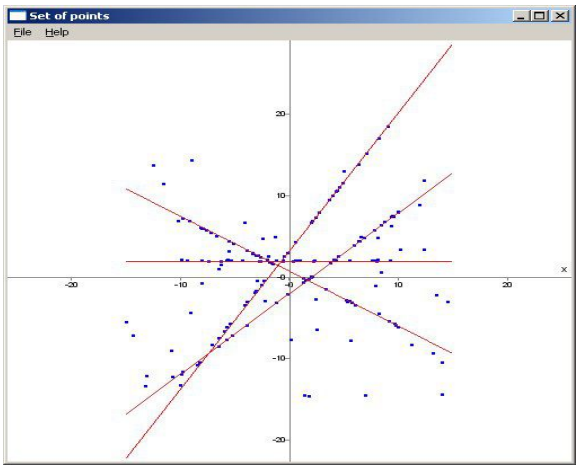
该方法尝试在一片点云中识别出未知数量的平面，需要给出判定某点是否为内点的阈值，以及假设平面有效所需要的内点最小数量。

在 Pentium M@2.0GHz 平台上，对 3 平面*300 内点 和 300 外点的平均执行时间是 60 毫秒。



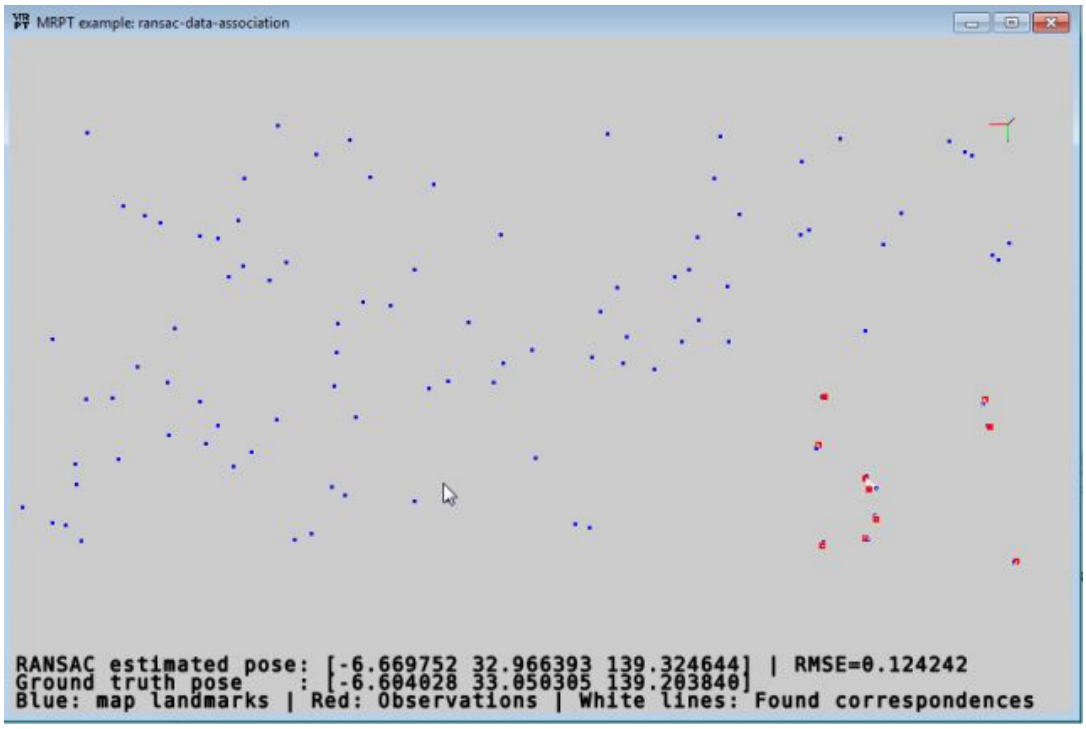
（2.3） 拟合多个 2D 线

这个示例的源码可以在此找到：
https://raw.github.com/jlblancoc/mrpt/master/samples/ransac-demo-applications.
这是线拟合的结果：



（2.4） 基于 RANSAC 的数据关联

请查看示例：Example:ransac-data-association.



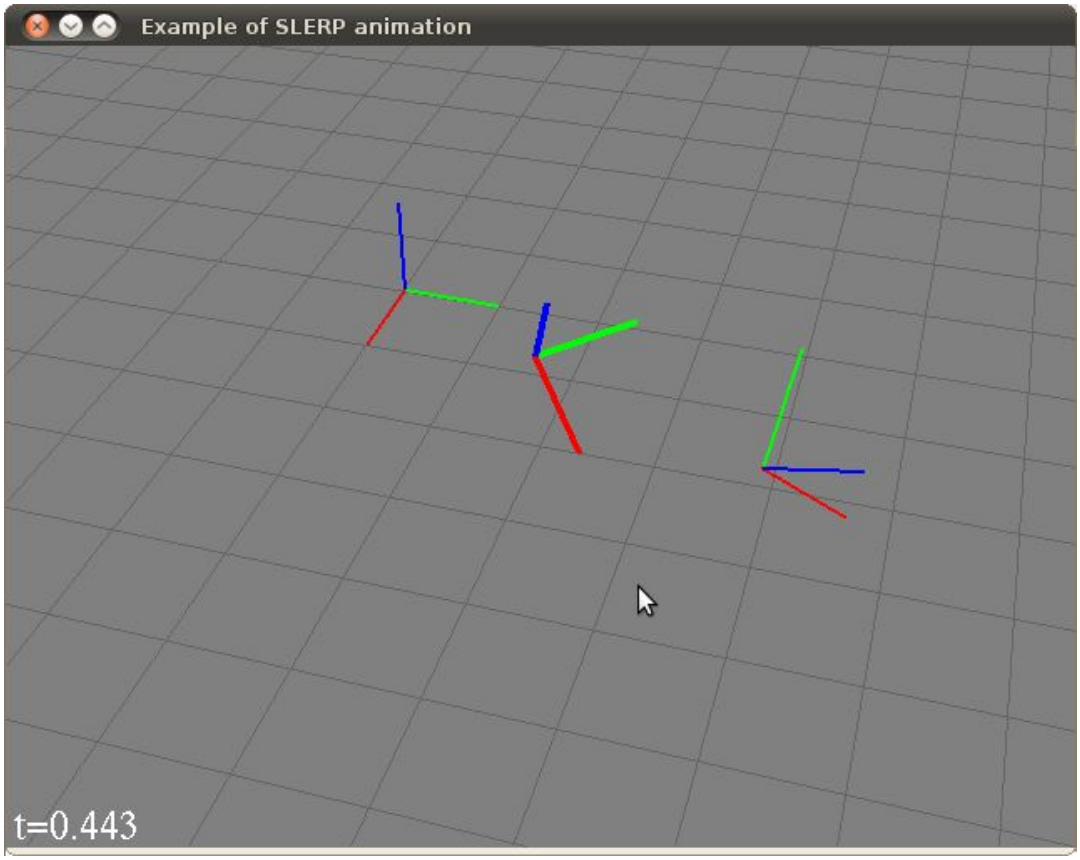
5.5 SLERP 插值

5.5-1 描述

SLERP 是球面线性插值(Spherical Linear Interpolation)，在两个 3D 旋转之间以数学合理的方式插入一条直观的平滑路径，这是一种非常流行的技术。（见 wikipedia 页面）

下面这个图是在给定的两个 3D 位姿之间插入的位姿动画的截图。

（参见示例：/samples/slerp_demo）



5.5-2 C++实现

SLERP 在 MRPT 中实现为 mrpt::math::slerp 函数，这是一个重载函数以支持四元数（纯旋转），也可完成 3D 平移+旋转（CPose3D and CPose3DQuat 类型）。

下面是当前的实现：

```
template <typename T>
void slerp(
        const CQuaternion<T>    & q0,
        const CQuaternion<T>    & q1,
        const double                t,
        CQuaternion<T>              & q)
{
        ASSERTDEB_(t>=0 && t<=1)
//See:http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/slerp/index.htm
        // Angle between q0-q1:
        double cosHalfTheta = q0[0]*q1[0]+q0[1]*q1[1]+q0[2]*q1[2]+q0[3]*q1[3];
        // if qa=qb or qa=-qb then theta = 0 and we can return qa
        if (std::abs(cosHalfTheta) >= 1.0)
        {
                q = q0;
                return;
        }
```

```
        bool reverse_q1 = false;
        if (cosHalfTheta < 0) // Always follow the shortest path
        {
                reverse_q1 = true;
                cosHalfTheta = -cosHalfTheta;
        }
        // Calculate temporary values.
        const double halfTheta = acos(cosHalfTheta);
        const double sinHalfTheta = std::sqrt(1.0 - square(cosHalfTheta));
        // if theta = 180 degrees then result is not fully defined
        // we could rotate around any axis normal to qa or qb
        if (std::abs(sinHalfTheta) < 0.001)
        {
                if (!reverse_q1)
                        for (int i=0;i<4;i++) q[i] = (1-t)*q0[i] + t*q1[i];
                else for (int i=0;i<4;i++) q[i] = (1-t)*q0[i] - t*q1[i];
                return;
        }
        const double A = sin((1-t) * halfTheta)/sinHalfTheta;
        const double B = sin(t*halfTheta)/sinHalfTheta;
        if (!reverse_q1)
                  for (int i=0;i<4;i++) q[i] = A*q0[i] + B*q1[i];
        else for (int i=0;i<4;i++) q[i] = A*q0[i] - B*q1[i];
}
```

5.5-3 参考

- http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternion…
- http://portal.acm.org/citation.cfm?doid=325334.325242

第六章 杂项

6.1 Kinect 和 MRPT

6.1-1 概述

MRPT 实现了一个使用 Xbox Kinect 的通用 C++接口。Kinect 这种新型的 RGB+D 传感器在机器人技术上有巨大的潜力。MRPT 的实现为用户提供了唯一统一的接口。在后台，MRPT 使用下面这些库中的其中一个实际访问传感器：

- 在 Unix / Linux 和 Windows 可使用 OpenKinect 的 libfreenect。
- 在 Windows 上可使用 CL NUI。

强度(RGB)图被自动转换为 OpenCV 的格式（MRPT :: utils:: CImage），10 位或 11 位的原始深度被转换为米，并提供处理两个摄像头的参数矩阵的方法（一个用于 RGB 摄像头和一个用于“范围图 range image”）。

被 MRPT 的 Kinect C++类抓取的观测数据，可以直接同其它数据类型一起保存在 rawlogs 机器人数据集文件中，或者可以在线用于任何实时应用。性能上，在 Linux 和 Windows 下使用 libfreenect 实现了全速 30fps 的速度，但如果在接收大数据量同时有其它需要实时处理的任务，则强烈建议使用多核处理器。

6.1-2 示例代码和文档

- **教程:**
  - 教程：抓取你自己的 Kinect 数据集。
  - 本教程解释了从 RGB+D 图像生成 3D 点云的不同途径。
  - Kinect 校正教程(New: Jun-2012).
- **示例:**
  - 示例应用: kinect_3d_view. 从 Kinect 抓取数据并在线 3D 点云渲染的小演示。
  - kinect-3d-slam: 通过移动 Kinect 构建小型 3D 地图的演示应用程序。
  - Example code ：如何在 Kinect 抓取（在线）和读取先前记录的数据集（离线）之间切换的示例代码。
  - 如何将 Kinect 的 3D 范围扫描结果转化为 2D“假”激光扫描？
- **相关 C++ API:**
  - mrpt::hwdrivers::CKinect: 这个 CKinect 类的 doxygen 文档提供了使用详情。
  - mrpt::slam::CObservation3DRangeScan: 从 CKinect 类得到的观测值数据类型。Doxygen 文档中有说明摄像头轴方向定义的图。

6.1-3 在 Windows 中安装

在 Windows 系统，要在 MRPT 上使能 Xbox Kinect，你需要 OpenKinect 的 libfreenect 或者 CL NUI SDK:http://codelaboratories.com/nui。

在 2011 年 1 月前(MRPT 0.9.3), Windows 系统中的唯一选择是 CL NUI SDK。这个 SDK 的安装比 freenect 简单，但是缺少一些重要的特性，比如抓取 IR 通道 (适当校准时需要), 因此我更愿意推荐 freenect。

（3.1）使用 OpenKinect 的 libfreenect（推荐）

A) 驱动安装(使用 MRPT 预编译包安装或者从源代码编译安装都需要)

1.下载最新的 libusb-win32-bin-x.x.x.x.zip(支持 32 和 64 位 Windows)。MRPT 需要使用里面的一些 DLLs 文件。所以将 libusb-win32-bin-x.x.x.x/bin/{x86 or amd64}添加到你的系统环境变量 PATH 中。

2.下载 libfreenect 源码 (例如: 从 github)。你只需要/platform 目录下的几个文件。

3. 按照下面的"驱动安装" 步骤 (复制于 libfreenect 中用于 Windows 的一步步安装指导):

> **Windows 7：一步一步的安装步骤（应该也适用于 Windows XP）**
>
> - 插上 Kinect。Windows 警告没有发现设备驱动（Kinect 上的 LED 灯不闪烁），如果 windows 提供了一个对话框，要求搜索驱动程序，取消它。
> - 打开设备管理器：开始>>控制面板>>系统与安全>>系统>>设备管理器
> - 在某个地方会发现一个“Xbox NUI Motor”（最有可能在“Other device”下）。在它的图标上带有一个黄色警告标识符“！”。单击右键，并选择“更新驱动程序软件...”,然后单击 “浏览计算机以查找驱动程序软件”。
> - “浏览” 选择 “XBox_NUI_Motor.inf”所在的文件夹(在 libfreenect 源文件夹的 /platform/windows/inf 里）单击“下一步”，如果出现关于安装的非安全驱动，选择始终安装。
> - 之后，Kinect 的 LED 的绿灯开始闪烁。现在需要在设备管理器中增加另外两个设备“Xbox NUI Camera”和“Xbox NUI Audio”，安装它们只需要重复以上步骤。

B) CMake 配置 (只有使用源码编译安装 MRPT 时需要)

