背景简介

书接上文，在完成ORBSLAM算法于无人机P450的部署测试后，便进入到课题的高潮阶段：着手扩展算法功能并在本机实验测试，为论文理论提供支持。在向指导老师进行阶段性汇报交谈后，我们客观审视当前实验条件，综合考虑影响课题进展的多方面因素，落实具体技术路线：在ROS框架下扩展ORBSLAM3算法实现实时定位与稠密建图。下文详细记载从此纸面蓝图到实现的关键过程know-how。

双目视差稠密建图

先澄清，ORB-SLAM系列算法聚焦于技术框架本身，将绝妙的理论思想以工程化的形式打包成一个完备的标准平台，供领域内后续研究者进一步挖掘使用（因此github和gitee上才有如此丰富的二次开发内容，本实验自然也是其中一员），一举将SLAM领域的研究范式向前推进一个版本，完成从0到1并指出后续方向，堪称学术界与工业界结合的典范。当然，在具体应用上，ORBSLAM系列并没有花费过多精力，将从1到N的工作留于生态内的其他角色。在测绘领域，ORBSLAM简单实现的稀疏重建不满足任何应用场景，若需稠密重建，还需进一步引入其他方案落实为点云等数据形式。

在纯视觉SLAM体系中，当前常见的传感器方案有单目、双目相机、RGBD相机，三者功能与价格均依次递增。考虑到当前实验资源限制(毕竟只是毕业设计，有什么用什么，充分挖掘手头的硬件资源以适应需要），本方案决定采用双目相机方案。当然，凡事都有两面性，RGB-D相机由于其测量范围和原理导致它主要适用于室内环境中；在室外环境中由于双目相机没有尺度问题，通常使用双目相机计算深度图。

想要从双目相机获取立体，就需要追溯到相关计算机视觉原理：视差。考虑到篇幅，这里就不过多展开了，简单说来就是仿照人眼立体成像原理，利用相似关系参考双眼直接的距离(基线长)计算双目影像中共同目标与相机的距离(深度)，凭借影像中的二维坐标加上深度信息便可计算出特点坐标系中三维数据。详情可参考此篇博文前半段。

为减少不必要的阻力，我们采用OpenCV全面内置的SGBM（Semi-Global Block Matching）算法。它基于全局能量最小化思想，结合局部匹配的效率和全局优化的精度。通过动态规划和路径聚合策略来计算每个像素的视差，从而生成稠密的视差图。详细流程讲解请自行网络搜索学习，这里直接给出相关模块代码。

```
#include <opencv2/opencv.hpp>
...
// 初始化SGBM
stereo_ = cv::StereoSGBM::create(0, 64, 11, 8 * 3 * 11 * 11, 32 * 3 * 11 * 11, 1, 10, 100, 32);
...
 // 计算视差图
cv::Mat disparity;
stereo_->compute(left_image_, right_image_, disparity);
...
// 生成深度图
float focal_length = 0.8;  // 相机焦距（单位：米）
float baseline = 0.1;      // 双目基线（单位：米）
cv::Mat depth = (focal_length * baseline) / (disparity / 16.0f);
```

