😜 Cube SLAM 代码总结：如何从 2D 目标检测恢复 3D 物体位姿

2019-02-22 · SLAM · code · cube slam · object slam · 约 7001 字 · 预计阅读 14 分钟

- 注：🌐 **Cube SLAM 系列论文，代码注释、总结汇总**
- 原始代码：https://github.com/shichaoy/cube_slam
- 个人注释：https://github.com/wuxiaolang/Cube_SLAM_wu

0. 函数

0.1. 函数调用

- `main_obj.cpp` 文件中 `detect_cuboid_obj.detect_cuboid();` **开始进行物体立方体结构检测**；
  - `detect_cuboid_obj` 是立方体检测类（定义在 `detect_3d_cuboid.h` 中） `detect_3d_cuboid` 的一个对象；

Code

```
detect_cuboid_obj.detect_cuboid(raw_rgb_img,transToWolrd,raw_2d_objs,all_lines_raw, frames_cuboids);
```

0.2. 函数定义

- `detect_cuboid()` 函数定义在 `box_proposal_detail.cpp` 文件中。
- **输入**：原始图像 `rgb_img` ； 相机位姿 `transToWolrd` ； 2D 检测框信息 `obj_bbox_coors` ； 线检测的边缘线信息 `all_lines_raw` ；
- **输出**：立方体提案： `std::vector<ObjectSet>& all_object_cuboids` （提案个数等于图像帧数）

---

1. 线段信息排序

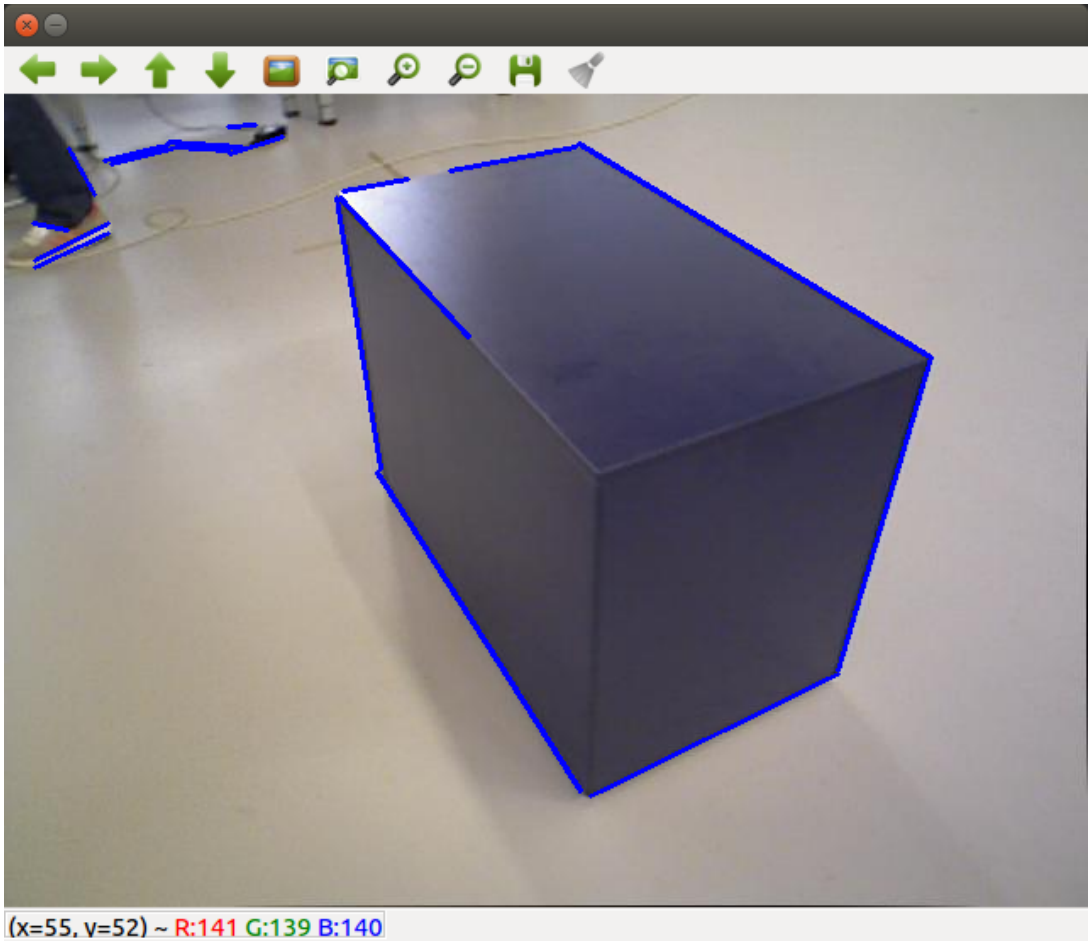
- **首先需要保证作为输入的线段**信息矩阵 `MatrixXd all_lines_raw` 中存储的所有线段的**两个端点是从左到右排序**的（ x 坐标）；
- `align_left_right_edges()` 函数在 `object_3d_util.cpp` 文件中；
- 比如，排序前后：

Code

```
# all_lines_raw 排序前:
     x_1         y_1         x_2         y_2
    33.6847     18.1416     25.6146    0.173099
    36.4537  -0.0483192     38.5934     20.0433
     65.885   0.0112345     68.7188     29.0275
    103.954     37.9624     102.454  -0.0179376
    78.8074     48.5287     102.018     39.0428
    532.903     149.891     363.809     12.2352
    475.029     337.319     532.423     152.131
    314.382     391.152     473.163     338.492
    229.763     193.524     312.705     389.125
    210.468     25.1893     231.075     191.991
    307.014     188.992     214.057     32.9661
```

```
# all_lines_raw 排序后:
   25.6146   0.173099    33.6847    18.1416
   36.4537 -0.0483192    38.5934    20.0433
    65.885  0.0112345    68.7188    29.0275
   102.454 -0.0179376    103.954    37.9624
   78.8074    48.5287    102.018    39.0428
   363.809    12.2352    532.903    149.891
   475.029    337.319    532.423    152.131
   314.382    391.152    473.163    338.492
   229.763    193.524    312.705    389.125
   210.468    25.1893    231.075    191.991
   214.057    32.9661    307.014    188.992
```

- 在原图上绘制检测到的线段： `plot_image_with_edges()` 函数



---

2. ground-wall 边界线

- 用 `(0,0,1,0)` 表示地平面 `ground_plane_world`
  - 平面的表示：一个**平面可以用一个齐次向量表示**：$\pi = (\pi_1, \pi_2, \pi_3, \pi_4)^T = (n^T, d)^T$ ，其中 n 是平面的法向量，d 是它到原点的距离；

Code

```
Vector4d ground_plane_world(0,0,1,0);
```

- 计算传感器（相机）的平面 `ground_plane_sensor`

Code

```
Vector4d ground_plane_sensor = cam_pose.transToWolrd.transpose() * ground_plane_world;
```

- **相机系中的平面**与世界系平面的转换关系为：

$$\pi_w = \left(T_{w,c}^{-}\right)^T \cdot \pi_c$$

---

3. 2D 检测框高度采样

3.1 YOLO 2D 检测框原始信息

- 左上角坐标： `(left_x_raw, top_y_raw)`
- 宽： `obj_width_raw`
- 高： `obj_height_raw`
- 右下角坐标： `(right_x_raw, down_y_raw)`

Code

```
obj_bbox_coors:
# x1    y1      宽      高
175     24      385     373     0.42
201     40      341     386     0.32
208     54      351     371     0.54
```