注意 MRPT 0.9.4+ 不需要"freenect"源代码, 因为已经包含了它的拷贝。

在 CMake(cmake-gui)配置 MRPT 时，使能 BUILD_KINECT, 点击 "Configure" 并验证 "BUILD_KINECT_USE_FREENECT" 已标记。同时两个新的变量会出现:

- LIBUSB_1_INCLUDE_DIR: 选择 libusb 的头文件夹"/include"的路径。
- LIBUSB_1_LIBRARY: 例如，如果为 MS Visual Studio 2010 配置, 则选择 /lib/msvc/libusb.lib，或者如果你打算使用 mingw, 则使用/lib/gcc/libusb.a 作为 libusb 库路径。

再单击一次“Configure”，应该不会出现错误。

如果在 MSVC 2008 中出现丢失 stdint.h 文件的错误，下载文件 pstdint.h，重命名为 stdint.h 并保存，例如，保存在 OPENKINECT_LIBFREENECT_DIR/include 目录中。你可以用#include <mrpt/utils/mrpt_stdint.h>替代#include <stdint.h>，这是个与编译器版本无关的保证可用文件。

注意在 Windows 系统中的 libusb:

- 如果你的系统是 win32(不是 win64), 至少在目前的 libusb 版本，你必须将 ibusb0_x86.dll 重命名为 libusb0.dll。
- 由于 libusb-win32-bin-1.2.5.0 中将 usb.h 换为了 lusb0_usb.h，这将导致现有代码错误：
  `致命错误 C1083: 无法打开包含的文件: 'usb.h': 没有这样的文件或目录。`

一个非常简单的解决方法是创建一个 libusb-win32-bin-1.x.x.0\include\usb.h ，在里面仅添加一行:

```
#include "lusb0_usb.h"
```

（3.2）使用 CL NUI SDK

下载安装后，当你配置 MRPT 时 CMake 会找到它。如果不能自动识别，使能 BUILD_KINECT，点击"Configure"，标记"BUILD_KINECT_USE_CLNUI"，再 "Configure" ，设置 CLNUI_SDK_DIR 指向 CL NUI 的安装目录。**记住增加 "CL NUI Platform\SDK\Bin" 目录到系统环境变量 PATH 中。**

6.1-4 在 Unix/Linux 中安装

在 Ubuntu 中，安装 libusb-1.0-0-dev：

```
sudo apt-get install libusb-1.0-0-dev
```

OpenKinect 的 libfreenect 目前已被嵌入在 MRPT 源代码中。

访问设备时会要求 root 权限(执行 "sudo")。为了规避这个，安装下面这个文件到 /etc/udev/rules.d/中：

http://mrpt.googlecode.com/svn/trunk/scripts/51-kinect.rules

6.1-5 数据集

- 这里（网页）有 33 个带有 6D 大地坐标的 Kinect 数据集，为 Rawlog 格式(如何读取)。来自 CVPR team at TUM。
- 参见: RGBD_dataset2rawlog 转换工具。

6.2 Kinect 校正

这部分介绍对 Kinect 两个摄像头（RGB 和 IR）的内部参数的校正，以及精确测量它们之间的相对 6D 位姿。

交互 GUI 应用 kinect-stereo-calib 让用户很容易使用棋盘完成 kinect 传感器的校正，无论是在线校正还是使用已经抓取的图片。

。。。。。。

下面是校正文件的示例，可以被类 mrpt::hwdrivers::CKinect 加载（区段名字中的前缀“CAMERA_”可以更改为其它名称）：

```
# Calibration file
# ------------------------------------------------------
# Left camera (IR/Depth in Kinect) calibration parameters
[CAMERA_PARAMS_LEFT]
resolution = [640 488]
cx              = 314.649173
cy              = 240.160459
fx              = 572.882768
fy              = 542.739980
dist            = [-4.747169e-03 -4.357976e-03 0.000000e+00 0.000000e+00 0.000000e+00]

// The order is: [K1 K2 T1 T2 K3]
```

```
# Right camera (RGB in Kinect) calibration parameters
[CAMERA_PARAMS_RIGHT]
resolution = [640 480]
cx                = 322.515987
cy                = 259.055966
fx                = 521.179233
fy                = 493.033034
dist              = [5.858325e-02 3.856792e-02 0.000000e+00 0.000000e+00 0.000000e+00]
// The order is: [K1 K2 T1 T2 K3]

# Relative pose of the right camera wrt to the left camera:
# This assumes that both camera frames are such that +Z points
# forwards, and +X and +Y to the right and downwards.
[CAMERA_PARAMS_LEFT2RIGHT_POSE]
pose_quaternion = [0.025575 -0.000609 -0.001462 0.999987 0.002038 0.004335 -0.001693]
# ------------------------------------------------------------
```

6.3 关于计量单位

虽然已经明确说明，在 MRPT 库和应用中无处不用的计量单位如下表所示：

MEASURE	UNIT
Coordinates, distances 坐标，距离	Meters 米
Angles 角度	Radians 弧度
Time intervals 时间	Seconds 秒
Linear speeds 线速度	m/sec 米每秒
Angular speeds 角速度	rad/sec 弧度每秒

6.4 从 RGB+D 观测生成 3D 点云（CObservation3DRangeScan 对象）

6.4-1 参数校正

首先，你必须非常清楚校正过程中涉及的参数。

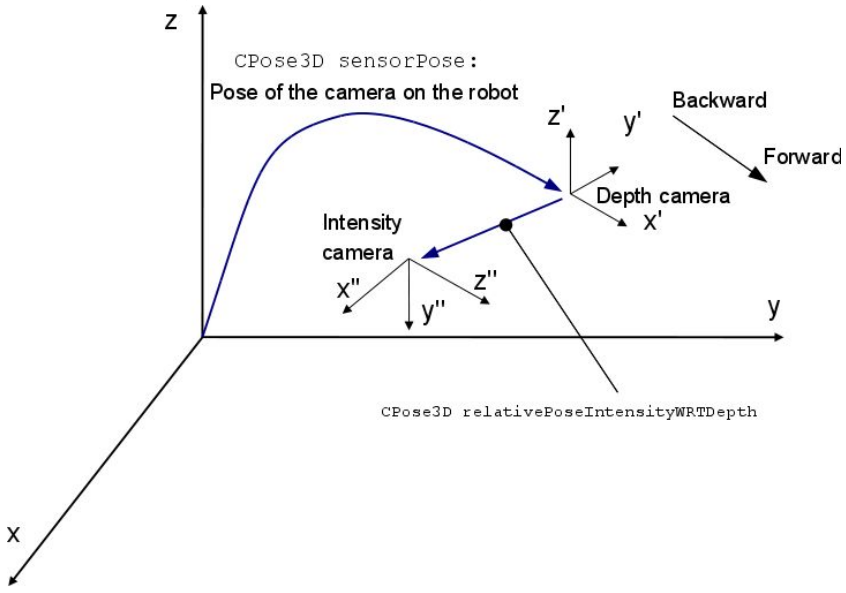
每个"RGB+D observation" 都是作为 mrpt::slam::CObservation3DRangeScan 类(点击查看它的整体描述)的一个对象被保存在 MRPT 中，本教程中设定其包括下面 3 部分:

- RGB 图像（mrpt::utils::CImage intensityImage;）以及它的摄像头参数（mrpt::utils::TCamera cameraParams;)。
- 深度图像（mrpt::math::CMatrix rangeImage;)以及它的摄像头参数(mrpt::utils::TCamera cameraParamsIntensity;)。
- 两个摄像头之间的相对 6D 位姿(mrpt::poses::CPose3D relativePoseIntensityWRTDepth;)

这些观测值可以从传感器(如 Kinect)中被实时捕获，或者从数据集中载入。如此处（网页）所述。

任何情况下，两个摄像头的校准参数已被填上正确的值(或,至少是用户在捕获/抓取数据集时提供的!)

在从 RGB+D 观测值生成精确 3D 点云之前，这三组（两组摄像头参数和一组相对位姿参数）必须是已校正过的(见教程 Kinect calibration)。



6.4-2 投影方程

。。。待写。。。

请参考源代码在：http://mrpt.googlecode.com/svn/trunk/libs/obs/
/include/mrpt/slam/CObservation3DRangeScan_project3D_impl.h

和示例:http://mrpt.googlecode.com/svn/trunk/samples
/kinect_online_offline_demo/

（2.1）一般情况

。。。待写。。。

（2.2）“校正的”深度图

。。。待写。。。

6.4-3 相关 MRPT APIs

存在不同的转换可能：

（3.1）RGB+D －> local 3D point cloud

```
CObservation3DRangeScanPtr obsRGBD = ...  // Get a smart pointer to the RGB+D observation data from wherever
obsRGBD->project3DPointsFromDepthImage(); // Project points into the internal buffers of "obsRGBD"

// Points are now in the object buffers as std::vector<float>:
cout << obsRGBD->points3D_x[0] << endl;
cout << obsRGBD->points3D_y[0] << endl;
cout << obsRGBD->points3D_z[0] << endl;
```

（3.2）RGB+D －> local 3D point cloud －> CPointsMap (& derived)

```
CObservation3DRangeScanPtr obsRGBD = ...  // Get a smart pointer to the RGB+D observation data from wherever
obsRGBD->project3DPointsFromDepthImage(); // Project points into the internal buffers of "obsRGBD"

// Load into an XYZ point map:
CSimplePointsMap pntsMap;
pntsMap.minDistBetweenLaserPoints = 0.05; // In meters
pntsMap.loadFromRangeScan(*obsRGBD);

// *OR* into an XYZ+RGB point map:
CColouredPointsMap pntsMap;
pntsMap.minDistBetweenLaserPoints = 0.05; // In meters
pntsMap.colorScheme.scheme = CColouredPointsMap::cmFromIntensityImage;
pntsMap.loadFromRangeScan(*obsRGBD);
```

（3.3）RGB+D －> CPointsMap (& derived)

```
CObservation3DRangeScanPtr obsRGBD = ...  // Get a smart pointer to the RGB+D
```

```
observation data from wherever
    CSimplePointsMap pntsMap;    // Or any other class derived from CPointsMap. If it has RGB data it will get filled.
    obsRGBD->project3DPointsFromDepthImageInto(
        pntsMap,
        false /* without obs.sensorPose */
        );
```

（3.4）RGB+D －> mrpt::opengl::CPointCloud

```
    CObservation3DRangeScanPtr obsRGBD = ...    // Get a smart pointer to the RGB+D observation data from wherever
    mrpt::opengl::CPointCloudPtr gl_points = mrpt::opengl::CPointCloud::Create();    // Smart pointer to opengl point cloud (with XYZ data)
    obsRGBD->project3DPointsFromDepthImageInto(
        *gl_points,
        false /* without obs.sensorPose */
        );
```

（3.5）RGB+D －> mrpt::opengl::CPointCloudColoured

```
    CObservation3DRangeScanPtr obsRGBD = ...    // Get a smart pointer to the RGB+D observation data from wherever
    mrpt::opengl::CPointCloudColouredPtr gl_points = mrpt::opengl::CPointCloudColoured::Create();    // Smart pointer to opengl point cloud (with XYZ & RGB data)
    obsRGBD->project3DPointsFromDepthImageInto(
        *gl_points,
        false /* without obs.sensorPose */
        );
```

（3.6）RGB+D －> pcl::PointCloud<PointXYZ>

```
    CObservation3DRangeScanPtr obsRGBD = ...    // Get a smart pointer to the RGB+D observation data from wherever
    pcl::PointCloud<pcl::PointXYZ> cloud;
    obsRGBD->project3DPointsFromDepthImageInto(
        cloud,
        false /* without obs.sensorPose */
        );
```

（3.7）RGB+D －> pcl::PointCloud<PointXYZRGB>

```
    CObservation3DRangeScanPtr obsRGBD = ...    // Get a smart pointer to the RGB+D observation data from wherever
```

```
pcl::PointCloud<pcl::PointXYZRGB> cloud;
obsRGBD->project3DPointsFromDepthImageInto(
    cloud,
    false /* without obs.sensorPose */
    );
```

6.5 元编程：类型名 to 字符串

6.5-1 描述

MRPT 提供了强大的模板，可获得任何类型的字符串表示形式。原理想法是，对于任何类型 T，都可以调用 TTypeName<T>::get( )得到该类型的字符串表示：

```
template<typename T>
struct TTypeName
{
  static std::string get();
};
```

这个模板完全依赖于元编程技术（递归模板和部分特化），与 RTTI（运行时类型信息）和编译器特定扩展都无关。支持的类型有：

- POD（Plain Old Data）：int，float，double，etc.....
- 大部分 MRPT 类：所有继承自 mrpt::utils::CSerializable 的类，以及轻型的几何结构体，还有更多。
- STL 容器
- std::pair 对组
- 以上类型的任意复杂组合

如果你想扩展这个机制到自定义类或者结构体（非模板），仅需要使用预定义宏 DECLARE_CUSTOM_TTYPENAME:

```
class MyClass { ... };
DECLARE_CUSTOM_TTYPENAME(MyClass)
cout << TTypeName<MyClass>::get() << endl; // "MyClass"
```

要想扩展到模板，需要定义你自己的 mrpt::utils::TTypeName<>。

6.5-2 示例

完整的代码示例在目录： MRPT/samples/type_name

```
#include <mrpt/base.h>


using namespace mrpt;
using namespace mrpt::utils;
using namespace mrpt::poses;
using namespace std;
```

```
void test()
{
  cout << "Type: " << TTypeName<int32_t>::get() << endl;
  cout << "Type: " << TTypeName<double>::get() << endl;
  cout << "Type: " << TTypeName<CPose2D>::get() << endl;
  // cout << "Type: " << TTypeName<mrpt::slam::COccupancyGridMap2D>::get() << endl;

  cout << "Type: " << TTypeName<vector_double>::get() << endl;
  cout << "Type: " << TTypeName<vector<int32_t> >::get() << endl;
  cout << "Type: " << TTypeName<set<double> >::get() << endl;

  cout << "Type: " << TTypeName<set< vector<double> > >::get() << endl;

  cout << "Type: " << TTypeName<pair<int32_t,int32_t> >::get() << endl;
  cout << "Type: " << TTypeName<pair<int32_t, pair<int32_t,int32_t> > >::get() << endl;

  cout << "Type: " << TTypeName<map< double, set<int32_t> > >::get() << endl;

  cout << "Type: " << TTypeName<set< multimap<double, pair<CPose3DPDFGaussian,
/*COccupancyGridMap2D*/ CPose2D > > > >::get() << endl;
}
```

