面向自动驾驶的C++实战
第5章 自动驾驶常用工具库

主　讲：姜朝峰

公众号：自动驾驶之心

自动驾驶之心
全栈知识矩阵

自动驾驶之心，专注自动驾驶与AI

公众号

干货每日放送

B站

不定期直播+最新视频

知识星球

海量图文教程和Paper分享

视频号

技术视频一站式分享

主要内容

1. 实践与领域知识
2. EIGEN库
3. APOLLO基础工具库
4. 其他知名库介绍

1 实践与领域知识

1. 实践与领域知识

1.1 我为什么还是看不懂C++代码?

- 我开始学习了计算机基础知识，为什么完全看不懂代码？——**不明白C++语法**
- 我已经学习了C++的主要语法，为什么还是看不懂大部分代码？——**不熟悉标准库**
- 我继续学习了C++ STL，为什么还是有些语句不明白？——**不了解第三方库**
- 我接着学习了用到的第三方库，为什么还是不理解某些类的意义？——**不了解编程范式和设计模式**
- 我跟随学习了编程范式和设计模式，为什么还是看不懂代码的意图？——**不了解项目的领域知识**
- 我最后学习了项目的领域知识，为什么还不了解整个项目的设计？——**不了解项目的整体设计方案**

1. 实践与领域知识

1.2 编码实践需要的知识

- **计算机知识：**包括操作系统、计算机组成、计算机网络、数据库等主要知识。
- **C++语法**
- **标准库和第三方库：**除了C++语言本身的语法，很多C++代码还大量使用标准库和第三方库。如果不了解这些库的用法，理解代码会有困难。
- **编程范式：**不同的代码风格和范式会影响代码的组织方式和表达方式。
- **设计模式：**复杂的C++项目常常采用各种设计模式，掌握常见设计模式也有助于理解代码架构。
- **领域知识：**C++应用于各个领域，不同领域的代码会涉及特定的名词、算法、数据结构、架构模式等。
- **项目设计：**大型项目的需求分析、方案设计、代码结构、模块划分等信息也会影响代码的可读性。

1. 实践与领域知识

1.3 领域知识

各个领域的知识

是指在特定领域或行业中所需要的专业知识、经验和技能，每个领域都有自己独特的概念、术语、理论、方法和工具。一些常见的领域及其所需的知识如下示例：

- **医疗健康：**解剖学、生理学、药理学、疾病诊断和治疗等。
- **金融：**财务报表分析、风险管理、投资策略、交易流程、金融工具等。
- **法律：**法律条文、案例分析、诉讼流程、合同拟定等。
- **机械/材料工程：**材料力学、热力学、流体力学、机械设计等。

软件开发人员需要对其所开发软件所应用的特定领域的有一定的理解。例如，开发医疗软件需要了解医疗流程、术语和规范；开发金融交易系统需要熟悉金融领域的业务逻辑和规则。

1. 实践与领域知识

1.3 领域知识

互联网领域知识范畴

1. **网络协议**：互联网的运作基于一系列网络协议，如TCP/IP、HTTP、FTP等。理解这些协议的原理和实现对于开发网络应用程序至关重要。
2. **数据库**：互联网应用通常需要存储和管理大量数据，这需要熟悉关系型数据库（如MySQL）和非关系型数据库（如MongoDB）的设计和使用。
3. **网络安全**：互联网应用需要重视安全性，防范各种攻击和威胁，如SQL注入、跨站脚本攻击、DDoS攻击等。这需要了解加密、认证、授权等安全技术。
4. **用户体验**：互联网应用需要关注用户体验，提供直观、流畅、个性化的交互。这需要了解用户研究、交互设计、视觉设计等方面的知识。
5. **网络营销**：互联网已成为重要的营销渠道，网络营销需要了解搜索引擎优化、社交媒体营销、网络广告等方面的知识和策略。
6. ……

1. 实践与领域知识

1.3 领域知识

自动驾驶领域知识范畴

1. **车辆工程**：自动驾驶系统需要深入理解车辆的各个组成部分，如动力系统、制动系统、转向系统等的工作原理和控制方法。
2. **机器视觉**：自动驾驶汽车需要图像处理、目标检测、跟踪和识别等机器视觉技术。
3. **路况和交通规则**：自动驾驶系统需要理解和遵守交通规则，如红绿灯、车道线、限速标识等，同时还要能够处理复杂的路况，如拥堵、事故、恶劣天气等。
4. **地图和定位**：自动驾驶汽车需要精确的地图和定位系统，以确定自己的位置和规划行驶路线。这需要卫星定位、惯性导航、激光雷达定位等技术。
5. **人机交互**：自动驾驶系统需要与乘客进行有效的交互，如语音控制、行车现实、紧急接管等。
6. ……

