CubeSLAM代码整理

cubeslam源码是基于ros的一个工程，我们将依次介绍cubeslam中的orb_object_slam的launch文件，orb_object_slam中3d bbox调用的代码，与detect_3d_cubiod的3d bbox生成方案。orb_object_slam主要是在orb_slam上做了修改，我们将对orb_slam中原本的内容介绍的简略些，重点放在cubeslam添加的内容上。

源码介绍

Cubeslam

- .vscode //配置文件
- dependency //依赖项，主要是tictoc_profiler计时
- detect_3d_cuboid //输入2d bbox信息与线检测结果

  --data //放入图片与线检测结果(源码里并没有放入2d bbox数据，但仍可以跑通3d生成的demo)

  --src && include
- line_lbd //轻便的线检测工具，CubeSLAM跑kitti数据 中给出了批处理的一个脚本

  --data //放入图片数据

  --launch //通过launch文件建立线检测的ros节点
- object_slam //基于3d bbox信息给出rviz可视化，未结合orb_slam
- orb_object_slam //嵌入到orb_slam后的结果，结构与orb_slam很像

  --Examples //提供kitti,TUM等常用数据集的一些参数yaml

  --include && src

  --launch //通过launch文件运行orb_object_slam，参考demo

  --Thirdparty //第三方库，包括DBoW2和g2o

  --Vocabulary //orb_slam词袋
- Preprocessing //2d bbox批处理生成，可参考CubeSLAM跑kitti数据 中的批处理替代
- install_dependencies.sh //shell脚本安装orb_object_slam下的第三方库

Ros launch文件

在CubeSLAM示例编译运行 中，我们使用mono.launch与mono_dynamic.launch。两个launch文件分为建立节点与传递参数两个部分

- 节点建立

Bash

```
# ROS_NAMESPACE=mono rosrun image_proc image_proc   bayer to bgr8
   <node pkg="image_proc" type="image_proc" name="image_proc" ns="mono" args="bayer to bgr8" />
   <node pkg="orb_object_slam" type="ros_mono" name="ros_mono"  output="screen"  args="$(find orb_object_slam)/Vocabulary/ORBvoc.bin
      $(find orb_object_slam)/Examples/Monocular/KITTI04-12_me.yaml">  # KITTI_09_26 KITTI04-12_me kinect_mev2 TUM_mono  tamu_corridor
   <remap from="/camera/image_raw" to ="/kitti/left/image_raw"/>
   # /kitti/left/image_raw   /camera/rgb/image_raw /camera/rgb/image_color  /mono/image_rect_color /kinect2/qhd/image_color_rect /camera/mono/image_raw
   </node>
```

image_proc节点是调用ros自带的mono image_proc中的bayer to bgr8，实现bayer格式转bgr格式

orb_object_slam节点是调用cubeslam源码中的ros_mono，传递参数为词袋ORBvoc.bin的路径，Example中对应的yaml参数，包括相机内参，orb特征提取的参数等
等，/kitti/left/image_raw为制作的rosbag中的相机话题，代码中将其remap到节点中，即orb_object_slam节点会订阅/kitti/left/image_raw这一话题。使用不同rosbag时，需要info查看话题信息，并在launch文件中修改，参考 CubeSLAM跑kitti数据

- 传递参数

PowerShell

```
   <param name="enable_loop_closing" value="false" />  # false true
   <param name="enable_viewer" value="true" />   <param name="enable_viewmap" value="true" />   <param name="enable_viewimage" value="true" />

   <param name="parallel_mapping" value="true" />  # if false, may reduce bag rate

   <rosparam file="$(find orb_object_slam)/launch/object_params/kitti.yaml" command="load"/>   # initial pose, folder name
   <param name="base_data_folder" value="/home/shichao/ysc_space/dataset/processed_third/slam/kitti/kitti_odom/seq_07" />

   <param name="whether_detect_object" value="true" />
   <param name="whether_read_offline_cuboidtxt" value="true" />  # for kitti, I read offline data.
   <param name="associate_point_with_object" value="true" />
   <param name="obj_det_2d_thre" value="0.5" />  # for online 3D detection


```

```
    <param name="bundle_object_opti" value="true" />
    <param name="build_worldframe_on_ground" value="true" />
    <param name="camera_object_BA_weight" value="2.0" />  #2.0 default


    # for dynamic object
    <param name="whether_dynamic_object" value="false" />
    <param name="remove_dynamic_features" value="false" />
    <param name="use_dynamic_klt_features" value="false" />  # not orb features
    <param name="object_velocity_BA_weight" value="0.5" />

    <param name="use_truth_trackid" value="false" />   # use offline tracking id  if false, need to initialize feature point, tracking, obj depth init.
    <param name="triangulate_dynamic_pts" value="false" />
    <param name="ba_dyna_pt_obj_cam" value="true" />  # need depth init
    <param name="ba_dyna_obj_velo" value="true" />
    <param name="ba_dyna_obj_cam" value="true" />

    # for depth initialization
    <param name="mono_firstframe_truth_depth_init" value="false" />
    <param name="mono_firstframe_Obj_depth_init" value="false" />
    <param name="mono_allframe_Obj_depth_init" value="false" />   # may not need for kitti

    # for ground height scaling
    <param name="enable_ground_height_scale" value="true" />  # for kitti
    <param name="ground_everyKFs" value="10" />
    <param name="ground_roi_middle" value="3.0" />  # 3(1/3) or 4(1/2)
    <param name="ground_roi_lower" value="3.0" />   # 2 or 3
    <param name="ground_inlier_pts" value="20" />
    <param name="ground_dist_ratio" value="0.08" />

    # save result
    <param name="whether_save_online_detected_cuboids" value="false" />
    <param name="whether_save_final_optimized_cuboids" value="false" />

    # gui drawing parameters  usually set to true, for paper, set to false
    <param name="draw_map_truth_paths" value="true" />
    <param name="draw_nonlocal_mappoint" value="true" />

```

上例是mono.launch中传递的参数，mono_dynamic.launch会在whether_dynamic_object，remove_dynamic_features，mono_firstframe_truth_depth_init，enable_ground_height_scale四个参数上有所不同

detect_3d_object模块

detect_3d_object为cubeslam中通过2d bbox，线检测数据与tranToWorld生成立方体提案的C++模块

主要内容包括：matrix_utils提供基本的矩阵操作组件，object_3d_util提供生成立方体提案时所需的基本函数，包括计算VP点与求直线交点等组件，头文件detect_3d_cuboid给出cuboid类与detect_3d_cuboid类的定义，box_proposal_detail提供detect_3d_cuboid类中detect函数的细节,main函数给出detect_3d_cuboid的接口，传递内参，transToWorld，2d bbox，线检测以及detect的参数

代码思路

读入数据与参数，设定相关阈值，进行一些基本的检查后，开始遍历所有的2d bbox数据。读入2d bbox后进行yaw角采样，yaw角用于vp点的计算。如果我们考虑2d bbox的不准确，则进行高度采样，并扩展边界框，扩展的边界框用于基于canny边缘做DT变换。然后我们对线检测的数据进行一些处理，比如筛选扩展检测框内的线条，对线条进行合并，对短线条进行剔除，计算线条的中点与角度用于VP点支撑边的检测，为角度误差的计算做好铺垫。同时使用opencv的canny检测与DT变换为距离误差计算做好准备，采样cam_roll和cam_pitch（注意我们建world系时，obj相对于world只有yaw角，而cam相对于world yaw角为0)。我们遍历cam_roll，cam_pitch，yaw的采样，cam_roll，cam_pitch主要影响2d坐标恢复到3d坐标。根据yaw与cam_pose计算VP点，并调用相关函数计算VP点的支撑边。而后遍历上顶点采样，根据上顶点与VP点坐标，可以复原8个顶点的2d坐标，复原的过程根据不同的情形config_id与vp1的左右有所不同。成功复原2d坐标后，我们可以根据所给函数计算距离误差与角度误差（它们都是基于2d图像的）。我们将提案的这些误差，2d坐标，基本参数，以及采样的值保存到一个整体的Matrix all_configs_error_one_objH，包含了我们采样生成所有提案的信息。最后我们对其中的误差信息融合(找出两个误差都比较小的，加权并归一化)，找出所谓good_proposal。而目前只有2d坐标信息，但根据Matrix中的cam_pose等参数，我们可以复原3d坐标与3d信息(change_2d_corner_to_3d_object)，复原后的立方体提案以cuboid类的格式保存到容器中，cuboid类定义在detect_3d_cuboid.h中。对于容器中的cuboid，我们再考虑它们的长宽比，重新计算最后的分数。根据分数进行部分排序，我们选取分数最低的那个(即误差最小)作为最后的立方体提案。

简要表示下2d bbox生成立方体提案时的循环结构，区分开采样与遍历

----#数据，参数，阈值

----**for** 2d bbox #遍历2d bbox数据，一个bbox代表一个物体

--------读入2d bbox，高度采样，yaw角采样

--------**for** height_sample #遍历高度采样，高度采样可以开关

------------重新计算2d bbox，并对2d bbox扩展，用于dist_map，顶边采样（上顶点）

------------整合线检测数据（合并或剔除），计算线角度与中点

------------opencv canny边缘检测，距离变换DT得到dist_map，采样cam_roll，cam_pitch

------------**for** yaw roll pitch sample #遍历yaw角，cam_roll，cam_pitch 可以设定不采样后两者

----------------计算VP点，求VP点的支撑边(支撑边是线检测数据中形成VP点的边)

----------**for** top sample #遍历上顶点的采样

----------根据vp1与上顶点1求顶点2，根据顶点2的左右判断vp1的左右

----------**for** config_id #对求透视的情形进行判断，cofig_id=1,2代表三面与两面

----------根据不用的config_id依次计算剩余6个顶点

----------计算距离误差与角度误差，距离误差与可见边相关，从而与config_id相关

----------遍历的参数，误差等全部储存到all_configs_error_one_objH

----------**end** of config_id

----------**end** of top sample

----------**end** of yaw roll pitch sample

----------融合误差信息，将两个误差都比较高的舍弃，留下good_proposal_id，并归一化

----------根据all_configs_error_one_objH中2d信息，复原3d信息为cuboid类，只复原good_id

--------**end** of height smaple

--------计入长宽比惩罚，重新计算最后的误差combined_error

--------基于combined_error部分排序，只要最好的立方体提案

----**end** of 2d bbox

detect_3d_cuboid.h

detect_3d_cuboid作为头文件声明了cuboid类与detect_3d_cuboid类中的成员与参数等

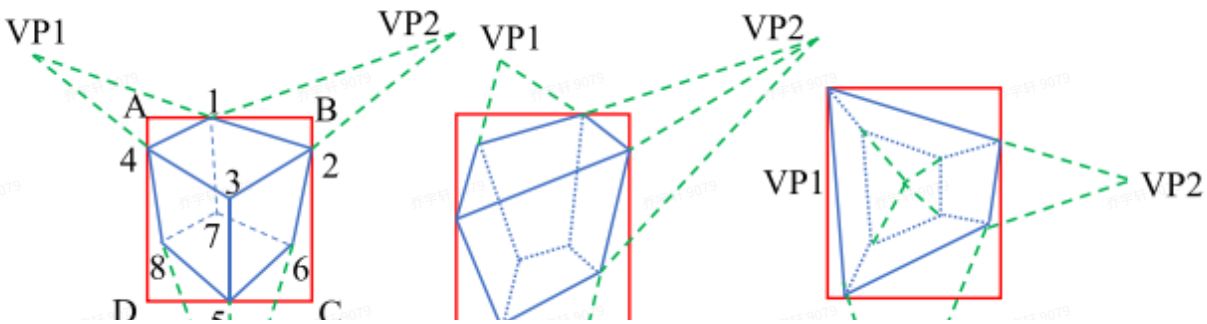
C++

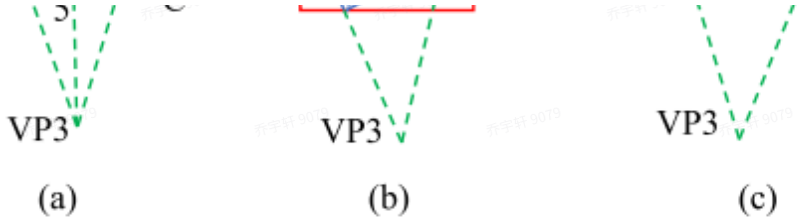
```cpp
class cuboid
{
    public:
      Eigen::Vector3d pos;
      Eigen::Vector3d scale;//长宽高的一半
      double rotY;//假设cuboid都在地面上，在地面上建世界坐标系，只有yaw一个自由度

      Eigen::Vector2d box_config_type;        //box的透视类型，标记vp1在左侧还是右侧
      Eigen::Matrix2Xi box_corners_2d;        // 8个顶点的2d坐标 2*8
      Eigen::Matrix3Xd box_corners_3d_world; // 8个顶点的3d坐标 3*8

      Eigen::Vector4d rect_detect_2d; //2d bbox数据
      double edge_distance_error; //canny距离误差
      double edge_angle_error; //角度误差
      double normalized_error; //距离角度加权后的normalized误差
      double skew_ratio; //长宽比，高长宽比在考虑误差时会得到惩罚
      double down_expand_height; //扩大边界框的大小
      double camera_roll_delta;
      double camera_pitch_delta;
      //对于每个obj提案生成时如果采样camera roll与pitch的话，记录与raw的差

      void print_cuboid(); // 输出cuboid信息，定义在object_3d_util.cpp
};
typedef std::vector<cuboid *> ObjectSet; // for each 2D box, the set of generated 3D cuboids 使用容器vector存放所有得到的cuboid提案到ObjectSet中

struct cam_pose_infos
{
      Eigen::Matrix4d transToWolrd; //相机坐标系到世界坐标系的变换矩阵
      Eigen::Matrix3d Kalib; //内参

      Eigen::Matrix3d rotationToWorld; //变换矩阵中的旋转矩阵
      Eigen::Vector3d euler_angle; //欧拉角
      Eigen::Matrix3d invR;
      Eigen::Matrix3d invK;
      Eigen::Matrix<double, 3, 4> projectionMatrix; //投影矩阵
      Eigen::Matrix3d KinvR; // K*invR
      double camera_yaw;
};
```

PHP

```php
class detect_3d_cuboid
{
    public:
      cam_pose_infos cam_pose;
      cam_pose_infos cam_pose_raw;
      //考虑pose的采样时，cam_pose_raw为原pose

      void set_calibration(const Eigen::Matrix3d &Kalib); //传递内参
      void set_cam_pose(const Eigen::Matrix4d &transToWolrd); //传递变换矩阵

      // object detector needs image, camera pose, and 2D bounding boxes(n*5, each row: xywh+prob)  long edges: n*4.  all number start from 0
      void detect_cuboid(const cv::Mat &rgb_img, const Eigen::Matrix4d &transToWolrd, const Eigen::MatrixXd &obj_bbox_coors, Eigen::MatrixXd edges,
                         std::vector<ObjectSet> &all_object_cuboids);
      //实现detect_3d_cuboid的函数，需要传递img,transToWorld,2d bbox,线检测edges
      //生成的提案存放在容器all_object_cuboids中

      bool whether_plot_detail_images = false;
      bool whether_plot_final_images = false;
      bool whether_save_final_images = false;
      cv::Mat cuboids_2d_img; // save_final_image保存到的变量
      bool print_details = false; //不输出生成过程的细节

      // important mode parameters for proposal generation.
      bool consider_config_1 = true; // false true
      bool consider_config_2 = true;
      bool whether_sample_cam_roll_pitch = false; //对cam_pose进行采样
      bool whether_sample_bbox_height = false;    //对bbox的高度进行采样

      int max_cuboid_num = 1;         //final return best N cuboids
      double nominal_skew_ratio = 1; //设置先验的长宽比，根据先验做出惩罚
      double max_cut_skew = 3;
};
```