输出结果：

```
Type: int32_t
Type: double
Type: CPose2D
Type: std::vector<double>
Type: std::vector<int32_t>
Type: std::set<double>
Type: std::set<std::vector<double>>
Type: std::pair<int32_t,int32_t>
Type: std::pair<int32_t,std::pair<int32_t,int32_t>>
Type: std::map<double,std::set<int32_t>>
Type: std::set<std::multimap<double,std::pair<CPose3DPDFGaussian,CPose2D>>>
```

6.6 mrpt-ros-pkg

我们正在努力提供支持机器人操作系统（ROS）的 MRPT 功能封装包。
要了解如何获取代码并生成包，可以去这个项目的网页：

http://code.google.com/p/mrpt-ros-pkg/

另外可参阅在 ROS.org 上的 mrpt-ros-pkg 的文档，或 wiki 页。下面是已有的栈：

- mrpt_common
- mrpt_slam

- mrpt_hwdrivers

也可以阅读：*How to use MRPT libraries from a ROS node?*

6.7 在读取运行时 Kinect 数据和读取 RGBD 数据集文件之间的切换

由于很多机器人应用程序都涉及到机器人传感器，很难正确的实时调试它，因此使用离线开发更方便。只有当应用程序在离线状态下运行一切正常后，才切换到实时在线运行。

下面的例子将告诉您如何做到这一点的，这并不影响你的程序结构：切换将通过一个简单的 bool 变量来控制。

在此处（网页）阅读关于机器人数据集 "rawlog" 的格式介绍，还有这里（网页）关于记录 Kinect 数据集的介绍。

完整的示例源码：

https://raw.github.com/jlblancoc/mrpt/master/samples/kinect_online_offline_demo/

6.8 在 MRPT 中单元测试

每 MRPT 的库（见列表）都有自己的基准测试，以验证并保证每一个类的函数的行为与预期相符。MRPT 使用谷歌的 GTEST 单元测试库（在维基百科的这篇文章中阅读更多关于单元测试的概念）。

添加新的单元测试的方式很简单，因为 CMake 脚本会辨认出那些实现单元测试的源文件，并将它们从普通 MRPT 库中提取出来放入一个特定的程序中，这个程序在测试时自动执行。结果就是，在 Visual Studio 下，用户可以通过向“test”目标发送一个“build”命令去测试 MRPT 行为的正确性。在 Unix/Linux/MacOS 下，发送“make test”命令也可以实现同样的效果。

目前，大约有超过 100 个单元测试，从元素矩阵操作到基于预定义数据集的复杂 SLAM 算法的执行都有所涵盖。在很多测试例子中使用了随机数据和一些输入数据集。在所有测试中，结果同预期值比较，并将不匹配的误差报告。当然，即便是单元测试代码中也可能仍然存在 bug，测试不总是 100%有效的。但多亏这些测试，可以大大减少因为以后的改动导致损坏东西的机会。

同样需要注意的是，在 Debian/Fedora/Ubuntu 软件库里的 MRPT 以及适用于 Windows 的 MRPT 二进制安装包，都是已经在发布前测试过的，以确保它们能按预期正常工作。所以，发布的稳定版本一般是可以保证正常工作的，但我们仍一直在改进它们并在每个 MRPT 版本中增加更多的单元测试。

第七章 里程和运动模型

7.1 概率运动模型

7.1-1 介绍

在粒子滤波中，样本在每一个时间步用给定的建议分布进行繁殖。移动机器人通常直接采用概率运动模型作为这个建议。

在 MRPT 中有两种概率 2D 运动模型，在 mrpt::slam::CActionRobotMovement2D 中实现。要使用时，只需要填写配置结构体“motionModelConfiguration”，并在“CActionRobotMovement2D::TMotionModelOptions::modelSelection”中选择此配置。

可能的使用示例如下：

```
using namespace mrpt::slam;
using namespace mrpt::poses;

CPose2D actualOdometryReading(0.20f, 0.05f, DEG2RAD(1.2f) );

// Prepare the "options" structure:
CActionRobotMovement2D                                actMov;
CActionRobotMovement2D::TMotionModelOptions opts;

opts.modelSelection = CActionRobotMovement2D::mmThrun;
opts.thrunModel.alfa3_trans_trans = 0.10f;

// Create the probability density distribution (PDF) from a 2D odometry reading:
actMov.computeFromOdometry( actualOdometryReading, opts );

// For example, draw one sample from the PDF:
CPose2D sample;
actMov.drawSingleSample( sample );
```

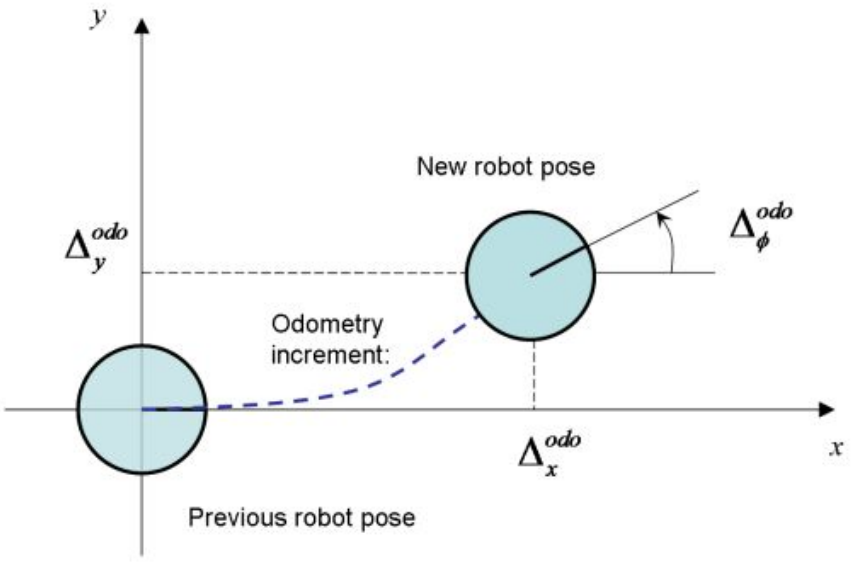
本部分介绍这些方法的内部原理：

7.1-2 高斯概率运动模型

假设里程以机器人 2D 位姿变化增量的形式读取。里程计读数表示为：

$$\left(\Delta_x^{odo}\ \Delta_y^{odo}\ \Delta_\phi^{odo}\right)$$

这些变量的模型见下图所示：



在里程增量变化后，机器人新位姿$(x'\ y'\ \phi')$和先前位姿$(x\ y\ \phi)$的关系方程如下：（基于[1]的建议）

$$\begin{pmatrix} x' \\ y' \\ \phi' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \phi \end{pmatrix} + \begin{pmatrix} \cos(\phi + \frac{\Delta_\phi^{odo}}{2}) & -\sin(\phi + \frac{\Delta_\phi^{odo}}{2}) & 0 \\ \sin(\phi + \frac{\Delta_\phi^{odo}}{2}) & \cos(\phi + \frac{\Delta_\phi^{odo}}{2}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta_x^{odo} \\ \Delta_y^{odo} \\ \Delta_\phi^{odo} \end{pmatrix}$$

在这里，考虑到先前位姿有一个已知值（它是被繁殖的粒子），我们的目标是获得新位姿的多变量高斯分布。在这种情况下，我们可以仅建模如何从先前位姿为（0，0，0）的状态绘制粒子，这之后的采样就可以使用实际的先前位姿。

使用这种简化：

$$\begin{pmatrix} x' \\ y' \\ \phi' \end{pmatrix} = \begin{pmatrix} \cos\frac{\Delta_\phi^{odo}}{2} & -\sin\frac{\Delta_\phi^{odo}}{2} & 0 \\ \sin\frac{\Delta_\phi^{odo}}{2} & \cos\frac{\Delta_\phi^{odo}}{2} & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta_x^{odo} \\ \Delta_y^{odo} \\ \Delta_\phi^{odo} \end{pmatrix} = H \begin{pmatrix} \Delta_x^{odo} \\ \Delta_y^{odo} \\ \Delta_\phi^{odo} \end{pmatrix}$$

高斯平均值可以用先前位姿和里程增量的组合很简单的计算出来。对于协方差，我们需要估计三个里程增量的方差。我们将其建模为具有独立的零均值的高斯误差，该误差由非理想里程计和潜在漂移影响这两项组成。

我们使用Σ表示一个包括三个里程增量方差的对角矩阵，建模为：

$$\sigma_{\Delta_x^{odo}} = \sigma_{\Delta_y^{odo}} = \sigma_{xy}^{min} + \alpha_1 \sqrt{(\Delta_x^{odo})^2 + (\Delta_y^{odo})^2} + \alpha_2 |\Delta_{\phi}^{odo}|$$
$$\sigma_{\Delta_{\phi}^{odo}} = \sigma_{\phi}^{min} + \alpha_3 \sqrt{(\Delta_x^{odo})^2 + (\Delta_y^{odo})^2} + \alpha_4 |\Delta_{\phi}^{odo}|$$

默认参数（构造函数载入的和 RawLogViewer 应用中的）为：

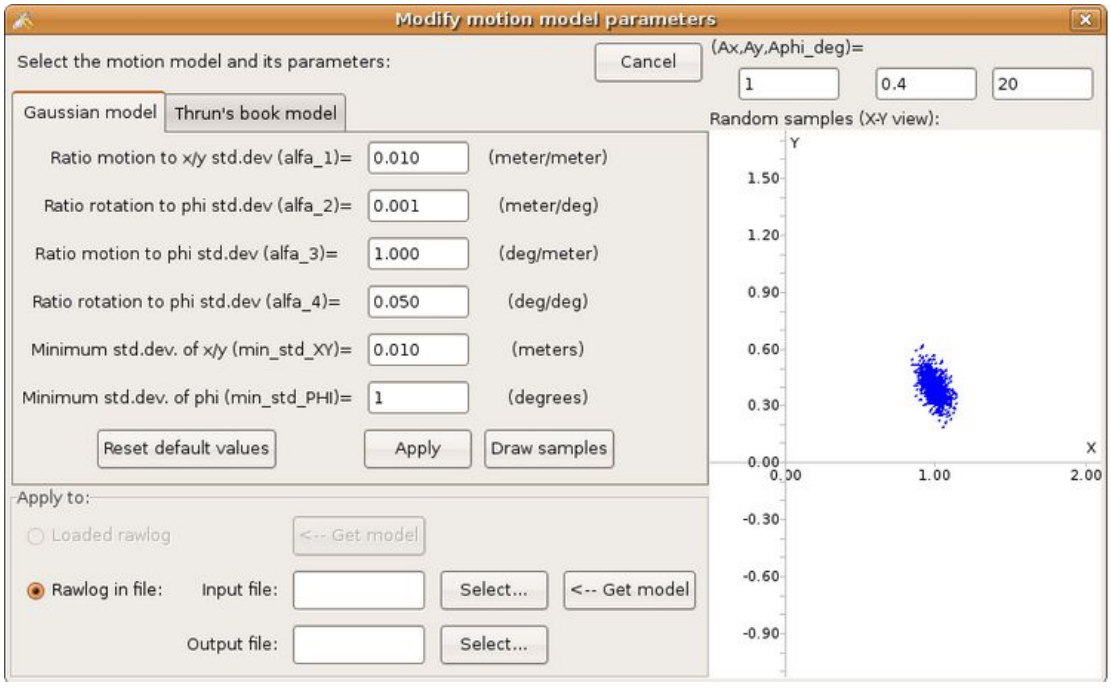
$$\begin{aligned}
\alpha_1 &= 0.05\ meters/meter \\
\alpha_2 &= 0.001\ meters/degree \\
\alpha_3 &= 5\ degrees/meter \\
\alpha_4 &= 0.05\ degrees/degree \\
\sigma_{xy}^{min} &= 0.01\ meters \\
\sigma_{\phi}^{min} &= 0.20\ degrees
\end{aligned}$$

最终，应用里程增量后的新位姿的协方差（C）用下式计算得到：

$$C = J \Sigma J^t$$

其中，J 代表矩阵 H 的雅可比矩阵。参见[3]关于这个公式的误差传播的推导。

下图是在 RawLogViewer 中使用本模型获得的采样示例：

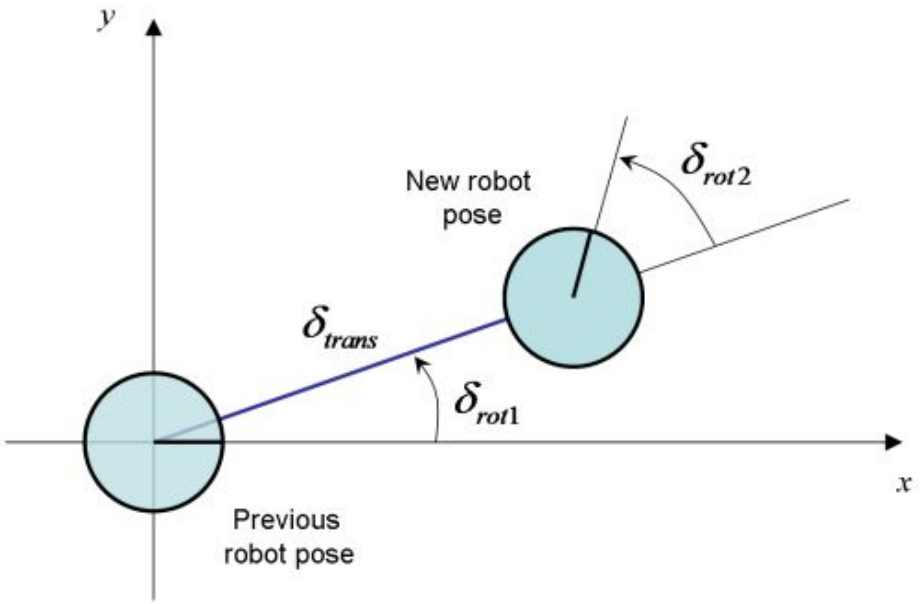


7.1-3 Thrun, Fox & Burgard's book：粒子运动模型

如上，里程计读数表示为 $\left(\Delta_x^{odo}\ \Delta_y^{odo}\ \Delta_{\phi}^{odo}\right)$，我们假设机器人的之前位姿是(0 0 0)，意为我们要抽取机器人里程增量的样本，而不是机器人最终位姿（不失一般性为了简化方程）。然后，我们希望用于样本抽取的机器人的新位姿是：

$$\begin{pmatrix} x' \\ y' \\ \phi' \end{pmatrix} = \begin{pmatrix} \cos\hat{\delta}_{rot1} & 0 & 0 \\ \sin\hat{\delta}_{rot1} & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix} \begin{pmatrix} \hat{\delta}_{trans} \\ \hat{\delta}_{rot1} \\ \hat{\delta}_{rot2} \end{pmatrix}$$