每个领域都有一系列**特定的**概念、理论、技术和方法，构成了该**领域的知识体系**。掌握这些领域知识，才能读懂实际的代码。

2 EIGEN库

2. Eigen库

2.1 简介

线性代数在自动驾驶中的应用

1. **坐标转换：**自动驾驶需要在不同坐标系（如车辆坐标系、世界坐标系、相机坐标系等）之间进行转换，这通常使用旋转矩阵、平移向量等线性代数工具实现。
2. **地图构建与位姿估计：**许多SLAM算法如图优化、粒子滤波等，都离不开矩阵和向量的运算，卡尔曼滤波、非线性优化等姿态估计算法也通常使用矩阵和向量来表示状态和观测。**例如，**图优化SLAM使用信息矩阵来表示位姿和地图特征之间的约束关系，扩展卡尔曼滤波EKF使用雅可比矩阵来线性化非线性系统，并使用矩阵运算进行状态预测和更新。
3. **运动规划：**自动驾驶需要规划车辆的运动轨迹，避免碰撞，达到目标位置。运动规划算法如基于采样的方法、优化型方法等，通常使用矩阵和向量来表示状态空间、约束条件、代价函数等。**例如，**模型预测控制MPC算法使用矩阵不等式来表示车辆动力学约束，使用二次型矩阵来表示控制目标的代价函数。
4. **感知和识别：**自动驾驶需要从传感器数据中感知环境，识别车道线、交通标志、障碍物等。许多感知算法如卷积神经网络（CNN）、支持向量机（SVM）等，都使用矩阵运算来实现特征提取和分类决策。**例如，**CNN使用卷积矩阵和池化矩阵来实现图像特征的提取和下采样。
5. ……

可以看到，线性代数几乎渗透到自动驾驶的每个方面。矩阵和向量不仅是表示数据的基本工具，也是实现各种算法的数学基础。

2. Eigen库

2.1 简介

其他语言对线性代数计算的支持

许多编程语言都有优秀的线性代数库，比如：

- **Python**
  - **NumPy**：事实上的Python科学计算标准库，提供了强大的多维数组对象和相关的线性代数运算。
  - **SciPy**：基于NumPy，提供了更高层的科学计算算法，包括优化、线性代数、积分等。
  - **生态位**：Python凭借在数据分析、机器学习领域的广泛应用，也拥有了非常完善和流行的线性代数计算生态。
- **MATLAB**
  - **内置支持**：MATLAB本身就是为线性代数和数值计算设计的，内置了丰富的矩阵运算和线性代数函数。
  - **LAPACK和BLAS**：MATLAB的许多线性代数函数都是基于这两个底层Fortran库实现的。
  - **生态位**：在科学计算和数值分析研究领域，MATLAB和Fortran传统上拥有最成熟、丰富的线性代数库。
- **R**
  - base包：R的基础包提供了向量、矩阵数据类型和基本的线性代数运算。
  - Matrix包：提供了更全面的稠密和稀疏矩阵运算，以及矩阵分解等高级功能。

而C++作为在工程实现和产品落地方面具有显著优势的语言，也有了自己的主流线性代数计算库——**Eigen**。

2. Eigen库

2.1 简介

概述

Eigen是一个开源的C++模板库，主要用于线性代数计算。它提供了一整套高效、灵活的向量、矩阵及相关算法。

1. Eigen是**纯粹的模板库，只包含头文件**，没有链接的代码，使用方便。
2. 支持动态矩阵和静态矩阵，矩阵大小可在**编译时确定**，减小开销。
3. 语法直观，**接近数学表达**。支持各种矩阵运算和数值分析算法。
4. 支持多种数值类型，包括整型、浮点型和复数等。
5. 支持密集矩阵和稀疏矩阵存储，以及特殊矩阵如对角阵等。
6. 通过表达式模板等技术优化，运行效率很高。
7. 经过严格测试，有很好的数值稳定性。可移植性强，支持多种编译器和平台。

线性代数就是领域知识，Eigen是领域知识在C++的落地实现；

作为库使用者，需要更多了解**领域知识**；作为库开发者，需要更多研究如何优化。

2. Eigen库

2.1 简介

应用领域

1. **计算机视觉**：如相机标定、3D重建、物体跟踪、图像配准等。
2. **机器人学**：机器人运动学、动力学建模、SLAM、机器人控制等。
3. **游戏和虚拟现实**：3D渲染、物理模拟等。
4. **机器学习**：特征提取、降维、聚类等基本算法，深度学习框架的底层实现。
5. **科学计算**：物理仿真、金融工程、生物信息等领域的数值计算。

2. Eigen库

2.2 Eigen的使用

- **Overview概览**
  - 为了解决什么问题而诞生的？
  - 大概有哪些优缺点？

- **Get Started入门**
  - Download&Install下载安装
  - Simple Demo简单示例
  - Explanation基本说明

- **Specification详细规范**
  - 介绍基本的数据对象
  - 矩阵和数组的各种操作
  - 矩阵的线性问题求解和矩阵分解
  - 稀疏矩阵相关的话题
  - 几何操作相关的话题
  - ……

2. Eigen库

2.2 Eigen的使用

- **Other Topics拓展话题**
  - 拓展Matrix类的定义
  - 拓展基础数据类型
  - 更好的性能
  - 断言
  - 多线程处理
  - 在CUDA中使用
  - ……
- **代码API说明**