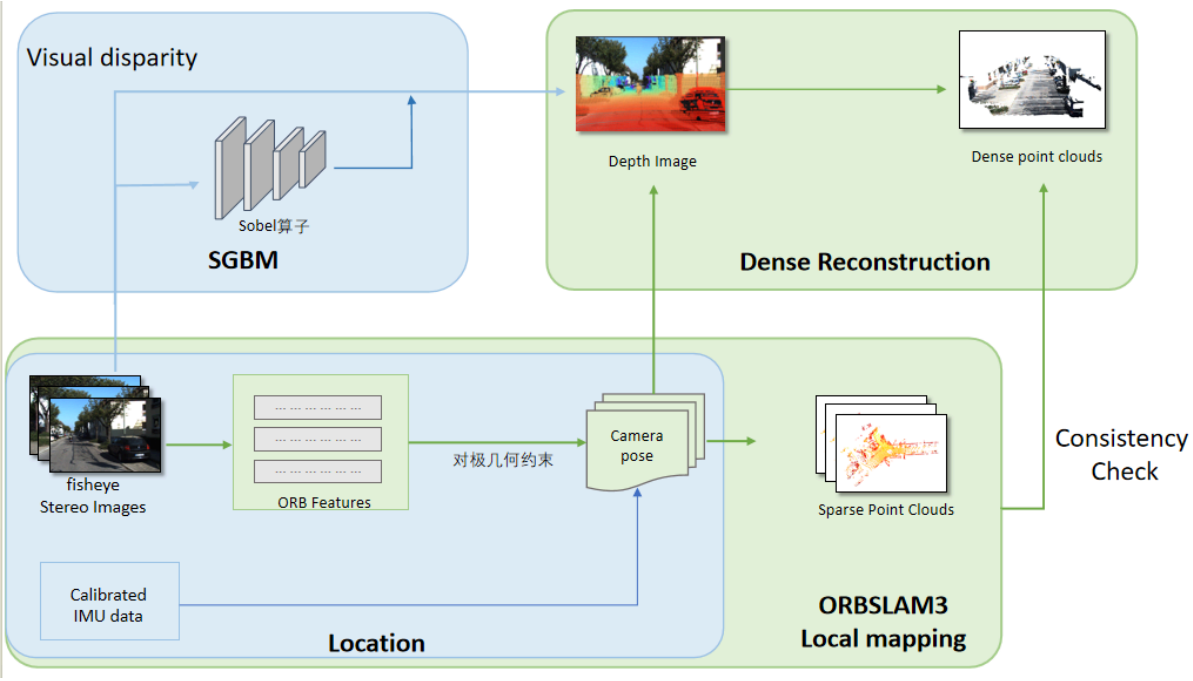
可见，得益于OpenCV良好的工程架构设计，用户对SGBM算法的调用具有类似API的高封装特性，使用十分简洁方便，感谢Intel开源~。

ROS工程结构管理

凑齐了技术路线上所有拼图块，我们还需要一套框架去规范拼图的过程，从而及时检验与校正performance。而在SLAM领域，最方便全面的框架就是ROS(Robot Operating System)。上篇文章虽有提及ROS，但仅仅是将其作为一个简单的工具，这里就我个人的理解对其进行简介。

在机器人看来，ROS是操作系统；在计算机（机器人端嵌入式）看来，ROS是应用程序；而在开发人员看来，ROS是工具箱，可以利用通信工具与二者进行数据接发传输，可以利用面板工具与前二者切换不同视角上的观测，甚至可以查看工具箱说明书去有针对性地补充特定应用下缺失的工具。总之，ROS是三个角色之间互动的标准，遵循这套标准可以绕开底层细节将精力集中在应用逻辑的设计开发上。

本实验采用ROS通信中基础的话题通信模型，把建图系统分为了三个节点。第一个节点利用ORB-SLAM算法主要做相机位姿估计；第二个节点作为视差计算节点，接收原始双目图像数据计算深度图；第三个节点作为建图节点，收集第一和第二节点传入的深度图数据和位姿数据，进行点云拼接。将数据流以话题形式发布串连起所有节点，以实现全过程。整体技术路线示意如图。



具体实现上，可将上述三个节点分作三个ROS功能包放在同一工作空间下进行管理。但本人为了方便管理依赖项编译的相互影响，将后续两个节点与ORBSLAM算法分开为两个工作空间下的两个功能包。关于ROS工作空间、功能包与节点的概念与使用请自行网络搜索与实践结合摸索。