这里的变量与机器人位姿增量的对应如下图所示：



在这里，变量 $\hat{\delta}_{trans}$, $\hat{\delta}_{rot1}$ and $\hat{\delta}_{rot2}$ 是对实际里程计读数加入了高斯、零均值随机噪声的结果：

$$\begin{aligned} \hat{\delta}_{trans} &= \delta_{trans} + \epsilon_{trans} & \epsilon_{trans} &\sim \mathcal{N}(0, \sigma^2_{trans}) \\ \hat{\delta}_{rot1} &= \delta_{rot1} + \epsilon_{rot1} & \epsilon_{rot1} &\sim \mathcal{N}(0, \sigma^2_{rot1}) \\ \hat{\delta}_{rot2} &= \delta_{rot2} + \epsilon_{rot2} & \epsilon_{rot2} &\sim \mathcal{N}(0, \sigma^2_{rot2}) \end{aligned}$$

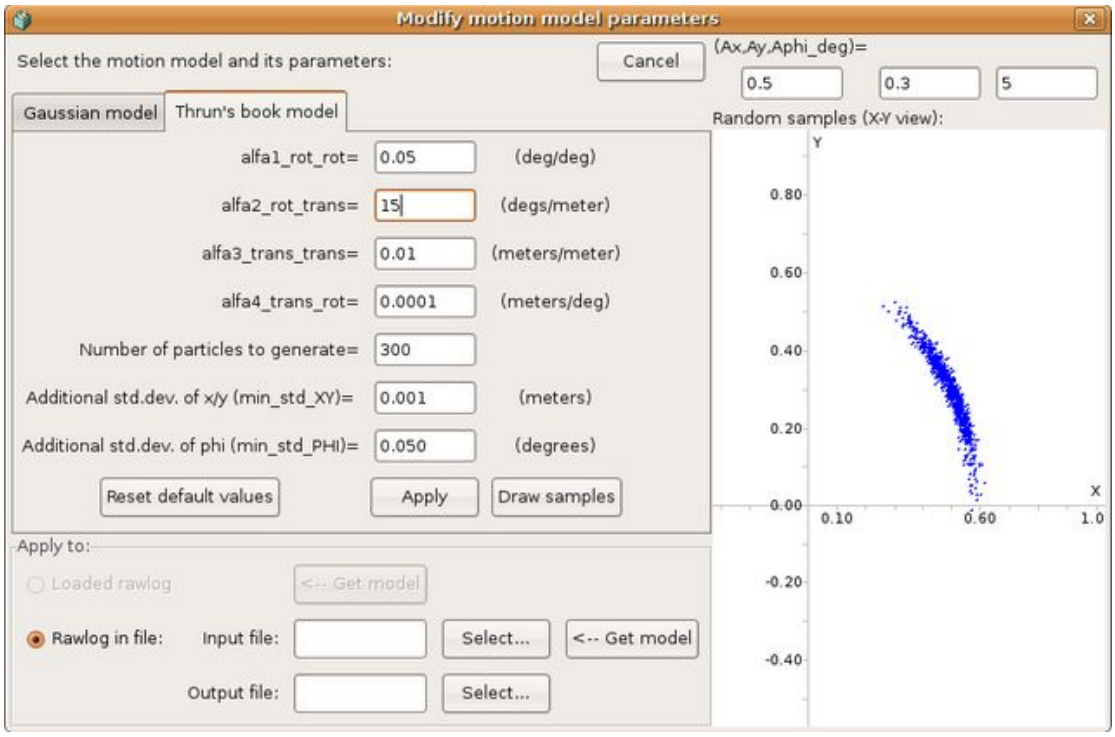
文献[2]中描述的模型使用了下面这个近似值作为上面的方程需要的标准偏差值：

$$\begin{aligned} \sigma_{rot1} &= \alpha_1 |\delta_{rot1}| + \alpha_2 \delta_{trans} \\ \sigma_{trans} &= \alpha_3 \delta_{trans} + \alpha_4 (|\delta_{rot1}| + |\delta_{rot2}|) \\ \sigma_{rot2} &= \alpha_1 |\delta_{rot2}| + \alpha_2 \delta_{trans} \end{aligned}$$

这个模型已经在 CActionRobotMovement2D 中实现，只需将“CActionRobotMovement2D::TMotionModelOptions::modelSelection”设为“mmThrun”即可。实际上，一个小的额外误差被累加到位姿的每个成员上。为了避免这种问题，在空里程增量时所有粒子运动变为绝对零值，但这也会导致粒子滤波器退化。

下图是使用本模型生成采样的演示，这里设定了一个超大的α_2值（一个非常大的“滑

移”）。使用 RawlogViewer 生成：



参考文献

[1] Eliazar, A.I. and Parr, R. Learning probabilistic motion models for mobile robots, 2004. (ACM portal).

[2] Thrun S. and Burgard W. and Fox D. Probabilistic Robotics (book), 2005.

[3] Arras, K.O., “An Introduction to Error Propagation: Derivation, Meaning, and Examples of Equation cy= fx cx fx”, Lausanne: Swiss Federal Institute of Technology Lausanne (EPFL), 1998.

第八章 路径和运动规划

8.1 避障

8.1-1 概述

MRPT 包括一些与避障和被动导航相关的类：

用于现实的机器人：

- 基于 PTG 被动导航引擎，其内部采用下面的完整约束导航。
- 完整的 GUI 模拟器: ReactiveNavigationDemo

用于完整约束机器人(即能向任何方向移动):见应用 holonomic-navigator-demo 和 C++类:

- Nearness-Diagram 逼近图实现
- VFF (虚拟力场 or 势场)

8.1-2 参考

- Blanco, J., González, J., and Fernández-Madrigal, J. 2008. Extending obstacle avoidance methods through multiple parameter-space transformations. Auton. Robots 24, 1 (Jan. 2008), 29-48. DOI=http://dx.doi.org/10.1007/s10514-007-9062-7
- Nearness diagram (ND) navigation: collision avoidance in troublesome scenarios, IEEE Transactions on Robotics and Automation, Minguez, J. and Montano, L., vol. 20, no. 1, pp. 45-59, 2004.

8.2 占据栅格地图下的路径规划

8.2-1 描述

MRPT 在类 CPathPlanningCircularRobot 中实现了用于搜索最短路径的基本的值迭代算法[1]，适用于占据栅格地图和圆形机器人。

该方法包括两个步骤：

- 由机器人半径决定障碍物增长。这使得仅一个自由元格(cell)就足够确保机器人不发生碰撞。
- 值迭代算法。从源位置开始，迭代增加最短路径覆盖的区域，直至到达目标元格(cell)。

注意，这是一个非常简单的方法，不适合于机器人形状与圆形差异较大 和/或 在比较杂乱的环境运动的情况。

可查看 obstacle avoidance methods.

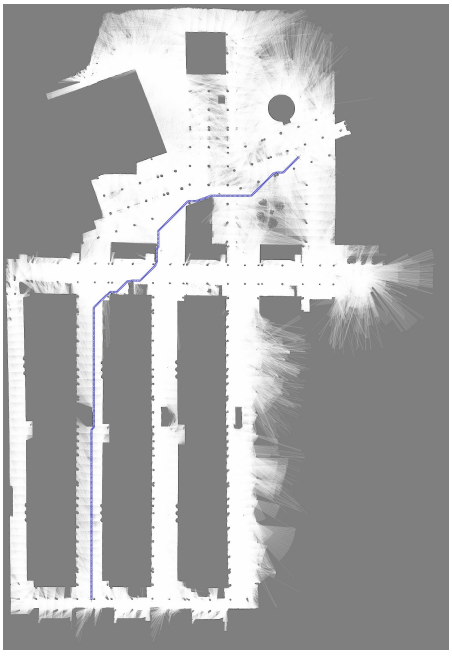
8.2-2 使用

使用的基本要求是，声明栅格地图、CPathPlanningCircularRobot 对象、设置机器人半径、并调用 CPathPlanningCircularRobot:: computePath( )。下面的代码片段取自示例 MRPT/samples/pathPlanning.：

```
// Load,create,... the grid map:
COccupancyGridMap2D gridmap;
// ...

// Find path:
CPathPlanningCircularRobot pathPlanning;
pathPlanning.robotRadius = 0.30f;

std::deque
thePath;
bool notFound;

CPoint2D origin( 20, -110 );
CPoint2D target( 90, 40 );

pathPlanning.computePath( &gridmap, &origin, &target, thePath, notFound, 100.0f /* Max. distance */ );

// Process the output path in "thePath"
// ....
```

8.2-3 结果

下图是示例 pahtPlanning 的输出结果：



参考文献

[1] Value iteration algorithm (Wikipedia) -

http://en.wikipedia.org/wiki/Markov_decision_process#Value_iteration.

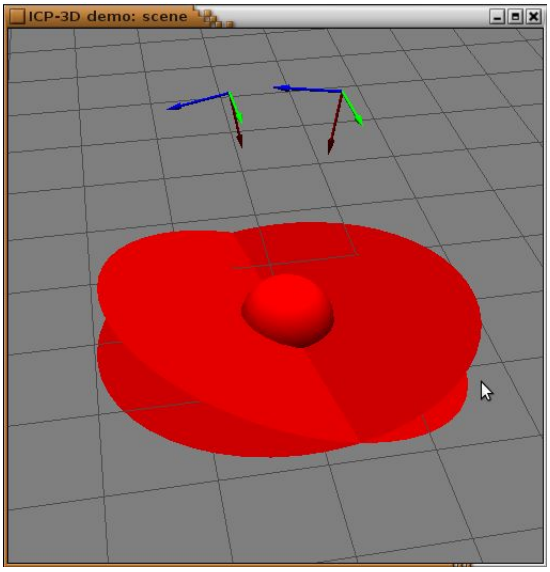
第九章 概率编程

。。。待写 。。。

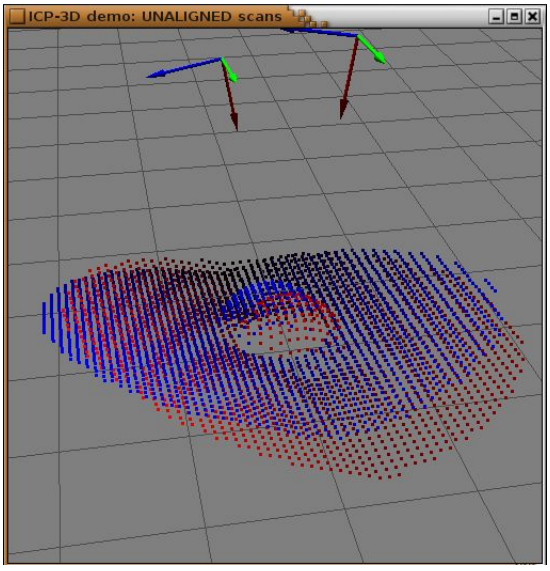
第十章 扫描匹配和 ICP

10.1 3D-ICP 示例

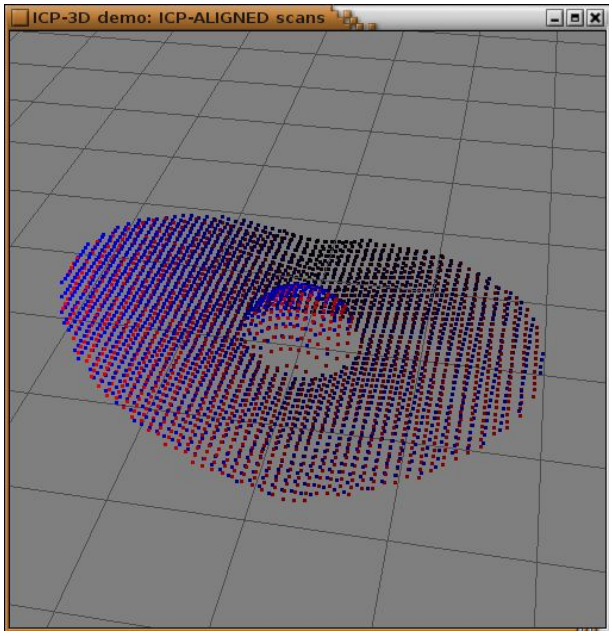
这个示例中用到的主要方法是 CICP::Align3DPDF。这个例子首先通过从两个不同位姿进行光线追踪来虚拟了两组 3D 点云：



然后，将一个小偏差注入到一组点云的所有六维坐标，制造了一个未对齐的点云：



对这些未对齐的地图应用 ICP-3D 算法之后，算法成功恢复了正确的对齐，如下图所示：



最新的示例代码可以在此找到：
https://raw.github.com/jlblancoc/mrpt/master/samples/icp3D/

10.2 迭代最近点（ICP）和其它匹配算法

10.2-1 迭代最近点（ICP）算法

最初在文献[1]中被介绍，ICP 算法的目的是通过最小化对应的实体之间的平方差来找到一片点云和某个参考表面（或另一片点云）之间的变换。

ICP 的“迭代”来自于一个事实，即最优解趋向局部最小偏差后对应关系要被重新考核。同任何梯度下降方法一样，ICP 的适用性以我们有一个相对较好的起点为前提。否则，它将陷入第一个局部最小而得到无用解。