官方文档：https://eigen.tuxfamily.org/index.php?title=Main_Page#Documentation

使用思路：

- 实际问题 -> 选择算法 -> 具体实现过程 -> 具体工具
- 图像处理、位姿变换等 -> Principle/Physics/Learning based algorithm -> 某些线性代数计算和求解 -> Eigen

3 APOLLO基础工具库

3. Apollo基础工具库

3.1 日志（log）库

- 基于Google Log（glog）
- 提供**宏定义**来简化日志记录语句的编写
- Log库提供了灵活的日志记录功能，支持不同级别的日志（如INFO、WARNING、ERROR等）
- 可以方便地配置日志格式、输出目标（如控制台、文件）等
- 支持条件日志记录，即只有当条件满足时才记录日志，避免过多的日志输出影响性能

Code: https://github.com/ApolloAuto/apollo/blob/master/cyber/common/log.h

关于**Google Log**

- Github Repo: https://github.com/google/glog
- Document: https://google.github.io/glog/stable/

3. Apollo基础工具库

3.2 Time库

人人都知道**时间**的概念，但是却没有意识到，我们有时间点、时间段、周期、定时器等多种丰富的需求。

- **Time时间**
  - 将某个特定的时间信息封装到Time类中。
  - 提供加减、计算等各种常用操作。
  - 方便不同单位、不同表示（Unix时间戳和自然时间）的时间互相转换。
  - 提供时间的表示、时间的计算、时间的格式化输出等。
- **Clock时钟**
  - 修改时钟的时间
  - 查看/获取时钟的当前时间
- 特殊的时间表示
  - **Duration时长/时间段**——对单个时间段做加减计算
  - **Rate周期**——多次循环中，保持一个固定的时间间隔
- **定时器Timer**
  - 单个定时任务，到达特定时间或每隔一段时间，触发某个事件
  - 多个定时器，多个时间触发——TimeWheel时间轮

代码位置

- **STL chrono: Date and Time Utility, https://cplusplus.com/reference/chrono/**
- https://github.com/ApolloAuto/apollo/blob/master/cyber/time
- https://github.com/ApolloAuto/apollo/blob/master/cyber/timer

3. Apollo基础工具库

3.3 Math库

Math库提供了常用的数学表达、函数和运算。

- **几何计算（Geometry）：** 提供了基本的几何形状和操作，如点（Point）、线段（Line Segment）、矩形（Bounding Box）、多边形（Polygon）、圆（Circle）等，提供几何变换，如平移、旋转、缩放等。
- **坐标转换（Coordinate Transformations）：** 提供了各种坐标系之间的转换，如世界坐标系、局部坐标系、车体坐标系等。
- **轨迹和路径（Trajectory and Path）：** 提供了轨迹生成和优化工具，支持路径插值、平滑、拟合等操作。
- **随机过程和概率（Random Processes and Probability）：** 提供随机数生成、概率分布等，支持贝叶斯滤波、卡尔曼滤波等常用的概率算法。
- **统计分析（Statistical Analysis）：** 提供常用的统计分析工具，如均值、方差、协方差、相关系数等，支持数据拟合、回归分析等。
- **优化算法（Optimization Algorithms）：** 提供常见的优化算法，如基本的线性规划、非线性规划、约束优化等。
- **矩阵和线性代数（Matrix and Linear Algebra）**

代码位置

- https://github.com/ApolloAuto/apollo/blob/master/modules/common/math/

3. Apollo基础工具库

3.4 其他Util库

其他功能实现

- **复杂的数据结构**，如无锁哈希表AtomicHashMap、线程安全读写锁AtomicRWLock、环形队列BoundedQueue、对象池ObjectPool、线程池ThreadPool、线程安全队列ThreadSafeQueue等。
- **各种实用的工具函数和类**，如特定格式文件的读写、获取环境变量信息、字符串处理、编码转换、哈希计算等常用功能。

Utility和Tool的区别

- Util(Utility)
  1. 目录中的代码通常是**通用的**、**可重用的**功能模块。
  2. 目录中的代码通常是**函数库**或**类库**，提供辅助功能。
  3. 项目的各个部分可能会广泛**依赖**、使用 util 目录中的代码。
  4. 示例：string_utils.cpp/.h: 字符串处理函数，file_utils.cpp/.h: 文件操作函数，需要被其他代码依赖和引用，不会单独编译。
- Tool
  1. 目录中的代码通常是**独立的**工具或程序，执行**特定的**、**不重复的**任务或功能。
  2. 目录中的代码通常是完整的**程序**或**脚本**，能够**独立运行**。
  3. 工具通常可以**依赖**项目中的**其他**代码，不一定被其他代码所依赖。
  4. 示例：data_generator.cpp/.h: 数据生成工具；benchmark_tool.cpp/.h: 性能测试工具，编译后单独生成一个binary，可以被独立启动执行。