关于consider_config_1与consider_config_2两个变量的含义参加下图：





(a) (b) (c)

物体相对于相机的透视情况有三种，如上图。consider_config_1代表是否考虑情况(a)，consider_config_2代表是否考虑情况(b)

matrix_utils.h

matrix_utils提供基本向量和矩阵运算的组件，函数的定义在matrix_utils.cpp中

Rust

```
template <class T>
Eigen::Quaternion<T> zyx_euler_to_quat(const T &roll, const T &pitch, const T &yaw);

template <class T>
void quat_to_euler_zyx(const Eigen::Quaternion<T> &q, T &roll, T &pitch, T &yaw);

template <class T>
void rot_to_euler_zyx(const Eigen::Matrix<T, 3, 3> &R, T &roll, T &pitch, T &yaw);

template <class T>
Eigen::Matrix<T, 3, 3> euler_zyx_to_rot(const T &roll, const T &pitch, const T &yaw);
//欧拉角，四元数，旋转矩阵互相转换

// input is 3*n (or 2*n)  output is 4*n (or 3*n)
template <class T>
Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> real_to_homo_coord(const Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &pts_in);
template <class T>
void real_to_homo_coord(const Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &pts_in, Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &pts_homo_out);
template <class T> // though vector can be casted into matrix, to make output
```

```
 clear to be vector, it is better to define a new function.
Eigen::Matrix<T, Eigen::Dynamic, 1> real_to_homo_coord_vec(const Eigen::Matrix<T, Eigen::Dynamic, 1> &pts_in);

// input is 3*n (or 4*n)  output is 2*n(or 3*n)
template <class T>
Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> homo_to_real_coord(const Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &pts_homo_in);
template <class T>
void homo_to_real_coord(const Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &pts_homo_in, Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &pts_out);

template <class T> // though vector can be casted into matrix, to make output clear to be vector, it is better to define a new function.
Eigen::Matrix<T, Eigen::Dynamic, 1> homo_to_real_coord_vec(const Eigen::Matrix<T, Eigen::Dynamic, 1> &pts_homo_in);
//真实坐标与齐次坐标的转换

//vertically stack a matrix to a matrix, increase rows, keep columns
void vert_stack_m(const Eigen::MatrixXd &a_in, const Eigen::MatrixXd &b_in, Eigen::MatrixXd &combined_out);
void vert_stack_m_self(Eigen::MatrixXf &a_in, const Eigen::MatrixXf &b_in);
//两个列数相同矩阵上下合并

void fast_RemoveRow(Eigen::MatrixXd &matrix, int rowToRemove, int &total_line_number);
//移去特定某行

// make sure column size is given. not check here. row will be adjusted automatically. if more cols given, will be zero.
template <class T>
bool read_all_number_txt(const std::string txt_file_name, Eigen::Matrix<T, Eigen::Dynamic, Eigen::Dynamic> &read_number_mat);

// each line: one string, several numbers. make sure column size is correct.
bool read_obj_detection_txt(const std::string txt_file_name, Eigen::MatrixXd &read_number_mat, std::vector<std::string> &strings);
// each line several numbers then one string . make sure column size is correct.
bool read_obj_detection2_txt(const std::string txt_file_name, Eigen::MatrixXd &read_number_mat, std::vector<std::string> &strings);
//2d bbox数据与线检测数据的读取

// (partial) sort a vector, only need top_K element, result is stored in idx by default increasing
void sort_indexes(const Eigen::VectorXd &vec, std::vector<int> &idx, int top_k);
```

```
52  void sort_indexes(const Eigen::VectorXd &vec, std::vector<int> &idx);
53  //对序列进行递增排序，只需要前top k的部分
54
55  // change [-180 180] to [-90 90] by +-90
56  template <class T>
57  T normalize_to_pi(T angle);
58
59
60  template <class T>
61  void print_vector(const std::vector<T> &vec);
62
63  //TODO could be replaced by eigen linespace
64  template <class T>
65  void linespace(T starting, T ending, T step, std::vector<T> &res);
66  //linespace函数实现采样
```

object_3d_util.h && object_3d_util.cpp

object_3d_util中声明并定义了关于obj生成的相关函数，将分部分介绍

首先是绘制有关，一般调用的是plot_image_with_cuboid，即在图像上绘制cuboid，以及plot_image_with_edges，即在图像上绘制线检测edges的信息

C++

```
1  void get_object_edge_visibility(MatrixXi &visible_hidden_edge_pts, const Vecto
   r2d &box_config_type, bool new_object_type = false);
2  void get_cuboid_draw_edge_markers(MatrixXi &edge_markers, const Vector2d &box_
   config_type, bool new_object_type = false);
3  void plot_image_with_cuboid_edges(cv::Mat &plot_img, const MatrixXi &box_corne
   rs_2d, const MatrixXi &edge_markers);
4  void plot_image_with_cuboid(cv::Mat &plot_img, const cuboid *cube_obj);
5  void plot_image_with_edges(const cv::Mat &rgb_img, cv::Mat &output_img, Matrix
   Xd &all_lines, const cv::Scalar &color);
```

关于相似矩阵与3d顶点计算，主要是通过物体cuboid的信息还原8个顶点的3d坐标

C++

```cpp
Matrix4d similarityTransformation(const cuboid &cube_obj)
{
    Matrix3d rot;
    rot << cos(cube_obj.rotY), -sin(cube_obj.rotY), 0,
        sin(cube_obj.rotY), cos(cube_obj.rotY), 0,
        0, 0, 1; //yaw角转化为旋转矩阵
    Matrix3d scale_mat = cube_obj.scale.asDiagonal();
    //scale向量转化成对角矩阵，相当于对单位立方体做拉伸

    Matrix4d res = Matrix4d::Identity();
    res.topLeftCorner<3, 3>() = rot * scale_mat;
    res.col(3).head(3) = cube_obj.pos;//物体位置
    return res;
}
Matrix3Xd compute3D_BoxCorner(const cuboid &cube_obj)
{
    MatrixXd corners_body;
    corners_body.resize(3, 8);
    corners_body << 1, 1, -1, -1, 1, 1, -1, -1,
        1, -1, -1, 1, 1, -1, -1, 1,
        -1, -1, -1, -1, 1, 1, 1, 1;//以立方体中心为原点的坐标系，再考虑拉伸
    MatrixXd corners_world = homo_to_real_coord<double>(similarityTransformation(cube_obj) * real_to_homo_coord<double>(corners_body));
    //立方体坐标系坐标转换到世界坐标系坐标
    return corners_world;
}
```

obj_3d_util提供了一些校验与标准化的函数

JavaScript

```cpp
bool check_inside_box(const Vector2d &pt, const Vector2d &box_left_top, const Vector2d &box_right_bottom)
{
    return box_left_top(0) <= pt(0) && pt(0) <= box_right_bottom(0) && box_left_top(1) <= pt(1) && pt(1) <= box_right_bottom(1);
}//检查某点是否在2d bbox内，用于2d顶点的生成


void align_left_right_edges(MatrixXd &all_lines)
{
    for (int line_id = 0; line_id < all_lines.rows(); line_id++)
    {
        if (all_lines(line_id, 2) < all_lines(line_id, 0))
        {
```

```
            Vector2d temp = all_lines.row(line_id).tail<2>();
            all_lines.row(line_id).tail<2>() = all_lines.row(line_id).head<2>();
            all_lines.row(line_id).head<2>() = temp;
        }
    }
}//检查线段数据格式，是否保持起点在左，终点在右

void normalize_to_pi_vec(const VectorXd &raw_angles, VectorXd &new_angles)
{
    new_angles.resize(raw_angles.rows());
    for (int i = 0; i < raw_angles.rows(); i++)
        new_angles(i) = normalize_to_pi<double>(raw_angles(i));
}//将角度转化到[-90,90]的区间,主要用于求角度误差

void atan2_vector(const VectorXd &y_vec, const VectorXd &x_vec, VectorXd &all_angles)
{
    all_angles.resize(y_vec.rows());
    for (int i = 0; i < y_vec.rows(); i++)
        all_angles(i) = std::atan2(y_vec(i), x_vec(i)); // don't need normalize_to_pi, because my edges is from left to right, always [-90 90]
}//对于vector的atan2

// remove the jumping angles from -pi to pi.   to make the raw angles smoothly change.
void smooth_jump_angles(const VectorXd &raw_angles, VectorXd &new_angles)
{
    new_angles = raw_angles;
    if (raw_angles.rows() == 0)
        return;

    double angle_base = raw_angles(0); // choose a new base angle.   (assume that the all the angles lie in [-pi pi] around the base)
    for (int i = 0; i < raw_angles.rows(); i++)
    {
        if ((raw_angles(i) - angle_base) < -M_PI)
            new_angles(i) = raw_angles(i) + 2 * M_PI;
        else if ((raw_angles(i) - angle_base) > M_PI)
            new_angles(i) = raw_angles(i) - 2 * M_PI;
    }
}//更改跳跃点的值，使一组角度相差不会太大，如[-30,0,170]会被改为[-30,0,-10]

```

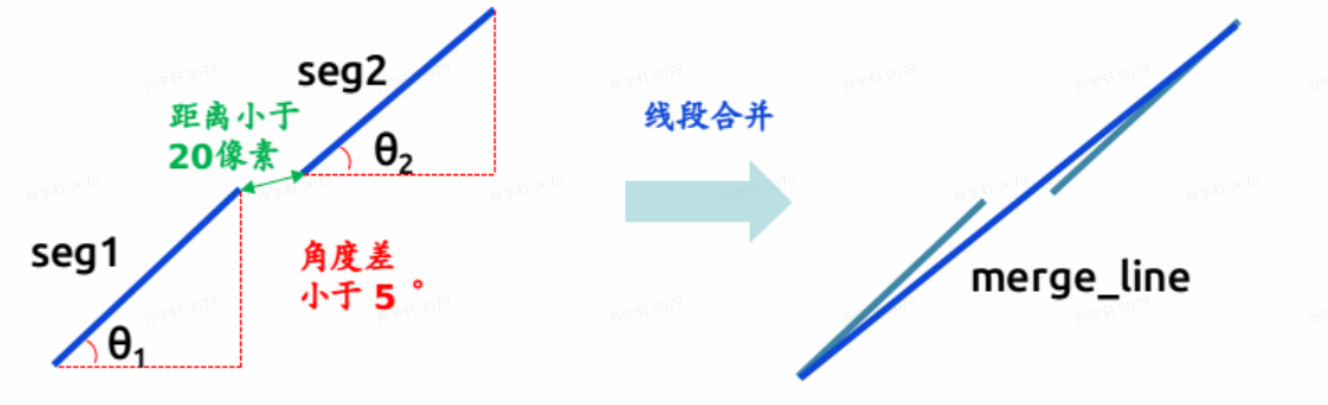
基本的交点处理，包括直线与bbox的交点，以及两条直线的交点

```
C++
```

```cpp
Vector2d seg_hit_boundary(const Vector2d& pt_start, const Vector2d& pt_end, const Vector4d& line_segment2 )//直线与bbox的交点，需要考虑bbox的边是垂直还是水平
{
    // 线段 line_segment2 的起点和终点的y坐标.
    Vector2d boundary_bgn = line_segment2.head<2>();
    Vector2d boundary_end = line_segment2.tail<2>();

    // 消失点与上边缘采样点构成线段的长度（x和y的长度）.
    Vector2d direc = pt_end - pt_start;
    Vector2d hit_pt(-1,-1);//找不到交点时返回的值

    // line equation is (p_u,p_v)+lambda*(delta_u,delta_v)  parameterized by lambda

    // 如果是水平边缘，两个点的y坐标相等.
    if ( boundary_bgn(1) == boundary_end(1) )   // if an horizontal edge
    {
        double lambd = (boundary_bgn(1)-pt_start(1))/direc(1);
        if (lambd >= 0)  // along ray direction
        {
            Vector2d hit_pt_tmp = pt_start + lambd * direc;
            if ( (boundary_bgn(0) <= hit_pt_tmp(0)) && (hit_pt_tmp(0) <= boundary_end(0)) )  // 得到的交点有可能在线段的延长线上，需要判断交点是否在线段内
            {
                hit_pt = hit_pt_tmp;
                hit_pt(1) = boundary_bgn(1);  // floor operations might have un-expected things
            }
            }
    }

    // 如果是垂直边缘.
    if ( boundary_bgn(0) == boundary_end(0) )   // if an vertical edge
    {
        double lambd=(boundary_bgn(0)-pt_start(0))/direc(0);
        if (lambd>=0)  // along ray direction
        {
            Vector2d hit_pt_tmp = pt_start+lambd*direc;
            if ( (boundary_bgn(1)<=hit_pt_tmp(1)) && (hit_pt_tmp(1)<=boundary_end(1)) )  //同上进行判断
            {
                hit_pt = hit_pt_tmp;
                hit_pt(0)= boundary_bgn(0);  // floor operations might have un-expected things
            }
        }
    }
```

```
    return hit_pt;
}

// compute two line intersection points, a simplified version compared to matlab
// 计算两条线的交点.
Vector2d lineSegmentIntersect(const Vector2d& pt1_start, const Vector2d& pt1_end,const Vector2d& pt2_start, const Vector2d& pt2_end,bool infinite_line)

Vector2f lineSegmentIntersect_f(const Vector2f& pt1_start, const Vector2f& pt1_end, const Vector2f& pt2_start, const Vector2f& pt2_end,float& extcond_1, float& extcond_2, bool infinite_line)

cv::Point2f lineSegmentIntersect_f(const cv::Point2f& pt1_start, const cv::Point2f& pt1_end, const cv::Point2f& pt2_start, const cv::Point2f& pt2_end, float& extcond_1, float& extcond_2, bool infinite_line)
```