ORBSLAM3源码扩展

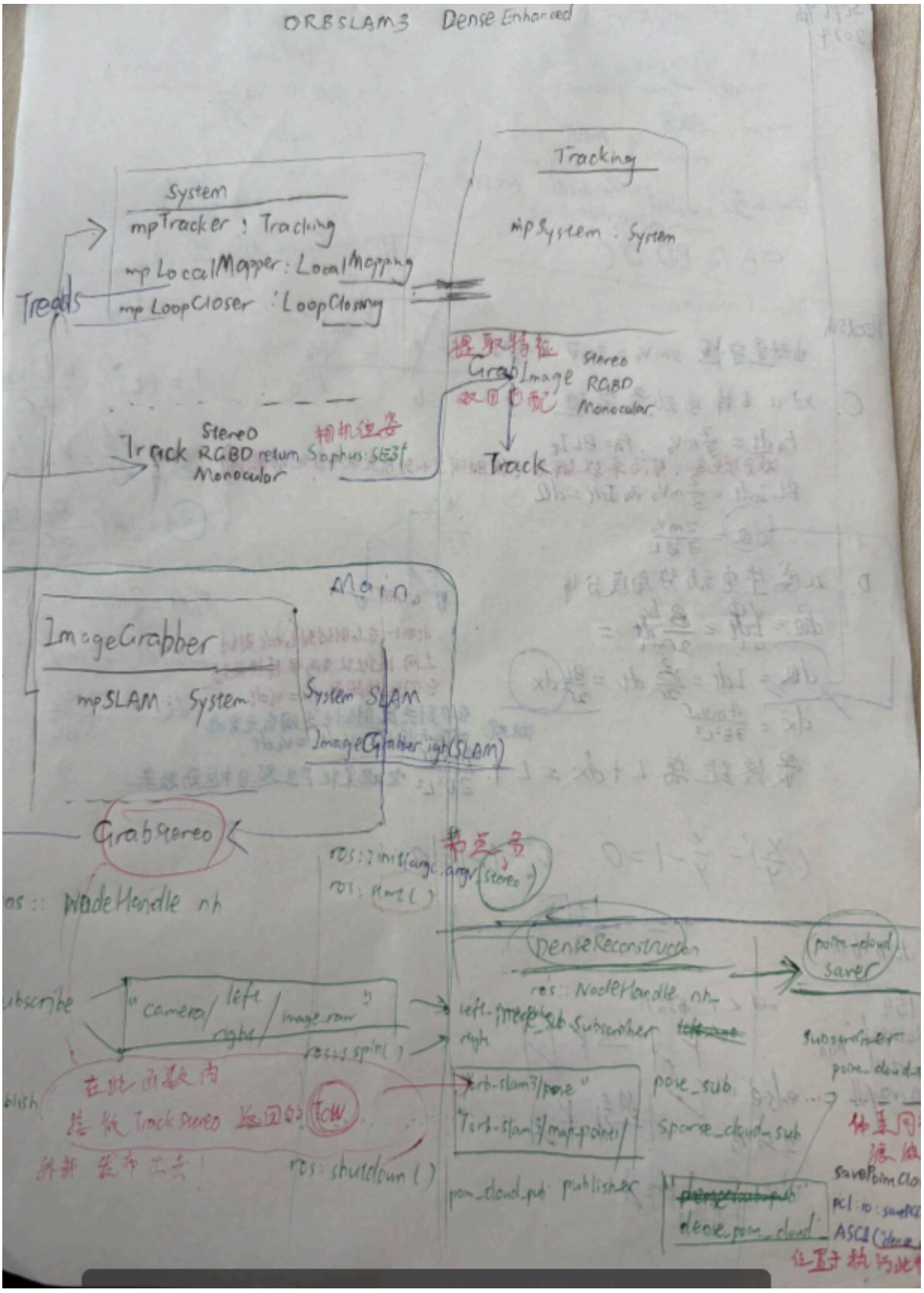
如前文所言，ORBSLAM3算法只支持稀疏建图，若想实现稠密建图。需如上图技术路线提取出算法实时计算出的相机位姿并传入后续节点参与建图计算。本节讲解如何实现此功能。

作为一个完备平台，ORBSLAM发布官方提供了IT界较标准的代码框架说明；而作为一个发布近十年的标杆算法，互联网上也已经积累了大量的相关解读。然而鉴于其不可小觑的工程复杂性，要想将其真正理解透彻，个人建议如下：

先积累相关知识储备，不要小看这一点。只有具备了足够的互信息，通信的效率才有可能提上去。不客气的讲，若是对诸如 多线程、特征提取、李群、四元数等不同领域的相关概念先没有基础的理解，是没资格真正疏通整个ORBSLAM框架的。

