3D 激光SLAM ->loam_velodyne论文与代码解析Lidar Odometry and Mapping

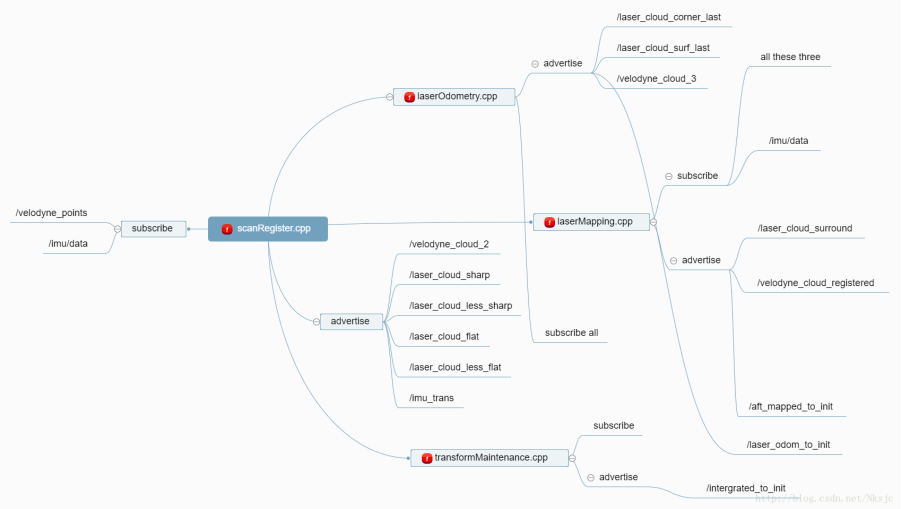
原创 2017年07月30日 18:45:50 标签：开源 / 结构 / 3d / slam的原理 7894

3D SLAM ->loam_velodyne论文与代码解析

一直做2D 的激光SLAM，最近终于接触到3D的了，想彻底的看透一个开源的3D 激光SLAM，选择了Loam_velodyne从论文到代码彻底看一下。论文：LOAM Lidar Odometry and Mapping in Real-time。

结构关系：订阅与发布节点关系

其中最终输出构建3D地图的点云是/laser_cloud_surround;需要更改的是最左侧输入、即订阅的/velodyne_points的topic名称。Remap成为需要订阅的主题。从图中可以看出scanRegister的工作是将源数据处理成/laser_cloud_sharp & /laser_cloud_flat等等，这些点云是具有特征信息的点云。



具体的操作为在论文中给出这样一幅图：

Nksjc
关注

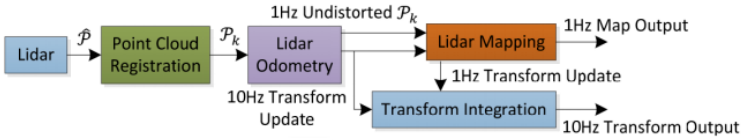


Fig. 3. Block diagram of ... dar odometry and mapping software system.

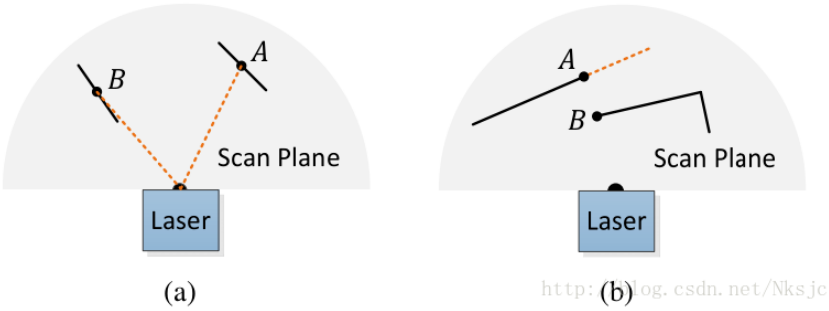
由此可以知道laserOdometry从特征 ... 与计运动（10HZ），然后整合数据发送给laserMapping（1HZ），在这个过程中，算是一种数据的降采样吧。也是为了保证运行online的时效性，以及节省内存占用的大小。

具体部分的实现思想

相对于其它直接匹配两个点云，loar ... 过提取特征点匹配后计算坐标变换。

特征点提取

一次扫描的点通过曲率（c）值来分 ... 特征点是c的最大值点–边缘点；特征点是c的最小值点–平面点。为了使特征点均匀的分布在环境中，将一次 ... 划分为4个独立的子区域。每个子区域最多提供2个边缘点和4个平面点，同时特征点的选择需要满足阈值的要求。
同时需要考虑特征点选择中的一些越是：比如如果一个点被其它特征点包围，那么就不被选择；以及一些点满足c的要求不过是不稳定的特征点，比如断点等。不稳定的点如下图：