对于读取的线检测edges数据，object_3d_util提供了线段合并与剔除的函数



C++

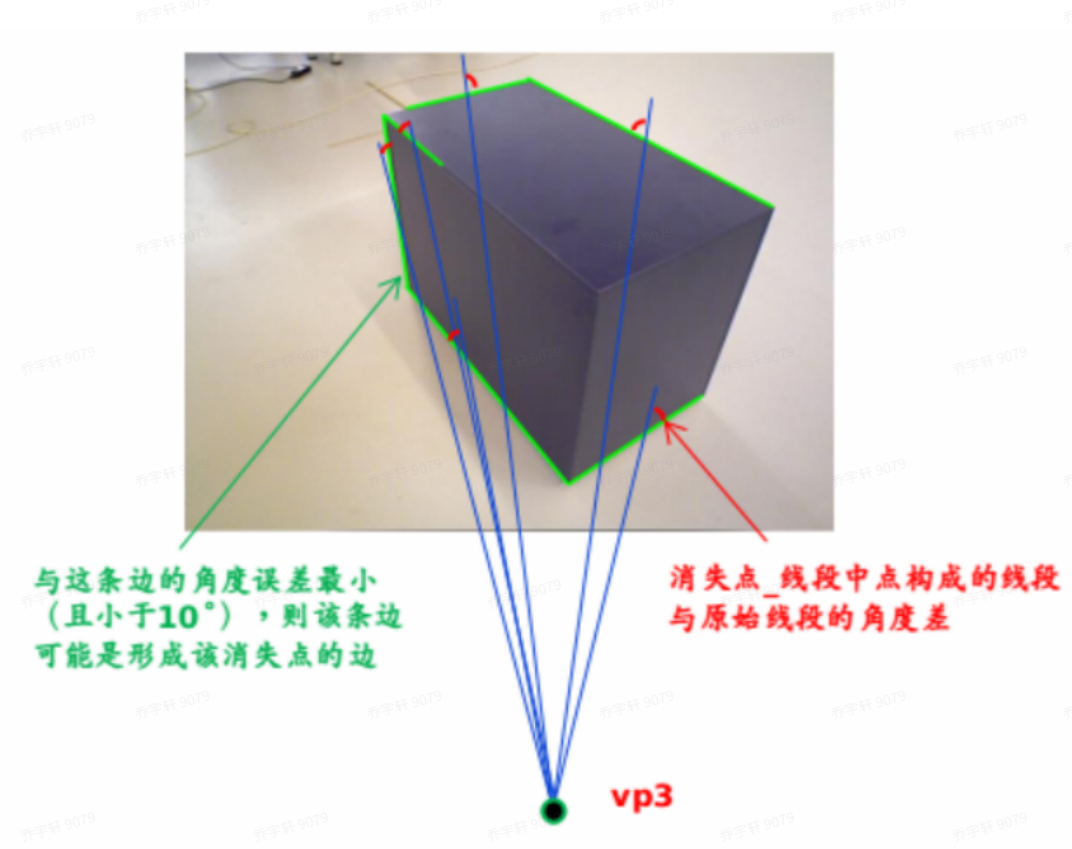
```
void merge_break_lines( const MatrixXd& all_lines,  //输入的所有在矩阵框内的线段矩阵
                        MatrixXd& merge_lines_out,  //输出的合并后的线段矩阵//
                        double pre_merge_dist_thre, //两条线段之间的距离阈值 20像素
                        double pre_merge_angle_thre_degree, //角度阈值 5//
                        double edge_length_threshold)       //长度阈值 30//
{
    bool can_force_merge = true;
    merge_lines_out = all_lines;
    int total_line_number = merge_lines_out.rows(); //线段条数:total_line_number将越来越小
    int counter = 0;
    double pre_merge_angle_thre = pre_merge_angle_thre_degree / 180.0 * M_PI;
    //角度转弧度
    while ((can_force_merge) && (counter < 500))//线段的合并
```

```
    {
        counter++;
        can_force_merge = false;
        MatrixXd line_vector =
merge_lines_out.topRightCorner(total_line_number, 2) -
merge_lines_out.topLeftCorner(total_line_number, 2);
        //线段向量：所有线段的右边点的坐标-左边点的坐标=每条线段的水平x长度和竖直y长度
        VectorXd all_angles;
        atan2_vector(line_vector.col(1), line_vector.col(0), all_angles);
        //根据线段向量求解每个线段的角度，线段已经规范化所以角度已经normalized
[-90,90]

        for (int seg1 = 0; seg1 < total_line_number - 1; seg1++)
        //开始对每个线段单独处理
        {
            for (int seg2 = seg1 + 1; seg2 < total_line_number; seg2++)
            {
                //遍历剩余线段，考虑两个线段
                double diff = std::abs(all_angles(seg1) - all_angles(seg2));
                double angle_diff = std::min(diff, M_PI - diff);
                //计算相邻两条线段的角度差
                if (angle_diff < pre_merge_angle_thre)//角度差与阈值比较
                {
                    double dist_1ed_to_2 = (merge_lines_out.row(seg1).tail(2)
- merge_lines_out.row(seg2).head(2)).norm();
                    double dist_2ed_to_1 = (merge_lines_out.row(seg2).tail(2)
- merge_lines_out.row(seg1).head(2)).norm();
                    //dist_1ed_to_2：线1尾到线2头的距离
                    //dist_2ed_to_1：线2尾到线1头的距离
                    if ((dist_1ed_to_2 < pre_merge_dist_thre) ||
(dist_2ed_to_1 < pre_merge_dist_thre))
                    {
                        //足够近的两条线段才能合并
                        Vector2d merge_start, merge_end;
                        if (merge_lines_out(seg1, 0) < merge_lines_out(seg2,
0))
                            merge_start = merge_lines_out.row(seg1).head(2);
                        else
                            merge_start = merge_lines_out.row(seg2).head(2);

                        if (merge_lines_out(seg1, 2) > merge_lines_out(seg2,
2))
                            merge_end = merge_lines_out.row(seg1).tail(2);
                        else
                            merge_end = merge_lines_out.row(seg2).tail(2);
                        //确定好合并之后线段的两个端点

                        double merged_angle = std::atan2(merge_end(1) -
```

```
merge_start(1), merge_end(0) - merge_start(0));
                        //求合并之后线段的角度

                        double temp = std::abs(all_angles(seg1) -
merged_angle);
                        double merge_angle_diff = std::min(temp, M_PI - temp);
                        //求原线段1与合并之后线段的角度差
                        if (merge_angle_diff < pre_merge_angle_thre)
                        {
                            //当该角度差也小于阈值时，进行合并
                            merge_lines_out.row(seg1).head(2) = merge_start;
                            merge_lines_out.row(seg1).tail(2) = merge_end;
                            fast_RemoveRow(merge_lines_out, seg2,
total_line_number);
                            //合并后的线段储存在merge_lines_out代替seg1,删去seg2
                            can_force_merge = true;
                            break;//每个线段每次只合并一个线段
                        }
                    }
                }
            }
            if (can_force_merge)
                break;
        }
    }
    if (edge_length_threshold > 0)
    {
        MatrixXd line_vectors =
merge_lines_out.topRightCorner(total_line_number, 2) -
merge_lines_out.topLeftCorner(total_line_number, 2);
        //重新计算合并之后线段的线段向量
        VectorXd line_lengths = line_vectors.rowwise().norm();
        int long_line_number = 0;
        MatrixXd long_merge_lines(total_line_number, 4);
        //存储每条线段的长度，将长线条保存在long_merge_lines中
        for (int i = 0; i < total_line_number; i++)
        {
            if (line_lengths(i) > edge_length_threshold)//比较阈值
            {
                long_merge_lines.row(long_line_number) =
merge_lines_out.row(i);
                long_line_number++;
            }
        }
        merge_lines_out = long_merge_lines.topRows(long_line_number);
        //将长线段放回merge_lines_out
    }
```

```
95      else
96          merge_lines_out.conservativeResize(total_line_number, NoChange);
97  }
```

回忆我们计算角度误差时，需要寻找边对应的消失点。计算消失点-线段中点所构成线段的角度，与检测到的线段的角度进行比较，如果角度差在阈值内，则该条线段可能是形成消失点的线段（理论上消失点和所支持的线段的中点是在一条线上的），记录下这条边的偏角和线段 ID。



object_3d_util中提供了VP_support_edge_infos来考虑VP点的支撑边

C++

```
1  Eigen::MatrixXd VP_support_edge_infos(  Eigen::MatrixXd& VPs,//消失点矩阵
2                                          Eigen::MatrixXd& edge_mid_pt//每条线段的
   中点
3                                          Eigen::VectorXd& edge_angles,//每条线段
   的偏角
4                                          Eigen::Vector2d
   vp_support_angle_thres)//消失点与边的夹角阈值
5  {
6      MatrixXd all_vp_bound_edge_angles = MatrixXd::Ones(3, 2) * nan(""); //
   initialize as nan  use isnan to check
7      if (edge_mid_pts.rows() > 0)
8      {
9          //遍历三个VP点，依次处理
10         for (int vp_id = 0; vp_id < VPs.rows(); vp_id++)
11         {
12             double vp_angle_thre;
```

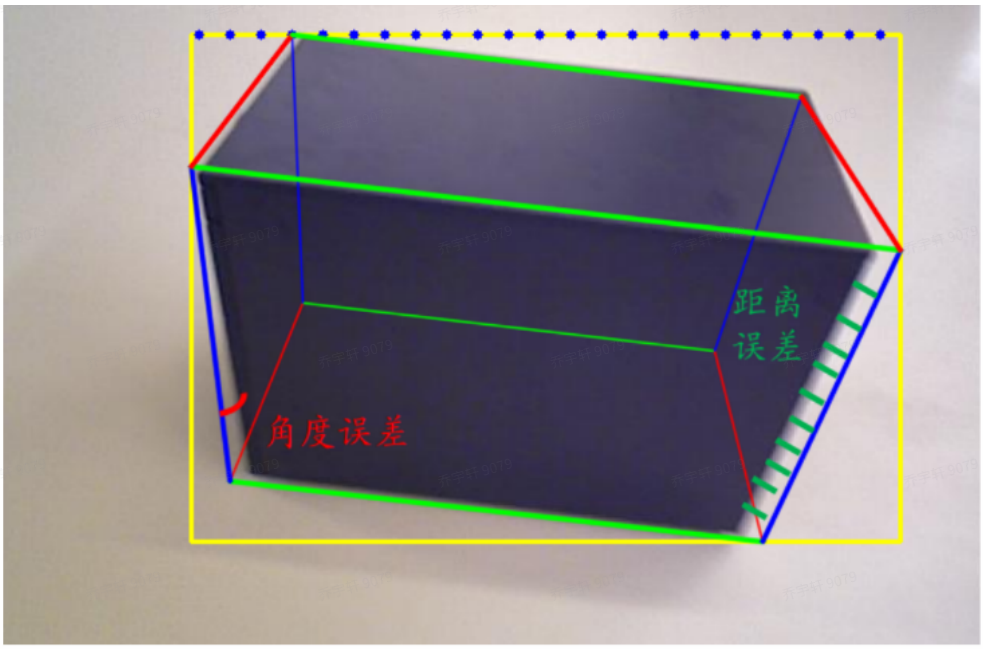
```
            if (vp_id != 2)//消失点 1 2 的夹角阈值 15
                vp_angle_thre = vp_support_angle_thres(0) / 180.0 * M_PI;
            else //消失点 3 的夹角阈值 10
                vp_angle_thre = vp_support_angle_thres(1) / 180.0 * M_PI;

            std::vector<int> vp_inlier_edge_id;//存储在范围内的边的 id
            VectorXd vp_edge_midpt_angle_raw_inlier(edge_angles.rows());
            // 边与第 vp_id 个消失点角度差在范围内的角度

            for (int edge_id = 0; edge_id < edge_angles.rows(); edge_id++)
            {
                double vp1_edge_midpt_angle_raw_i = atan2(edge_mid_pts(edge_id, 1) - VPs(vp_id, 1), edge_mid_pts(edge_id, 0) - VPs(vp_id, 0));
                double vp1_edge_midpt_angle_norm_i = normalize_to_pi<double>(vp1_edge_midpt_angle_raw_i);
                double angle_diff_i = std::abs(edge_angles(edge_id) - vp1_edge_midpt_angle_norm_i);
                //求消失点到中点连线的角度并normalize到[-90,90],因为不确定左右需要normalize
                //计算消失点与中点连线与边角度的角度差
                angle_diff_i = std::min(angle_diff_i, M_PI - angle_diff_i);
                if (angle_diff_i < vp_angle_thre)
                {
                    vp_edge_midpt_angle_raw_inlier(vp_inlier_edge_id.size()) = vp1_edge_midpt_angle_raw_i;
                    vp_inlier_edge_id.push_back(edge_id);
                }//角度差小于阈值时，将edge_id放到容器中，并记录角度
            }
            if (vp_inlier_edge_id.size() > 0) // if found inlier edges
            {
                VectorXd vp1_edge_midpt_angle_raw_inlier_shift;

                smooth_jump_angles(vp_edge_midpt_angle_raw_inlier.head(vp_inlier_edge_id.size()),
                                   vp1_edge_midpt_angle_raw_inlier_shift);
                //对角度进行平滑处理，因为提高泛化性能通过选取最大和最小，保证这一目的需要平滑
                int vp1_low_edge_id;

                vp1_edge_midpt_angle_raw_inlier_shift.maxCoeff(&vp1_low_edge_id);
                int vp1_top_edge_id;

                vp1_edge_midpt_angle_raw_inlier_shift.minCoeff(&vp1_top_edge_id);
                //考虑角度最大的边与角度最小的边
                if (vp_id > 0)
                    std::swap(vp1_low_edge_id, vp1_top_edge_id);
```

```
    // match matlab code
49                  all_vp_bound_edge_angles(vp_id, 0) =
    edge_angles(vp_inlier_edge_id[vp1_low_edge_id]); // it will be 0*1 matrix if
    not found inlier edges.
50                  all_vp_bound_edge_angles(vp_id, 1) =
    edge_angles(vp_inlier_edge_id[vp1_top_edge_id]);
51                  //将角度数据保存作为函数返回，用于计算角度误差
52              }
53          }
54      }
55      return all_vp_bound_edge_angles;
56  }
```

生成提案还需考虑距离误差，计算距离误差时需要object_3d_util中的box_edge_sum_dists函数

C++

```cpp
double box_edge_sum_dists(  const cv::Mat& dist_map,            //距离变换图
                            const MatrixXd& box_corners_2d,     //8 个顶点的 2D 坐标
                            const MatrixXi& edge_pt_ids,        //可见的边
                            bool  reweight_edge_distance)
{
    // give some edges, sample some points on line then sum up distance from dist_map
    // input: visible_edge_pt_ids is n*2  each row stores an edge's two end point's index from box_corners_2d
    // if weight_configs: for configuration 1, there are more visible edges compared to configuration2, so we need to re-weight
    // [1 2;2 3;3 4;4 1;2 6;3 5;4 8;5 8;5 6];  reweight vertical edge id 5-7 by 2/3, horizontal edge id 8-9 by 1/2
    float sum_dist = 0;
    for (int edge_id = 0; edge_id < edge_pt_ids.rows(); edge_id++)//遍历可见边
    {
        Vector2d corner_tmp1 = box_corners_2d.col(edge_pt_ids(edge_id, 0));
        Vector2d corner_tmp2 = box_corners_2d.col(edge_pt_ids(edge_id, 1));
        //每条可见边两个点的2d 坐标

        for (double sample_ind = 0; sample_ind < 11; sample_ind++)
        {
            //循环进行采样，每次只采样一个点
            Vector2d sample_pt = sample_ind / 10.0 * corner_tmp1 + (1 - sample_ind / 10.0) * corner_tmp2;
            float dist1 = dist_map.at<float>(int(sample_pt(1)), int(sample_pt(0))); //make sure dist_map is float type
            //在dist_map上找到对应距离
            if (reweight_edge_distance)
            {
                if ((4 <= edge_id) && (edge_id <= 5))
                    dist1 = dist1 * 3.0 / 2.0;
                if (6 == edge_id)
                    dist1 = dist1 * 2.0;
            }
            //如果在dist计算中reweight_edge_distance=true，则对5,6,7条边更加信任
            sum_dist = sum_dist + dist1;
        }
    }
    return double(sum_dist);
}
```



除了距离误差外，生成提案还需考虑角度误差，object_3d_util提供了box_edge_alignment_angle_error来计算立方体边角度与VP点支撑边角度的角度差

$$\phi_{angle}(O,I) = \Sigma_{i=1:3} \| < l_{i\_ms}, l_{i\_mt} > - < VP_i, l_{i\_mt} > \| + \| < l_{i\_ns}, l_{i\_nt} > - < VP_i, l_{i\_nt} > \| \tag{18}$$