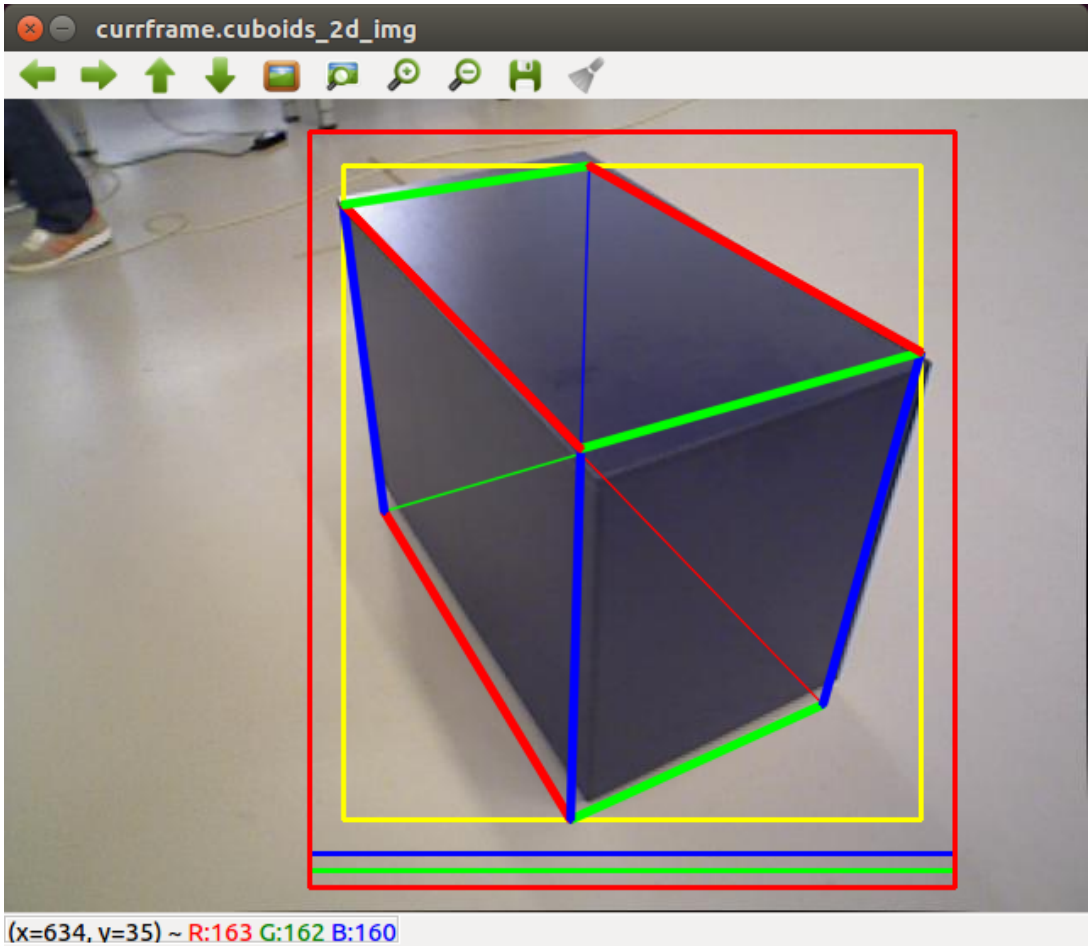
3.2 扩大边界框大小

- 2D 目标检测的检测框可能不准确，是否对检测框的高度进行采样： `std::vector<int> down_expand_sample_all;`
  - 不采样的话： `down_expand_sample_all = 0` ；
  - 如果采样高度： `down_expand_sample_all = 0, 10, 20;`
  - 开关 `whether_sample_bbox_height` 在 `detect_3d_cuboid.h` 和 `main_obj.cpp` 中；
- 事实上，开启了高度采样之后好像效果更差了？
- `down_expand_sample_all` 的 size 是需要计算的次数，不采样时计算一次，采样 10 像素再计算一次，采样 20 像素再计算一次。

3.3 边界距离宽度

- 如下图所示，黄色的框是 YOLO 原始的 2D 检测框，其他颜色的框是拓宽 20 像素的边界框；

- 蓝色、绿色和红色的检测框分别是采样了高度之后，也就是高度采样只往 y 下方进行采样了。



---

4. 物体偏航角 yaw 采样

- 物体的偏航角直接**初始化**为面向相机，与光轴对齐

Code

```
1  double yaw_init = cam_pose.camera_yaw - 90.0 / 180.0 * M_PI;
```

- 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 个偏航角**。

Code

```
1  std::vector<double> obj_yaw_samples;
2  // BRIEF linespace()函数从 a 到 b 以步长 c 产生采样的 d.
3  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_sa
```

---

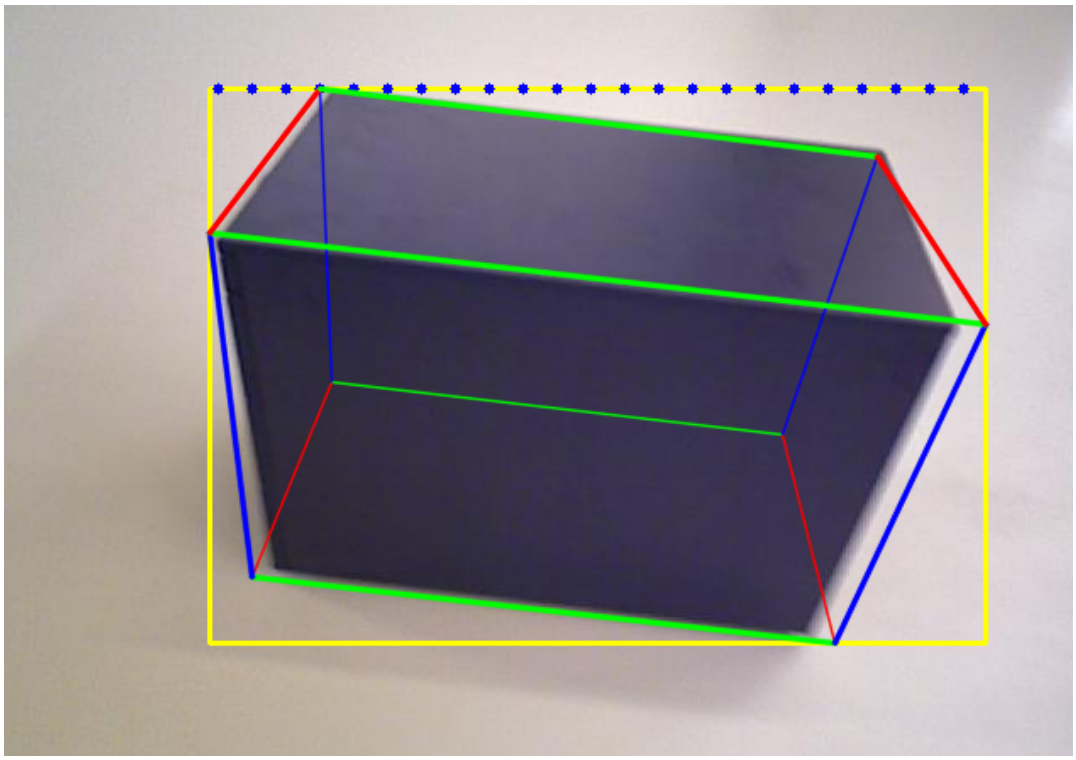
5. 上边缘点采样

- 为确定物体的“顶点”需要从原始边界框的最左边 `left_x_raw+5` 到最右边 `right_x_raw-5` 每隔 `top_sample_resolution` （20像素）的距离采样一个点 `top_x_samples[i]` .