3. Apollo基础工具库

3.4 其他Util库

使用

在项目中，各种utility库是通过明确的模块化设计和依赖管理被其他库或binary引用和使用的。

1. **库的链接。**在构建过程中，通过构建系统将不同的库链接在一起。这样，在最终生成的binary中，各个模块可以互相调用对方的函数和类。
   - **CMake构建系统。**在CMake文件中，不同模块通过添加依赖来引用utility库。
   - **Bazel构建系统。**通过Bazel，依赖关系在BUILD文件中定义。

```
// CMakeLists.txt
add_library(my_module
    my_module.cc
)

target_link_libraries(my_module
    common::util
)
```

```
// BUILD
cc_library(
    name = "my_module",
    srcs = ["my_module.cc"],
    deps = [
        "//modules/common/util:string_util",
    ],
)
```

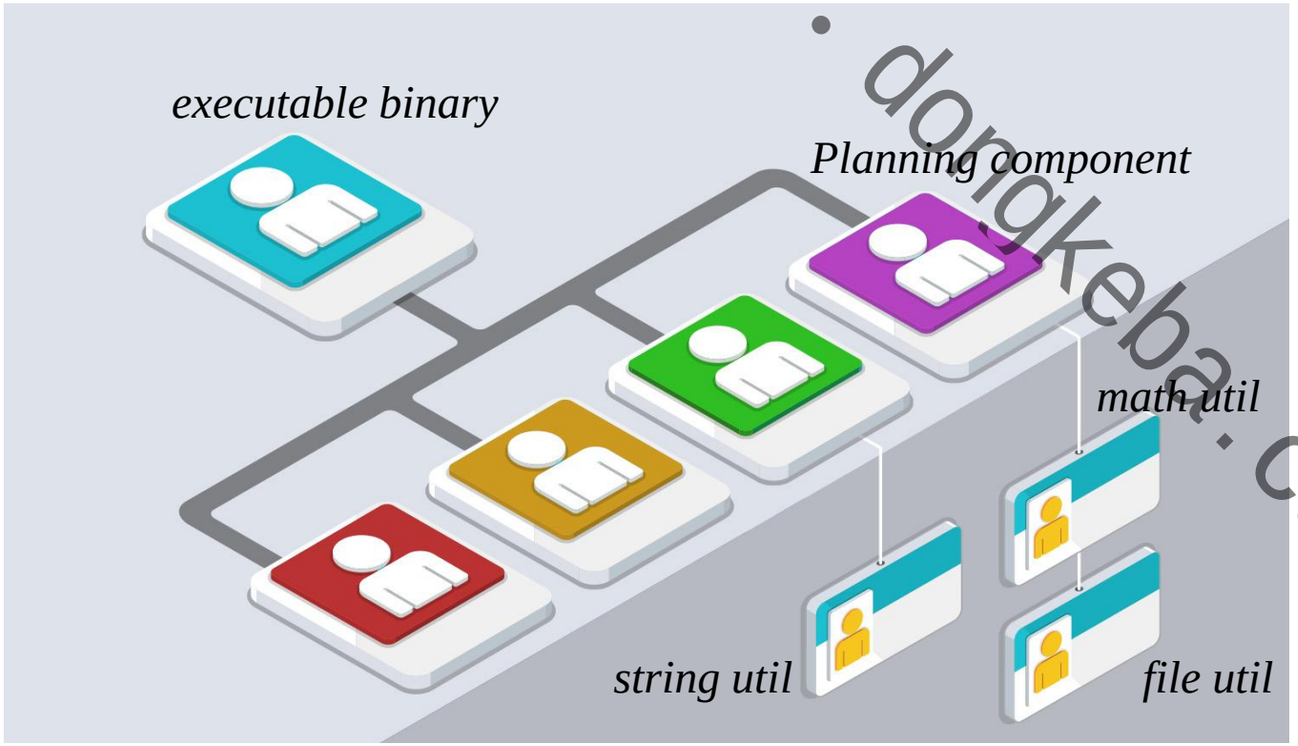
2. **头文件引用。**每个utility库通常会有对应的头文件，这些头文件定义了库中的函数和类接口。其他模块通过包含这些头文件来使用库中的功能。
3. **函数调用。**

```
#include "modules/common/util/string_util.h"
void MyFunction() {
    std::string result =
apollo::common::util::SomeUtilityFunction();
}
```