C++

```cpp
double box_edge_alignment_angle_error(  const MatrixXd& all_vp_bound_edge_angles, //消失点与边的两个角度
                                        const MatrixXi& vps_box_edge_pt_ids,//每个消失点来源的两条边，6条边，12个顶点序号
                                        const MatrixXd& box_corners_2d) //8 个顶点的 2D坐标
{
    // compute the difference of box edge angle with angle of actually VP aligned image edges. for evaluating the box
    // all_vp_bound_edge_angles: VP aligned actual image angles. 3*2  if not found, nan.      box_corners_2d: 2*8
    // vps_box_edge_pt_ids: % six edges. each row represents two edges [e1_1 e1_2   e2_1 e2_2;...] of one VP
    double total_angle_diff = 0;
    double not_found_penalty = 30.0 / 180.0 * M_PI * 2;//找不到VP点支撑边时固定误差
    for (int vp_id = 0; vp_id < vps_box_edge_pt_ids.rows(); vp_id++)
    {
        //遍历三个VP点分别对应的两条边
        Vector2d vp_bound_angles = all_vp_bound_edge_angles.row(vp_id);
        //读取VP点对应的两个支撑边的角度
```

```
        //读取VP点对应的两个支撑边的角度
        std::vector<double> vp_bound_angles_valid;
        for (int i = 0; i < 2; i++)
            if (!std::isnan(vp_bound_angles(i)))
                vp_bound_angles_valid.push_back(vp_bound_angles(i));//存在支撑边，放入valid
        if (vp_bound_angles_valid.size() > 0) //  exist valid edges
        {
            for (int ee_id = 0; ee_id < 2; ee_id++)
            // find cloeset from two boundary edges. we could also do left-left right-right compare. but pay close attention different vp locations
            {
                Vector2d two_box_corners_1 = box_corners_2d.col(vps_box_edge_pt_ids(vp_id, 2 * ee_id));     // [ x1;y1 ]
                Vector2d two_box_corners_2 = box_corners_2d.col(vps_box_edge_pt_ids(vp_id, 2 * ee_id + 1)); // [ x2;y2 ]
                //ee_id=0时对应第1 第2个点的坐标，ee_id=1时对应第3 第4个点的坐标

                double box_edge_angle = normalize_to_pi(atan2(two_box_corners_2(1) - two_box_corners_1(1),
                        two_box_corners_2(0) - two_box_corners_1(0))); // [-pi/2 -pi/2]
                //计算边的角度并进行normalize

                double angle_diff_temp = 100;
                for (int i = 0; i < vp_bound_angles_valid.size(); i++)
                {
                    double temp = std::abs(box_edge_angle - vp_bound_angles_valid[i]);
                    //计算角度误差，消失点与支撑边顶点的角度 边的角度 两者之差
                    temp = std::min(temp, M_PI - temp);
                    if (temp < angle_diff_temp)
                        angle_diff_temp = temp; //误差设定不超过100，避免离群点影响
                }
                total_angle_diff = total_angle_diff + angle_diff_temp;
            }
        }
        else
            total_angle_diff = total_angle_diff + not_found_penalty;
            //如果没有找到VP点的valid支撑边，则固定误差
    }
    return total_angle_diff;
}
```

对于距离误差与角度误差，我们分配权重进行融合，fuse_normalize_scores_v2

```
C++
```

```cpp
void fuse_normalize_scores_v2(  const VectorXd& dist_error,          //距离误差
                                const VectorXd& angle_error,         // 角度误差
                                VectorXd& combined_scores,           // 综合得分
                                std::vector<int>& final_keep_inds,   // 最终纳入计算的测量的ID
                                                                     //ID是box proposal的ID
                                double weight_vp_angle,              // 角度误差的权重
                                bool whether_normalize)              // 是否归一化两个误差

    int raw_data_size = dist_error.rows();//选择距离误差的数量作为raw_data的大小
    if (raw_data_size > 4)
    {
        int breaking_num = round(float(raw_data_size) / 3.0 * 2.0);
        //breaking_num为需要部分排序的数量，设置为前2/3
        std::vector<int> dist_sorted_inds(raw_data_size);
        std::iota(dist_sorted_inds.begin(), dist_sorted_inds.end(), 0);
        std::vector<int> angle_sorted_inds = dist_sorted_inds;
        //dist_sorted_inds和angle_sorted_inds都为生成的有序向量

        sort_indexes(dist_error, dist_sorted_inds, breaking_num);
        sort_indexes(angle_error, angle_sorted_inds, breaking_num);
        //对距离误差和角度误差进行递增排序取前2/3

        std::vector<int> dist_keep_inds = std::vector<int>(dist_sorted_inds.begin(), dist_sorted_inds.begin() + breaking_num - 1); // keep best 2/3
        //根据距离误差的排序保存前2/3到dist_keep_inds

        if (angle_error(angle_sorted_inds[breaking_num - 1]) > angle_error(angle_sorted_inds[breaking_num - 2]))
        {
            //判断[breaking_num-1]与[breaking_num-2]个角度误差的大小关系，根据排序有两种关系
            //如果是小于，认为到末尾仍未达到max=100;如果是等于，则认为末端都已达到max=100

            std::vector<int> angle_keep_inds = std::vector<int>(angle_sorted_inds.begin(), angle_sorted_inds.begin() + breaking_num - 1); // keep best 2/3
            //未达到max的话，我们将角度误差的前2/3保存

            std::
```

```
34              std::sort(dist_keep_inds.begin(), dist_keep_inds.end());
35              std::sort(angle_keep_inds.begin(), angle_keep_inds.end());
36              //对距离误差和角度误差的ID进行排序
37
38              std::set_intersection(dist_keep_inds.begin(),
    dist_keep_inds.end(),
39                                    angle_keep_inds.begin(),
    angle_keep_inds.end(),
40                                    std::back_inserter(final_keep_inds));
41              //寻找两个ID序列的交集放到最后的final_keep_inds,即纳入考量的ID集合
42          }
43          else //don't need to consider angle.   my angle error has maximum. may
    already saturate at breaking pt.
44          {
45              final_keep_inds = dist_keep_inds;
46              //如果认为达到了max,考量ID则全由距离误差提供
47          }
48      }
49      else
50      {
51          //如果raw_data_size本来就很小，就将它们全部纳入考量
52          final_keep_inds.resize(raw_data_size); //don't change anything.
53          std::iota(final_keep_inds.begin(), final_keep_inds.end(), 0);
54      }
55
56      int new_data_size = final_keep_inds.size();
57      // find max/min of kept errors.
58      double min_dist_error = 1e6;
59      double max_dist_error = -1;
60      double min_angle_error = 1e6;
61      double max_angle_error = -1;
62      VectorXd dist_kept(new_data_size);
63      VectorXd angle_kept(new_data_size);
64      for (int i = 0; i < new_data_size; i++)
65      {
66          double temp_dist = dist_error(final_keep_inds[i]);
67          double temp_angle = angle_error(final_keep_inds[i]);
68          min_dist_error = std::min(min_dist_error, temp_dist);
69          max_dist_error = std::max(max_dist_error, temp_dist);
70          min_angle_error = std::min(min_angle_error, temp_angle);
71          max_angle_error = std::max(max_angle_error, temp_angle);
72          dist_kept(i) = temp_dist;
73          angle_kept(i) = temp_angle;
74      }
75      //dist_kept,angle_kept保存纳入考量ID对应的距离误差与角度误差
76      //记录角度误差，距离误差的最大最小值
77
78      if (whether_normalize && (new_data_size > 1)) //判断是否需要归一化，避免量纲影
```

```
    响
    {
        combined_scores = (dist_kept.array() - min_dist_error) /
(max_dist_error - min_dist_error);
        //距离误差归一化  （所有的距离 - 最小距离值）/（最大距离 - 最小距离）
        if ((max_angle_error - min_angle_error) > 0)
        {
            angle_kept = (angle_kept.array() - min_angle_error) /
(max_angle_error - min_angle_error);
            //角度误差归一化  （所有角度误差 - 最小角度误差） /  （最大角度误差 - 最小角
度误差）
            combined_scores = (combined_scores + weight_vp_angle * angle_kept)
/ (1 + weight_vp_angle);
            //联合角度与距离  （距离误差 + 角度权重×角度误差）/（1 + 角度权重）
        }
        else
            combined_scores = (combined_scores + weight_vp_angle * angle_kept)
/ (1 + weight_vp_angle);
            //最大角度误差=最小角度误差时，联合分数由距离误差决定
    }
    else
        combined_scores = (dist_kept + weight_vp_angle * angle_kept) / (1 +
weight_vp_angle);
        //不需要归一化的的话，进行粗暴的加权
}
```

关于线与平面的交点计算，以及2d坐标恢复3d坐标，object_3d_util也提供了相关函数

设直线过(m_1,m_2,m_3),方向向量为(v_1,v_2,v_3)，平面过点(n_1,n_2,n_3),法线为$(vp_1，vp_2，vp_3)$,
交点坐标为$x=m_1+v_1t,y=m_2+v_2t,z=m_3+v_3t,$
其中$t=\frac{((n_1-m_1)vp_1+(n_2-m_2)vp_2+(n_3-m_3)vp_3)}{(vp_1v_1+vp_2v_2+vp_3v_3)}$.
在我们的问题中，选取相机参考系，射线过$(0,0,0)$,方向向量为$K^{-1}p_5$,平面法向为$\mathbf{n}$，设平面经过点$(0,0,a)$,则根据距离m求得$a=-m/n_3,\mathbf{n}=[n_1,n_2,n_3]$,从而$P_1=-\frac{m}{\mathbf{n}^T(K^{-1}p_5)}K^{-1}p_1$.

C++

```
void ray_plane_interact(const MatrixXd &rays,//射线为3*n，每列都是从原点开始的一条
射线
                        const Eigen::Vector4d &plane,//平面为4个参数构成的向量，
[n,m]
                        MatrixXd &intersections)//返回交点坐标 3*n
{
    VectorXd frac = -plane[3] / (plane.head(3).transpose() * rays).array();
//n*1
    intersections = frac.transpose().replicate<3,1>().array() * rays.array();
}

```

```
void plane_hits_3d(const Matrix4d &transToWolrd, //相机关于wolrd的变换矩阵，相机位姿
                   const Matrix3d &invK, //内参的逆矩阵
                   const Vector4d &plane_sensor, //传感器坐标系中的平面参数，一般是相机坐标系
                         MatrixXd pixels,  //平面上四个点的2d坐标
                         Matrix3Xd &pts_3d_world) //输出世界坐标系下的3d坐标
// compute ray intersection with plane in 3D.
// transToworld: 4*4 camera pose.   invK: inverse of calibration.   plane: 1*4 plane equation in sensor frame.
// pixels  2*n; each column is a pt [x;y] x is horizontal,y is vertical
outputs: pts3d 3*n in world frame
{
    pixels.conservativeResize(3, NoChange);
    pixels.row(2) = VectorXd::Ones(pixels.cols());//2d坐标齐次化
    MatrixXd pts_ray = invK * pixels; //每个点反投影出射线
    MatrixXd pts_3d_sensor;
    ray_plane_interact(pts_ray, plane_sensor, pts_3d_sensor);//反投影射线与平面的交点，相机坐标系下3d坐标
    pts_3d_world = homo_to_real_coord<double>(transToWolrd *
real_to_homo_coord<double>(pts_3d_sensor)); //
    //相机坐标系下的3d坐标变换到世界坐标系下，再转换到真实坐标
}

Vector4d get_wall_plane_equation(const Vector3d &gnd_seg_pt1,
                                 const Vector3d &gnd_seg_pt2)
// 1*6 a line segment in 3D. [x1 y1 z1  x2 y2 z2]  z1=z2=0  or  two 1*3
//根据地面上一条线段立起垂直于地面的墙，返回墙的sensor参数
{

    Vector3d partwall_normal_world = (gnd_seg_pt1 -
gnd_seg_pt2).cross(Vector3d(0, 0, 1)); // [0,0,1] is world ground plane
    //墙的法向垂直于线段方向以及地面法向，我们对两个向量做叉积则得到墙的法向
    partwall_normal_world.array() /= partwall_normal_world.norm();//单位化
    double dist = -partwall_normal_world.transpose() * gnd_seg_pt1;//求第四个参数距离
    Vector4d plane_equation;
    plane_equation << partwall_normal_world, dist; // wall plane in world frame
    if (dist < 0)
        plane_equation = -plane_equation; // make all the normal pointing inside the room. neamly, pointing to the camera
    return plane_equation;
}

void change_2d_corner_to_3d_object(const MatrixXd &box_corners_2d_float, //8个顶点的2d坐标
                                   const Vector3d &configs, //模式（1或2）,vp1的
```

```
位置，yaw角
                                const Vector4d &ground_plane_sensor,//相机系下的地平面
                                const Matrix4d &transToWolrd, //变换矩阵
                                const Matrix3d &invK, //内参的逆
                                Eigen::Matrix<double, 3, 4> &projectionMatrix,//投影矩阵
                                cuboid &sample_obj)//3d 提案
{
    Matrix3Xd obj_gnd_pt_world_3d;
    plane_hits_3d(transToWolrd, invK, ground_plane_sensor, box_corners_2d_float.rightCols(4), obj_gnd_pt_world_3d); //% 3*n each column is a 3D point  floating point
    //通过底部的四个顶点的2d坐标计算它们的3d坐标

    double length_half = (obj_gnd_pt_world_3d.col(0) - obj_gnd_pt_world_3d.col(3)).norm() / 2; // along object x direction   corner 5-8
    double width_half = (obj_gnd_pt_world_3d.col(0) - obj_gnd_pt_world_3d.col(1)).norm() / 2;  // along object y direction   corner 5-6
    //根据地面计算length_half与width_half，即scale中的两项

    Vector4d partwall_plane_world = get_wall_plane_equation(obj_gnd_pt_world_3d.col(0), obj_gnd_pt_world_3d.col(1)); //% to compute height, need to unproject-hit-planes  formed by 5-6 corner
    Vector4d partwall_plane_sensor = transToWolrd.transpose() * partwall_plane_world;                                  // wall plane in sensor frame
    //通过5,6点计算世界坐标系与相机坐标系中的wall_plane

    Matrix3Xd obj_top_pt_world_3d;
    plane_hits_3d(transToWolrd, invK, partwall_plane_sensor, box_corners_2d_float.col(1), obj_top_pt_world_3d); // should match obj_gnd_pt_world_3d  % compute corner 2
    //根据wall_plane平面计算其中两个顶点的3d坐标

    double height_half = obj_top_pt_world_3d(2, 0) / 2;
    //计算高度的一般，得到了scale

    double mean_obj_x = obj_gnd_pt_world_3d.row(0).mean();
    double mean_obj_y = obj_gnd_pt_world_3d.row(1).mean();
    //求底部长方形的中心，来求box的平移量

    double vp_1_position = configs(1);//传递vp1的位置，左或者右
    double yaw_esti = configs(2); //传递yaw角
    sample_obj.pos = Vector3d(mean_obj_x, mean_obj_y, height_half);
```

```
78      sample_obj.rotY = yaw_esti;
79      sample_obj.scale = Vector3d(length_half, width_half, height_half);
80      sample_obj.box_config_type = configs.head<2>();
81      //将所有cuboid相关参数传递到sample_obj
82
83      VectorXd cuboid_to_raw_boxstructIds(8);//8个点的编号
84      if (vp_1_position == 1) // vp1 on left, for all configurations
85          cuboid_to_raw_boxstructIds << 6, 5, 8, 7, 2, 3, 4, 1;
86      if (vp_1_position == 2) // vp1 on right, for all configurations
87          cuboid_to_raw_boxstructIds << 5, 6, 7, 8, 3, 2, 1, 4;
88
89      Matrix2Xi box_corners_2d_int = box_corners_2d_float.cast<int>();
90      //将float类型的2d坐标转换成int类型
91      sample_obj.box_corners_2d.resize(2, 8);
92      for (int i = 0; i < 8; i++)
93          sample_obj.box_corners_2d.col(i) =
    box_corners_2d_int.col(cuboid_to_raw_boxstructIds(i) - 1); // minius one to
    match index
94          //按照编号顺序将2d 坐标放入cuboid中
95
96      sample_obj.box_corners_3d_world = compute3D_BoxCorner(sample_obj);
97      //根据现有cuboid提案可以计算8个点的3d坐标
98  }
99
```