代码部分:主要是在scanRegistration.cpp这个文件的任务是提取特征点，并且发布出去。
代码具体的阅读理解：
① 论文中存储每个点的曲率用的是数组，因此需要考虑数组的大小：

```
float cloudCurvature[80000];
int cloudSortInd[80000];
int cloudNeighborPicked[80000];
int cloudLabel[80000];
```

前言：N_SCANS是将3D的激光点云按照激光的接受器做了个划分，比如N_SCANS是16表明是16线的激光（程序中的默认值，作者用过velodyne16）。
对于一堆点云并不是像LaserScan（二维的数据结构）那样按照角度给出个距离值，保证每次的扫描都能够有相同大小的数据量。PointCloud2接受到的点云的大小在变化，因此在数据到达需要一些运算来判断点的一些特性。例如下面这段通过计算pitch角度值将点划分到不同的“线”中。代码坑，point.x=laserCloudIn.points[i].y；做个迷惑的赋值。Pitch=atan(z/(x2+y2));和代码中只是形式的不同。

```
  PointType point;
  std::vector<pcl::PointCloud<PointType> > laserCloudScans(N_SCANS);
  for (int i = 0; i < cloudSize; i++) {
    point.x = laserCloudIn.points[i].y;
    point.y = laserCloudIn.points[i].z;
    point.z = laserCloudIn.points[i].x;

    float angle = atan(point.y / sqrt(point.x * point.x + point.z * point.z)) * 180 /
```

```
    int scanID;
    int roundedAngle = int(angle + (angle<0.0?-0.5:+0.5));
    if (roundedAngle > 0){
      scanID = roundedAngle;
    }
    else {
      scanID = roundedAngle + (N_SCANS - 1);
    }
    if (scanID > (N_SCANS - 1) || scanID < 0 ){
      count--;
      continue;
    }
```

算出scan ID后，又将intensity属性扩展用起来，整数部分：scan ID，小数部分：每个点扫描的时间（在startOri->endOri按照均匀划分）

```
float relTime = (ori  rtOri) / (endOri - startOri);
point.intensity = sca  scanPeriod * relTime;
```

然后，将点压入到每个线中，同时更新的点云laserCloud。

```
 ……
 laserCloudScans[scanID].push_back(point);
}
cloudSize = count;

pcl::PointCloud<PointType>::Ptr laserCloud(new pcl::PointCloud<PointType>());
for (int i = 0; i < N_SCANS; i++) {
  *laserCloud += laserCloudScans[i];
}
```

含义说明：cloudCurvature是存储每个点的曲率，由上面的laserCloud+=……，可以知道的是这个时候的点云是按照线的数据方式存储的，线的数量自己定义。作者给出的计算公式：

```
    float diffX = laserCloud->points[i - 5].x + laserCloud->points[i - 4].x
                + laserCloud->points[i - 3].x + laserCloud->points[i - 2].x
                + laserCloud->points[i - 1].x - 10 * laserCloud->points[i].x
                + laserCloud->points[i + 1].x + laserCloud->points[i + 2].x
                + laserCloud->points[i + 3].x + laserCloud->points[i + 4].x
                + laserCloud->points[i + 5].x;
    float diffY = laserCloud->points[i - 5].y + laserCloud->points[i - 4].y
                + laserCloud->points[i - 3].y + laserCloud->points[i - 2].y
                + laserCloud->points[i - 1].y - 10 * laserCloud->points[i].y
                + laserCloud->points[i + 1].y + laserCloud->points[i + 2].y
                + laserCloud->points[i + 3].y + laserCloud->points[i + 4].y
                + laserCloud->points[i + 5].y;
    float diffZ = laserCloud->points[i - 5].z + laserCloud->points[i - 4].z
                + laserCloud->points[i - 3].z + laserCloud->points[i - 2].z
                + laserCloud->points[i - 1].z - 10 * laserCloud->points[i].z
                + laserCloud->points[i + 1].z + laserCloud->points[i + 2].z
                + laserCloud->points[i + 3].z + laserCloud->points[i + 4].z
                + laserCloud->points[i + 5].z;
    //jc : cloudCurvature calculate
    cloudCurvature[i] = diffX * diffX + diffY * diffY + diffZ * diffZ;
```

因为是按照线的序列存储，因此接下来能够得到起始和终止的index；在这里滤除前五个和后五个。

```
if (int(laserCloud->points[i].intensity) != scanCount) {
    scanCount = int(laserCloud->points[i].intensity);
    //N_SCANS is 16
    if (scanCount > 0 && scanCount < N_SCANS) {
  //std::vector<int> scanStartInd(N_SCANS, 0);
  //std::vector<int> scanEndInd(N_SCANS, 0);
      scanStartInd[scanCount] = i + 5;
      scanEndInd[scanCount - 1] = i - 5;
    }
  }
```