3. Apollo基础工具库

3.4 其他Util库

使用



Visualize Tree Structures, link

Burj Khalifa Building, in Dubai, link

4 其他知名库介绍

4. 其他知名库介绍

4.1 OpenCV

OpenCV（Open Source Computer Vision Library）是一个以**C++**语言开发的**计算机视觉**库。

主要支持：

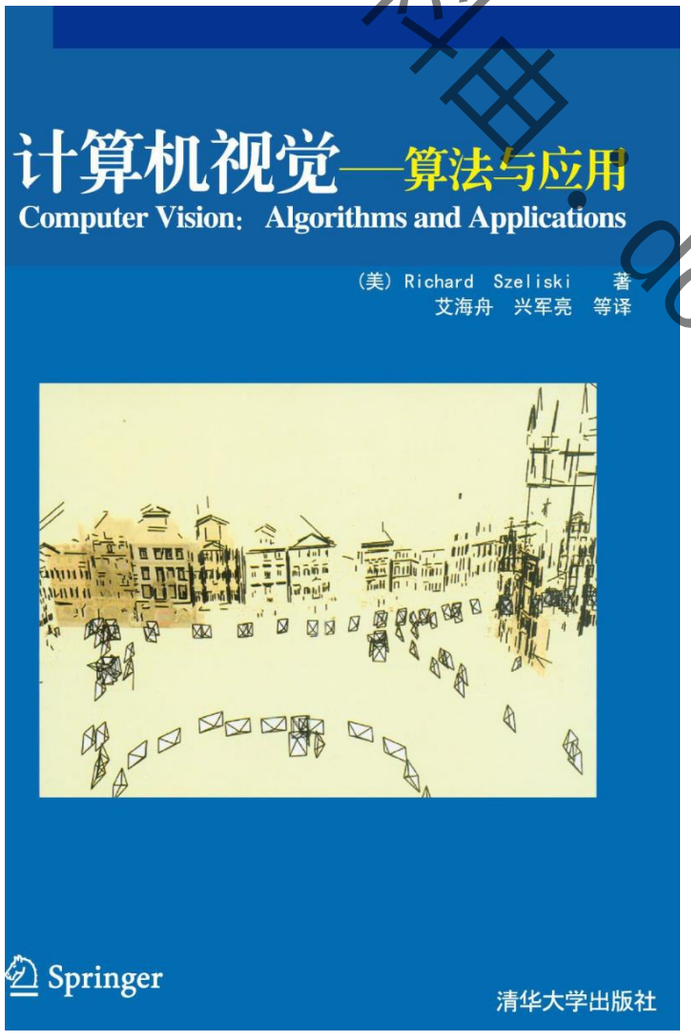
- **图像处理：**支持基本的图像处理操作，如图像变换、色彩空间转换、滤波、几何变换等。
- **目标检测和跟踪：**提供了多种目标检测和跟踪算法，如Haar特征分类器、HOG特征和CNN等。
- **特征提取和描述：**支持关键点检测（如SIFT、SURF、ORB）和描述子生成，用于图像配准和物体识别。
- **机器学习支持：**集成了一些机器学习算法，如k最近邻（kNN）、支持向量机（SVM）等，用于图像分类和识别任务。
- **摄像头标定和视觉几何：**提供了标定工具和几何变换算法，支持摄像头标定、立体视觉和结构光等应用。
- **图像和视频的I/O操作：**支持从文件、摄像头和网络流中读取和写入图像和视频数据。
- **并行处理和优化：**利用多线程和硬件加速（如CUDA）进行性能优化，加快图像处理和计算速度。

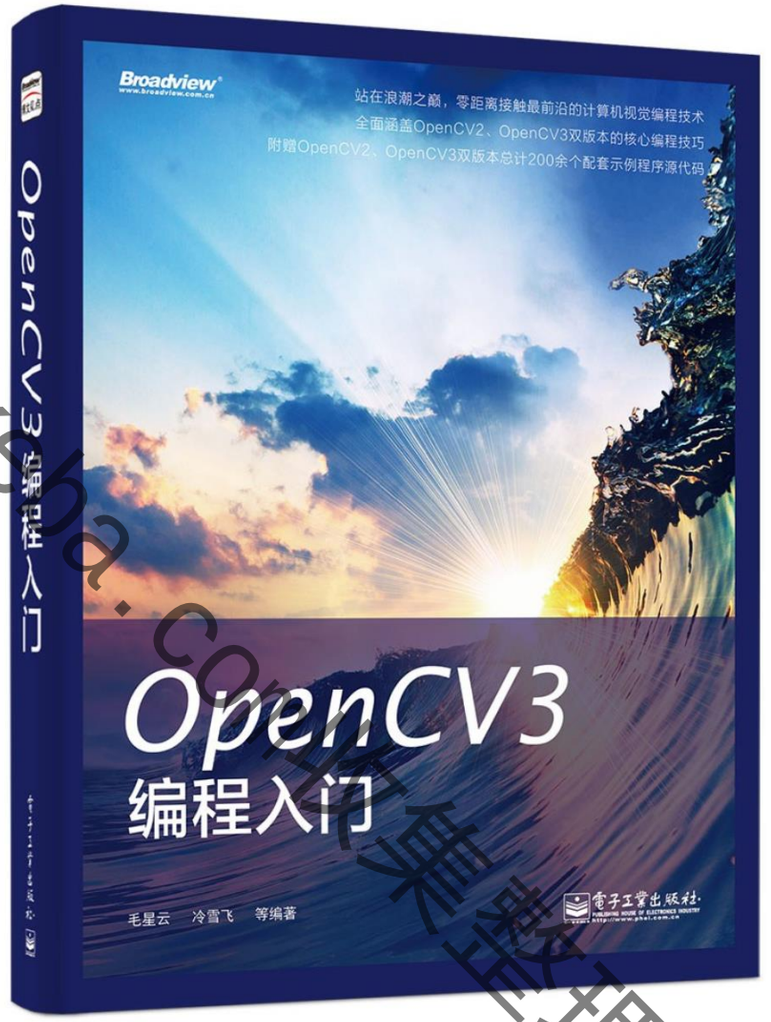
资料

1. 官方网站： https://opencv.org/
2. 代码位置： https://github.com/opencv/opencv
3. 需要结合《*计算机视觉*》进行学习
4. 很多人的入门书籍：《OpenCV3编程入门》-毛星云