Code

```
1  int top_sample_resolution = round(min(20,obj_width_raw/10 )); //  20 pixels
2  std::vector<int> top_x_samples;
3  linespace<int>(left_x_raw+5, right_x_raw-5, top_sample_resolution, top_x_samples);
```

- 为确定物体的顶部，**提供至少 10 个采样点（后期通过最小误差得到最合适的点）**，对于越小的物体需要越精细的采样。上边缘采样点如下图所示：



- **疑问：为什么在原始检测框的顶部采样？不在拓宽边界之后的顶部采样呢？效果是否会更好。**

---

6. 线段的处理

6.1 保留在扩大后边界内的线段

- 线检测得到的所有线段： `all_lines_raw`，筛选之后在范围内的线段： `all_lines_inside_object`；
- `check_inside_box()` 函数检测线段是否在矩形框内，位于 `object_3d_util.cpp` 文件中；
- 要求线段的两个端点需要同时都在框内（是否可改进呢）

Code

```
for (int edge_id = 0; edge_id < all_lines_raw.rows(); edge_id++)
// 判断 all_lines_raw 矩阵中第 edge_id 线段的一个端点.head<2> 是否在区域中.
if (check_inside_box(all_lines_raw.row(edge_id).head<2>(), expan_distmap_lefttop, expan_distmap_rig
// 判断 all_lines_raw 矩阵中第 edge_id 线段的另一个端点.tail<2> 是否在区域中.
if (check_inside_box(all_lines_raw.row(edge_id).tail<2>(),expan_distmap_lefttop, expan_distmap_righ
{
        // 存储所有在扩大后的边界框内的线段.
        all_lines_inside_object.row(inside_obj_edge_num) = all_lines_raw.row(edge_id);
        inside_obj_edge_num++;
}
```

6.2 线段合并与剔除

- 短边合并与剔除在 `merge_break_lines()` 函数中实现，位于 `object_3d_util.cpp` 文件中；

Code

```
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*/
```

```
4                           double pre_merge_angle_thre_degree, /*角度阈值 5°*/
5                           double edge_length_threshold)       /*长度阈值 30*/
```

6.2.1 线段合并

- **首先要求两条线段的角度误差小于 5 °**；
  - `atan2_vector()` 函数根据线段的水平和竖直投影**计算线段的角度** [−90, 90] 范围内；

Code

```
// BRIEF 根据线段的 水平和竖直长度x_vec y_vec 计算出角度 all_angles
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
}
```

  - **计算两条线段的角度差**，并保证角度差小于阈值 `pre_merge_angle_thre` （5 °）

Code

```
// 相邻两条选段的角度差 angle_diff.
double diff = std::abs(all_angles(seg1) - all_angles(seg2));
double angle_diff = std::min(diff, M_PI - diff);
```

- **然后要求两条线段的距离差小于 20 像素**；
  - 计算两条线段的距离：

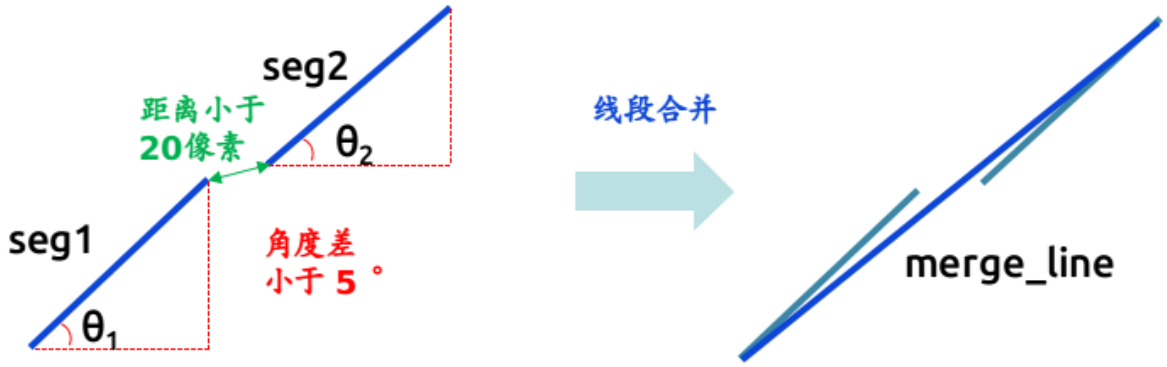
Code

```
// 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；
double dist_1ed_to_2 = (merge_lines_out.row(seg1).tail(2) - merge_lines_out.row(seg2).head(2)).
double dist_2ed_to_1 = (merge_lines_out.row(seg2).tail(2) - merge_lines_out.row(seg1).head(2)).
```

  - 要求“线1尾到线2头的距离”**或**“线2尾到线1头的距离”小于阈值 `pre_merge_dist_thre` （20像素）
- 同时满足以上两个条件的**两条线段进行合并**

Code

```
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);
// 融合之后的新的线段的角度 merged_angle.
double merged_angle = std::atan2(merge_end(1)-merge_start(1),merge_end(0)-merge_start(0));
```