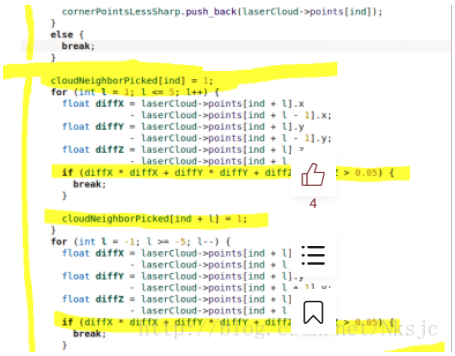
cloudSortInd是对曲率排序得到的序列：这里作者将每一线划分为等间距的6段分别处理，在每一段升序排列。
变量说明sp startPoint；ep endPoint.

```
for (int i = 0; i < N_SCANS; i++) {
    pcl::PointCloud<PointType>::Ptr surfPointsLessFlatScan(new pcl::PointCloud<Point
    for (int j = 0; j <  ; j++) {

      int sp = (scanStartInd[i] * (6 - j)  + scanEndInd[i] * j) / 6;
      int ep = (scanSta    [i] * (5 - j)  + scanEndInd[i] * (j + 1)) / 6 - 1;

      for (int k = sp +  ; k <= ep; k++) {
        for (int l = k;    > sp + 1; l--) {
          if (cloudCurvature[cloudSortInd[l]] < cloudCurvature[cloudSortInd[l - 1]])
            int temp =    SortInd[l - 1];
            cloudSortIn     1] = cloudSortInd[l];
            cloudSortIn     = temp;
          }
        }
      }
```

cloudNeighborPicked是考虑一个特征点的周围不能再设置成特征点约束的判断标志位。
cloudLabel按照如下计算，选择最大曲率的作为sharp和lessSharp，因为是按照升序排列的cloudSortInd，因此从ep->sp逐个选择：

```
    int largestPickedNum = 0;
      for (int k = ep; k >= sp; k--) {
        int ind = cloudSortInd[k];
        // jc : in cloudNeighborPicked array  1 is nerighbor and 0 is alone
        // jc : if it 's  alone and the curudCurvature is bigger then 0.1,
        if (cloudNeighborPicked[ind] == 0 &&
            cloudCurvature[ind] > 0.1) {

          largestPickedNum++;
          if (largestPickedNum <= 2)
         {
            cloudLabel[ind] = 2;    // jc : what is the difference between 2 and 1
            cornerPointsSharp.push_back(laserCloud->points[ind]);
            cornerPointsLessSharp.push_back(laserCloud->points[ind]);
          }
          else if (largestPickedNum <= 20) {
            cloudLabel[ind] = 1;
            cornerPointsLessSharp.push_back(laserCloud->points[ind]);
          }
          else {
            break;
          }

          cloudNeighborPicked[ind] = 1;
```

同时如果一个点添加到了特征点中（sharp以及lessSharp），周围的点不是特征点了；通过cloudNeighborPicked做标志来判断：接着上面的截图（一个for循环内）：



② 关于scanRegistration.cpp中的i
需要发布一个topic /imu_trans类型是 or_msgs::PointCloud2而不是sensor_msgs::Imu
作者用4个pcl::Point XYZ类型的数组 储IMU的信息，作者pcl用的6，坑神走起。

```
pcl::PointCloud<pcl::PointXYZ> imuTrans(4, 1);
imuTrans.points[0].x = imuPitchStart;
imuTrans.points[0].y = imuYawStart;
imuTrans.points[0].z = imuRollStart;

imuTrans.points[1].x = imuPitchCur;
imuTrans.points[1].y = imuYawCur;
imuTrans.points[1].z = imuRollCur;

imuTrans.points[2].x = imuShiftFromStartXCur;
imuTrans.points[2].y = imuShiftFromStartYCur;
imuTrans.points[2].z = imuShiftFromStartZCur;

imuTrans.points[3].x = imuVeloFromStartXCur;
imuTrans.points[3].y = imuVeloFromStartYCur;
imuTrans.points[3].z = imuVeloFromStartZCur;

sensor_msgs::PointCloud2 imuTransMsg;
pcl::toROSMsg(imuTrans, imuTransMsg);
imuTransMsg.header.stamp = laserCloudMsg->header.stamp;
imuTransMsg.header.frame_id = "/camera";
pubImuTrans.publish(imuTransMsg);
```

特征点关联

特征点的关联就是计算历程数据odometry推演，将t时刻的点云和t+1时刻的点云关联起来的一种方法（如果有imu或者里程计可以先进行一个初步的变换，作为初始信息用于匹配）。在t时刻得到的边缘点生成的线和t+1时刻提取的边缘点对应起来，对于t+1时刻的点，在t时刻找到最近的两个点（同时满足两点的线段有意义）；同样的考虑平面特征点。t时刻的特征点通过3D的KD-tree建立快速索引。计算这一时刻对应上一时刻最近的两个点，求点到线的距离；同样的考虑点到面的距离。