4. 其他知名库介绍

4.1 OpenCV





4. 其他知名库介绍

4.1 OpenCV

领域知识

概念和应用

编程实践

API的调用和运行

```cpp
#include <opencv2/opencv.hpp>

int main() {
    // 加载彩色图像
    cv::Mat colorImage = cv::imread("path_to_image.jpg");
    // 检查图像是否成功加载
    if(colorImage.empty()) {
        std::cerr << "Error: Loading image" << std::endl;
        return -1;
    }
    // 创建一个Mat对象用于存储灰度图像
    cv::Mat grayImage;
    // 将彩色图像转换为灰度图像
    cv::cvtColor(colorImage, grayImage, cv::COLOR_BGR2GRAY);
    // 显示灰度图像
    cv::imshow("Gray Image", grayImage);
    // 等待按键事件
    cv::waitKey(0);
    return 0;
}
```

背后的理论、算法和公式

灰度图像上每个像素的颜色值又称为灰度，指黑白图像中点的颜色深度，范围一般从0到255，白色为255，黑色为0。所谓灰度值是指色彩的浓淡程度，灰度直方图是指一幅数字图像中，对应每一个灰度值统计出具有该灰度值的象素数。

灰度就是没有色彩，RGB色彩分量全部相等。如果是一个二值灰度图像，它的象素值只能为0或1，我们说它的灰度级为2。用个例子来说明吧：一个256级灰度的图像，如果RGB三个量相同时，如：RGB(100,100,100)就代表灰度为100，RGB(50,50,50)代表灰度为50。

代码的实现

```cpp
//////////////////////////////////////////////////////////////////////////////////////////
//                                The main function                                     //
//////////////////////////////////////////////////////////////////////////////////////////

/* void cvtColor */( InputArray _src, OutputArray _dst, int code, int dcn )
{
    CV_INSTRUMENT_REGION();

    CV_Assert( !_src.empty() );

    if(dcn <= 0)
        dcn = dstChannels(code);

    CV_OCL_RUN( _src.dims() <= 2 && _dst.isUMat() &&
                !(CV_MAT_DEPTH(_src.type()) == CV_8U && (code == COLOR_Luv2BGR || code == COLOR_Luv2RGB)),
                ocl_cvtColor(_src, _dst, code, dcn) )

    switch( code )
    {
        case COLOR_BGR2BGRA: case COLOR_RGB2BGRA: case COLOR_BGRA2BGR:
        case COLOR_RGBA2BGR: case COLOR_RGB2BGR: case COLOR_BGRA2RGBA:
            if(_src.channels() == 1)
                cvtColorGray2BGR(_src, _dst, dcn);
            else
                cvtColorBGR2BGR(_src, _dst, dcn, swapBlue(code));
            break;

        case COLOR_BGR2BGR565:  case COLOR_BGR2BGR555:  case COLOR_BGRA2BGR565:  case COLOR_BGRA2BGR555:
        case COLOR_RGB2BGR565:  case COLOR_RGB2BGR555:  case COLOR_RGBA2BGR565:  case COLOR_RGBA2BGR555:
            cvtColorBGR25x5(_src, _dst, swapBlue(code), greenBits(code));
            break;
```

4. 其他知名库介绍

4.2 PCL

基本信息

PCL（Point Cloud Library）是一个开源的**点云处理库**，用于3D点云的处理与分析和2D/3D图像处理。
PCL广泛应用于**计算机视觉、机器人、自动驾驶、三维重建、医学成像**等领域。

1. **Overview:** https://pointclouds.org/
2. **Get Started:** https://pcl.readthedocs.io/projects/tutorials/en/master/basic_structures.html#basic-structures
3. **Specification Documents:** https://pcl.readthedocs.io/projects/tutorials/en/master/#features
4. **Advanced Topics:** https://pcl.readthedocs.io/projects/tutorials/en/master/#visualization
5. **API:** https://pointclouds.org/documentation/