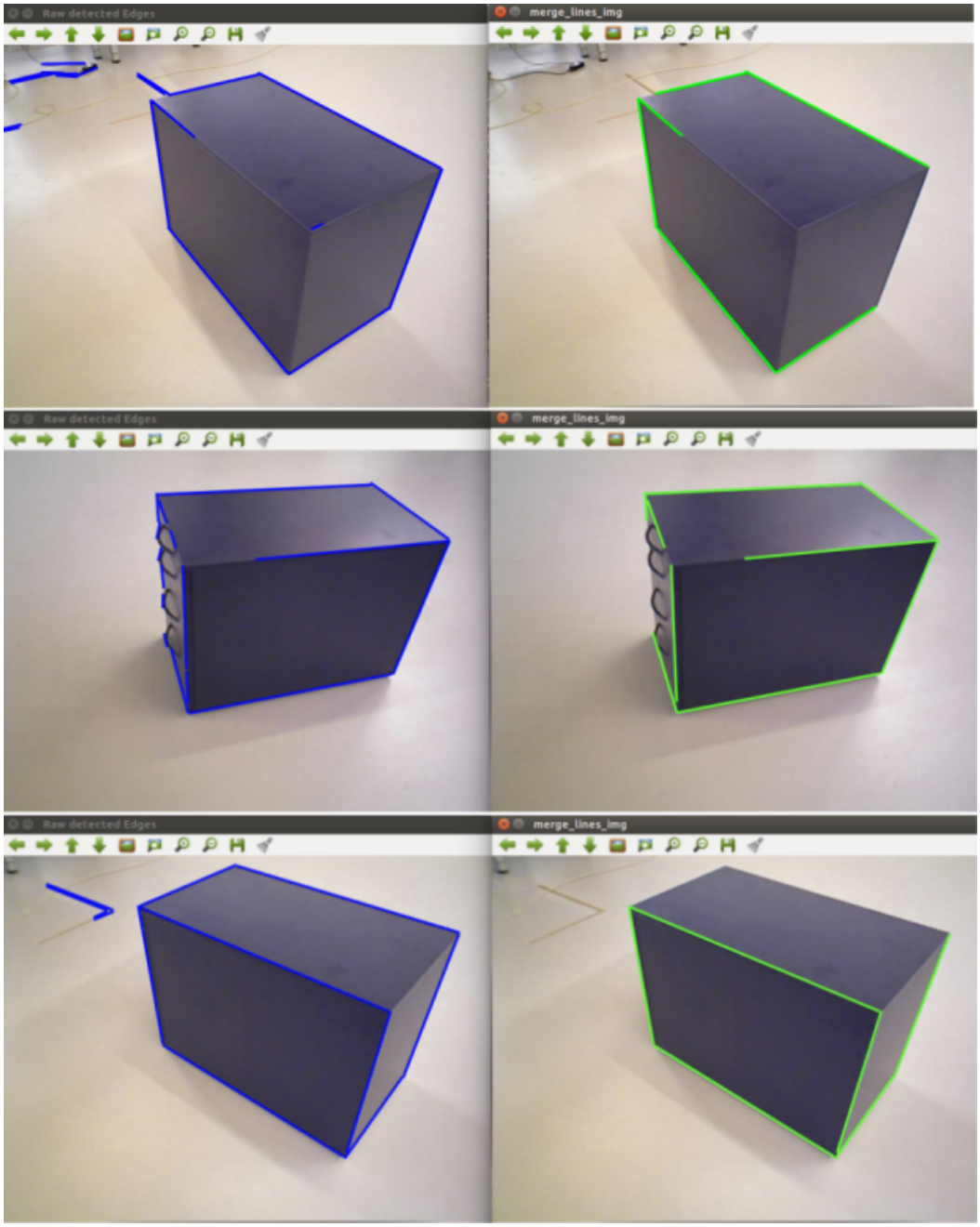
6.2.2 线段剔除

- 计算每条线段的长度，并要求大于长度阈值 `edge_length_threshold` （30 像素）

Code

```
// 重新计算合并之后的线段向量：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.
MatrixXd line_vectors = merge_lines_out.topRightCorner(total_line_number,2) - merge_lines_out.topLef
// @PARAM line_lengths 存储每条线段的长度
VectorXd line_lengths = line_vectors.rowwise().norm();
```

- 最终经过线段合并与筛选之后的效果



---

7. 线段、边缘检测、距离变换

7.1 计算筛选之后的线段角度和中点

Code

```
// 计算每条边缘线段的角度和中点.
// @PARAM lines_inobj_angles    线段角度.
// @PARAM edge_mid_pts                  线段的中点.
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), a
        edge_mid_pts.row(i).head<2>() = (all_lines_merge_inobj.row(i).head<2>()+all_lines_merge_inc
}
```

7.2 Canny 边缘检测

Code

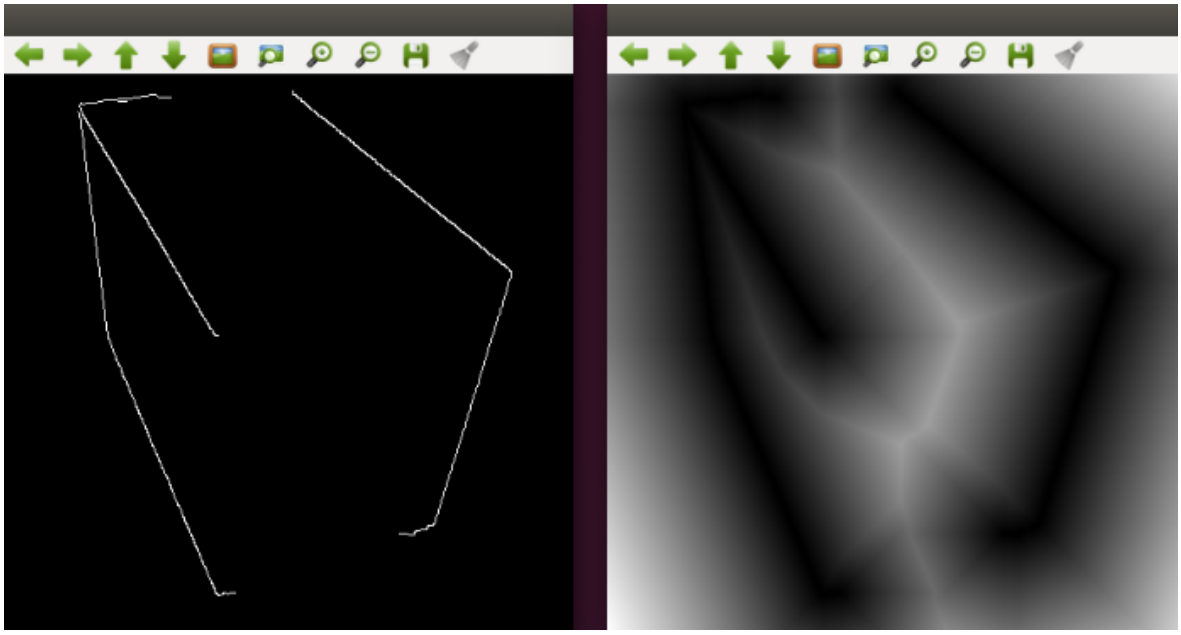
```
cv::Mat im_canny;
// @PARAM               object_bbox             扩大后的检测框.
cv::Rect object_bbox = cv::Rect(left_x_expan_distmap, top_y_expan_distmap, width_expan_distmap, heig
cv::Canny(gray_img(object_bbox), im_canny, 80, 200); // low thre, high thre    im_canny 0 or 255   [
```

7.3 构建距离图

- 2D 长方体边缘应与实际图像的边缘匹配。利用 Canny 边缘检测方法构建距离图，然后再长方体边缘倒角距离（Chamfer distance）进行累加求和，再通过 2D 框的大小进行归一化。

Code

```
cv::Mat dist_map;
cv::distanceTransform(255 - im_canny, dist_map, CV_DIST_L2, 3); // dist_map is float datatype
```



---

8. 立方体提案生成

8.1 采样相机的翻滚角 roll 和俯仰角 pitch

- 以相机偏角为中心的**正负 6 ° 范围内，每隔 3 ° 采样一个值**（如果不采样的话就直接使用相机的 roll 和 pitch 角），也就是分别采样了 5 个角度；

```
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
        linespace<double>(cam_pose_raw.euler_angle(1)-6.0/180.0*M_PI, cam_pose_raw.euler_angle(1)+6
}
else
{
        cam_roll_samples.push_back(cam_pose_raw.euler_angle(0));
```

```
11          cam_pitch_samples.push_back(cam_pose_raw.euler_angle(1));
12  }
```

- 同时，保存**新的相机位姿**，计算新的**相机平面**

Code

```
1  // 将当前角度值转换成旋转矩阵.
2  // NOTE 这里yaw为什么不用采样的值而用相机的值？：yaw 采样是真对物体的，并没有对相机的 yaw 值进行采样。
3  transToWolrd_new.topLeftCorner<3,3>() = euler_zyx_to_rot<double>(cam_roll_samples[cam_roll_id], cam_
4  set_cam_pose(transToWolrd_new);
5  // TODO 平面？？
6  ground_plane_sensor = cam_pose.transToWolrd.transpose() * ground_plane_world;
```

8.2 计算三个消失点

- 消失点计算函数为 `getVanishingPoints()`

Code

```
1  void getVanishingPoints(const Matrix3d& KinvR,  /* Kalib*invR */
2                          double yaw_esti,        /* 采样的物体偏航角 */
3                          Vector2d& vp_1,         /* 输出的消失点 */
4                          Vector2d& vp_2,
5                          Vector2d& vp_3)
```