生成立方体提案必不可少的需要vp点的坐标，坐标的计算如下

$$vp_x = K \cdot R^{-1} \cdot (\cos(yaw), \sin(yaw), 0)^T$$

$$vp_y = K \cdot R^{-1} \cdot (-\sin(yaw), \cos(yaw), 0)^T$$

$$vp_z = K \cdot R^{-1} \cdot (0, 0, 1)^T$$

C++

```
void getVanishingPoints(const Matrix3d &KinvR, //K*invR
                        double yaw_esti, //yaw角
                        Vector2d &vp_1, Vector2d &vp_2, Vector2d &vp_3) //vp点坐标
{
    vp_1 = homo_to_real_coord_vec<double>(KinvR * Vector3d(cos(yaw_esti), sin(yaw_esti), 0));  // for object x axis
    vp_2 = homo_to_real_coord_vec<double>(KinvR * Vector3d(-sin(yaw_esti), cos(yaw_esti), 0)); // for object y axis
    vp_3 = homo_to_real_coord_vec<double>(KinvR * Vector3d(0, 0, 1));
// for object z axis
}
```

最后，object_3d_util提供了如下两个函数

C++

```
float bboxOverlapratio(const cv::Rect &rect1, const cv::Rect &rect2)
//求两个长方形的重叠比例
int pointBoundaryDist(const cv::Rect &rect, const cv::Point2f &kp)
//求某点到长方形四条边距离中的最小
```

box_proposal_detail.cpp

detect_3d_cuboid的最后一个部分是box_proposal_detail，主要包括参数传递与主接口detect_cuboid

参数传递主要包括设定内参，设定cam_pose，并求欧拉角，旋转矩阵等

C++

```
void detect_3d_cuboid::set_calibration(const Matrix3d &Kalib)//内参
{
    cam_pose.Kalib = Kalib;
    cam_pose.invK = Kalib.inverse();
}

void detect_3d_cuboid::set_cam_pose(const Matrix4d &transToWolrd)
//cam_pose，相机坐标系到世界坐标系的变换矩阵
{
    cam_pose.transToWolrd = transToWolrd;
    cam_pose.rotationToWorld = transToWolrd.topLeftCorner<3, 3>();
    Vector3d euler_angles;
    quat_to_euler_zyx(Quaterniond(cam_pose.rotationToWorld), euler_angles(0), euler_angles(1), euler_angles(2));
    cam_pose.euler_angle = euler_angles;
    cam_pose.invR = cam_pose.rotationToWorld.inverse();
    cam_pose.projectionMatrix = cam_pose.Kalib * transToWolrd.inverse().topRows<3>(); // project world coordinate to camera
    cam_pose.KinvR = cam_pose.Kalib * cam_pose.invR;
    cam_pose.camera_yaw = cam_pose.euler_angle(2);
    //TODO relative measure? not good... then need to change transToWolrd.
}
```

下面为box_proposal_detail的主要部分，也是detect_3d_object模块的主接口detect_cuboid

PHP

```
void detect_3d_cuboid::detect_cuboid(const cv::Mat &rgb_img, //输入图片
                                     const Matrix4d &transToWolrd, //变换矩阵
                                     const MatrixXd &obj_bbox_coors, //2d bbox
                                     MatrixXd all_lines_raw, //线检测数据
                                     std::vector<ObjectSet> &all_object_cuboids)
                                     //立方体提案都存放在all_object_cuboids容器中
```

我们先看循环之前的数据读取与参数设定

C++

```
set_cam_pose(transToWolrd);
cam_pose_raw = cam_pose;
//在detect_3d_cuboid类中传递cam_pose，即transToWorld

cv::Mat gray_img;
if (rgb_img.channels() == 3)
```

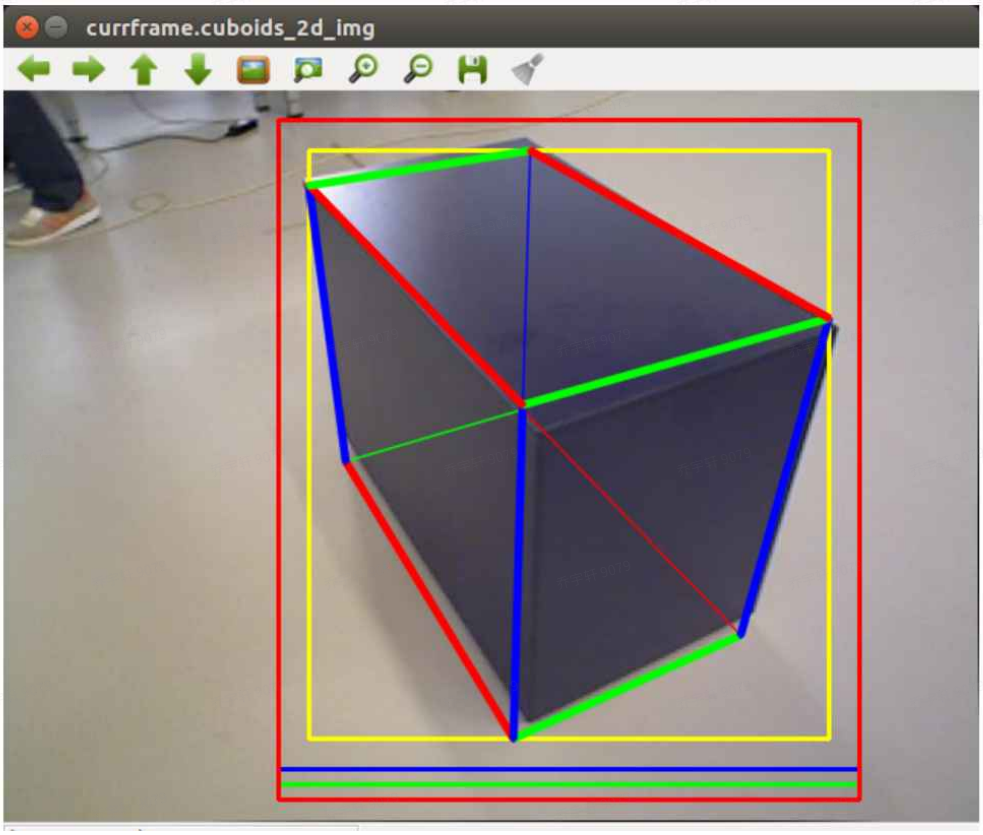
```
if (rgb_img.channels() == 3)
    cv::cvtColor(rgb_img, gray_img, CV_BGR2GRAY);
else
    gray_img = rgb_img;
//转换成灰度图保存到gray_img

int img_width = rgb_img.cols;
int img_height = rgb_img.rows;
//图像宽度高度

int num_2d_objs = obj_bbox_coors.rows();//2d bbox数量
all_object_cuboids.resize(num_2d_objs);//存储2d bbox数据

vector<bool> all_configs;//考虑三面情形与两面情形
all_configs.push_back(consider_config_1);
all_configs.push_back(consider_config_2);

// parameters for cuboid generation
double vp12_edge_angle_thre = 15;
double vp3_edge_angle_thre = 10;     //求支撑边时的阈值
double shorted_edge_thre = 20;       //剔除短边时的阈值
bool reweight_edge_distance = true; //计算距离误差时的权重分配


// parameters for proposal scoring
bool whether_normalize_two_errors = true;//启用归一化
double weight_vp_angle = 0.8; //角度误差权重
double weight_skew_error = 1.5; //长宽比惩罚权重
// if also consider config2, need to weight two erros, in order to compare two configurations

align_left_right_edges(all_lines_raw); // this should be guaranteed when detecting edges
//检查是否起点在左，终点在右，保证不需要normalize

if (whether_plot_detail_images)
{
    cv::Mat output_img;
    plot_image_with_edges(rgb_img, output_img, all_lines_raw, cv::Scalar(255, 0, 0));
    cv::imshow("Raw detected Edges", output_img); //cv::waitKey(0);
}//绘制带线检测的图像

// find ground-wall boundary edges
Vector4d ground_plane_world(0, 0, 1, 0); //地平面在世界坐标系下的表示
Vector4d ground_plane_sensor = cam_pose.transToWolrd.transpose() * ground_plane_world;
```

读取了cam_pose，内参，bbox等数据，设定好参数后，进入循环遍历2d bbox数据

C++

```cpp
for (int object_id = 0; object_id < num_2d_objs; object_id++)
{
    //根据2d bbox数据遍历object_id
    ca::Profiler::tictoc("One 3D object total time");//计时
    int left_x_raw = obj_bbox_coors(object_id, 0);
    int top_y_raw = obj_bbox_coors(object_id, 1);
    int obj_width_raw = obj_bbox_coors(object_id, 2);
    int obj_height_raw = obj_bbox_coors(object_id, 3);
    int right_x_raw = left_x_raw + obj_bbox_coors(object_id, 2);
    int down_y_raw = top_y_raw + obj_height_raw;
    //读取bbox的左侧右侧的x值，以及顶端与底端的y值

    std::vector<int> down_expand_sample_all;
    down_expand_sample_all.push_back(0);
    if (whether_sample_bbox_height) //2d检测可能不准确，添加bbox的高度采样
    {
        int down_expand_sample_ranges = max(min(20, obj_height_raw - 90), 20);
        down_expand_sample_ranges = min(down_expand_sample_ranges, img_height - top_y_raw - obj_height_raw - 1); // should lie inside the image  -1 for c++ index
        //图像高度 - 边界框左上角y轴坐标 - 检测框高度 - 1
        if (down_expand_sample_ranges > 10)
// if expand large margin, give more samples.
            down_expand_sample_all.push_back(round(down_expand_sample_ranges / 2));
        down_expand_sample_all.push_back(down_expand_sample_ranges);
    }

    // NOTE later if in video, could use previous object yaw..., also reduce search range
    double yaw_init = cam_pose.camera_yaw - 90.0 / 180.0 * M_PI; // yaw init is directly facing the camera, align with camera optical axis
    std::vector<double> obj_yaw_samples;
    linespace<double>(yaw_init - 45.0 / 180.0 * M_PI, yaw_init + 45.0 / 180.0 * M_PI, 6.0 / 180.0 * M_PI, obj_yaw_samples);
    //然后以初始化的yaw角为中心的-45到+45的90范围内，每隔6采样一个值，采样得到15个yaw

    MatrixXd all_configs_errors(400, 9);
    MatrixXd all_box_corners_2ds(800, 8);   // initialize a large eigen matrix
    int valid_config_number_all_height = 0; // all valid objects of all height samples
    ObjectSet raw_obj_proposals; //ObjectSet为cuboid存储的容器
    raw_obj_proposals.reserve(100);序列长度为100
    //      int sample_down_expan_id=1;
```

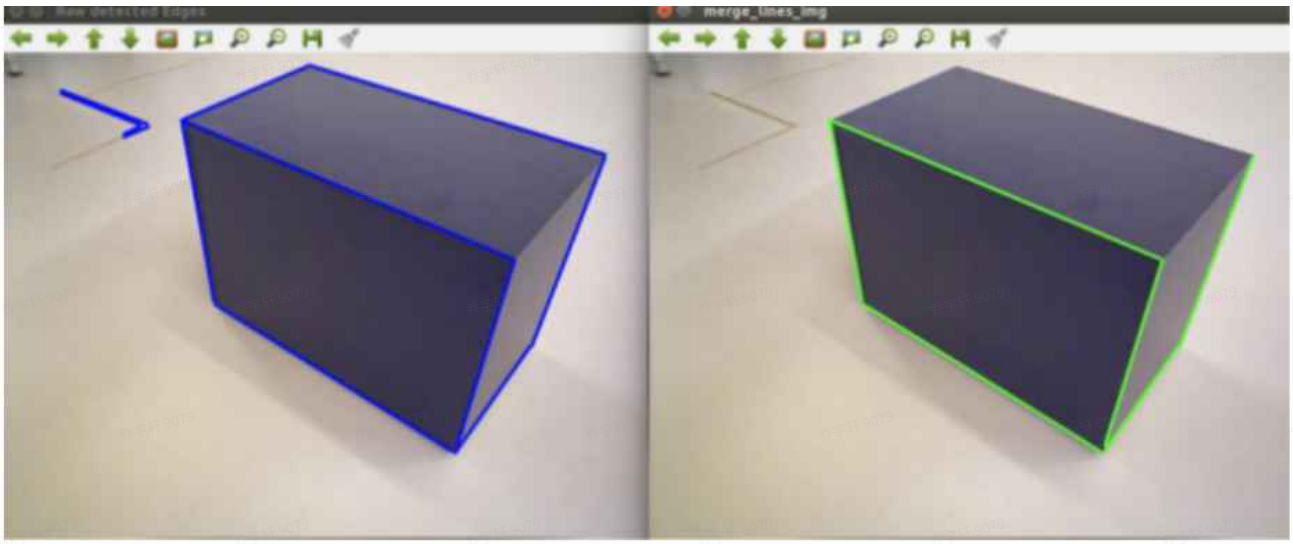
```
    //      int sample_down_expan_id = 1;
    for (int sample_down_expan_id = 0; sample_down_expan_id < down_expand_sample_all.size(); sample_down_expan_id++)
    {
        //不进行采样的话，只进入一次循环，如果两次push_back的话进行三次循环
        int down_expand_sample = down_expand_sample_all[sample_down_expan_id];
        int obj_height_expan = obj_height_raw + down_expand_sample;
        int down_y_expan = top_y_raw + obj_height_expan;
        double obj_diaglength_expan = sqrt(obj_width_raw * obj_width_raw + obj_height_expan * obj_height_expan);
        //读入采样扩展的高度，计算扩展后的底边与对角线情况

        // sample points on the top edges, if edge is too large, give more samples. give at least 10 samples for all edges. for small object, object pose changes lots
        int top_sample_resolution = round(min(20, obj_width_raw / 10)); //  25 pixels
        std::vector<int> top_x_samples;
        linespace<int>(left_x_raw + 5, right_x_raw - 5, top_sample_resolution, top_x_samples);
       //进行上边缘点采样，如果边很长，则提供更多样本。为所有边缘提供至少10个样本。
        MatrixXd sample_top_pts(2, top_x_samples.size());
        for (int ii = 0; ii < top_x_samples.size(); ii++)
        {
            sample_top_pts(0, ii) = top_x_samples[ii];
            sample_top_pts(1, ii) = top_y_raw;
        }
        //存储顶边采样的点

        // expand some small margin for distance map  [10 20]
        int distmap_expand_wid = min(max(min(20, obj_width_raw - 100), 10), max(min(20, obj_height_expan - 100), 10));
        int left_x_expan_distmap = max(0, left_x_raw - distmap_expand_wid);
        int right_x_expan_distmap = min(img_width - 1, right_x_raw + distmap_expand_wid);
        int top_y_expan_distmap = max(0, top_y_raw - distmap_expand_wid);
        int down_y_expan_distmap = min(img_height - 1, down_y_expan + distmap_expand_wid);
        int height_expan_distmap = down_y_expan_distmap - top_y_expan_distmap;
        int width_expan_distmap = right_x_expan_distmap - left_x_expan_distmap;
        Vector2d expan_distmap_lefttop = Vector2d(left_x_expan_distmap, top_y_expan_distmap);
        Vector2d expan_distmap_rightbottom = Vector2d(right_x_expan_distmap, down_y_expan_distmap);
        //拓宽检测框的边界,并计算拓宽后的各项数据
        //扩展检测框主要用于局部DT变换
```