在移动机器人领域，ICP 已经被广泛使用于 2D 激光匹配，一个名为“扫描匹配”的问题。

MRPT 中的 ICP 算法可以使用以下作为输入：

- 两个平面（2D）地图，可以是：
  - 一个点云地图作为参考地图，待对齐地图也是点云地图。或
  - 一个占据栅格地图作为参考地图，待对齐地图是点云地图。
- 两个 3D 地图，都用点云表示。

在点云地图的情况下，KD 树被用来加速最近邻搜索。

ICP 方法在类 mrpt::slam::CICP 中被实现，输出是两个地图之间相对位姿的概率密度函数 PDF（probability density function），即，一个关联到最佳配准的不确定性约束也被计算出来。

一个典型应用示例（也可在目录 MRPT/samples/icp 下查看）：

```
CICP icp;
// set ICP parameters:
```

```
icp.options.maxIterations = 50;
...

// Reference map:
CSimplePointsMap refMap;

// Map to be aligned:
CSimplePointsMap alignMap;

// Initial guess, used in the first ICP iteration:
CPose2D initialGuess(0,0,0);
CPosePDFPtr pdf = icp.AlignPDF(
   &refMap, // Reference map
   &alignMap, // Map to be aligned
   initialGuess // Starting estimate
   );
CPose2D icpEstimateMean = pdf->getMeanVal();
cout << icpEstimateMean << endl;
```

在 MRPT C++库中实现的不同 ICP 算法是：

- “经典”ICP
- Levenberg-Marquardt 迭代方法

（1.1）使用示例

- 2D 对齐，查看：https://raw.github.com/jlblancoc/mrpt/master/samples/icp/
- 3D 对齐（全 6D 位姿），查看：https://raw.github.com/jlblancoc/mrpt/master/samples/icp3D/

（1.2）“经典”ICP 算法

在 MRPT 中这个算法可以通过方法 mrpt::slam::CICP::AlignPDF(), ::Align() (或它们的 3D 版本) 被调用，需要在结构体 CICP::options 中设定 ICP_algorithm = CICP::icpClassic。

在 MRPT 中实现的具体算法在接近收敛时是表现为逐步细化的。如果你希望禁用细化阶段，可以设置参数 ALFA=0.

我们先用伪代码展示简化版算法，然后我们会介绍更多感兴趣的部分，最后我们给出完整的参数列表极其含义：

```
i=0          // Iteration counter
P(i)=P0    // Initial guess given by the user
WHILE( i<max_iterations OR thres_dist>thres_dist_min )
   Matchings = ComputeMatching of m1 with m2 displaced by P(i) with thres_dist & thres_ang
   P(i+1) = LeastSquare(Matchings)
   IF (all components of |P(i+1)-P(i)|<1e-6)
      thres_dist *= alpha
```

```
        thres_ang    *= alpha
        IF (thres_dist<thres_dist_min)
           BREAK;    // End of the WHILE loop: we reached convergence
        END-IF
      END-IF
      i++;
    END-WHILE
```

意思是：变换后的点云与作为参考的点云地图用 thres_dist 和 thres_ang 决定匹配，然后执行求解器以获得最佳配对的 2D 或 3D 变换，重复直至收敛。如果 ALFA>0（默认值），则 tresholds 将降低，重复整个过程。

上面的算法被 mrpt::slam::CICP::options 的以下这些参数控制：

- TICPAlgorithm ICP_algorithm: …
- bool onlyClosestCorrespondences: …
- bool onlyUniqueRobust: …
- unsigned int maxIterations: …
- float thresholdDist,thresholdAng: 当判定两片点云匹配时，附件两个点被视为“配对候选”只有当它们的距离小于 thresholdDist+ D* thresholdAng，其中 d 为在“待对齐”地图上的点到地图坐标原点的距离。在数学上，这构造了两片点云之间的位姿角度分量的不确定度。
- float ALFA: …
- float smallestThresholdDist: …
- float covariance_varPoints: …
- bool skip_cov_calculation: …
- bool doRANSAC: …
- unsigned int ransac_minSetSize,ransac_maxSetSize,ransac_nSimulations: …
- float ransac_mahalanobisDistanceThreshold: …
- float normalizationStd: …
- bool ransac_fuseByCorrsMatch: …
- float ransac_fuseMaxDiffXY, ransac_fuseMaxDiffPhi: …
- float kernel_rho: …
- bool use_kernel: …
- float Axy_aprox_derivatives: …
- float LM_initial_lambda: …
- uint32_t corresponding_points_decimation: M2 的每个点都通过一个 kd 树测试在 M1 的最近邻。查询 kd 树实际上是整个 ICP 备过程中最耗时的部分。这就是为什么当 M2 是一个密集点云时去降采样它会是一个好主意。这个参数控制降采样（默认值=5），但可改为 1 来执行精确匹配搜索。然而，试探法有很好的效果和很强的时间改善，所以建议在不降低结果准确度的情况下将该参数设置的尽可能高。请注意，只有“对应点抽取”出的一个点会去和 M1 匹配，但每次当阈值被“alfa”调节后，这些点索引的偏移也被转移，因此一个完整的 ICP 对齐后，所有 M2 点都已被照顾到。只是，不是同一时间而已。

（1.3）Levenberg-Marquardt 的 ICP 算法

与上面的伪代码唯一不同的地方是将下面这句：

```
LeastSquare(Matchings)
```

替换为：

```
NonLinearLeastSquare(Matchings)
```

其中，最大限度地减少配对之间的均方差的优化器是用 Levenberg-Marquardt 算法来实现的。雅可比矩阵的数值确定是为尽可能好的捕捉点的实际分布。

MRPT 对这个算法的实现是基于 Dr. Paul Newman 博士的代码。

10.2-2 对应关系组优化

（2.1）最小二乘刚性变换（2D+方向）

给定两组点之间的一系列对应关系，该方法计算平方差最小的变换。实现在 scanmatching:: leastSquareErrorRigidTransformation 中。

（2.2）最小二乘刚性变换（6D）

给定两组点之间的一系列对应关系，该方法计算平方差最小的变换。实现在 scanmatching:: leastSquareErrorRigidTransformation6D 中。

（2.3）鲁棒刚性变换（2D+方向）

给定两组点之间的一系列对应关系，该方法使用鲁棒 RANSAC 级计算可能变换的高斯和（SOG）。实现在 scanmatching::robustRigidTransformation 中。

参考文献

[1] P.J. Besl, H.D. McKay, A method for registration of 3-D shapes, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1992

[2] A. Censi, “An ICP variant using a point-to-line metric”, ICRA’08

[3] Segal, A. and Haehnel, D. and Thrun, S. “Generalized-ICP” (gICP, G-ICP). RSS 2009. (C++ Implementation by Alex Segal)

第十一章 串行化

11.1 串行化机制

介绍 mrpt::utils::CSerializable 类，并介绍如何实现可串行化类。

11.1-1 基础

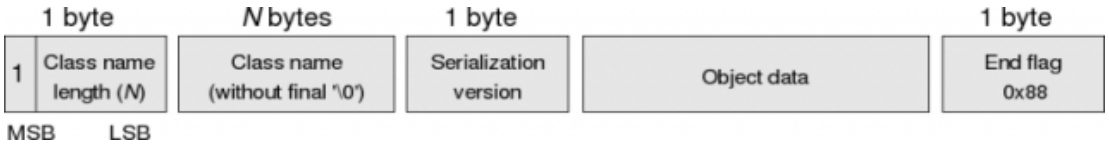
串行化过程包括提取现有对象，并以给定的任意格式将它转换成一个字节序列，对象的内容和状态可以在以后被重建或反串行化[1]。有很多可以用于串行化的 C++库（比如 boost），而 MRPT C++库使用了比较简单的定制实现，基于以下目的：

- 简洁：只有几个小的核心功能。
- 版本化：如果一个类以后有改变（很可能），会给它的串行化结果中分配一个新的版本号，但旧版本数据仍然可以被导入。
- 独立于 C++编译器：只使用标准化数据长度。比如，“int”类型在不同机器平台有不同的长度，因此不允许串行化“int”型变量除非将它强制转换为已知长度。

目前，串行化唯一支持的格式是二进制的，即，不支持 XML。原因是，对于运行在实时系统中的机器人应用，更重要的是节省数据大小（和传输时间）。请注意，在 MRPT 中存在特殊的“stream”类，所以标准的 std:: istream 和 std:: ostream 仅保留用于文本输入和输出（大多只是供人检查或调试），而 MRPT 自己的 stream 类（几乎仅）用于二进制串行化。

（1.1）类

每个串行化对象的实际二进制帧大致如下：



注意：在 MRPT 0.5.5 之前的版本中，没有末尾标识（End flag），而且第一和第三字段是 4 个字节宽(不是仅 1 个字节)。然而，用旧格式保存的数据仍能够被无误加载。当一个对象被串行化后,通过 CStream 类，它的内容被写入到一个目标中。当前已实现的 CStream 类请见 mrpt::utils::CStream。

（1.2）POD（普通旧数据类型）和特例

在上面提到的"对象数据 object data "字段中，每个类都可以完全控制存储到此处的数据。通常，一个类会将其内部包含的每一个对象放置到"对象数据 object data "字段中，这些内部对象基于不同的类，有自己的串行化数据格式，因此类的串行化数据格式是递归的。

然而，为了避免帧头帧尾的额外占用，对我们已知的一些长期不变的基础通用数据类型进行特殊处理。以下数据类型的串行化区域直接包含了一段按照小端模式排列的变量内存块(即使在大端架构中)。

- bool
- uint8_t,int8_t

- uint16_t,int16_t
- uint32_t,int32_t
- uint64_t,int64_t
- float
- double
- long double(如果在使用的编译器中有定义)

对于 float 和 double 类型,格式假定一个低级 IEEE 754 机器编码(事实上所有的现代架构)。 注意：为何上面没有列出 int 和 short，这是由于这些类型的大小依赖于系统架构。当处理串行化时，请尽量使用数据大小定义好的类型。

以下的基本类型在 MRPT 中也有特殊的串行化格式:

- const char *：字符串。它的串行化二进制格式首先包含一个 uint32_t 值，用于存储字符串长度（不包括末尾′\0′）,接下来是字符串的所有字符，不包括末尾′\0′。
- std::string：字符串。同 const char * 。
- 基础数据类型 vector：这些 vector 的串行化二进制格式首先包含一个 uint32_t 值，用于存储元素数量，接着是每个元素的串接(注意：为了存储效率，这些串行化格式是基于更通用的 STL 串行化机制实现的特殊版本，下面所列):
  - std::vector<float>
  - std::vector<double>
  - std::vector<int8_t>
  - std::vector<int16_t>
  - std::vector<int32_t>
  - std::vector<int64_t>
  - std::vector<uint8_t>
  - std::vector<uint16_t>
  - std::vector<uint32_t>

（1.3）存储基础数据类型的数组

如果说你想保存和加载一个基础数据类型的 C 数组（POD，见上所述），重点是注意那些 POD 类型的字节存储顺序（小端 or 大端）。举例说明，如果像下面这样写入整个数组的内存块：

```
float v[100];
void write(CStream &amp;s)
{
s.WriteBuffer(&amp;v,100*sizeof(float)); // Bad: DON'T do this!!!
}
void read(CStream &amp;s)
{
s.ReadBuffer(&amp;v,100*sizeof(float)); // Bad: DON'T do this!!!
}
```

将导致跨越不同的字节存储顺序的系统时出现二进制格式不兼容问题。为解决这个问题，MRPT 提供了两个可对字节重排序的方法，如果需要的话：

```
float v[100];
void write(CStream &amp;s)
```

```
{
   s.WriteBufferFixEndianness(&amp;amp;v,100);    // OK
}
void read(CStream &amp;amp;s)
{
   s.ReadBufferFixEndianness(&amp;amp;v,100);    // OK
}
```

欲了解更多信息，请参考 MRPT:: utils::CStream 的文档及其方法。另外请注意，如果你的 Vectors 是在 STL 容器中，而不是在普通的 C 数组中，你可以使用下面介绍的 STL 串行化机制，更安全更清晰。

（1.4）基本用法

将现有对象串行化存储到（比如）文件中的典型用法是使用 CStream 类的<<操作符。

```
#include
#include

using namespace mrpt;
using namespace mrpt::slam;
using namespace mrpt::math;
using namespace mrpt::utils;

int main()
{
        // Declare serializable objects:
        COccupancyGridMap2D    grid;
        CMatrix                M(6,6);

        // Do whatever...

        // Serialize it to a file:
        CFileStream("saved.gridmap", fomWrite) <<grid<< M;

        return 0;
}
```

要恢复已保存的对象，你可以使用两种方法，这取决于你是否确知将被从流中读取的对象的类。如果您了解待读取对象的类，你可以对现有对象简单地使用>>运算符，这个对象的参考和内容将被流中读出的数据覆盖。举例如下：

```
// Declare serializable objects:
COccupancyGridMap2D    grid;
CMatrix                M;

// Load from the file:
```

```
CFileInputStream(“saved.gridmap”) >>grid >>M;
```

另一种情况，你不知道要读取的对象的类。在这种情况下，它必须被声明为一个指向通用 utils::CSerializable 对象的智能指针（初始化为 NULL，表明它为空），在使用>>操作后，它将被指向一个新创建的反串行化对象：

```
// Declare serializable objects:
CSerializablePtr obj;   // NULL pointer

// Load from the file:
CFileInputStream(“saved.gridmap”) >>obj;

std::cout <<“Loaded an object of class: ”<<obj->GetRuntimeClass()->className;
```

接下来的部分解释了 utils::CSerializable 最重要的方法和运行时类的信息。在加载未知类的对象时，重点是了解 MRPT 注册机制以及何时需要手动调用。注意,这些示例代码没有捕捉可能的异常(更多关于 MRPT 中异常管理 见网页)。

除了使用 utils:: CStream 中的<<和>>操作符之外，还有两个独立的函数 utils:: ObjectToString 和 utils:: StringToObject，分别是将对象串行化和反串行化到标准的 STL 字符串（std::string）。与正常的 CStream 串行化函数不同的是，这两个函数将二进制数据流进行编码以避免空字符’\0’，因此结果可以被转化为 char*字符串。除非绝对必要，否则尽量避免使用这两个函数，因为它们引入了额外的处理延迟。