***代码部分：** laserOdometry.cpp的实现分析

首先将上页的坑，作者补上了，转换成可以看的数据

```
void imuTransHandler(const sensor_msgs::PointCloud2ConstPtr& imuTrans2)
{
  timeImuTrans = imuTrans2->header.stamp.toSec();

  imuTrans->clear();
  pcl::fromROSMsg(*imuTrans2, *imuTrans);

  imuPitchStart = imuTrans->points[0].x;
  imuYawStart = imuTrans->points[0].y;
  imuRollStart = imuTrans->points[0].z;

  imuPitchLast = imuTrans->points[1].x;
  imuYawLast = imuTrans->points[1].y;
```

```
  imuRollLast = imuTrans->points[1].z;

  imuShiftFromStartX = imuTrans->points[2].x;
  imuShiftFromStartY = imuTrans->points[2].y;
  imuShiftFromStartZ = imuTrans->points[2].z;

  imuVeloFromStartX = imuTrans->points[3].x;
  imuVeloFromStartY = i  ns->points[3].y;
  imuVeloFromStartZ = i  ns->points[3].z;

  newImuTrans = true;
}
```

以及数组的定义：其中pointSelCorn 和pointSelSurfInd并没有用到……
剩下的pointSearchCornerInd1和poi rchCornerInd2是存储最近的两个点的索引（corner） 。 另一组自然是用来存储最近点的索引（平面Sur

判断点的个数选择处理方式，如果上 刻的点的个数（边缘点个数大于10，平面特征点大于100）：

```
// jc : make sure there are    gh feature points for t+1 time to match
if (laserCloudCornerLastNum    && laserCloudSurfLastNum > 100) {
  std::vector<int> indices;
  pcl::removeNaNFromPointCloud(*cornerPointsSharp,*cornerPointsSharp, indices);
  int cornerPointsSharpNum = cornerPointsSharp->points.size();
  int surfPointsFlatNum = surfPointsFlat->points.size();
  for (int iterCount = 0; iterCount < 25; iterCount++)
  {
    laserCloudOri->clear();
    coeffSel->clear();
    //jc:功能是为了计算第i个点的最近的两个点（构成线段）
    for (int i = 0; i < cornerPointsSharpNum; i++)  {
      //jc:pointSel个人感觉pointSearch
      TransformToStart(&cornerPointsSharp->points[i], &pointSel);
      //jc:当iterCount是0的时候也执行。功能是为了计算第i个点的最近的两个点（构成线段）
      //pointSearchCornerInd1和pointSearchCornerInd2是用来存储点的索引的数组
      if (iterCount % 5 == 0)
      {
      //jc:如果找到了对应的最近的线段，构造coeffSel，即点到直线距离的误差  laserCloudOri
      if (pointSearchCornerInd2[i] >= 0)
      {
    }
    //jc:功能是为了计算第i个点的最近的三个点（构成平面）
    for (int i = 0; i < surfPointsFlatNum; i++) {
      TransformToStart(&surfPointsFlat->points[i], &pointSel);
      //jc:功能是为了计算第i个点的最近的三个点（构成平面）
      if (iterCount % 5 == 0) {
      //jc:如果找到了对应的最近的线段，构造coeffSel，即点到平面距离的误差  laserCloudOri
      if (pointSearchSurfInd2[i] >= 0 && pointSearchSurfInd3[i] >= 0) {
    }

    int pointSelNum = laserCloudOri->points.size();

    if (pointSelNum < 10) {
    cv::Mat matA(pointSelNum, 6, CV_32F, cv::Scalar::all(0));
    cv::Mat matAt(6, pointSelNum, CV_32F, cv::Scalar::all(0));
    cv::Mat matAtA(6, 6, CV_32F, cv::Scalar::all(0));
    cv::Mat matB(pointSelNum, 1, CV_32F, cv::Scalar::all(0));
    cv::Mat matAtB(6, 1, CV_32F, cv::Scalar::all(0));
    cv::Mat matX(6, 1, CV_32F, cv::Scalar::all(0));
    // pointSelNum是有多少个对应约束
    for (int i = 0; i < pointSelNum; i++) {

    cv::transpose(matA, matAt);
    matAtA = matAt * matA;
    matAtB = matAt * matB;
    cv::solve(matAtA, matAtB, matX, cv::DECOMP_QR);

    if (iterCount == 0) {

    if (isDegenerate) {

    transform[0] += matX.at<float>(0, 0);
    transform[1] += matX.at<float>(1, 0);
    transform[2] += matX.at<float>(2, 0);
    transform[3] += matX.at<float>(3, 0);
    transform[4] += matX.at<float>(4, 0);
    transform[5] += matX.at<float>(5, 0);
```


在迭代的过程中计算transform；iterCound%5==0的意思也就出来了，因为在迭代的过程中最近的点的匹配是随

着transform的更新逐渐变化的，因此作者采用了5次迭代（计算量的一种平衡吧，每次更新transform后都更新一次最近匹配计算资源消耗大？）后再计算一次对应的最近点。这样下次通过更新的最近点匹配对来完成新的计算。