黄色的是原始的2D检测边框，蓝色、绿色和红色的检测框分别是采样了高度之后拓宽的边框

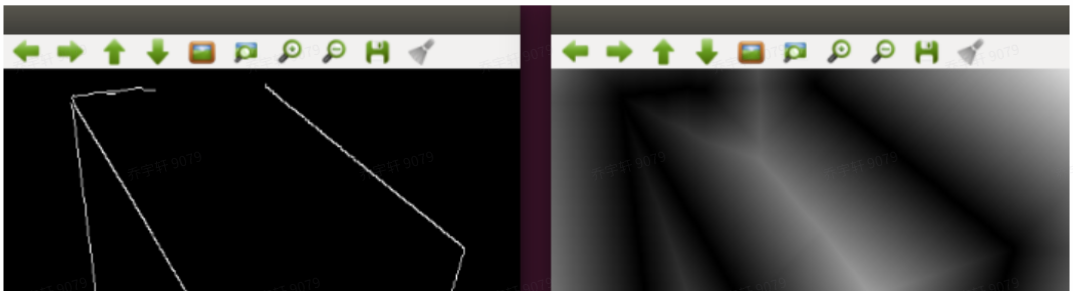
C++

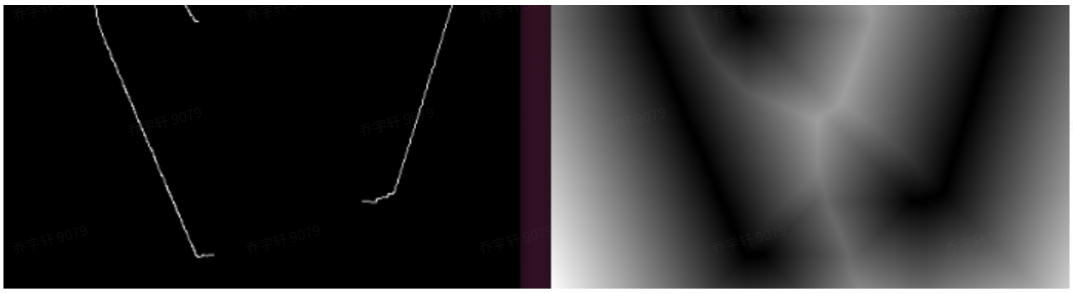
```
        // find edges inside the object bounding box
        MatrixXd all_lines_inside_object(all_lines_raw.rows(),
all_lines_raw.cols()); // first allocate a large matrix, then only use the
toprows to avoid copy, alloc
        int inside_obj_edge_num = 0;
        for (int edge_id = 0; edge_id < all_lines_raw.rows(); edge_id++)
            if (check_inside_box(all_lines_raw.row(edge_id).head<2>(),
expan_distmap_lefttop, expan_distmap_rightbottom))
                if (check_inside_box(all_lines_raw.row(edge_id).tail<2>(),
expan_distmap_lefttop, expan_distmap_rightbottom))
                {
                    all_lines_inside_object.row(inside_obj_edge_num) =
all_lines_raw.row(edge_id);
                    inside_obj_edge_num++;
                }
        //遍历线检测数据，利用check_inside_box筛选两个端点都在bbox内的线条

        // merge edges and remove short lines, after finding object edges.
edge merge in small regions should be faster than all.
        double pre_merge_dist_thre = 20;
        double pre_merge_angle_thre = 5;
        double edge_length_threshold = 30;
        MatrixXd all_lines_merge_inobj;

merge_break_lines(all_lines_inside_object.topRows(inside_obj_edge_num),
all_lines_merge_inobj, pre_merge_dist_thre,
                          pre_merge_angle_thre, edge_length_threshold);

        //设定线段合并与剔除的阈值，并进行线段的合并与剔除
```



C++

```
        // compute edge angels and middle points
        VectorXd lines_inobj_angles(all_lines_merge_inobj.rows());
        MatrixXd edge_mid_pts(all_lines_merge_inobj.rows(), 2);
        for (int i = 0; i < all_lines_merge_inobj.rows(); i++)
        {
            lines_inobj_angles(i) = std::atan2(all_lines_merge_inobj(i, 3) - all_lines_merge_inobj(i, 1), all_lines_merge_inobj(i, 2) - all_lines_merge_inobj(i, 0)); // [-pi/2 -pi/2]
            edge_mid_pts.row(i).head<2>() = (all_lines_merge_inobj.row(i).head<2>() + all_lines_merge_inobj.row(i).tail<2>()) / 2;
        }
        //对所有在bbox内的线条求角度与中点，用于计算vp点的支撑边与角度误差

        // TODO could canny or distance map outside sampling height to speed up!!!!   Then only need to compute canny onces.
        // detect canny edges and compute distance transform  NOTE opencv canny maybe different from matlab. but roughly same
        cv::Rect object_bbox = cv::Rect(left_x_expan_distmap, top_y_expan_distmap, width_expan_distmap, height_expan_distmap); //
        cv::Mat im_canny;
        cv::Canny(gray_img(object_bbox), im_canny, 80, 200); // low thre, high thre    im_canny 0 or 255   [80 200  40 100]
        cv::Mat dist_map;
        cv::distanceTransform(255 - im_canny, dist_map, CV_DIST_L2, 3); // dist_map is float datatype
        //拓宽后的边界框放入object_bbox,检测canny边缘放入im_canny,做DT变换放入dist_map
        if (whether_plot_detail_images)
        {
            cv::imshow("im_canny", im_canny);
            cv::Mat dist_map_img;
            cv::normalize(dist_map, dist_map_img, 0.0, 1.0, cv::NORM_MINMAX);
            cv::imshow("normalized distance map", dist_map_img);
            cv::waitKey();
        }
        //绘制图像
```





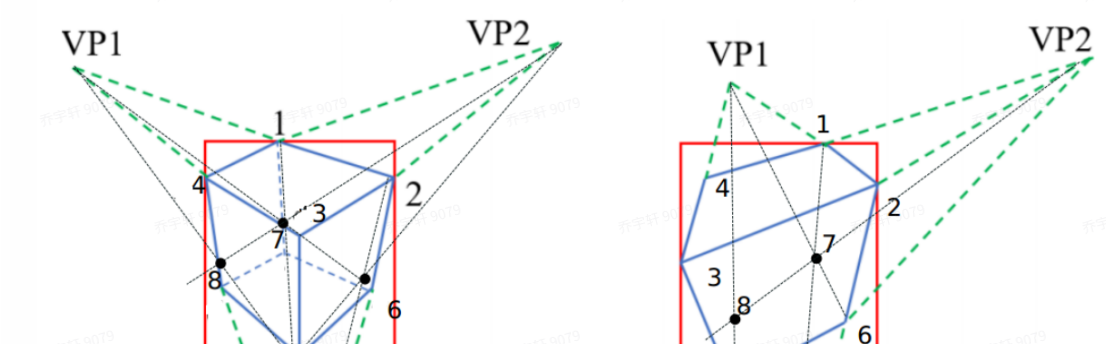
C++

```
        // Generate cuboids
        MatrixXd all_configs_error_one_objH(200, 9);
        MatrixXd all_box_corners_2d_one_objH(400, 8);
        int valid_config_number_one_objH = 0;

        std::vector<double> cam_roll_samples;
        std::vector<double> cam_pitch_samples;
        if (whether_sample_cam_roll_pitch)
        {
            linespace<double>(cam_pose_raw.euler_angle(0) - 6.0 / 180.0 * M_PI, cam_pose_raw.euler_angle(0) + 6.0 / 180.0 * M_PI, 3.0 / 180.0 * M_PI, cam_roll_samples);
            linespace<double>(cam_pose_raw.euler_angle(1) - 6.0 / 180.0 * M_PI, cam_pose_raw.euler_angle(1) + 6.0 / 180.0 * M_PI, 3.0 / 180.0 * M_PI, cam_pitch_samples);
        }
        //采样相机 roll pitch 角，在相机偏角的+-6每隔3采样一个值或者直接使用相机的roll pitch角.
        else
        {
            cam_roll_samples.push_back(cam_pose_raw.euler_angle(0));
            cam_pitch_samples.push_back(cam_pose_raw.euler_angle(1));
        }
        // different from matlab. first for loop yaw, then for configurations.
        //        int obj_yaw_id=8;
        for (int cam_roll_id = 0; cam_roll_id < cam_roll_samples.size(); cam_roll_id++)
            for (int cam_pitch_id = 0; cam_pitch_id < cam_pitch_samples.size(); cam_pitch_id++)
                for (int obj_yaw_id = 0; obj_yaw_id < obj_yaw_samples.size(); obj_yaw_id++)
```

```
24                  {
25                      //根据cam_roll,cam_pitch,cam_yaw的采样生成提案
26                      if (whether_sample_cam_roll_pitch)
27                      {
28                          Matrix4d transToWolrd_new = transToWolrd;
29                          transToWolrd_new.topLeftCorner<3, 3>() =
    euler_zyx_to_rot<double>(cam_roll_samples[cam_roll_id],
    cam_pitch_samples[cam_pitch_id], cam_pose_raw.euler_angle(2));
30                          set_cam_pose(transToWolrd_new);
31                          ground_plane_sensor =
    cam_pose.transToWolrd.transpose() * ground_plane_world;
32                      }
33                      //采样roll pitch的话，根据采样重新计算相机坐标系下的地平面参数
34
35                      double obj_yaw_esti = obj_yaw_samples[obj_yaw_id];//取出yaw
    角
36
37                      Vector2d vp_1, vp_2, vp_3;
38                      getVanishingPoints(cam_pose.KinvR, obj_yaw_esti, vp_1,
    vp_2, vp_3); // for object x y z  axis
39                      //根据采样的yaw角计算vp点
40
41                      MatrixXd all_vps(3, 2);
42                      all_vps.row(0) = vp_1;
43                      all_vps.row(1) = vp_2;
44                      all_vps.row(2) = vp_3;
45                      //        std::cout<<"obj_yaw_esti  "<<obj_yaw_esti<<"  "
    <<obj_yaw_id<<std::endl;
46                      MatrixXd all_vp_bound_edge_angles =
    VP_support_edge_infos(all_vps, edge_mid_pts, lines_inobj_angles,
47
    Vector2d(vp12_edge_angle_thre, vp3_edge_angle_thre));
48                      //根据vp点坐标，边的中点，以及边的角度，我们计算vp点的支撑边
49                      //        int sample_top_pt_id=15;
50                      for (int sample_top_pt_id = 0; sample_top_pt_id <
    sample_top_pts.cols(); sample_top_pt_id++)
51                      {
52                          //根据顶边上顶点的采样生成立方体提案
53                          //            std::cout<<"sample_top_pt_id "
    <<sample_top_pt_id<<std::endl;
54                          Vector2d corner_1_top =
    sample_top_pts.col(sample_top_pt_id);
55                          bool config_good = true;
56                          int vp_1_position = 0; // 0 initial as fail,  1  on
    left   2 on right
57                          //开始计算第二个点
58                          Vector2d corner_2_top = seg_hit_boundary(vp_1,
    corner_1_top, Vector4d(right_x_raw, top_y_raw, right_x_raw, down_y_expan));
```

```
59                          //检查vp1-上边缘采样点的射线是否与右边线段有交集
60                          if (corner_2_top(0) == -1)
61                          { // vp1-corner1 doesn't hit the right boundary. check whether hit left
62                              //没有交点的话检查左侧，与左侧有交点的话，vp1在右
63                              corner_2_top = seg_hit_boundary(vp_1, corner_1_top, Vector4d(left_x_raw, top_y_raw, left_x_raw, down_y_expan));
64                              if (corner_2_top(0) != -1) // vp1-corner1 hit the left boundary   vp1 on the right
65                                  vp_1_position = 2;
66                          }
67                          else // vp1-corner1 hit the right boundary   vp1 on the left
68                              vp_1_position = 1;
69
70                          config_good = vp_1_position > 0;//检查vp1情况
71                          if (!config_good)
72                          {
73                              if (print_details)
74                                  //vp_1_position=0表示左边与右边的线段上都找不到交点
75                                  printf("Configuration fails at corner 2, outside segment\n");
76                              continue;
77                          }
78                          if ((corner_1_top - corner_2_top).norm() < shorted_edge_thre)
79                          {
80                              if (print_details)
81                                  printf("Configuration fails at edge 1-2, too short\n");
82                              continue;
83                          }//考虑顶点1与顶点2形成的边，若长度过短
84                          //              cout<<"corner_1/2    "<<corner_1_top.transpose()<<"   "<<corner_2_top.transpose()<<endl;
85                          //              int config_ind=0; // have to consider config now.
```

8.4 计算物体 8 个点的 2D坐标



注意到对于两个情形，顶点3，4的生成方式是不同的

C++

```
                    for (int config_id = 1; config_id < 3; config_id++) // configuration one or two of matlab version
                    {    //对上图中的两种情形考虑，config_id=1,2
                        if (!all_configs[config_id - 1])
                            continue;//默认设置两个都是true
                        Vector2d corner_3_top, corner_4_top;
                        if (config_id == 1)
                        {
                            //情形1下，顶点4是顶点2所在边的另一边上的交点
                            if (vp_1_position == 1) // then vp2 hit the left boundary
                                corner_4_top = seg_hit_boundary(vp_2, corner_1_top, Vector4d(left_x_raw, top_y_raw, left_x_raw, down_y_expan));
                            else // or, then vp2 hit the right boundary
                                corner_4_top = seg_hit_boundary(vp_2, corner_1_top, Vector4d(right_x_raw, top_y_raw, right_x_raw, down_y_expan));
                            if (corner_4_top(1) == -1)
                            {
                                config_good = false;
                                //如果另一侧找不到交点的话，则生成出错
                                if (print_details)
                                    printf("Configuration %d fails at corner 4, outside segment\n", config_id);
                                continue;
                            }
                            if ((corner_1_top - corner_4_top).norm() < shorted_edge_thre)
```