在积累了一定量的储备后形成领域整体认知（不可能有彻底完善的一天，足够理解后剩下的边实践边补学），便可以设法阅览较权威的文献与解读去理解ORBSLAM算法整体的设计框架了。

最后一步，沉下心来复习C++语言规范（现在有类Cursor等AI工具非常方便），对照代码校正自身理解，必要时绘制类似UML的工具图，不必严格规范，足够条分缕析有效帮助理解即可，本人手工绘制的如下图（整体预览效果即可，不必深入细节，个人偏好不一定规范）。



对照着上方手工绘图，就如图战场上有了地图，得以精确锁定打击目标。具体对ROS功能包中ros_stereo.cc补充一个方法以发布从TrackStereo返回的Sophus库SE3f标准格式数据Tcw即可。

发布相关修改如下，详见[github完整版]

```
void ImageGrabber::GrabStereo(const sensor_msgs::ImageConstPtr& msgLeft,const sensor_msgs::ImageConstPtr& msgRight)
{
    Sophus::SE3f Tcw;
...
    if(do_rectify)
    {
        cv::Mat imLeft, imRight;
        cv::remap(cv_ptrLeft->image,imLeft,M1l,M2l,cv::INTER_LINEAR);
        cv::remap(cv_ptrRight->image,imRight,M1r,M2r,cv::INTER_LINEAR);
       Tcw = mpSLAM->TrackStereo(imLeft,imRight,cv_ptrLeft->header.stamp.toSec());
    }
    else
    {
       Tcw = mpSLAM->TrackStereo(cv_ptrLeft->image,cv_ptrRight->image,cv_ptrLeft->header.stamp.toSec());
    }
    this->PublishPose(Tcw);
}


void ImageGrabber::PublishPose(const Sophus::SE3f& Tcw) {

  Sophus::SE3f Twc = Tcw.inverse();

  // **构造 ROS 位姿消息**

  geometry_msgs::PoseStamped pose_msg;

  pose_msg.header.stamp = ros::Time::now();

  pose_msg.header.frame_id = "map";

  // **提取平移量**

  Eigen::Vector3f t = Twc.translation();

  pose_msg.pose.position.x = t.x();

  pose_msg.pose.position.y = t.y();

  pose_msg.pose.position.z = t.z();

  // **提取旋转矩阵，并转换为四元数**

  Eigen::Quaternionf q(Twc.unit_quaternion());

  pose_msg.pose.orientation.x = q.x();

  pose_msg.pose.orientation.y = q.y();

  pose_msg.pose.orientation.z = q.z();

  pose_msg.pose.orientation.w = q.w();

```

```
  // **发布位姿**

  pose_pub.publish(pose_msg);

}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "slam");
    ros::start();

    ros::NodeHandle nh;
    // Create SLAM system. It initializes all system threads and gets ready to process frames.
    ORB_SLAM3::System SLAM(argv[1],argv[2],ORB_SLAM3::System::STEREO,true);
    mpSLAM = &SLAM;
    ImageGrabber igb(&SLAM);
     // 初始化发布器
    pose_pub = nh.advertise<geometry_msgs::PoseStamped>("/orb_slam3/pose", 10);
    message_filters::Subscriber<sensor_msgs::Image> left_sub(nh, "/camera/left/image_raw", 1);
    message_filters::Subscriber<sensor_msgs::Image> right_sub(nh, "/camera/right/image_raw", 1);
    typedef message_filters::sync_policies::ApproximateTime<sensor_msgs::Image, sensor_msgs::Image> sync_pol;
    message_filters::Synchronizer<sync_pol> sync(sync_pol(10), left_sub,right_sub);

 sync.registerCallback(boost::bind(&ImageGrabber::GrabStereo,&igb,_1,_2));

    ros::spin();
    // Stop all threads
    SLAM.Shutdown();

    // Save camera trajectory
    SLAM.SaveKeyFrameTrajectoryTUM("KeyFrameTrajectory_TUM_Format.txt");
    SLAM.SaveTrajectoryTUM("FrameTrajectory_TUM_Format.txt");
    SLAM.SaveTrajectoryKITTI("FrameTrajectory_KITTI_Format.txt");

    ros::shutdown();

    return 0;
}
```