在当前帧中遍历上一帧中最近的两个点。作者没有使用nearKSearch函数直接求最近的2个点的原因是，nearKSearch求出的两个点不一定能构成合理的直线。

```
        //jc:功能是为了计 个点的最近的两个点(构成线段)
        for (int i = 0; i < cornerPointsSharpNum; i++)  {
        //jc:pointSel个.'求'pointSearch
            TransformTo (&cornerPointsSharp->points[i], &pointSel);
            //jc:当iterCount是0的时候也执行。功能是为了计算第i个点的最近的两个点(构成线段)
        //pointSearchCo ind1和pointSearchCornerInd2是用来存储点的索引的数组
            if (iterCount % 5 == 0)
        {
              std::vect t> indices;
              pcl::remo FromPointCloud(*laserCloudCornerLast,*laserCloudCornerLa
          //jc:kdtreeCo ast由上次的LessSharp的corner点构造
              kdtreeCor st->nearestKSearch(pointSel, 1, pointSearchInd, pointSea
              int close ntInd = -1, minPointInd2 = -1;
          //jc:两桢之间的 不会太剧烈,25作为阈值
              if (pointSearchSqDis[0] < 25) {
                closestPointInd = pointSearchInd[0];
        //pointSel对应上一桢的那一线?closestPointScan
                int closestPointScan = int(laserCloudCornerLast->points[closestPoint
                float pointSqDis, minPointSqDis2 = 25;
        //jc:找到closestPointInd后找次之的点的索引
        //这里作者没有直接nearKSearch找最近的两个点,也为了考虑这两个点要有效的吧,
        //下面是不同约束,不能差3线以上,不能距离过大。
                for (int j = closestPointInd + 1; j < cornerPointsSharpNum; j++) {
                  if (int(laserCloudCornerLast->points[j].intensity) > closestPointS
                    break;
                  }
                  pointSqDis = (laserCloudCornerLast->points[j].x - pointSel.x) *
                               (laserCloudCornerLast->points[j].x - pointSel.x) +
                               (laserCloudCornerLast->points[j].y - pointSel.y) *
                               (laserCloudCornerLast->points[j].y - pointSel.y) +
                               (laserCloudCornerLast->points[j].z - pointSel.z) *
                               (laserCloudCornerLast->points[j].z - pointSel.z);
                  if (int(laserCloudCornerLast->points[j].intensity) > closestPointS
                    if (pointSqDis < minPointSqDis2) {
                      minPointSqDis2 = pointSqDis;
                      minPointInd2 = j;
                    }
                  }
                }
                for (int j = closestPointInd - 1; j >= 0; j--) {
                  if (int(laserCloudCornerLast->points[j].intensity) < closestPointS
                    break;
                  }
                  pointSqDis = (laserCloudCornerLast->points[j].x - pointSel.x) *
                               (laserCloudCornerLast->points[j].x - pointSel.x) +
                               (laserCloudCornerLast->points[j].y - pointSel.y) *
                               (laserCloudCornerLast->points[j].y - pointSel.y) +
                               (laserCloudCornerLast->points[j].z - pointSel.z) *
                               (laserCloudCornerLast->points[j].z - pointSel.z);
                  if (int(laserCloudCornerLast->points[j].intensity) < closestPointS
                    if (pointSqDis < minPointSqDis2) {
                      minPointSqDis2 = pointSqDis;
                      minPointInd2 = j;
                    }
                  }
                }
              }
              pointSearchCornerInd1[i] = closestPointInd;
              pointSearchCornerInd2[i] = minPointInd2;
            }
```

同理还有遍历当前帧中的Surf点在上一帧找最近的平面在循环
for (int i = 0; i < surfPointsFlatNum; i++) 中完成。。。Surf的思想和corner的大同小异，这里以及下面以corner的对应点计算以及误差计算来表述作者的思想。