- 消失点计算方法：

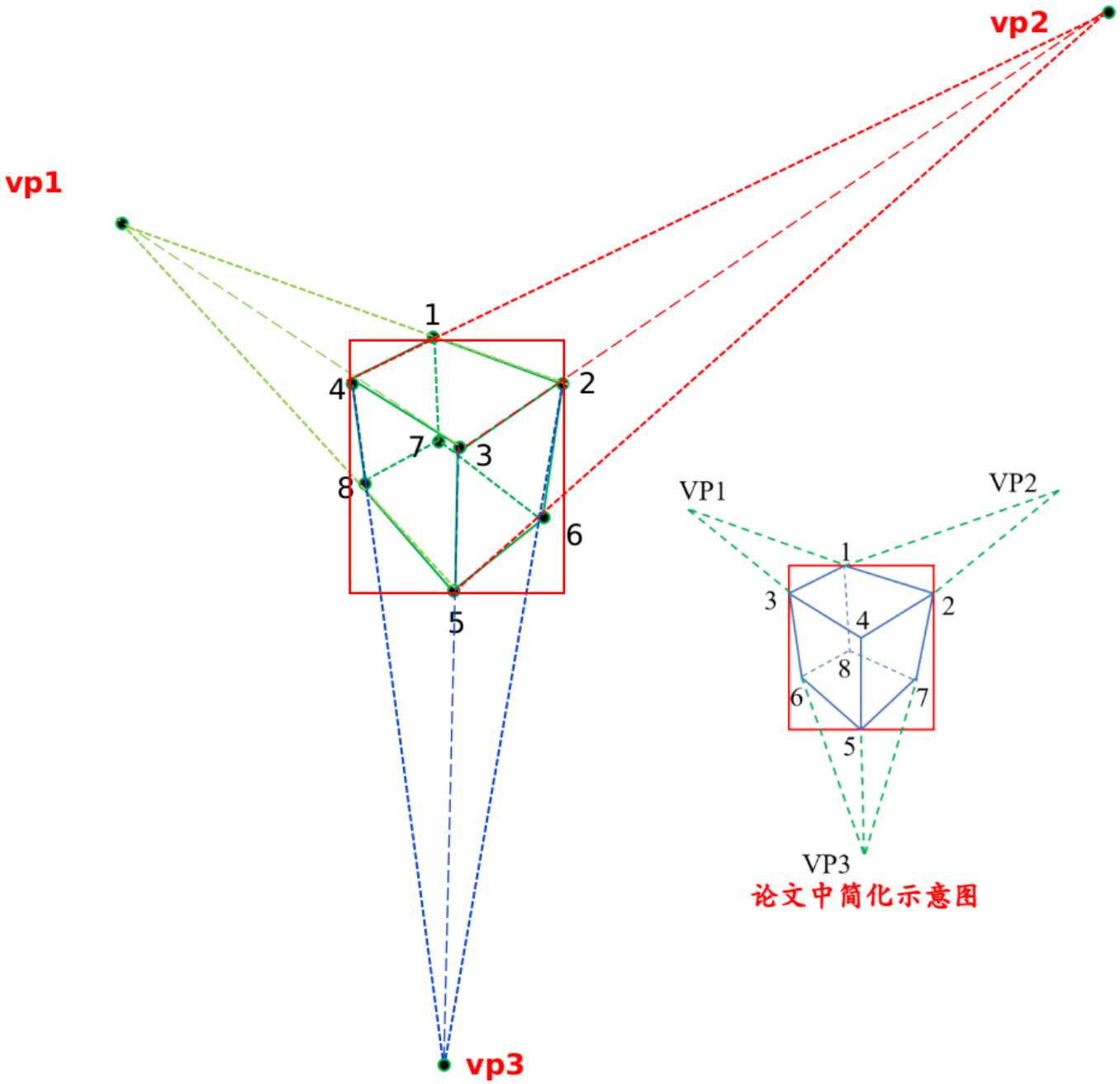
$$vp_x = K \cdot R^{-1} \cdot (\cos(yaw), \sin(yaw), 0)^T$$

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

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

Code

```
1  vp_1 = homo_to_real_coord_vec<double>( KinvR * Vector3d(cos(yaw_esti), sin(yaw_esti), 0) );     // f
2  vp_2 = homo_to_real_coord_vec<double>( KinvR * Vector3d(-sin(yaw_esti), cos(yaw_esti), 0) );    // f
3  vp_3 = homo_to_real_coord_vec<double>( KinvR * Vector3d(0,0,1) );                               // f
```



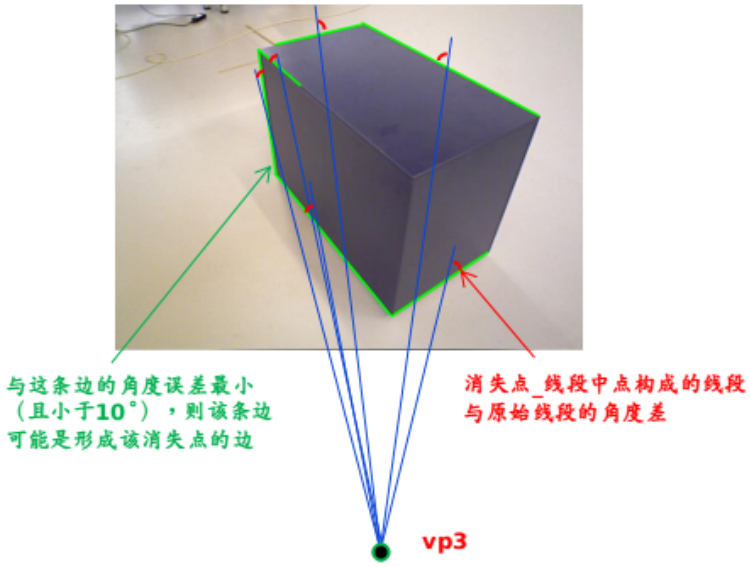
8.3 寻找形成消失点的边

- 在函数 `VP_support_edge_infos()` 中实现
  - 输入：消失点的坐标； 每条线段的中点； 每条线段的角度； 角度偏差阈值；
  - 消失点 1, 2 的角度偏差阈值为 15 °，消失点 3 的角度偏差阈值为 10 °；

Code

```
Eigen::MatrixXd VP_support_edge_infos(  Eigen::MatrixXd& VPs,                   /* 消失点矩阵 3*2 */
                                        Eigen::MatrixXd& edge_mid_pts,          /* 每条线段的中点 n×2
                                        Eigen::VectorXd& edge_angles,           /* 每条线段的偏角 n×1
                                        Eigen::Vector2d vp_support_angle_thres) /* 消失点与边的夹角阈值
```

- 思想：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。如下图所示：

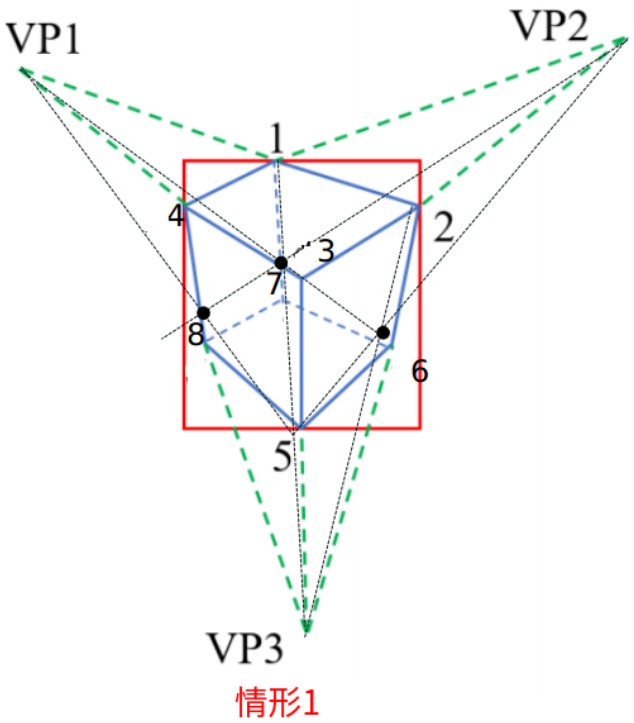


- 角度平滑 `smooth_jump_angles()`
- “支撑线”选择，如果检测到多条满足角度要求的边，则选择角度最大和最小的，作为形成消失点的两条“支撑线”，存储在 `MatrixXd all_vp_bound_edge_angles` 矩阵中。