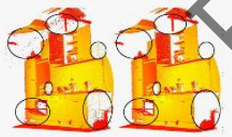
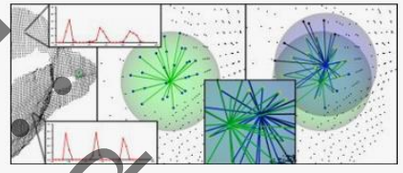
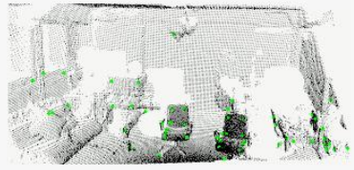
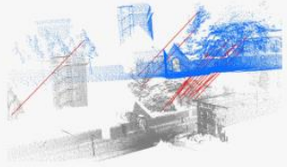
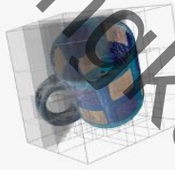
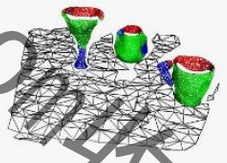
资料

- 官方网站：Point Cloud Library (PCL)
- 代码仓库：PCL GitHub
- 教程文档：PCL Documentation
- **PCL in ROS:** https://wiki.ros.org/pcl

4. 其他知名库介绍

4.2 PCL

应用

filters	features	keypoints
		
registration	**kdtree**	**octree**
		
segmentation	**sample_consensus**	**surface**
		
recognition	**io**	**visualization**
		

The most important set of released modules in PCL, link

4. 其他知名库介绍

4.3 Boost

Boost和C++ STL的关系

- Boost是对C++ STL的一个重要补充，提供了许多C++ STL中没有的功能和特性。
- 许多Boost的开发者也是C++标准委员会的成员，他们在Boost中的经验和成果直接影响了C++标准的制定。
- Boost中的许多库和技术最终被纳入了C++标准库。

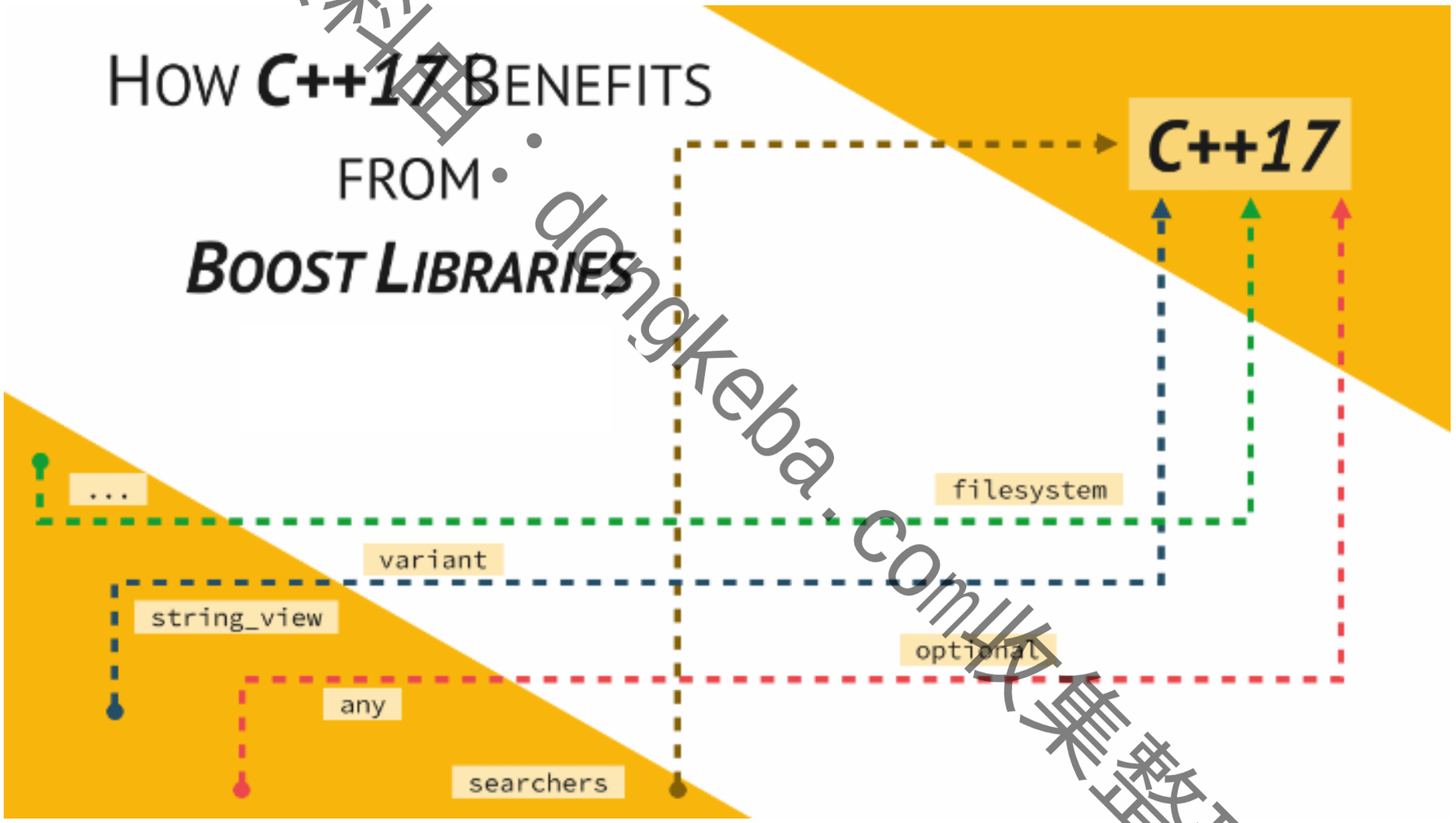
所以，可以把Boost理解为C++ STL的预览版本、尝鲜版本，只是把STL Beta版这个概念叫做Boost。