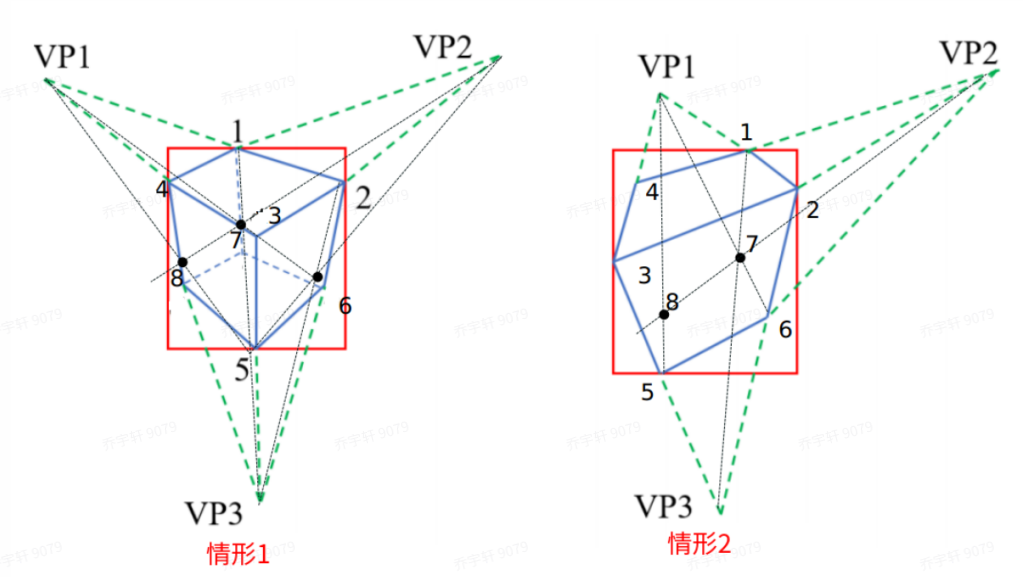
```
shorted_edge_thre)
                                    {
                                        if (print_details)
                                            printf("Configuration %d fails at edge 1-4, too short\n", config_id);
                                        continue;
                                    }//同样的考虑边的长度，后面将省略

                                    // compute the last point in the top face
                                    corner_3_top = lineSegmentIntersect(vp_2, corner_2_top, vp_1, corner_4_top, true);
                                    //情形1的顶点3考虑vp2与顶点2连线，vp1与顶点4连线，两者交点
                                    if (!check_inside_box(corner_3_top, Vector2d(left_x_raw, top_y_raw), Vector2d(right_x_raw, down_y_expan)))
                                    { // check inside boundary. otherwise edge visibility might be wrong
                                        //顶点3不是bbox上的点，我们需要计算是否inside box
                                        config_good = false;
                                        if (print_details)
                                            printf("Configuration %d fails at corner 3, outside box\n", config_id);
                                        continue;
                                    }
                                    if (((corner_3_top - corner_4_top).norm() < shorted_edge_thre) || ((corner_3_top - corner_2_top).norm() < shorted_edge_thre))
                                    {
                                        if (print_details)
                                            printf("Configuration %d fails at edge 3-4/3-2, too short\n", config_id);
                                        continue;
                                    }
                                    //                cout<<"corner_3/4   "<<corner_3_top.transpose()<<"   "<<corner_4_top.transpose()<<endl;
                                }
                                if (config_id == 2)
                                //情形2的计算实际上只是顶点3与顶点4计算方法的互换
                                {
                                    if (vp_1_position == 1) // then vp2 hit the left boundary
                                        corner_3_top = seg_hit_boundary(vp_2, corner_2_top, Vector4d(left_x_raw, top_y_raw, left_x_raw, down_y_expan));
                                    else // or, then vp2 hit the right boundary
                                        corner_3_top = seg_hit_boundary(vp_2, corner_2_top, Vector4d(right_x_raw, top_y_raw, right_x_raw, down_y_expan));
                                    if (corner_3_top(1) == -1)
```

```
                                {
                                    config_good = false;
                                    if (print_details)
                                        printf("Configuration %d fails at corner 3, outside segment\n", config_id);
                                    continue;
                                }
                                if ((corner_2_top - corner_3_top).norm() < shorted_edge_thre)
                                {
                                    if (print_details)
                                        printf("Configuration %d fails at edge 2-3, too short\n", config_id);
                                    continue;
                                }
                                // compute the last point in the top face
                                corner_4_top = lineSegmentIntersect(vp_1, corner_3_top, vp_2, corner_1_top, true);
                                if (!check_inside_box(corner_4_top, Vector2d(left_x_raw, top_y_expan_distmap), Vector2d(right_x_raw, down_y_expan_distmap)))
                                {
                                    config_good = false;
                                    if (print_details)
                                        printf("Configuration %d fails at corner 4, outside box\n", config_id);
                                    continue;
                                }
                                if (((corner_3_top - corner_4_top).norm() < shorted_edge_thre) || ((corner_4_top - corner_1_top).norm() < shorted_edge_thre))
                                {
                                    if (print_details)
                                        printf("Configuration %d fails at edge 3-4/4-1, too short\n", config_id);
                                    continue;
                                }
                                //                cout<<"corner_3/4   "<<corner_3_top.transpose()<<"   "<<corner_4_top.transpose()<<endl;
                            }
                            // compute first bottom points    computing bottom points is the same for config 1,2
                            Vector2d corner_5_down = seg_hit_boundary(vp_3, corner_3_top, Vector4d(left_x_raw, down_y_expan, right_x_raw, down_y_expan));
                            //计算bbox边上的底部点顶点5，vp3与顶点3的连线与底边的交点
                            //底部的四个点计算方法对于两种情形而言相同，考虑高度采样
                            if (corner_5_down(1) == -1)
```

```
89                              {
90                                  config_good = false;
91                                  if (print_details)
92                                      printf("Configuration %d fails at corner
    5, outside segment\n", config_id);
93                                  continue;
94                              }
95                              if ((corner_3_top - corner_5_down).norm() <
    shorted_edge_thre)
96                              {
97                                  if (print_details)
98                                      printf("Configuration %d fails at edge 3-
    5, too short\n", config_id);
99                                  continue;
100                             }
101                             Vector2d corner_6_down =
    lineSegmentIntersect(vp_2, corner_5_down, vp_3, corner_2_top, true);
102                             //vp2和底部点5的连线与消失点vp3与顶部点2连线的交点
103                             if (!check_inside_box(corner_6_down,
    expan_distmap_lefttop, expan_distmap_rightbottom))
104                             {
105                                 //同样检查是否inside_box内，先前的check用的不是
    expan拓展框
106                                 config_good = false;
107                                 if (print_details)
108                                     printf("Configuration %d fails at corner
    6, outside box\n", config_id);
109                                 continue;
110                             }
111                             if (((corner_6_down - corner_2_top).norm() <
    shorted_edge_thre) || ((corner_6_down - corner_5_down).norm() <
    shorted_edge_thre))
112                             {
113                                 if (print_details)
114                                     printf("Configuration %d fails at edge 6-
    5/6-2, too short\n", config_id);
115                                 continue;
116                             }
117                             Vector2d corner_7_down =
    lineSegmentIntersect(vp_1, corner_6_down, vp_3, corner_1_top, true);
118                             //vp1和底部点6的连线与消失点vp3与顶部点1连线的交点
119                             if (!check_inside_box(corner_7_down,
    expan_distmap_lefttop, expan_distmap_rightbottom))
120                             { // might be slightly different from matlab
121                                 config_good = false;
122                                 if (print_details)
123                                     printf("Configuration %d fails at corner
    7, outside box\n", config_id);
```

```
7, outside box\n", config_id);
                                continue;
                            }
                            if (((corner_7_down - corner_1_top).norm() < shorted_edge_thre) || ((corner_7_down - corner_6_down).norm() < shorted_edge_thre))
                            {
                                if (print_details)
                                    printf("Configuration %d fails at edge 7-1/7-6, too short\n", config_id);
                                continue;
                            }
                            Vector2d corner_8_down = lineSegmentIntersect(vp_1, corner_5_down, vp_2, corner_7_down, true);
                            //vp1和底部点5的连线与消失点vp2与底部点7连线的交点
                            if (!check_inside_box(corner_8_down, expan_distmap_lefttop, expan_distmap_rightbottom))
                            {
                                config_good = false;
                                if (print_details)
                                    printf("Configuration %d fails at corner 8, outside box\n", config_id);
                                continue;
                            }
                            if (((corner_8_down - corner_4_top).norm() < shorted_edge_thre) || ((corner_8_down - corner_5_down).norm() < shorted_edge_thre) || ((corner_8_down - corner_7_down).norm() < shorted_edge_thre))
                            {
                                if (print_details)
                                    printf("Configuration %d fails at edge 8-4/8-5/8-7, too short\n", config_id);
                                continue;
                            }

                            MatrixXd box_corners_2d_float(2, 8);
                            box_corners_2d_float << corner_1_top, corner_2_top, corner_3_top, corner_4_top, corner_5_down, corner_6_down, corner_7_down, corner_8_down;
                            //            std::cout<<"box_corners_2d_float \n"<<box_corners_2d_float<<std::endl;
                            //存储物体的8个顶点
                            MatrixXd box_corners_2d_float_shift(2, 8);
                            box_corners_2d_float_shift.row(0) = box_corners_2d_float.row(0).array() - left_x_expan_distmap;
                            box_corners_2d_float_shift.row(1) = box_corners_2d_float.row(1).array() - top_y_expan_distmap;
                            //计算偏移shift,8个顶点x坐标距离左边拓展边界框边界的距离
```

```
和y坐标距离上边拓展边界的距离
156
```



C++

```
1                           MatrixXi visible_edge_pt_ids, vps_box_edge_pt_ids;
2                           double sum_dist;
3                           if (config_id == 1)
4                           {
5                               visible_edge_pt_ids.resize(9, 2);
6                               visible_edge_pt_ids << 1, 2, 2, 3, 3, 4, 4, 1,
    2, 6, 3, 5, 4, 8, 5, 8, 5, 6;
7                               //情形1的9条可见边1-2,2-3,3-4,1-4,2-6,3-5,4-8,5-
    6,5-8
8                               vps_box_edge_pt_ids.resize(3, 4);
9                               vps_box_edge_pt_ids << 1, 2, 8, 5, 4, 1, 5, 6,
    4, 8, 2, 6; // six edges. each row represents two edges [e1_1 e1_2   e2_1
    e2_2;...] of one VP
10                              //形成vp点的边，1-2,5-8生成vp1;1-4,5-6生成vp2;4-
    8,2-6生成vp3
11                              visible_edge_pt_ids.array() -= 1;
12                              vps_box_edge_pt_ids.array() -= 1; // change to
    c++ index
13                              sum_dist = box_edge_sum_dists(dist_map,
    box_corners_2d_float_shift, visible_edge_pt_ids);
14                              //dist_map是在扩展边界框里做的，故2d corners坐标要
    用_shift
15                              //基于可见边采样求距离误差
16                          }
17                          else
18                          {
19                              visible_edge_pt_ids.resize(7, 2);
20                              visible_edge_pt_ids << 1, 2, 2, 3, 3, 4, 4, 1,
```

```
2, 6, 3, 5, 5, 6;
                                //情形2相比于情形1少了两条边5-8,4-8，都不可见
                                vps_box_edge_pt_ids.resize(3, 4);
                                vps_box_edge_pt_ids << 1, 2, 3, 4, 4, 1, 5, 6,
3, 5, 2, 6; // six edges. each row represents two edges [e1_1 e1_2   e2_1
e2_2;...] of one VP
                                //1-2,3-4;4-1,5-6;3-5,2-6 形成vp点的边
                                visible_edge_pt_ids.array() -= 1;
                                vps_box_edge_pt_ids.array() -= 1;
                                sum_dist = box_edge_sum_dists(dist_map,
box_corners_2d_float_shift, visible_edge_pt_ids, reweight_edge_distance);
                            }
                            double total_angle_diff =
box_edge_alignment_angle_error(all_vp_bound_edge_angles, vps_box_edge_pt_ids,
box_corners_2d_float);
                            //all_vp_bound_edge_anglr是形成vp点的支撑边角度，来自
线检测
                            //vps_box_edge_pt为理论图中形成消失点的编号，根据2d坐标
也可求角度


all_configs_error_one_objH.row(valid_config_number_one_objH).head<4>() =
Vector4d(config_id, vp_1_position, obj_yaw_esti, sample_top_pt_id);

all_configs_error_one_objH.row(valid_config_number_one_objH).segment<3>(4) =
Vector3d(sum_dist / obj_diaglength_expan, total_angle_diff,
down_expand_sample);
                            //all_configs_error_one_objH存储所有的误差项以及参数
                            //config情形，vp_1左或右,使用的yaw角，使用的顶点
                            //除以对角线长的距离误差，角度误差，高度采样后的底边
                            if (whether_sample_cam_roll_pitch)

all_configs_error_one_objH.row(valid_config_number_one_objH).segment<2>(7) =
Vector2d(cam_roll_samples[cam_roll_id], cam_pitch_samples[cam_pitch_id]);
                            else

all_configs_error_one_objH.row(valid_config_number_one_objH).segment<2>(7) =
Vector2d(cam_pose_raw.euler_angle(0), cam_pose_raw.euler_angle(1));
                            //根据是否采样cam_roll和cam_pitch存储相应值
                            all_box_corners_2d_one_objH.block(2 *
valid_config_number_one_objH, 0, 2, 8) = box_corners_2d_float;
                            //all_box_corners_2d_one_objH储存8个点的2d坐标
                            valid_config_number_one_objH++;
                            //有效立方体提案数量+1 如果生成时continue则不会+1
                            if (valid_config_number_one_objH >=
all_configs_error_one_objH.rows())
                            {