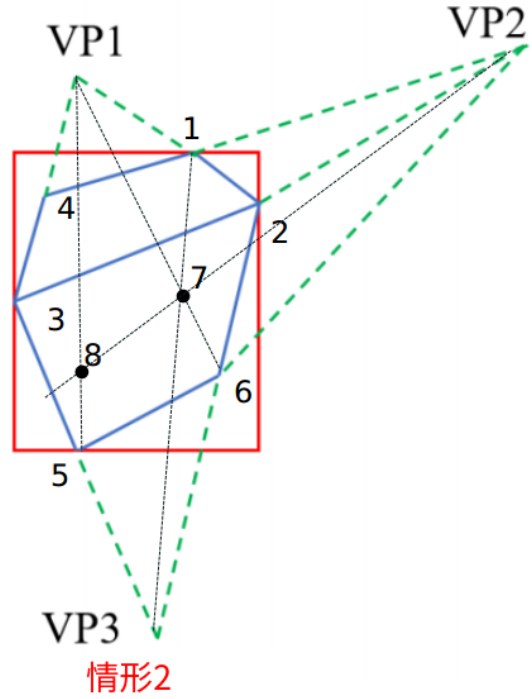
Code

```
// NOTE 如果有多条边满足要求（可能检测到多条支撑线），比较得到角度最大和最小的，作为两条支撑线.
// 角度最大和最小的边的id
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);
// TODO 第 2 3 个消失点时交换最大和最小值
if (vp_id > 0)
    std::swap(vp1_low_edge_id, vp1_top_edge_id);  // match matlab code
// NOTE 输出：消失点两边的夹角.
all_vp_bound_edge_angles(vp_id,0) = edge_angles(vp_inlier_edge_id[vp1_low_edge_id]);   // it will b
all_vp_bound_edge_angles(vp_id,1) = edge_angles(vp_inlier_edge_id[vp1_top_edge_id]);
```

8.4 计算物体 8 个点的 2D坐标



情形1



情形2

8.4.1 顶部点 1

- **顶部第一个点**由上边缘采样的点确定，**可能在检测框顶部的某一个位置**，可以参考上面顶边采样的图。

8.4.2 顶部点 2

- **检查消失点 1 在左边还是在右边**
  - `seg_hit_boundary()` 函数两个点构成的射线是否与一条边有交点，没有交集则返回 [–1 –1]

Code

```
Vector2d seg_hit_boundary(  const Vector2d& pt_start,
                            const Vector2d& pt_end,
                            const Vector4d& line_segment2 )
```

  - 如果消失点 1 与上边缘采样点的射线与边界框的右边界有交点，则位于左边；
  - 如果消失点 1 与上边缘采样点的射线与边界框的左边界有交点，则位于右边；
  - 参数 `vp_1_position` 表示消失点的左右位置：

Code

```
int vp_1_position = 0;  // 0 initial as fail,  1  on left   2 on right
```

- 顶部第二个点：即上一步射线与左右边界的交点

8.4.3 第一种情形下的顶部第 3, 4 个点

- 第一种情形：可以观察到物体的三个面；
- **顶部点 4**：消失点 2 与上边缘采样点的射线与检测框左边（或右边）界的交点；
  - 如果没有产生交点，则中断本次检测，说明采样的上边缘点不准确。
  - 点 1 与点 4 的距离小于阈值 `shorted_edge_thre` （默认为 20 ），也失败；
- **顶部点 3**：消失点 1 和点 3 构成的射线与消失点 2 和点 2 构成的射线的交点；

- 函数 `lineSegmentIntersect()` 计算**两条射线的交点**

```
Vector2d lineSegmentIntersect(  const Vector2d& pt1_start, const Vector2d& pt1_end,     /* 线段
                                const Vector2d& pt2_start, const Vector2d& pt2_end,     /* 线段
                                bool infinite_line)
```

  - 如果交点不在检测框内，则失败；
  - 如果点 3–4 或点 3–2 的长度小于阈值，则失败；

8.4.4 第二种情形下的顶部第 3, 4 个点

- 第二种情形：可以观察到物体的两个面；
- **顶部点 3**：消失点 2 与顶部点 2 的连线与检测框左边界（或右）的交点；
  - 若没有交点，则失败；
  - 若点 2–3 的距离小于阈值，则失败；
- **顶部点 4**：消失点 1 和顶部点 3 构成的射线与消失点 2 和顶部点 1 构成的射线的交点；
  - 若交点不在检测框内，则失败；
  - 如果点 3–4 或点 4–1 的长度小于阈值，则失败；

8.4.5 底部点 5

- 底部点情形 1， 2 的计算方法相同；
- **底部点 5**：消失点 3 和顶部点 3 的连线与检测框底边（是否进行了高度采样）的交点；
  - 如果没有交点，则失败；
  - 如果点 3–5 的距离小于阈值，则失败。

8.4.6 底部点 6

- **底部点 6**：消失点 2 和底部点 5 的连线与消失点 3 与顶部点 2 连线的交点；
  - 若交点不在检测框内，则失败；
  - 若点 6–2 或点 6–5 的距离小于阈值，则失败；

8.4.7 底部点 7

- **底部点 7**：消失点 1 和底部点 6 的连线与消失点 3 与顶部点 1 连线的交点；
  - 若交点不在检测框内，则失败；
  - 若点 7–1 或点 7–6 的距离小于阈值，则失败；

8.4.8 底部点 8

- **底部点 8**：消失点 1 和底部点 5 的连线与消失点 2 与顶部点 7 连线的交点；
  - 若交点不在检测框内，则失败；
  - 若点 8–4 或点 8–5 的距离小于阈值，则失败；
- 物体 **8 个点的 2D坐标**存储在 `box_corners_2d_float` 矩阵中

```
MatrixXd box_corners_2d_float(2,8);
box_corners_2d_float << corner_1_top, corner_2_top, corner_3_top, corner_4_top,
```

```
3                           corner_5_down, corner_6_down, corner_7_down, corner_8_down;
```

8.5 误差计算（一）

8.5.0 误差存储

- 前四项

Code

```
1  Vector4d(   config_id,          /* 模式情形 */
2              vp_1_position,      /* vp1的位置 */
3              obj_yaw_esti,       /* 采样的物体偏航角 */
4              sample_top_pt_id);  /* 上边缘采样点 */
```

- 往后三项

Code

```
1  Vector3d(   sum_dist/obj_diaglength_expan,      /* 平均距离误差?  */
2              total_angle_diff,                   /* 角度误差 */
3              down_expand_sample);                /* 高度采样，是否扩充底部 0,  10,  20 */
```

- 后两项，采样的相机 roll pitch 角（不采样则直接取相机的角度）

Code

```
1  Vector2d(   cam_roll_samples[cam_roll_id],      /* 采样相机 roll 角 */
2              cam_pitch_samples[cam_pitch_id]);   /* 采样相机 pitch 角 */
```

8.5.1 距离误差

- 距离误差计算函数为： `box_edge_sum_dists()`

Code

```
1  double box_edge_sum_dists(  const cv::Mat& dist_map,            /* 距离变换图 */
2                              const MatrixXd& box_corners_2d,     /* 8 个顶点的 2D坐标(以检测框为轴) */
3                              const MatrixXi& edge_pt_ids,        /* 可见的边 */
4                              bool  reweight_edge_distance)
```

- 将 8 个点在图像中的坐标转换成在**检测框中的坐标（以检测框左上角为坐标原点）**

Code

```
1  MatrixXd box_corners_2d_float_shift(2,8);
2  box_corners_2d_float_shift.row(0) = box_corners_2d_float.row(0).array() - left_x_expan_distmap;
3  box_corners_2d_float_shift.row(1) = box_corners_2d_float.row(1).array() - top_y_expan_distmap;
```