运动估计

假设：雷达的运动是连续的。将所有**对应到的点求到直线的距离到面的距离之和最短然后按照Levenberg-Marquardt算法迭代计算**，得到两帧之间 换，最后通过累计计算odom。在这里，需要得到的是距离对坐标变换的偏导数。
首先是计算偏导数，这一部分手打 烦，暂时省去，……下面是作者推到偏导数然后得到的表达式，厉害了，具体含义鞋子图片中了……：

```
//jc:判断第二个点存在，即找到了对应的最近的线段，构造coeffSel，即点到直线距离的误差

if (pointSearchCornerI                    ] >= 0)
{ //jc:最近的两个点tripod1和tripod2
  tripod1 = laserCloud                    erLast->points[pointSearchCornerInd1[i]];
  tripod2 = laserCloud                    erLast->points[pointSearchCornerInd2[i]];
  float x0 = pointSel.
  float y0 = pointSel.
  float z0 = pointSel.
  float x1 = tripod1.x
  float y1 = tripod1.y,
  float z1 = tripod1.z;
  float x2 = tripod2.x;
  float y2 = tripod2.y;
  float z2 = tripod2.z;
  //jc: a012= |(pointSel-tripod1)*(pointSel-tripod2)|差乘后的向量的模
  float a012 = sqrt(((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
             * ((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
             + ((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))
             * ((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))
             + ((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1))
             * ((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1)));
  //jc: tripod1与tripod2的距离l12
  float l12 = sqrt((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2) + (z1 - z2)*(z1 - z2));
  //jc:la、lb、lc分别是ld2(距离)对x0\y0\z0的偏导数；最终要对transform的偏导
  float la = ((y1 - y2)*((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
           + (z1 - z2)*((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))) / a012 / l12;
  float lb = -((x1 - x2)*((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
           - (z1 - z2)*((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1))) / a012 / l12;
  float lc = -((x1 - x2)*((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))
           + (y1 - y2)*((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1))) / a012 / l12;
  float ld2 = a012 / l12;//jc: ld2是按照距离公式计算得到的
  pointProj = pointSel;  //jc：这个变量好像没用到……
  pointProj.x -= la * ld2;
  pointProj.y -= lb * ld2;
  pointProj.z -= lc * ld2;
  float s = 1;
  //点到直线距离小，权重大；
  if (iterCount >= 5) {
    s = 1 - 1.8 * fabs(ld2);
  }
  //jc:s是系数，具体为的原因，可能是平衡边缘点和平面点？平面部分也有这个系数，数值不同
  //jc:
  coeff.x = s * la;
  coeff.y = s * lb;
  coeff.z = s * lc;
  coeff.intensity = s * ld2;
  if (s > 0.1 && ld2 != 0) {
    laserCloudOri->push_back(cornerPointsSharp->points[i]);
    coeffSel->push_back(coeff);
  }
 }
}
```

```
// pointSelNum是有多少个对应约束
for (int i = 0; i < pointSelNum; i++) {
  pointOri = laserCloudOri->points[i];
  coeff = coeffSel->points[i];

  float s = 1;

  float srx = sin(s * transform[0]);
  float crx = cos(s * transform[0]);
  float sry = sin(s * transform[1]);
  float cry = cos(s * transform[1]);
  float srz = sin(s * transform[2]);
  float crz = cos(s * transform[2]);
  float tx = s * transform[3];
  float ty = s * transform[4];
  float tz = s * transform[5];
   //jc：偏导数……
  float arx = (-s*crx*sry*srz*pointOri.x + s*crx*crz*sry*pointOri.y + s*srx*sry*pointOri.z
            + s*tx*crx*sry*srz  ty*crx*crz*sry - s*tz*srx*sry) * coeff.x
            + (s*srx*srz*point(  - s*crz*srx*pointOri.y + s*crx*pointOri.z
            + s*ty*crz*srx - s*tz*crx - s*tx*srx*srz) * coeff.y
            + (s*crx*cry*srz*p  ri.x - s*crx*cry*crz*pointOri.y - s*cry*srx*pointOri.z
            + s*tz*cry*srx + s  rx*cry*crz - s*tx*crx*cry*srz) * coeff.z;

  float ary = ((-s*crz*sry - s  srx*srz)*pointOri.x
            + (s*cry*crz*srx -  y*srz)*pointOri.y - s*crx*cry*pointOri.z
            + tx*(s*crz*sry + s  *srx*srz) + ty*(s*sry*srz - s*cry*crz*srx)
            + s*tz*crx*cry) * (  .x
            + ((s*cry*crz - s*srx*sry*srz)*pointOri.x
            + (s*cry*srz + s*crz*srx*sry)*pointOri.y - s*crx*sry*pointOri.z
            + s*tz*crx*sry - ty*(s*cry*srz + s*crz*srx*sry)
            - tx*(s*cry*crz - s*srx*sry*srz)) * coeff.z;

  float arz = ((-s*cry*srz - s*crz*srx*sry)*pointOri.x + (s*cry*crz - s*srx*sry*srz)*pointOri.y
            + tx*(s*cry*srz + s*crz*srx*sry) - ty*(s*cry*crz - s*srx*sry*srz)) * coeff.x
            + (-s*crx*crz*pointOri.x - s*crx*srz*pointOri.y
            + s*ty*crx*srz + s*tx*crx*crz) * coeff.y
            + ((s*cry*crz*srx - s*sry*srz)*pointOri.x + (s*crz*sry + s*cry*srx*srz)*pointOri.y
            + tx*(s*sry*srz - s*cry*crz*srx) - ty*(s*crz*sry + s*cry*srx*srz)) * coeff.z;

  float atx = -s*(cry*crz - srx*sry*srz) * coeff.x + s*crx*srz * coeff.y
            - s*(crz*sry + cry*srx*srz) * coeff.z;

  float aty = -s*(cry*srz + crz*srx*sry) * coeff.x - s*crx*crz * coeff.y
            - s*(sry*srz - cry*crz*srx) * coeff.z;

  float atz = s*crx*sry * coeff.x - s*srx * coeff.y - s*crx*cry * coeff.z;

  float d2 = coeff.intensity;

  matA.at<float>(i, 0) = arx;
  matA.at<float>(i, 1) = ary;
  matA.at<float>(i, 2) = arz;
  matA.at<float>(i, 3) = atx;
  matA.at<float>(i, 4) = aty;
  matA.at<float>(i, 5) = atz;
  matB.at<float>(i, 0) = -0.05 * d2;
}
```