注意，left_sub，right_sub为接收图像信息的话题，而"/orb_slam3/pose"为发布位姿的话题，需在稠密建图相关节点接收呼应(ROS框架类似，这里就不贴了，详见github)，具体需因实验环境而调整。

至此，拼图已经按照框架摆放有序。下面便可通过ROS环境编译对工作空间输入catkin_make生成可执行程序，测试运行！接通了全流程，方案可行性已经得到充分验证，但为了让作品拼图更稳定可靠，无论什么环境都不会轻易散落功亏一篑，还需进行最后的Fine-Tune: 工程调优。

运行结果演示

照惯例，运行指令如下：

```
Roscore
rosbag play --pause V1_01_easy.bag /cam0/image_raw:=/camera/left/image_raw
/cam1/image_raw:=/camera/right/image_raw
rosrun ORB_SLAM3 Stereo Vocabulary/ORBvoc.txt Examples_old/Stereo/EuRoC.yaml
true
rosrun dense_reconstruction dense_reconstruction
rosrun dense_reconstruction point_cloud_saver
```

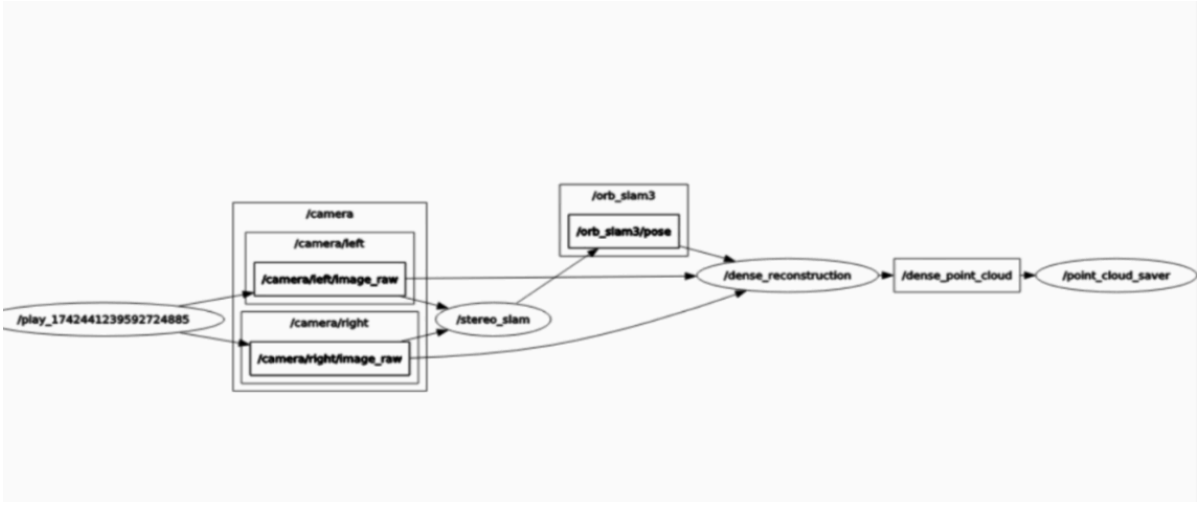
其中第二句为播放EuRoC数据集，运行后 需按下空格键开始播放。

注意，第五句命令类似第三句设计，在数据播放完毕后人为输入ctrl+C，程序判断为正常关闭并保存结果数据(稠密点云地图)。

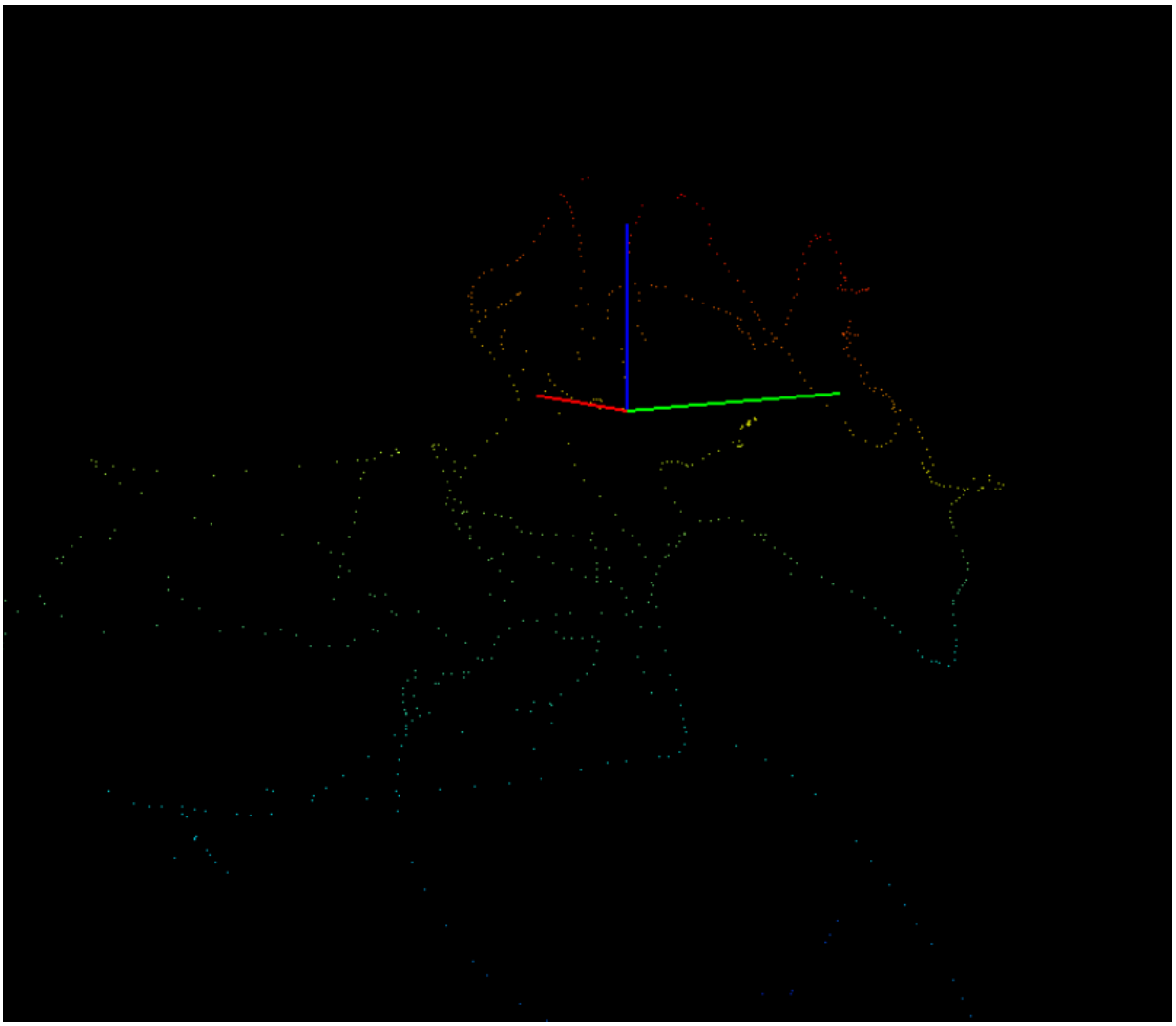
若终端未检测到ORBSLAM功能包路径，输入如下指令刷新。

```
`export ROS_PACKAGE_PATH=/home/amov/zagho_ws/src:$ROS_PACKAGE_PATH`
```

再打开一个终端输入rqt_graph监测ROS节点与话题，在如图左上角选择Nodes/Topics(all)，其中矩形为话题，椭圆为节点。



最终绘制得到三维稠密点云地图如下。



当然，由于EuRoC数据集本身轻量级方便调试的特性，数据量相对不足导致所建点云地图也不够丰富。有兴趣的读者可以自行下载与加工KITTI数据集或自行采集数据进行测试。

实机运行只需参考前一篇博客“阿木P450双目T265运行ORBSLAM”调整图像来源即可。

至此，毕业设计课题实验部分告一段落，期待后续实飞！不过除此之外，后续主要精力切换到论文撰写当中。