- **情形 1**:
  - 情形 1 下可观察到 3 个面，共 9 条**可见边**:

    Code

    ```
    1  visible_edge_pt_ids.resize(9,2);
    2  visible_edge_pt_ids << 1,2,  2,3,  3,4,  4,1,  2,6,  3,5,  4,8,  5,8,  5,6;
    ```

    - **形成消失点的线段**

Code

```
// 三个消失点所需要的两条边
vps_box_edge_pt_ids.resize(3,4);
// 1_2 与 8_5 交点得到vp1 4_1 与 5_6 交点得到vp1
vps_box_edge_pt_ids << 1,2,8,5,  4,1,5,6,  4,8,2,6;
```

- **情形 2**：
  - 情形 2 下可观察到 2 个面，共 7 条**可见边**：

Code

```
visible_edge_pt_ids.resize(7,2);
visible_edge_pt_ids<<1,2, 2,3, 3,4, 4,1, 2,6, 3,5, 5,6;
```

  - **形成消失点的线段**

Code

```
// 三个消失点所需要的两条边
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;
```

- **距离误差计算**：采样可见边上的点，计算点在距离变换得到的图中的值，累加得到误差值；
  - 从前面的距离图中可以看出，距离图显示图像中每一个非零点距离离自己最近的零点的距离，**图像上越亮的点，代表了离零点（边缘）的距离越远**；（疑问：canny检测没检测出边，导致距离图不准确，但没检测出的边被LSD边缘检测出了呢？那这条边岂不是误差会特别大？？）
  - 在**可见边**（不是检测到的线段，是绘制的边）上**采样不同的点**：

Code

```
Vector2d sample_pt = sample_ind/10.0 * corner_tmp1 + (1-sample_ind/10.0) * corner_tmp2;
```

  - 计算**每个点的距离值**

Code

```
float dist1 = dist_map.at<float>(int(sample_pt(1)),int(sample_pt(0)));
```

  - 是否**加权**，然后**累加**

Code

```
// 是否重新加权
// TODO 第5,6,7条边的测量更值得信赖？？
if (reweight_edge_distance)
 {
     if ((4<=edge_id) && (edge_id<=5))       // 对第 5,6 条边 × 1.5
          dist1 = dist1 * 3.0 / 2.0;
     if (6==edge_id)                         // 对第 7 条边  × 2
          dist1 = dist1 * 2.0;
 }
```

  - 误差累加，并求**平均**

Code

```
sum_dist = sum_dist + dist1;
mean_dist = sum_dist / obj_diaglength_expan;  //obj_diaglength_expan是边界框对角线长度.
```

8.5.2 角度误差

- 角度误差函数： `box_edge_alignment_angle_error()`

Code

```
double box_edge_alignment_angle_error(  const MatrixXd& all_vp_bound_edge_angles,   /* 消失点与边的两个
                                        const MatrixXi& vps_box_edge_pt_ids,        /* 每个消失点来源的
                                        const MatrixXd& box_corners_2d)             /* 8 个顶点的 2D坐
```

- **角度误差** = | 形成消失点的边（2D 点构成）的角度 – 线检测得到的对应边的角度 |

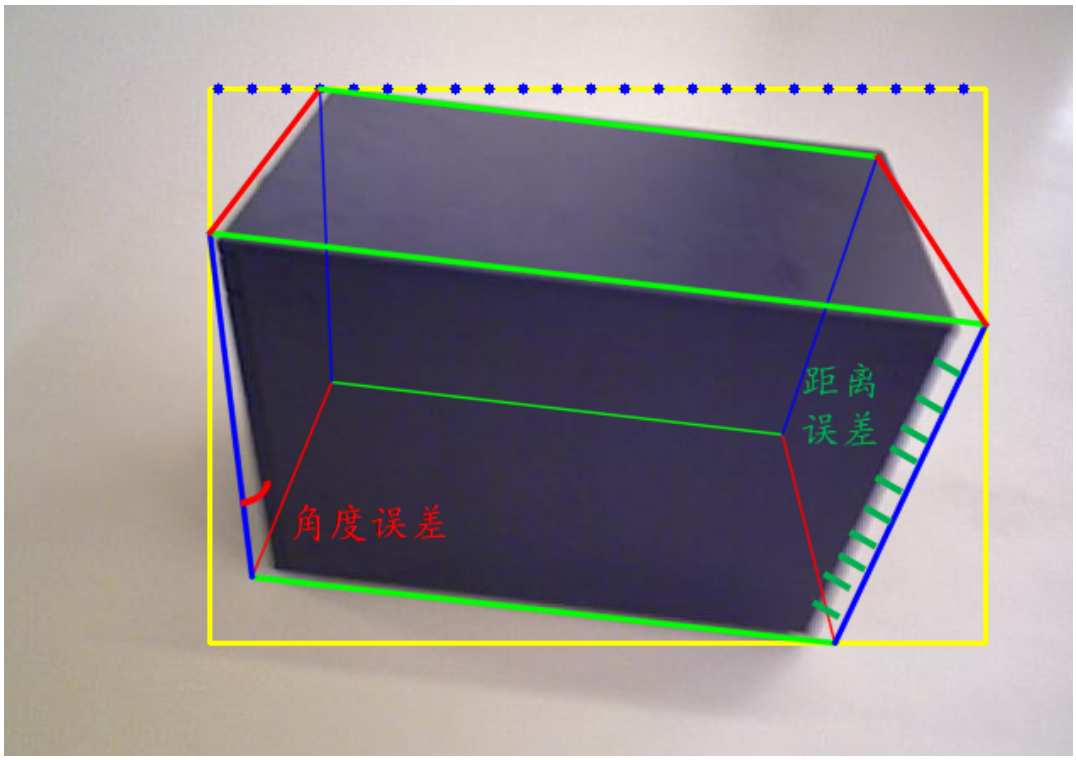
Code

```
for (int i = 0; i < vp_bound_angles_valid.size(); i++)
{
    // NOTE 【计算角度误差】，形成消失点的边的角度-消失点与检测到的边缘的角度.
    double temp = std::abs(box_edge_angle - vp_bound_angles_valid[i]);
    temp = std::min( temp, M_PI-temp );
    if (temp<angle_diff_temp)
        angle_diff_temp = temp;
}
total_angle_diff = total_angle_diff + angle_diff_temp;
```

- 如果没有找到形成消失点的边，则赋予固定的偏差 15 °：

Code

```
double not_found_penalty = 30.0/180.0*M_PI*2;
total_angle_diff = total_angle_diff + not_found_penalty;
```



8.5.3 距离–角度综合评分

- 归一化误差函数： `fuse_normalize_scores_v2()`

Code

```
// BRIEF 加权不同的误差评估 weighted sum different score
void fuse_normalize_scores_v2(  const VectorXd& dist_error,         /* 距离误差 */
                                const VectorXd& angle_error,        /* 角度误差 */
```

```
4                                   VectorXd& combined_scores,          /* 综合得分 */
5                                   std::vector<int>& final_keep_inds,  /* 最终纳入计算的测量的ID */
6                                   double weight_vp_angle,             /* 角度误差的权重 */
7                                   bool whether_normalize)             /* 是否归一化两个误差 */
```

- 分别对距离误差和角度误差以递增的方式取**所有测量次数的 2/3**；
- 再寻找这**两个序列的交集**，即两个误差都在前 2/3 的测量；
- 对交集中每一次测量**归一化两个误差**：