迭代更新transform即变换矩阵

```
1            cv::transpose(matA, matAt);
2            matAtA = matAt * matA;
3            matAtB = matAt * matB;
4            cv::solve(matAtA, matAtB, matX, cv::DECOMP_QR);
5
6            if (iterCount == 0) {
7              cv::Mat matE(1, 6, CV_32F, cv::Scalar::all(0));
8              cv::Mat matV(6, 6, CV_32F, cv::Scalar::all(0));
9              cv::Mat matV2(6, 6, CV_32F, cv::Scalar::all(0));
10
11             cv::eigen(matAtA, matE, matV);
12             matV.copyTo(matV2);
13
14             isDegenerate = false;
15             float eignThre[6] = {10, 10, 10, 10, 10, 10};
16             for (int i = 5; i >= 0; i--) {
17               if (matE.at<float>(0, i) < eignThre[i]) {
18                 for (int j = 0; j < 6; j++) {
19                   matV2.at<float>(i, j) = 0;
20                 }
```

```
                isDegenerate = true;
              } else {
                break;
              }
            }
            matP = matV.inv() * matV2;
          }

          if (isDegenerate) {
            cv::Mat matX2(6, 1, CV_32F, cv::Scalar::all(0));
            matX.copyTo(matX2);
            matX = matP * matX2;
          }

          transform[0] += matX.at<float>(0, 0);
          transform[1] += matX.at<float>(1, 0);
          transform[2] += matX.at<float>(2, 0);
          transform[3] += matX.at<float>(3, 0);
          transform[4] += matX.at<float>(4, 0);
          transform[5] += matX.at<float>(5, 0);
```

累计计算总的里程计数据：transform；之后会作为/laser_odom_to_init主题发布出去，程序中作者并不喜欢用tf变换，而是节点的订阅。mapping节点订阅。

laserMapping.cpp代码阅读解析

迭代计算的流程

1.从划分好的地图中加载当前激光扫描所在的区域部分

```
kdtreeCornerFromMap->setInputCloud(laserCloudCornerFromMap);
kdtreeSurfFromMap->setInputCloud(laserCloudSurfFromMap);
```

2.迭代开始`for(int iterCount=0;iterCount<10;iterCount++)`

```
laserCloudOri->clear();//迭代中的中间变量
coefSel->clear();
```

3.计算corner点到kdtreeCornerFromMap中提出来的5个最近点的匹配关系。

```
// pointOri是激光点转换到map坐标系下的表示
//每次迭代更新transformTobeMapped,而pointOri是在transformTobeMapped变换后的点
laserCloudOri->push_back(pointOri);
// coeff的四列分别是距离对(x,y,z)的导数和点到匹配直线的距离。
coeffSel->push_back(coeff);
```

4.计算surf点到kdtreeSurfFromMap中的点形成面的匹配关系

```
从中取最近的5个点->判断点能否拟合平面(planeValid) -> surf点到面的距离小于阈值
```

5.计算迭代的delta,根据pointOri和coeffSel计算之后更新transformTobeMapped

具体部分说明

1.空间划分，将新获得的corner和surf激光点分别映射到每个立方体laserCloudCornerArray和laserCloudSurfArray

```
int laserCloudCenWidth = 10;
int laserCloudCenHeight = 5;
int laserCloudCenDepth = 10;
const int laserCloudWidth = 21;
const int laserCloudHeight = 11;
const int laserCloudDepth = 21;
const int laserCloudNum = laserCloudWidth * laserCloudHeight * laserCloudDepth;
  for (int i = 0; i < laserCloudNum; i++) {
    laserCloudCornerArray[i].reset(new pcl::PointCloud<PointType>());
```

```
    laserCloudSurfArray[i].reset(new pcl::PointCloud<PointType>());
    laserCloudCornerArray2[i].reset(new pcl::PointCloud<PointType>());
    laserCloudSurfArray2[i].reset(new pcl::PointCloud<PointType>());
  }
        //以surf点为例,加入的方式
        for (int i = 0; i < laserCloudSurfStackNum; i++) {
          //根据transformTobeMapped将每个点变换到map坐标系下
          pointAssociat                p(&laserCloudSurfStack->points[i], &pointSel);
          //根据点在map生                的位置,计算点在laserCloudSurfArray中的索引值
          int cubeI = int((pointSel.x + 25.0) / 50.0) + laserCloudCenWidth;
          int cubeJ = i               ointSel.y + 25.0) / 50.0) + laserCloudCenHeight;
          int cubeK = i               ointSel.z + 25.0) / 50.0) + laserCloudCenDepth;
          if (pointSel.               5.0 < 0) cubeI--;
          if (pointSel.               5.0 < 0) cubeJ--;
          if (pointSel.z + 25.0 < 0) cubeK--;
          if (cubeI >=                cubeI < laserCloudWidth &&
              cubeJ >= 0 &&           cubeJ < laserCloudHeight &&
              cubeK >=                cubeK < laserCloudDepth) {
            int cubeInd               ubeI + laserCloudWidth * cubeJ + laserCloudWidth * laserC
            laserCloudS               ray[cubeInd]->push_back(pointSel);
          }
        }
```