```

```
all_configs_error_one_objH.conservativeResize(2 *
valid_config_number_one_objH, NoChange);

all_box_corners_2d_one_objH.conservativeResize(4 *
valid_config_number_one_objH, NoChange);
                            }//如果要存储不下了，进行resize
                        } // end of config loop
                    }     // end of top id
                }         // end of yaw

        //        std::cout<<"valid_config_number_one_hseight  "
<<valid_config_number_one_objH<<std::endl;
        //        std::cout<<"all_configs_error_one_objH  \n"
<<all_configs_error_one_objH.topRows(valid_config_number_one_objH)<<std::endl;
        //        MatrixXd all_corners =
all_box_corners_2d_one_objH.topRows(2*valid_config_number_one_objH);
        //        std::cout<<"all corners   "<<all_corners<<std::endl;

        VectorXd normalized_score;
        vector<int> good_proposal_ids;

fuse_normalize_scores_v2(all_configs_error_one_objH.col(4).head(valid_config_n
umber_one_objH),
all_configs_error_one_objH.col(5).head(valid_config_number_one_objH),
                                 normalized_score, good_proposal_ids,
weight_vp_angle, whether_normalize_two_errors);
        //进行归一化融合角度误差与距离误差，纳入考量的proposal放入good_proposal_id

        for (int box_id = 0; box_id < good_proposal_ids.size(); box_id++)
        {
            //遍历所有有效的立方体提案
            int raw_cube_ind = good_proposal_ids[box_id];

            if (whether_sample_cam_roll_pitch)
            {
                Matrix4d transToWolrd_new = transToWolrd;
                transToWolrd_new.topLeftCorner<3, 3>() =
euler_zyx_to_rot<double>(all_configs_error_one_objH(raw_cube_ind, 7),
all_configs_error_one_objH(raw_cube_ind, 8), cam_pose_raw.euler_angle(2));
                set_cam_pose(transToWolrd_new);
                ground_plane_sensor = cam_pose.transToWolrd.transpose() *
ground_plane_world;
            }//如果采样了roll角和pitch角，重新计算变换矩阵，用于计算坐标

            cuboid *sample_obj = new cuboid();
            change_2d_corner_to_3d_object(all_box_corners_2d_one_objH.block(2
* raw_cube_ind, 0, 2, 8), all_configs_error_one_objH.row(raw_cube_ind).head<3>
```

```
(),
                                                ground_plane_sensor,
cam_pose.transToWolrd, cam_pose.invK, cam_pose.projectionMatrix, *sample_obj);
            //调用2d转3d函数，保存的cuboid信息放入sample_obj
            //        sample_obj->print_cuboid();
            if ((sample_obj->scale.array() < 0).any())
                continue; // scale should be positive
            sample_obj->rect_detect_2d = Vector4d(left_x_raw, top_y_raw,
obj_width_raw, obj_height_raw);
            sample_obj->edge_distance_error =
all_configs_error_one_objH(raw_cube_ind, 4); // record the original error
            sample_obj->edge_angle_error =
all_configs_error_one_objH(raw_cube_ind, 5);
            sample_obj->normalized_error = normalized_score(box_id);
            double skew_ratio = sample_obj->scale.head(2).maxCoeff() /
sample_obj->scale.head(2).minCoeff();
            sample_obj->skew_ratio = skew_ratio;
            //计算长宽比并放入cuboid成员中
            sample_obj->down_expand_height =
all_configs_error_one_objH(raw_cube_ind, 6);
            //进行了高度采样的话放入成员down_expand_height中
            if (whether_sample_cam_roll_pitch)
            {
                sample_obj->camera_roll_delta =
all_configs_error_one_objH(raw_cube_ind, 7) - cam_pose_raw.euler_angle(0);
                sample_obj->camera_pitch_delta =
all_configs_error_one_objH(raw_cube_ind, 8) - cam_pose_raw.euler_angle(1);
            }
            else
            {
                sample_obj->camera_roll_delta = 0;
                sample_obj->camera_pitch_delta = 0;
            }
            //采样了roll和pitch的话更新成员中的delta
            raw_obj_proposals.push_back(sample_obj);
            //放入保存所有提案的容器raw_obj_proposals
        }
    } // end of differnet object height sampling

    // %finally rank all proposals. [normalized_error   skew_error]
    int actual_cuboid_num_small = std::min(max_cuboid_num,
(int)raw_obj_proposals.size());
    //max_cuboid_num=1,准确提案的数量为1
    VectorXd all_combined_score(raw_obj_proposals.size());
    for (int box_id = 0; box_id < raw_obj_proposals.size(); box_id++)
    {
        cuboid *sample_obj = raw_obj_proposals[box_id];
        double skew_error = weight_skew_error * std::max(sample_obj-
```

```
    >skew_ratio - nominal_skew_ratio, 0.0);
120         //（长宽比-先验）*权重，得到惩罚量
121         if (sample_obj->skew_ratio > max_cut_skew)
122             skew_error = 100;//长宽比过大的话，直接设置高惩罚进行排除
123         double new_combined_error = sample_obj->normalized_error +
    weight_skew_error * skew_error;
124         all_combined_score(box_id) = new_combined_error;
125     }//融合长宽比信息，进行最后打分
126 
127     std::vector<int> sort_idx_small(all_combined_score.rows());
128     iota(sort_idx_small.begin(), sort_idx_small.end(), 0);
129     //从0开始递增生成索引，用于排序
130     sort_indexes(all_combined_score, sort_idx_small, actual_cuboid_num_small);
131     //递增排序只要第一个，即误差最小
132     for (int ii = 0; ii < actual_cuboid_num_small; ii++) // use sorted index
133     {
134 
    all_object_cuboids[object_id].push_back(raw_obj_proposals[sort_idx_small[ii]])
    ;
135         //将最好的提案push_back到all_object_cuboids
136     }
137 
138     ca::Profiler::tictoc("One 3D object total time");//计算生成一个obj提案生成的
    时间
139 } // end of different objects
140 
141 if (whether_plot_final_images || whether_save_final_images)
142 {
143     cv::Mat frame_all_cubes_img = rgb_img.clone();
144     for (int object_id = 0; object_id < all_object_cuboids.size();
    object_id++)
145         if (all_object_cuboids[object_id].size() > 0)
146         {
147             plot_image_with_cuboid(frame_all_cubes_img,
    all_object_cuboids[object_id][0]);
148         }
149     if (whether_save_final_images)
150         cuboids_2d_img = frame_all_cubes_img;
151     if (whether_plot_final_images)
152     {
153         cv::imshow("frame_all_cubes_img", frame_all_cubes_img);
154         cv::waitKey(0);
155     }
156 }//绘制包含提案的图像，保存结果
```

orb_object_slam模块

orb_object_slam是在orb_slam上加入obj对象，其余基本实体对象与orb_slam相同，包括Frame，KeyFrame，MapPoint，Map，KeyFrame Database。每一个送入系统的视频帧都会构造一个 Frame，Frame 中比较重要的会设为 KeyFrame，每个 Frame 会提取很多 ORB 特征点（FeaturePoint），每一个 ORB 特征点可能会对应一个 MapPoint，同一个 MapPoint 会对应多个不同 Frame 中 ORB 特征，KeyFrames 和 MapPoints 构成了 Map，重要的 KeyFrame 会存入 KeyFrame Database，用于回环检测和重定位。

ros_mono.cc

cubeslam通过launch文件直接运行的是ros_mono，我们先看ros_mono.cc文件

C++

```
1
2  #include <iostream>
3  #include <algorithm>
4  #include <fstream>
5  #include <chrono>
6
7  #include <ros/ros.h>
8  #include <ros/package.h>
9  #include <cv_bridge/cv_bridge.h>
10
11 #include <opencv2/core/core.hpp>
12
13 #include "System.h"
14 #include "Parameters.h"
15 #include "tictoc_profiler/profiler.hpp"
16
17 using namespace std;
18
19 class ImageGrabber
20 {
21 public:
22     ImageGrabber(ORB_SLAM2::System *pSLAM) : mpSLAM(pSLAM) {}
23
24     void GrabImage(const sensor_msgs::ImageConstPtr &msg);
25
26     ORB_SLAM2::System *mpSLAM;
27 };
28
29 int main(int argc, char **argv)
30 {
31     ros::init(argc, argv, "Mono");
32     ros::start();
33     ca::Profiler::enable();
34
35     if (argc != 3)
```

```
35      if (argc != 3)
36      {
37          cerr << endl
38               << "Usage: rosrun ORB_SLAM2 Mono path_to_vocabulary
    path_to_settings" << endl;
39          ros::shutdown();
40          return 1;
41      }
42      ros::NodeHandle nh;
43
44      bool enable_loop_closing = true;
45      nh.param<bool>("enable_viewer", ORB_SLAM2::enable_viewer, true);
46      nh.param<bool>("enable_viewmap", ORB_SLAM2::enable_viewmap, true);
47      nh.param<bool>("enable_viewimage", ORB_SLAM2::enable_viewimage, true);
48      nh.param<bool>("enable_loop_closing", enable_loop_closing, true);
49      nh.param<bool>("parallel_mapping", ORB_SLAM2::parallel_mapping, true);
50
51      nh.param<bool>("whether_detect_object", ORB_SLAM2::whether_detect_object,
    false);
52      nh.param<bool>("whether_read_offline_cuboidtxt",
    ORB_SLAM2::whether_read_offline_cuboidtxt, false);
53      nh.param<bool>("associate_point_with_object",
    ORB_SLAM2::associate_point_with_object, false);
54
55      nh.param<bool>("whether_dynamic_object",
    ORB_SLAM2::whether_dynamic_object, false);
56      nh.param<bool>("remove_dynamic_features",
    ORB_SLAM2::remove_dynamic_features, false);
57
58      nh.param<bool>("mono_firstframe_truth_depth_init",
    ORB_SLAM2::mono_firstframe_truth_depth_init, false);
59      nh.param<bool>("mono_firstframe_Obj_depth_init",
    ORB_SLAM2::mono_firstframe_Obj_depth_init, false);
60      nh.param<bool>("mono_allframe_Obj_depth_init",
    ORB_SLAM2::mono_allframe_Obj_depth_init, false);
61
62      nh.param<bool>("enable_ground_height_scale",
    ORB_SLAM2::enable_ground_height_scale, false);
63      nh.param<bool>("use_dynamic_klt_features",
    ORB_SLAM2::use_dynamic_klt_features, false);
64
65      nh.param<bool>("bundle_object_opti", ORB_SLAM2::bundle_object_opti,
    false);
66      nh.param<double>("camera_object_BA_weight",
    ORB_SLAM2::camera_object_BA_weight, 1.0);
67      nh.param<double>("object_velocity_BA_weight",
    ORB_SLAM2::object_velocity_BA_weight, 1.0);
68
```

```
69      nh.param<bool>("draw_map_truth_paths", ORB_SLAM2::draw_map_truth_paths,
    true);
70      nh.param<bool>("draw_nonlocal_mappoint",
    ORB_SLAM2::draw_nonlocal_mappoint, true);
71
72      // temp debug
73      nh.param<bool>("ba_dyna_pt_obj_cam", ORB_SLAM2::ba_dyna_pt_obj_cam,
    false);
74      nh.param<bool>("ba_dyna_obj_velo", ORB_SLAM2::ba_dyna_obj_velo, true);
75      nh.param<bool>("ba_dyna_obj_cam", ORB_SLAM2::ba_dyna_obj_cam, true);
76
77      std::string scene_name;
78      ros::param::get("/scene_name", scene_name);
79      ros::param::get("/base_data_folder", ORB_SLAM2::base_data_folder);
80
81      if (scene_name.compare(std::string("kitti")) == 0)
82          ORB_SLAM2::scene_unique_id = ORB_SLAM2::kitti;
83
84      cout << "Base_data_folder:  " << ORB_SLAM2::base_data_folder << endl;
85
86      std::string packagePath = ros::package::getPath("orb_object_slam");
87
88      if (!enable_loop_closing)
89          ROS_WARN_STREAM("Turn off global loop closing!!");
90      else
91          ROS_WARN_STREAM("Turn on global loop closing!!");
92      // Create SLAM system. It initializes all system threads and gets ready to
    process frames.
93
94      ORB_SLAM2::System SLAM(argv[1], argv[2], ORB_SLAM2::System::MONOCULAR,
    enable_loop_closing);
95      //转到orb_slam中的主接口system.cc
96
97      ImageGrabber igb(&SLAM);
98
99      ros::Subscriber sub = nh.subscribe("/camera/image_raw", 10,
    &ImageGrabber::GrabImage, &igb);
100     //注意订阅的节点名
101
102     ros::spin(); //block here till I ctrl-C
103
104     // Stop all threads
105     SLAM.Shutdown();
106
107     // Save camera trajectory
108     //
    SLAM.SaveKeyFrameTrajectoryTUM(packagePath+"/Outputs/KeyFrameTrajectory.txt");
109
```

```
110     // Save camera trajectory
111     SLAM.SaveTrajectoryTUM(packagePath + "/Outputs/AllFrameTrajectory.txt");
112     if (ORB_SLAM2::scene_unique_id == ORB_SLAM2::kitti)
113         SLAM.SaveTrajectoryKITTI(packagePath +
    "/Outputs/AllFrameTrajectoryKITTI.txt");
114
115     ca::Profiler::print_aggregated(std::cout);
116
117     ros::shutdown();
118
119     return 0;
120 }
121
122 void ImageGrabber::GrabImage(const sensor_msgs::ImageConstPtr &msg)
123 {
124     // Copy the ros image message to cv::Mat.
125     cv_bridge::CvImageConstPtr cv_ptr;
126     try
127     {
128         cv_ptr = cv_bridge::toCvShare(msg);
129     }
130     catch (cv_bridge::Exception &e)
131     {
132         ROS_ERROR("cv_bridge exception: %s", e.what());
133         return;
134     }
135
136     mpSLAM->TrackMonocular(cv_ptr->image, cv_ptr->header.stamp.toSec(), msg-
    >header.seq);
137 }
```

ros_mono.cc中主要实现三个功能，一是初始化ros节点并将ros_mono中的预设参数传递到orb_slam中，二是将ros_mono的路径参数(Voc,setting等)转到orb_object_slam(orb_slam)的主接口system.cc中，三是定义了ImageGrabber这个类，数据成员为system指针，具有构造函数和一个void函数GrabImage，GrabImage实现了将ros中的messages转换为可供opencv处理的图片格式(cv_bridge::toCvShare)，同时将转换后的图片传递到成员下的TrackMonocular,而TrackMonocular是system下的数据入口(在orb_slam中是mono的数据入口，cubeslam只有mono)

ros_mono中订阅的话题为/camera/image_raw，对于rosbag中不同名的话题需要remap，参考上面roslaunch文件的解读

system.cc

cubeslam与orb_slam系统的入口都在system.cc，文件中主要包括四个函数:

1. System
2. TrackStereo

3. TrackRGBD
4. TrackMonocular

System函数是SLAM系统的构造函数，包括所有功能模块和所有线程的初始化。cubeslam中只使用到了TrackMonocular，即单目数据入口

简要介绍一下System构造函数的内容，主要如下

PHP

```
System::System(const string &strVocFile, const string &strSettingsFile, const eSensor sensor, bool use_loop_closing) : mSensor(sensor), use_loop_close(use_loop_closing), mbReset(false), mbActivateLocalizationMode(false), mbDeactivateLocalizationMode(false)
//传递Voc文件路径，Settings设置路径，传感器类型(mono)，是否回环检测(false,作者的demo中不考虑回环检测)，ros_mono中转入System ORB_SLAM2::System SLAM(argv[1], argv[2], ORB_SLAM2::System::MONOCULAR, enable_loop_closing);

mpVocabulary = new ORBVocabulary();
//加载orb词袋

mpKeyFrameDatabase = new KeyFrameDatabase(*mpVocabulary);
//创建KeyFrameDatabase,主要存放词袋模型里的值，用于闭环检测与重定位

mpFrameDrawer = new FrameDrawer(mpMap);
mpMapDrawer = new MapDrawer(mpMap, strSettingsFile);
//画Frame与画Map

mpTracker = new Tracking(this, mpVocabulary, mpFrameDrawer, mpMapDrawer,mpMap, mpKeyFrameDatabase, strSettingsFile, mSensor);
//跟踪线程

mpLocalMapper = new LocalMapping(mpMap, mSensor == MONOCULAR);
if (parallel_mapping)
    mptLocalMapping = new thread(&ORB_SLAM2::LocalMapping::Run, mpLocalMapper);
mpLoopCloser = new LoopClosing(mpMap, mpKeyFrameDatabase, mpVocabulary, mSensor != MONOCULAR);
if (use_loop_close)
    mptLoopClosing = new thread(&ORB_SLAM2::LoopClosing::Run, mpLoopCloser);
//初始化并创建局部建图与闭环检测线程，由参数控制

mpViewer = new Viewer(this, mpFrameDrawer, mpMapDrawer, mpTracker, strSettingsFile);
if (enable_viewer)
    mptViewer = new thread(&Viewer::Run, mpViewer);

mpTracker->SetViewer(mpViewer);
```

```
//创建窗口显示，并为跟踪分配窗口

mpTracker->SetLocalMapper(mpLocalMapper);
mpTracker->SetLoopClosing(mpLoopCloser);

mpLocalMapper->SetTracker(mpTracker);
mpLocalMapper->SetLoopCloser(mpLoopCloser);

mpLoopCloser->SetTracker(mpTracker);
mpLoopCloser->SetLocalMapper(mpLocalMapper);
//各个线程之间建立交互的关联
```

cubeslam在System.cc中除了构造SLAM系统，还提供了单目入口TrackMonocular，

PHP

```
cv::Mat System::TrackMonocular(const cv::Mat &im, const double &timestamp, int msg_seq_id)
//功能为进入入口后的确认与启动工作

//1)检测Localization模式(纯定位)，开启时停止局部建图，传递参数，关闭时只传递参数
mpLocalMapper->RequestStop();mpTracker->InformOnlyTracking(true);//开启
mpTracker->InformOnlyTracking(false);mpLocalMapper->Release();//关闭

//2)检测mbReset，即是否重启，需要重启的话进入track下的重启
mpTracker->Reset();//mbReset=true

//3)启动单目的跟踪，参数在ros_mono中传递
return mpTracker->GrabImageMonocular(im, timestamp, msg_seq_id);
```