11.1-2 运行时类识别

所有可串行化类都必须继承自虚拟类 utils::CSerializable，这个虚拟类提供了管理任意可串行化对象的标准方法，无需知道对象真正的类。最常见的操作是判断一个对象是否是给定的类型，这可以通过以下方式进行：

```
CSerializablePtr obj;
stream >>obj;

// Test if “obj”points to an object of class “CMatrix”
if ( IS_CLASS(obj, CLASS_ID( CMatrix ) )
...
```

如果进行检测的类没在当前命名空间中（没有 using namespace NAMESPACE;），你也可以使用 CLASS_ID_NAMESPACE，比如：

```
if ( obj->GetRuntimeClass() == CLASS_ID_NAMESPACE( CMatrix, UTILS) ) ...
```

方法 CSerializable::GetRuntimeClass()实际上返回一个指向 UTILS::TRuntimeClassId 数据结构体的指针，还包括其它有用的成员：

◆ 类名字符串：

```
 obj->GetRuntimeClass()->className;
```

■ 检查一个类是否是给定虚类的子类。示例：

```
void func( CMetricMap * aMap )
{
```

```
        if (IS_DERIVED(aMap, CPointsMap))
        {
                CPointsMap *pMap = (CPointsMap*) aMap;
                ...
        }
}
```

对于任何可串行化对象，另一个有用的方法是 CSerializable::duplicate，它用来制作对象的一个拷贝。内部数据、指针、等等.....将被真正复制，原对象可以被安全地删除。

11.1-3 如何实现新的可串行化类

下面介绍 CSerializable 类的内情，以及如何开发新的可串行化类。

（3.1）一般步骤

- 为类定义一个默认构造函数，即，无参数。你也可以在其中一个构造函数中为所有参数设定默认值。
- 从 utils::CSerializable 派生，或者从派生自 utils::CSerializable 的任何其它类派生。
- 在类定义中添加宏 DEFINE_SERIALIZABLE(class_name) (即“class”范围内部)，在类声明前添加宏 DEFINE_SERIALIZABLE_PRE(class_name)。
- 在类的实现文件添加宏 IMPLEMENTS_SERIALIZABLE(class_name,parent_class,namespace)。
- 在你的类中实现虚拟方法 UTILS::CSerializable::writeToStream() 和 UTILS::CSerializable::readFromStream()。这些方法负责[注入/解析]对象[到/从]二进制流。
  - virtual void writeToStream(CStream &out, int *getVersion) const = 0;
    - out：二进制流输出，数据被注入。
    - getVersion：若为 NULL，则对象数据被注入。否则，仅将版本号用这个指针返回。这可使对象版本化注入并向后兼容性之前保存的数据。
  - virtual void readFromStream(CStream &in, int version) = 0;
    - ▪ in：待读取的对象二进制流：通常对版本号“switch”以实现对所有流化版本的不同读取过程，以便允许二进制兼容不同版本保存的旧数据。
    - ▪ version：存储在数据流中的对象版本：在你的代码中使用此版本号以了解如何读取进来的数据。

下面的示例可以作为创建新的可串行化类的模板：

```
// CLASS DECLARATION (Typically in a &amp;quot;.h&amp;quot; file)
// ================================================
#include

namespace MyNamespace
{

        // This must be added to any CSerializable derived class:
        DEFINE_SERIALIZABLE_PRE( CMyPose2D )
```

```
        class CMyPose2D : public mrpt::utils::CSerializable
        {
                // This must be added to any CSerializable derived class:
                DEFINE_SERIALIZABLE( CMyPose2D )

        public:
                 // Constructor from an initial value of the pose.
                 CMyPose2D(float x=0,float y=0,float phi=0);

        protected:
                float m_x,m_y,m_phi;
    }; // End of class declaration

};

// CLASS IMPLEMENTATION (Typically in a ".cpp"file)
// ================================================
using namespace MyNamespace;

IMPLEMENTS_SERIALIZABLE(CMyPose2D, CSerializable, MyNamespace)

void    CMyPose2D::writeToStream(CStream &out,int *version) const
{
        if (version)
                *version = 0;      // This is the serialization version #0 for this class.
        else
        {
                // Save the data:
                out <<m_x <<m_y <<m_phi;
        }
}

void    CMyPose2D::readFromStream(CStream &in,int version)
{
        switch(version)
        {
        case 0:
                {
                        // Load the data:
                        in >>m_x>>m_y >>m_phi;
                } break;
        default:
                MRPT_THROW_UNKNOWN_SERIALIZATION_VERSION(version)
        };
```

```
}
```

（3.2）特殊情况

如果自定义的可串行化类是虚类，则将宏 DEFINE_SERIALIZABLE 和 IMPLEMENTS_SERIALIZABLE 分别替换为 DEFINE_VIRTUAL_SERIALIZABLE 和 IMPLEMENTS_VIRTUAL_SERIALIZABLE 。（DEFINE_SERIALIZABLE_PRE 不用变）

11.1-4 MRPT 中串行化的用途

- Rawlogs：由机器人收集的数据（“数据集”）被保存为 rawlog 格式（*.rawlog），实际上是串行化到文件中的一系列 action-observation 对组。这些文件有一个独立的应用程序来管理、可视化和编辑：RawLogViewer。
- 地图：所有在 MRPT 定义的地图可以串行化保存到文件（见 MRPT::slam:: CMetricMap）。
- 3D 场景也被保存为自定义的文件格式（*.3Dscene），这仅仅是一个串行化的 UTILS:: COpenGLScene 对象。有一个独立的应用程序用于可视化这些文件：3DSceneViewer。
- （地图，图片，传感数据，等...）对象通过 TCP/ IP 套接字进行传输或作为 BABEL 开发系统的参数。

11.1-5 MRPT 内部注册可串行化类

要从流中加载未知类的对象，它的类必须预先注册为 CSerializable 的实现（见 MRPT:: utils:: registerClass）。有时候这对获取现有类的清单是很有用的，比如，为给定的虚基类生成一个所有子类的清单。为此目的，可以使用 MRPT:: utils:: getAllRegisteredClasses。

11.1-6 串行化和 STL 容器

MRPT 完全支持对混合了 STL 容器、普通数据类型、MRPT 类的任意复杂数据结构进行串行化。示例如下：

```
std::multimap<double, std::pair<CPose3D,COccupancyGridmap2D>>    myVar;
file <<myVar;
```

上面的代码将编译并正常工作，无需用户为 multimap<>类型编写任何额外的代码。在使用 STL 容器的情况下，二进制格式包括：

- 注入一个 std :: string，其值为 STL 容器名称（使用上面说过的串行化格式注入）。
- 注入一些字符串，表示容器中持有的每个数据类型。（map 的 key 和 value，list 的 values,）。
- 容器中元素的个数（适用于所有容器，除 std::pair）。
- 各元素的递归注入。如果元素是 STL 容器，这里同样适用。对于基本的 MRPT 类，在这里使用上面说过的格式。

下面这个真实示例来说明其格式：

```
#include

int main()
{
    map<uint32_t, CPose2D> m1;
```

```
        m1[2] = CPose2D(1,2,0);
        m1[9] = CPose2D(-2,-3,1);

        CFileOutputStream f("m1.bin");
        f <<m1;

        return 0;
    }
```

下面是生成的输出：

```
00000000  08 00 00 00 73 74 64 3A  3A 6D 61 70 08 00 00 00  ....std::map....
00000010  75 69 6E 74 33 32 5F 74  07 00 00 00 43 50 6F 73  uint32_t....CPos
00000020  65 32 44 02 00 00 00 02  00 00 00 87 43 50 6F 73  e2D.........CPos
00000030  65 32 44 01 00 00 00 00  00 00 F0 3F 00 00 00 00  e2D........?....
00000040  00 00 00 40 00 00 00 00  00 00 00 00 88 09 00 00  ...@............
00000050  00 87 43 50 6F 73 65 32  44 01 00 00 00 00 00 00  ..CPose2D.......
00000060  00 C0 00 00 00 00 00 00  08 C0 00 00 00 00 00 00  ................
00000070  F0 3F 88                                          .?.
```

11.2 Rawlog 格式

本节描述了二进制文件".rawlog"的格式，用来存储机器人数据集(见资源库)，作为很多 MRPT 应用程序的输入用于离线处理。

11.2-1 现有操作工具

- **RawLogViewer**: 这个功能强大的 GUI 程序非常有用，可以快速可视化任意数据集的内容，修改、导入、导出为其它格式。
- **rawlog-grabber**: 这个命令行程序可以从多个传感器中抓取观测数据，并把所有观测数据组合存入一个时间戳排序的数据集文件。
- **rawlog-edit**: 一个命令行工具，与 RawlogViewer 功能类似。
- **carmen2rawlog**: 一个命令行工具，用于导入 CARMEN logs 转为 Rawlogs。
- rgbd_dataset2rawlog: 一个命令行工具，用于转换 TUM rgbd 数据集到 rawlogs。

11.2-2 FORMAT#1:一种贝叶斯友好的文件格式

Rawlog 文件的目的是为了尽可能准确的获取机器人的所有数据，通过自动导航或手动导引实现机器人在一个环境中移动获得的数据。

从贝叶斯 SLAM 的角度来看，这些数据被分成明显不同的两组：动作值（actions）及观测量(observations)，在文献里一般分别表示为 Uk 和 Zk 。

因此，为了更容易实现 MRPT 中的贝叶斯方法，一个 rawlog 文件被分成一系列动作，观测，动作(actions)，观测(observations)…. “动作(Actions)”通常包括机器人电机转动(里程)，也可以包括任何用户自定义的动作(如. 机械手臂动作)。"观测(Observations)"包括机器人各种传感器的读取值，如：激光扫描仪, 摄像头图像, 超声测距，等等。

需要注意的是，在连续两个动作(Actions)之间插入一组观测量的目的是为了确保它们（这组观测量）是同时被收集的，虽然每个独立观测量都有自己的时间戳。GUI 应用程序 RawLogViewer 提供了几个用于可视化和操作 rawlog 文件的工具。

（2.1）这种格式的".rawlog"文件的实际内容

这种 rawlog 文件是下面这两个类的对象交替二进制串行化：

- CActionCollection，一个或多个动作(actions) (比如，里程)
- CSensoryFrame，存储观测值。

11.2-3 FORMAT#2:一种时间戳排序的观测序列

前面的格式确实非常适用于贝叶斯方法，有着清晰独立的动作过程-观测过程。但有某些复杂的数据集：有很多不同的传感器，工作在不同的速率，也可能没有里程数据（SLAM 算法中典型的“动作 action”），这种情况下，只使用观测量序列作为数据集进行存储将更清晰。

（3.1）这种格式的".rawlog"文件的实际内容

在这种格式中，rawlog 文件是对类 slam::CObservation 派生对象的二进制串行化。这种情况下，里程数据也被存储为观测量。

应用 RawLogViewer, rawlog-grabber, 类 slam::CRawlog, 以及很多定位及 SLAM 应用，都支持这种格式。

11.2-4 压缩 Rawlog 文件

所有 rawlog 文件都是使用 gzip 算法压缩。

压缩级默认设为“最小压缩”，以尽可能减少计算负载，文件大小压缩率约为 3。如果需要兼容旧的 MRPT 版本（<MRPT 0.6.0），可以将文件格式重命名为“.rawlog.gz”，然后用标准工具解压缩。

11.2-5 生成 Rawlog 文件

现在已有个独立应用，用来从一组机器人传感器抓取 rawlog：MRPT: rawlog-grabber

本节介绍在自己的源码中生成 rawlog 文件的常用方法，这种方法常用于对将现有数据集转为 rawlog 格式或从机器人传感器中在线抓取数据。

```
#include <mrpt/utils.h>
#include <mrpt/obs.h>

using namespace mrpt;
using namespace mrpt::utils;
using namespace mrpt::slam;
using namespace mrpt::poses;

int main()
{
CFileGZOutputStream    f("my_dataset.rawlog");

while (there_is_more_data)
{
   CActionCollection actions;
   CSensoryFrame    SF;
```

```
        // Fill out the actions:
        // ----------------------------------
        CActionRobotMovement2D myAction;    // For example, 2D odometry
        myAction.computeFromOdometry( ... );

        actions.insert( myAction );

        // Fill out the observations:
        // ----------------------------------
        CObservation2DRangeScanPtr myObs = CObservation2DRangeScan::Create();
        // CObservation2DRangeScanPtr    myObs = CObservation2DRangeScanPtr(new
CObservation2DRangeScan());      // This is an alternative to the line above

        myObs->...    // Fill out the data

        SF.insert( myObs );    // memory of "myObs" will be automatically freed.

        // Save to the rawlog file:
        // ----------------------------------
        f << actions << SF;
    };

    return 0;
    }
```

11.2-6 读取 Rawlog 文件

（6.1）选项 A：文件流

这一般是首选的操作模式：动作(actions)和观测(observations)从文件中被顺序读取、处理、内存释放，等等。这样，在任何时候只载入需要的对象，在管理大型数据时这是强制性的（比如包含数千帧嵌入的图像、数以百万的激光扫描数据）。

下面演示了按这种方式载入一个 rawlog 的循环过程：

```
    CFileGZInputStream      rawlogFile(filename);      // "file.rawlog"
    CActionCollectionPtr action;
    CSensoryFramePtr         observations;
    CObservationPtr              observation;
    size_t                          rawlogEntry=0;
    bool              end = false;

    // Read from the rawlog:
    while ( CRawlog::getActionObservationPairOrObservation(
              rawlogFile,          // Input file
```

```
            action,                 // Possible out var: action of a pair action/obs
            observations,   // Possible out var: obs's of a pair action/obs
            observation,    // Possible out var: a single obs.
            rawlogEntry     // Just an I/O counter
            ) )
{
// Process action & observations
if (observation)
{
  // Read a single observation from the rawlog (Format #2 rawlog file)
  ...
}
else
{
  // action, observations should contain a pair of valid data (Format #1 rawlog file)
  ...
}

};
// Smart pointers will be deleted automatically.
```