资料

- **官方网站：** https://www.boost.org/
- **代码仓库：** https://github.com/boostorg/boost
- **内容范畴：** https://www.boost.org/doc/libs/1_85_0/?view=categorized&sort=cxxstd
- **例子：** https://www.fluentcpp.com/2019/11/22/how-c17-benefits-from-boost-libraries-part-two/

4. 其他知名库介绍

4.3 Boost



5

小结与作业

5. 小结与作业

小结

1. 对**编程能力**和**领域知识**进行了区分；
2. 结合官方文档，讲解**Eigen**线性代数计算库；
3. 结合Apollo的源码，讲解了Apollo中**log**、**time**、**math**及其他**util**代码；
4. 介绍了**OpenCV**、**PCL**、**Boost**库。

【修订】之前大纲提到的“实现自己的CNN神经网络”一节，挪动到“智能驾驶C++项目实战”一章里。

5. 小结与作业

感悟

库的分类

实际项目中，需要使用各种各样的库：

- **STL标准库和Boost库**。实现最基础、最常用的数据结构和算法；
- **基础功能库**。实现日志记录、特定格式文件读取等各领域通用的功能，如glog、TinyXML；
- **某领域的库**。实现特定行业、领域的大量规范和功能，比如计算机视觉领域的OpenCV、点云处理/SLAM领域的PCL；
- **项目内部的库**。在其他库基础上扩展、优化得到的库，或者重新开发实现某功能的库。

库的使用

- 库就是一个个模块、一块块积木，写代码就像盖房子或者搭积木，需要把各种功能模块去组合、拼接起来，实现我们想实现的复杂功能。
- 如果某个功能模块已经有成熟的解决方案，那我们尽量复用已经成熟的库；如果并没有，那么就去自己实现一个新的库。

5. 小结与作业

感悟

图书馆海量的书籍

拼图

5. 小结与作业

感悟

库的数量

讲到这里，大家应该已经明白，各种库是讲不完的，因为我们有大量的领域、大量不同的需求，更有大量丰富的解决方案。

库的学习

- 了解某个库属于通用领域，还是特定领域，先学习对应的领域知识；
- 如果是库使用者，根据官方文档，学习所需要、感兴趣的功能、API、配置；
- 进一步，如果想成为库开发者，可以阅读源码，了解背后的实现和优化。

使用各种往往并不难，但需要有耐心，最好参考官方的使用指南；内部实现往往比较复杂，考虑到了各种情况，包含大量的性能优化代码，有余力再研究。

5. 小结与作业

感悟

为什么不仔细讲解下载、安装的过程？

- 官网的说明更正确、完整、全面
- 参考官网的步骤，效率最高
- 没有多少技术信息量
- Docker可以地简化软件、工具、库的安装和管理过程

下载、安装、部署的过程麻烦耗时、容易失败，但不要灰心，多多尝试。

5. 小结与作业

作业

概要

选择Eigen、OpenCV、PCL中的一个库，进行git下载，按说明去编译，然后运行demo程序，并将运行的结果截图。要求：

- 必须添加或修改至少一行日志输出内容，添加带有个人信息的字符串，来说明自己的确在本地完成了编译和运行；
- 最好进一步添加更多的日志作为调试输出，以更完整、全面地了解整个项目，了解其中关键的C++代码步骤实现了什么。

demo（sample/example/test）代码源文件位于：

- Eigen/test: https://gitlab.com/libeigen/eigen/-/blob/master/test
- OpenCV/samples: https://github.com/opencv/opencv/tree/4.x/samples
- PCL/examples: https://github.com/PointCloudLibrary/pcl/tree/master/examples

请先按照官方文档的说明进行下载、构建、运行，demo可执行程序一般会在本地自动创建的./build或/bin目录下。

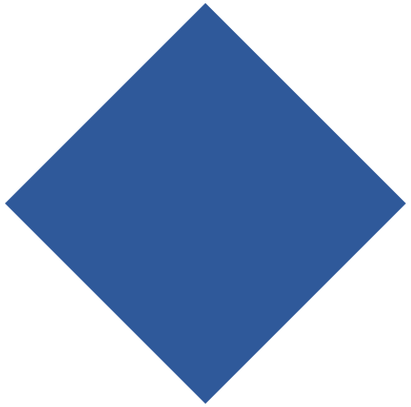
公众号

自动驾驶与计算机视觉
日常干货分享

知识星球

加入自动驾驶之心知识星球，
获取更多硬核干货

自动驾驶与AI全栈技术学习+领域大咖交流+职位内推！



End