Code

```
// STEP 误差归一化.
if (whether_normalize && (new_data_size > 1))
{
    // 距离误差              (所有的距离 - 最小距离值) / (最大距离 - 最小距离)
    combined_scores  = (dist_kept.array() - min_dist_error) / (max_dist_error - min_dist_error);
    if ((max_angle_error - min_angle_error) > 0)
    {
        // 角度误差          (所有角度误差 - 最小角度误差) / (最大角度误差 - 最小角度误差)
        angle_kept = (angle_kept.array() - min_angle_error) / (max_angle_error - min_angle_error);

        // NOTE 联合评分, (距离误差 + 角度权重x角度误差) / (1 + 角度权重)
                combined_scores = (combined_scores + weight_vp_angle * angle_kept) / (1 + weight_vp
    }
    else
        combined_scores = (combined_scores + weight_vp_angle*angle_kept)/(1+weight_vp_angle);
}
else
        combined_scores = (dist_kept + weight_vp_angle * angle_kept) / (1 + weight_vp_angle);
```

8.6 生成 3D 立方体提案

- 2D 顶点恢复 3D立方体信息： `change_2d_corner_to_3d_object()`

Code

```
// BRIEF    由2D顶点恢复出 3D 立方体信息.
void change_2d_corner_to_3d_object( const MatrixXd& box_corners_2d_float,   /* 8 个点的 2D 坐标*/
                                    const Vector3d& configs,                /* 模式, vp1的位置, 偏航角*
                                    const Vector4d& ground_plane_sensor,    /* 法平面? */
                                    const Matrix4d& transToWolrd,           /* 相机旋转 */
                                    const Matrix3d& invK,                   /* 相机内参的逆矩阵 */
                                    Eigen::Matrix<double, 3, 4>& projectionMatrix,  /* 投影矩阵 */
                                    cuboid& sample_obj)                     /* 3D提案 */
```

8.6.1 计算立方体底部四个 3D点

- `plane_hits_3d()` 函数参数：

Code

```
// @PARAM obj_gnd_pt_world_3d   计算世界坐标系中的 3D 点（立方体底部） .
Matrix3Xd obj_gnd_pt_world_3d;
plane_hits_3d(  transToWolrd,                       /* 相机旋转矩阵 */
                invK,                               /* 相机内参的逆矩阵 */
                ground_plane_sensor,                /* 相机系下的地平面*/
                box_corners_2d_float.rightCols(4), /* 立方体底部的 4 个 2D 点 */
                obj_gnd_pt_world_3d);               /* 立方体底部的 4 个 3D 点 */
```

- 首先将 2 * 4 的 4 个 **2D 坐标**转换成**齐次形式 3 × 4**;

```
齐次像素点:
344.614   528.2  429.72 255.345
    424 281.372 233.359 340.603
      1       1       1       1
```

- 计算像素点对应的**投影射线**（ Z 不确定就是一条射线，确定 Z 了就是一个确定的 3D 点）：

$$l_{ray} = K^{-1}(u, v, 1)^T$$

- 通过相机平面**确定底部 3D 点**
  - 计算相机系下的平面

```
// 相机系下的地平面
ground_plane_sensor = cam_pose.transToWolrd.transpose()*ground_plane_world;
```

  - 计算射线与平面的交点，确定 3D 点： `ray_plane_interact()` 函数

```
ray_plane_interact( pts_ray,        /* 投影射线 */
                    plane_sensor,   /* 相机系下的平面 */
                    pts_3d_sensor); /* 输出的 3D 点 */
```

  - 函数

```
// BRIEF 射线为 3×n ，每列都是从原点开始的一条射线  平面 (4*1) ，计算射线的交点 3×n.
//rays is 3*n, each column is a ray staring from origin  plane is (4, 1)  parameters, compute i
void ray_plane_interact(const MatrixXd &rays,
                        const Eigen::Vector4d &plane,
                        MatrixXd &intersections)
{
    VectorXd frac = -plane[3] / (plane.head(3).transpose() * rays).array();   //n*1
    intersections = frac.transpose().replicate<3,1>().array() * rays.array();
}
```

8.6.2 计算立方体顶部四个 3D点

- `Matrix3Xd obj_top_pt_world_3d` ， 同上调用上面的函数计算；

8.6.3 物体的 9 自由度表示

- x, y 坐标取四个点的平均值； z 坐标取 z 的一半；
- 方向：取最终提案对应的偏航角采样值；
- 尺度：长宽高；
- 模式：几个面，消失点 1 的位置。

8.6.4 计算 8 个点的 3D世界坐标

Code

```
sample_obj.box_corners_3d_world = compute3D_BoxCorner(sample_obj);
```

Code

```
// BRIEF  计算立方体的 3D 坐标.
Matrix3Xd compute3D_BoxCorner(const cuboid& cube_obj)
{
    //@PARAM    corners_body    存储立方体的3D坐标.
    MatrixXd corners_body;
    corners_body.resize(3,8);
    // 八个3D点       1   2   3   4   5   6   7   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;

    // 计算 3D 坐标                                          相似变换
    MatrixXd corners_world = homo_to_real_coord<double>(similarityTransformation(cube_obj) * real_t

    return corners_world;
}
```

8.7 误差计算（二）

8.7.1 歪斜比

- 歪斜比 = 长度 / 宽度；

Code

```
// NOTE 歪斜比：长/宽.
// head(2).maxCoeff()：尺度的前两项xy中较大者：长
// head(2).minCoeff()：尺度的前两项xy中较小者：宽
double skew_ratio = sample_obj->scale.head(2).maxCoeff()/sample_obj->scale.head(2).minCoeff();
```

8.7.2 计算所有误差

- 归一化提案的**总体误差：距离+角度+歪斜比误差**

Code

```
// 计算总体误差
for (int box_id = 0; box_id < raw_obj_proposals.size(); box_id++)
{
        cuboid* sample_obj = raw_obj_proposals[box_id];
        // 计算歪斜比误差 skew_ratio - 1，要求大于0
        double skew_error = weight_skew_error*std::max(sample_obj->skew_ratio - nominal_skew_ratio,

        // TODO 若大于最大歪斜比 3 ，则误差为100
        if (sample_obj->skew_ratio > max_cut_skew)
        skew_error = 100;

        // NOTE 新的误差：距离+角度+歪斜比误差
        double new_combined_error = sample_obj->normalized_error + weight_skew_error*skew_error;
        all_combined_score(box_id) = new_combined_error;
}
```

8.7.3 确定最终提案

- 对误差进行排序，**选择误差最小的一个作为最终提案**；

Code

```
// STEP 保留误差最小的提案.
// 提案的索引（ID）并从 0 开始递增赋值.
std::vector<int> sort_idx_small(all_combined_score.rows());
iota(sort_idx_small.begin(), sort_idx_small.end(), 0);

// 对 all_combined_score 的前 actual_cuboid_num_small（1） 项进行递增.
sort_indexes(   all_combined_score,             /* 检索比较的序列 */
                sort_idx_small,                         /* 按照all_combined_score中误差从小到达排序误差
                actual_cuboid_num_small);

for (int ii = 0; ii < actual_cuboid_num_small; ii++) // use sorted index
{
    all_object_cuboids[object_id].push_back(raw_obj_proposals[sort_idx_small[ii]]);
    //std::cout << "sort_idx_small[ii]: " << sort_idx_small[ii] << std::endl;
}
```

8.8 绘制提案

- `plot_image_with_cuboid()` 函数完成。

2019.02.22
wuyanminmax@gmail.com

‹ 论文阅读 | 使用物体补充的 BA 来恢复单目 SLAM 的稳定尺度
论文阅读 | Quadric SLAM：以目标检测获得的对偶二次曲面为面向物体 SLAM 的路标 ›