（6.2）选项 B：一次读取

一个 rawlog 文件可以使用类 slam::CRawlog 实现整个读取。

请注意，根据内存需求，对于非常大的数据集（例如：数十数百万条目）这可能是不切实际的，但是对于中等规模的数据集，它是肯定是载入 rawlogs 的最简单方法。

```
CRawlog dataset;
dataset.loadFromRawLogFile("file.rawlog");
cout << dataset.size() << " entries loaded." << endl;
```

11.2-7 关于里程数据的注意事项

请注意，里程数据的表示形式决定于 rawlog 格式是 FORMAT#1（传感帧 Sensor frames）还是 FORMAT#2（只有观测量）：

- **当使用格式 1（动作 Actions 及传感帧 Sensory frames）时：**里程计读数被存储为动作 action（MRPT::slam:: CActionRobotMovement2D），因此，存储的是里程增量数据，而不是里程计的绝对读数。
- **当使用格式 2（只有观测量 Observations）时：**里程计只是被当作普通传感器，这就是为什么在这种情况下，它被存储为 mrpt::slam:: CObservationOdometry 对象，包含了到目前为止的绝对里程位置读数。

第十二章 统计和贝叶斯滤波[Statistics and Bayes filtering]

12.1 平均似然对数值[Averaging Log-Likelihood Values]：数值稳定性[Numerical Stability]

12.1-1 加权似然对数值[Weighted log-likelihood values]

本节要解决的问题是计算：

$$\bar{l} = \frac{1}{\sum_i \omega_i} \sum_i \omega_i \cdot l_i$$

对动态扩展范围使用似然对数[log-likelihoods]（以及权重对数[log-weights]）是非常明智的。这可以避免权重变为零的问题。

然后，如果我们使用权重对数和各个似然对数作为输入，问题变为：

$$\log \bar{l} = \log \left(\frac{1}{\sum_i e^{l\omega_i}} \sum_i e^{l\omega_i} \cdot e^{ll_i} \right) = -\log \sum_i e^{l\omega_i} + \log \sum_i e^{l\omega_i + ll_i}$$

问题：非常 大/小 数值的指数将产生 下溢/上溢。

解决方法：将所有指数偏移，然后在对数和中校正。

$$l\omega_{max} = \max_i l\omega_i, \quad ll_{max} = \max_i ll_i$$

$$\log \bar{l} = -\log \sum_i e^{l\omega_i} + \log \sum_i e^{l\omega_i + ll_i}$$

$$= -\log \sum_i e^{l\omega_i - l\omega_{max}} + \log \sum_i e^{l\omega_i + ll_i - ll_{max} - l\omega_{max}} + ll_{max}$$

这个方法实现在 C++函数 mrpt::math::averageLogLikelihood.中。

12.1-2 未加权似然对数值（算术平均值）[Unweighted log-likelihood values]

如果所有采样的权重都相等，公式简化为：

$$\log \bar{l} = -\log N + \log \sum_{i=1}^{N} e^{ll_i - ll_{max}} + ll_{max}$$

12.2 卡尔曼滤波[Kalman Filters]

12.2-1 MRPT 中的卡尔曼滤波器

卡尔曼滤波算法（EKF（扩展卡尔曼滤波算法）,IEKF（迭代扩展卡尔曼滤波算法）...）都集中在 mrpt::bayes::CKalmanFilterCapable 这个单独的虚类中。这个类中包括系统状态向量和系统协方差矩阵，以及根据选择的算法执行一个完整迭代的通用方法。在实践中，解决一个特定问题时需要从这个虚类派生一个新的类，并实现一些新方法，比如通过传递模型转换状态向量，或者计算基于给定状态空间值的线性化观测模型的雅可比矩阵。

在这个类的成员 KF_options 中有一组问题独立的参数可以修改，其中最重要的参数是选择 KF 算法。目前，有下列选择 ：

- Naive EKF：基本的 EKF 算法。
- 迭代卡尔曼滤波算法（IKF）：该方法对状态向量的渐强精确值的雅可比矩阵进行重线性化。

这个类已被用于 6D-SLAM 的一个高效方案实现。

在 MRPT C++库中，贝叶斯滤波的另一种实现是粒子滤波器。

12.2-2 为特定问题写一个 KF 类

（2.1）派生新类

首先，必须从 mrpt::bayes::CKalmanFilterCapable 派生一个新的类。声明一个 public 方法作为用户操作入口，用于保存输入的数据，并调用父类中的 protected 方法 runOneKalmanIteration( )。

通过派生类用户可以构建两种基本系统类型：

- 普通问题：状态向量的大小不随时间改变。这个大小必须由虚方法“get_vehicle_size（）”返回。
- 地图问题：随着新的地图元素被导入，状态向量的大小随之增加。在这种情况下，第一个“get_vehicle_size()” 状态向量元素对应于 vehicle/robot/camera...的状态，其余状态向量是“get_feature_size()”子向量的全部数量，每个表示一个地图元素（路标，特征...）。

（2.2）算法内部流程

方法 mrpt::bayes::CKalmanFilterCapable::runOneKalmanIteration()会依次调用每个虚方法，根据下列顺序：

- 在 KF 预测阶段：
  - OnGetAction.
  - OnTransitionModel.
  - OnTransitionJacobian.
  - OnTransitionNoise.
- OnNormalizeStateVector：根据具体问题可选择实现。
- OnGetObservations：某些应用需要当前状态的估计值（比如在 MonoSLAM 中，去预测每个路标在图像中将出现的位置），这里是生成观测量的理想地方。在此，内部状态向量和协方差包含“先验分布”，即经过传递模型被更新。这里也是数据关联必须被解决的地方（在地图问题中）。

- 在 KF 更新阶段：
  - OnObservationModelAndJacobians：这个方法在 naive EKF 中会被调用一次，或者在 EKF a la Davison 被调用多次。
- OnNormalizeStateVector：
- 如果系统实现一个建图问题，而且“OnGetObservations”返回的数据关联表明了未建图观测量的存在，那么，对每个新特征点调用下面这个方法：
  - OnInverseObservationModel
- OnPostIteration：一个用户自定义代码区，放置用户希望在每次迭代后执行的代码（比如日志记录、可视化.....）。

12.2-3 一个示例

可以在示例目录 (samples/bayesianTracking) 内找到一个 KF 实现的例子，实现了从有噪声的方位数据中跟踪小车。这里将导出该实现所需的方程，以及它们在实际例子中如何实现。注意，在同一个示例中，这个问题也有粒子滤波实现，为的是一对一比较查看这两个方法的性能。

（3.1）问题描述

方位跟踪的问题是估算小车状态：

$$\mathbf{x} = \begin{pmatrix} x & y & v_x & v_y \end{pmatrix}$$

其中，x 和 y 是小车的直角坐标，Vx 和 Vy 是线速度。另外，我们使用一个简单的匀速模型，转移方程为：

$$\mathbf{x_k} = f(\mathbf{x_{k-1}}, \Delta t) = \begin{cases} x_{k-1} + v_{x_k}\Delta t \\ y_{k-1} + v_{y_k}\Delta t \end{cases}$$

观测向量 Z=(Zb,Zr)包含小车距离同一个点（任意设为原点）的方向和距离：

$$z_b = atan2(y, x)\; z_r = \sqrt{x^2 + y^2}$$

然后，直接获得需要的转移方程的雅可比矩阵：

$$\frac{\partial f}{\partial x} = \begin{pmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

以及观测模型的雅可比矩阵：

$$\frac{\partial h}{\partial x} = \begin{pmatrix} \frac{-y}{x^2+y^2} & \frac{1}{x\left(1+\left(\frac{y}{x}\right)^2\right)} & 0 & 0 \\ \frac{x}{\sqrt{x^2+y^2}} & \frac{y}{\sqrt{x^2+y^2}} & 0 & 0 \end{pmatrix}$$

（3.2）实现

最重要的实现方法详述如下。关于更多详情请参阅示例“bayesianTracking”的源代码。

（3.2.1）转移模型

匀速模型被简易实现为：

```
/** Implements the transition model $\hat{x}_{k|k-1} = f(\hat{x}_{k-1|k-1}, u_k)$
 * \param in_u The vector returned by OnGetAction.
 * \param inout_x At input has $\hat{x}_{k-1|k-1}$, at output must have $\hat{x}_{k|k-1}$.
 * \param out_skip Set this to true if for some reason you want to skip the prediction step (to do not modify either the vector or the covariance). Default:false
 */
void CRangeBearing::OnTransitionModel(
        const KFArray_ACT &in_u,
        KFArray_VEH       &inout_x,
        bool &out_skipPrediction
        ) const
{
        // in_u[0] : Delta time
        // in_out_x: [0]:x   [1]:y   [2]:vx   [3]: vy
        inout_x[0] += in_u[0] * inout_x[2];
        inout_x[1] += in_u[0] * inout_x[3];
}
```

（3.2.2）转移模型雅可比矩阵

```
/** Implements the transition Jacobian $\frac{\partial f}{\partial x}$
 * \param out_F Must return the Jacobian.
 *   The returned matrix must be $N \times N$latex with N being either the size of the whole state vector or get_vehicle_size().
 */
void CRangeBearing::OnTransitionJacobian(KFMatrix_VxV   &F) const
{
        F.unit();

        F(0,2) = m_deltaTime;
        F(1,3) = m_deltaTime;
}
```

（3.2.3）观测及观测模型

```
void CRangeBearing::OnGetObservationsAndDataAssociation(
```

```
        std::vector<KFArray_OBS>       &out_z,
        vector_int                     &out_data_association,
        const vector<KFArray_OBS>      &in_all_predictions,
        const KFMatrix                 &in_S,
        const vector_size_t            &in_lm_indices_in_S,
        const KFMatrix_OxO             &in_R
        )
    {
        out_z.resize(1);
        out_z[0][0] = m_obsBearing;
        out_z[0][1] = m_obsRange;

        out_data_association.clear(); // Not used
    }


    /** Implements the observation prediction . h_i(x).
       * \param idx_landmark_to_predict The indices of the landmarks in the map whose predictions are expected as output. For non SLAM-like problems, this input value is undefined and the application should just generate one observation for the given problem.
       * \param out_predictions The predicted observations.
       */
    void CRangeBearing::OnObservationModel(
        const vector_size_t            &idx_landmarks_to_predict,
        std::vector<KFArray_OBS>    &out_predictions
        ) const
    {
        // predicted bearing:
        kftype x = m_xkk[0];
        kftype y = m_xkk[1];

        kftype h_bear = atan2(y,x);
        kftype h_range = sqrt(square(x)+square(y));

        // idx_landmarks_to_predict is ignored in NON-SLAM problems
        out_predictions.resize(1);
        out_predictions[0][0] = h_bear;
        out_predictions[0][1] = h_range;
    }
```

12.3 粒子滤波算法

本节介绍 MRPT C++库中实现的粒子滤波算法的理论背景。也会看到不同的重采样机制。对应的 C++类见 Particle Filters.

12.3-1 顺序重要性重采样[Sequential Importance Resampling] - SIR（pfStandardProposal）

标准建议分布+基于似然函数的权重。

12.3-2 辅助粒子滤波[Auxiliary Particle Filter] - APF（pfAuxiliaryPFStandard）

这个方法是由 Pitt 和 Shephard 在 1999 年提出[1]。

我们假设滤波后验是由下面的加权样本描述的：

$$p(x_t|z_{1:t}) \approx \sum_{i=1}^{M} \omega_t^{(i)} \delta\left(x_t - x_t^{(i)}\right)$$

然后，该算法的每一步都包括先抽取一个粒子索引样本 k，它将从第(t-1)步传播到第(t)步。这些粒子索引是临时变量，仅用作中间步骤，算法名字由此得来。索引是根据某些参考点$u^{(i)}$的似然率抽取的，某种程度上和转移模型$(x_t \mid x_{(t-1)})$有关系（比如，均值、样本.....）

$$k^{(i)} \sim P(i = k|z_t) \propto \omega_t^{(i)} p(z_t|\mu_t^{(i)})$$

对（i=1,2,....）重复。我们现在可以用这些索引来抽取条件样本：

$$x_t^{(i)} \sim p(x_t|x_{t-1}^{k^{(i)}})$$

最终，权重被更新，以考虑实际采样的似然率和预测点之间的不匹配。

$$\omega_t^{(i)} \propto \frac{p(z_t|x_t^{(i)})}{p(z_t|\mu_t^{k^{(i)}})}$$

12.3-3 最优采样[Optimal Sampling]（pfOptimalProposal）

使用精确最优建议分布（如果有的话！通常使用近似）。在 RBPF-SLAM 的实现中，这个方法基于文献[2]。

12.3-4 逼近最优采样[Approximate Optimal Sampling]（pfAuxiliaryPFOptimal）

使用最优采样和一个辅助粒子滤波器（见文献[3]）。

参考文献

- [1] Pitt, M.K. and Shephard, N.,

[http://www.nuff.ox.ac.uk/economics/papers/1997/w13/sir.pdf Filtering Via Simulation: Auxiliary Particle Filters], Journal of the American Statistical Association, vol. 94, pp. 590-591, 1999.

- [2] Grisetti, G. and Stachniss, C. and Burgard, W., “Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling”, ICRA 2005.
- [3] J.-L. Blanco, J. González, and J.-A. Fernández-Madrigal,An Optimal Filtering Algorithm for Non-Parametric Observation Models in Robot Localization, IEEE International Conference on Robotics and Automation (ICRA), Pasadena (California, USA), May 19-23, 2008, pp. 461 - 466.

12.4 粒子滤波器

下面的 C++类是所有 MRPT 中实现的各个粒子滤波器类的基类。

- mrpt::bayes::CParticleFilterCapable:这个虚类定义了接口，任何基于粒子的 PDF 类都必须实现这个接口以便能被 CParticleFilter 执行。
- mrpt::bayes::CParticleFilter:这个类对一个 CParticleFilterCapable 对象执行迭代。
- mrpt::bayes::CparticleFilterData：这个容器模型被某些类使用，但并不一定使用粒子滤波进行处理。它只是表示一套 C++模板，用于任意给定类型的加权样本组，以及一般的内存管理任务。

无论是要运行的指定“粒子滤波算法”还是要使用的“重采样机制”，都可以在选项结构体 mrpt::bayes::CParticleFilter::TParticleFilterOptions 中单独选定。

- **粒子滤波算法** – 可参见该算法介绍。
  - pfStandardProposal：标准建议分布+基于似然函数的权重。
  - pfAuxiliaryPFStandard：一种辅助粒子滤波，使用标准建议分布。
  - pfOptimalProposal：使用精确最优建议分布（如果有的话！通常使用近似）。
  - pfAuxiliaryPFOptimal：使用最优建议分布和一种辅助粒子滤波（见文献）。
  - 此外，adaptiveSampleSize 可以选择是否使用动态样本数。需要注意，对某个问题可能只实现了所有算法组合的一个子集。
- **重采样机制** – 可参见该算法介绍。
  - prMultinomial（默认）：使用标准的选择和替换（抽取 m 个随机统一编号）
  - prResidual：剩余或“余数”的方法。
  - prStratified：分层重采样，对范围（0,1）中每个分支抽取统一样本。
  - prSystematic：在范围（0,1/ M]中的单个统一样本被抽取。

2D 追踪示例可查看： samples/bayesianTracking 。

12.5 空间表征概率密度分布

在很多地位和 SLAM 算法中，这些分布是机器人和地图元素的基本表示形式。

包括单高斯、高斯和、粒子集、栅格表示，以及它们之间的转换的方法，从任意分布中抽取任意数量样本的方法。查看下列类：

- poses::CPointPDF for 3D points.
- poses::CPosePDF for 2D + heading poses.
- poses::CPose3DPDF for 6D (3D + yaw,pitch,roll) poses.
- poses::CPose3DQuatPDF for 7D (3D + quaternion) poses.

上面这些抽象类，每个都实现了不同的 PDF 表示（粒子、单高斯、高斯和.....）。

下面这篇技术报告里有一些实现雅可比矩阵的推导：

J.L. Blanco, "*6D poses as Euler angles, transformation matrices and quaternions: equivalences, compositions and uncertainty*".

12.6 重采样机制

12.6-1 重采样算法

无论是要运行的指定“粒子滤波算法”还是要使用的“重采样机制”，都可以在选项结构体 mrpt::bayes::CParticleFilter::TParticleFilterOptions 中单独选定。

本节回顾四种不同的用于重采样一组粒子的策略，粒子的规范化权重由 ω [i] for i=1,…,N 给定。

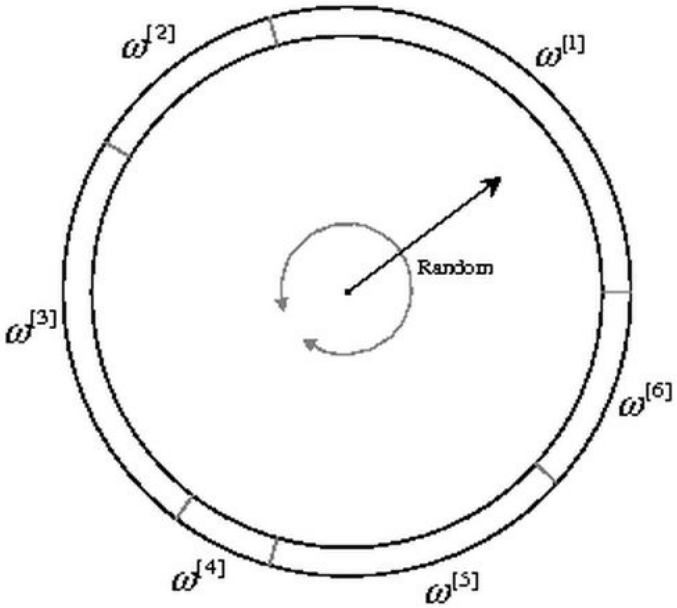
解释方法是使用一个“轮”作为可视化类比，它的周长被分配到不同的粒子。在这种方式下，关联到第“i”个粒子的弧长正比于它的权重ω[i]。因此，在“轮”中选择一个随机指向意味着选择一个例子的概率正比于其权重。

关于该方法的更正式介绍，请参阅 Douc, Cappé and Moulines.的优秀论文。

（1.1）prMultinomial (默认)

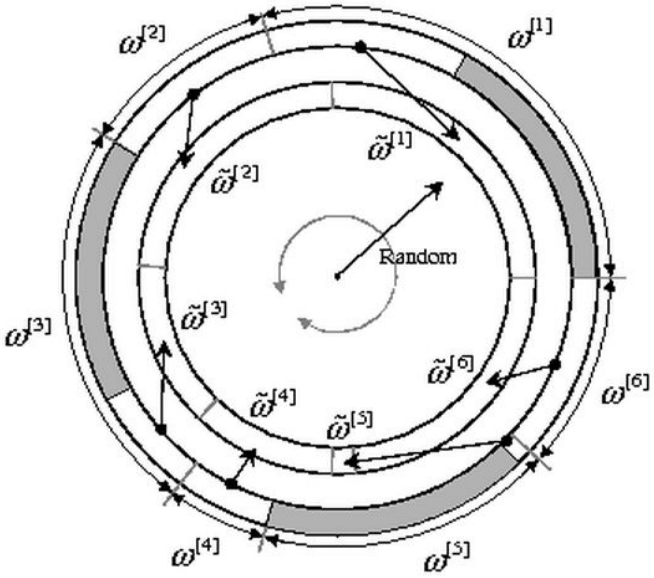
最简单直接的方法，生成 N 个独立随机数用于从旧粒子集中挑选粒子。在“轮”类比中，如下图所示，该方法包括挑出 N 个独立随机指向。

这种方法的名字来源于一个事实，即重复计数 Ni 概率密度函数是以权重作为参数的多项分布。



（1.2）prResidual

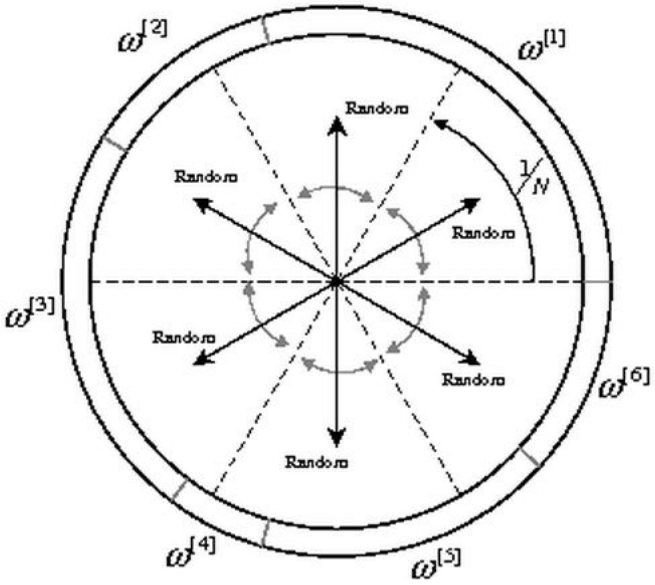
该方法包括两个阶段。首先，通过挑选第“i”个粒子的$N_i = \lfloor N\omega^{[i]} \rfloor$副本得到确定性采样粒子。然后，使用剩余权重$\tilde{\omega}^{[i]} = \omega^{[i]} - N_i/N$进行多项采样。



（1.3）prStratified

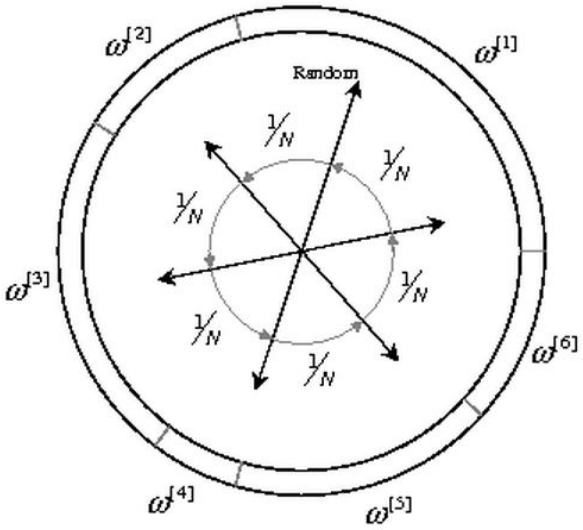
在该方法中，代表旧粒子集的“轮”被分成 N 个相等大小的段，如下图所示。

然后，类似多项采样独立生成 N 个统一编号，并不是将每个抽取都映射到整个圆周上，而是映射到对应的分区。



（1.4）prSystematic

也被称为“通用采样”，这种流行的技术只抽取一个随机数，即“轮”中的一个方向，其它 N-1 个方向被固定为从随机方向开始的 1/N 递增。



12.6-2 MATLAB 版本

这些算法的 MATLAB 版本已被出版在：

http://www.mathworks.com/matlabcentral/fileexchange/24968

参考文献

- Douc, R. and Cappe, O. and Moulines, E., “Comparison of Resampling Schemes for Particle Filtering”, Image and Signal Processing and Analysis, 2005.
- Blanco, J.L., Contributions to Localization, Mapping and Navigation in Mobile Robotics, PhD Thesis.

第十三章 有用的 MRPT 宏(C++预处理器)

13.1 用于异常处理的宏

13.1-1 异常类别

可以被 MRPT API 引发的所有异常都派生自 C++标准的 std::exception，所以如果你想捕获所有这些异常，你的代码应当含有（至少在顶层，比如，main()中）下面的 try/catch 块：

```
int main(int argc, char **argv)
{
 try
 {
    // do whatever...
    return 0;
 }
 catch(std::exception &e)
 {
     std::cerr << e.what() << std::endl;
     return 1;
 }
}
```

实际上，所有 MRPT 异常类型都是这些类型之一（派生自 std::exception）：

- mrpt::utils::CMRPTException
- std::logic_error

所以，如果你想对一个特殊类型异常用不同方式处理，可以按下面的做：

```
int main(int argc, char **argv)
{
 try
 {
 // do whatever...
 return 0;
 }
 catch(mrpt::utils::CExceptionEOF &e)
 {
    // This was a EOF error
    std::cerr << e.what() << std::endl;
    return 1;
 }
 catch(std::exception &e)
 {
```

```
    // All other errors
    std::cerr << e.what() << std::endl;
    return 1;
  }
}
```

13.1-2 异常引发宏（似断言宏）

```
// A compile-time assertion
MRPT_COMPILE_TIME_ASSERT( condition )

// A softer alternative to C assert() that doesn't
// abort the program, but raises an std::exception
ASSERT_( condition )

// As ASSERT_() but the exception will contain the given custom message
// instead of a textual representation of "condition"
ASSERTMSG_( condition, "custom message" )

// As ASSERT_(), but on fail displays a message like:
// ASSERT_EQUAL(w.size(),v.size()) failed with
// w.size()=10
// v.size()=12
ASSERT_EQUAL_( a, b )
ASSERT_NOT_EQUAL_( a, b )
ASSERT_BELOW_( a, b )
ASSERT_ABOVE_( a, b )

// As ASSERT_() but only has effect in DEBUG builds.
ASSERTDEB_( condition )

// As ASSERTMSG_() but only has effect in DEBUG builds.
ASSERTDEBMSG_( condition, "custom message" )

// Raises a std::logic_error exception with the custom message
// and the source code file name, line number, and function name
THROW_EXCEPTION("message")

// Just like THROW_EXCEPTION() but the message has one printf-like format
// specifier which is expanded with the second argument
THROW_EXCEPTION_CUSTOM_MSG1("message %i", myVar )

// Checks for the existence of a file, raising an exception with an
```

```
// explanatory message otherwise
ASSERT_FILE_EXISTS_(fileName)
```

注意，不需要在上述宏的后面加分号（;）!

13.1-3 用于异常传递的特殊宏

大多数 MRPT 函数和方法使用 MRPT_START/ MRPT_END 宏，它使能（除其它之外）自动显示某异常的逆调用堆栈，异常的 e.what（）字符串通常包括堆栈跟踪，比如：

```
================ MRPT EXCEPTION ==============
size_t mrpt::math::KDTreeCapable::kdTreeClosestPoint2D(float, float, float&, float&,
float&) const, line 201:
Assert condition failed: x0==10
virtual void mrpt::slam::CPointsMap::computeMatchingWith2D(const
mrpt::slam::CMetricMap*, const mrpt::poses::CPose2D&, float, float, const
mrpt::poses::CPose2D&, mrpt::utils::TMatchingPairList&, float&, float*, bool, bool) const, line
586:
mrpt::poses::CPosePDFPtr mrpt::slam::CICP::ICP_Method_Classic(const
mrpt::slam::CMetricMap*, const mrpt::slam::CMetricMap*, const
mrpt::poses::CPosePDFGaussian&, mrpt::slam::CICP::TReturnInfo&), line 562:
virtual mrpt::poses::CPosePDFPtr mrpt::slam::CICP::AlignPDF(const
mrpt::slam::CMetricMap*, const mrpt::slam::CMetricMap*, const
mrpt::poses::CPosePDFGaussian&, float*, void*), line 104:
void mrpt::slam::CMetricMapBuilderICP::processObservation(const
mrpt::slam::CObservationPtr&), line 376:
void MapBuilding_ICP(const std::string&), line 442:
```

13.2 关于宏 MRPT_START / MRPT_END

大多数 MRPT 函数和方法在它们代码里会有一对宏 MRPT_START 和 MRPT_END：

```
void Foo::method()
{
 MRPT_START
   ....
 MRPT_END
}
```

这些宏同时有两个目的：

- **异常传递**：构建异常的反向调用堆栈的文字表述。
- **全局分析**：如果 CMake 标志 MRPT_ENABLE_EMBEDDED_GLOBAL_PROFILER 启用，所有 MRPT_START/ MRPT_END 块将自动被分析，并在任何程序的执行后，一个包含统计信息的大表将被注入到控制台，包括所有函数的调用次数，最小、平均、最大执行次数。

用户也可以把这两个宏放在他们自己的代码中，以便将上面这两个效用延伸到他们自

己的程序中。