2.匹配对应点

```
laserCloudCornerFromMap和laserCloudSurfMap分别是从laser_cloud_corner_last和laser_cloud
对应在laserCloudCornerArray和laserCloudSurfArray中激活的区域的点的叠加。
              // 在已经划分好的空间内搜索最近的5个点
              kdtreeCornerFromMap->nearestKSearch(pointSel, 5, pointSearchInd, point
              // 如果5个最近点中最远的距离也小于1m,认为是潜在匹配线段
              // 构建这五个点的(x,y,z)方向的3*3的协方差矩阵,之后根据特征根来判断是否能拟合成直
              // 判断的方法是最大的特征根大于次大的特征根3倍。
              // 如果使用matlab,
              // eig(cov(x,y,z)),其中的x,y,z是点云的各个分量向量。
              if (pointSearchSqDis[4] < 1.0) {
                float cx = 0;
                float cy = 0;
                float cz = 0;
                for (int j = 0; j < 5; j++) {
                  cx += laserCloudCornerFromMap->points[pointSearchInd[j]].x;
                  cy += laserCloudCornerFromMap->points[pointSearchInd[j]].y;
                  cz += laserCloudCornerFromMap->points[pointSearchInd[j]].z;
                }
                cx /= 5;
                cy /= 5;
                cz /= 5;
                float a11 = 0;
                float a12 = 0;
                float a13 = 0;
                float a22 = 0;
                float a23 = 0;
                float a33 = 0;
                for (int j = 0; j < 5; j++) {
                  float ax = laserCloudCornerFromMap->points[pointSearchInd[j]].x -
                  float ay = laserCloudCornerFromMap->points[pointSearchInd[j]].y -
                  float az = laserCloudCornerFromMap->points[pointSearchInd[j]].z -
                  a11 += ax * ax;
                  a12 += ax * ay;
                  a13 += ax * az;
                  a22 += ay * ay;
                  a23 += ay * az;
                  a33 += az * az;
                }
                a11 /= 5;
                a12 /= 5;
                a13 /= 5;
                a22 /= 5;
                a23 /= 5;
                a33 /= 5;
```

```
                // 5个点的协方差矩阵
                matA1.at<float>(0, 0) = a11;
                matA1.at<float>(0, 1) = a12;
                matA1.at<float>(0, 2) = a13;
                matA1.at<float>(1, 0) = a12;
                matA1.at<float>(1, 1) = a22;
                matA1.at<float>(1, 2) = a23;
                matA1.a  at<float>(2, 0) = a13;
                matA1.a  at<float>(2, 1) = a23;
                matA1.at<float>(2, 2) = a33;
                // 特征  1,对应的特征向量matV1.
                cv::eig  tA1, matD1, matV1);
                if (mat  .<float>(0, 0) > 3 * matD1.at<float>(0, 1)) {
                  float  pointSel.x;
                  float  pointSel.y;
                  float  pointSel.z;
                  float x1 = cx + 0.1 * matV1.at<float>(0, 0);
                  float  cy + 0.1 * matV1.at<float>(0, 1);
                  float  cz + 0.1 * matV1.at<float>(0, 2);
                  float  cx - 0.1 * matV1.at<float>(0, 0);
                  float  cy - 0.1 * matV1.at<float>(0, 1);
                  float  cz - 0.1 * matV1.at<float>(0, 2);
                  // 计算(x0,y0,z0)到线段(x1,y1,z1)(x2,y2,z2)的距离。
                  float a012 = sqrt(((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
                              * ((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
                              + ((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))
                              * ((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))
                              + ((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1))
                              * ((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1)));
                  float l12 = sqrt((x1 - x2)*(x1 - x2) + (y1 - y2)*(y1 - y2) + (z1 -
                  // la=diff(ld2)/diff(x0),表示点point0到point1和point2直线距离对x0的偏
                  float la = ((y1 - y2)*((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
                           + (z1 - z2)*((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))) /
                  float lb = -((x1 - x2)*((x0 - x1)*(y0 - y2) - (x0 - x2)*(y0 - y1))
                           - (z1 - z2)*((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1))) /
                  float lc = -((x1 - x2)*((x0 - x1)*(z0 - z2) - (x0 - x2)*(z0 - z1))
                           + (y1 - y2)*((y0 - y1)*(z0 - z2) - (y0 - y2)*(z0 - z1))) /
                  // 点到线段距离公式
                  float ld2 = a012 / l12;
                  pointProj = pointSel;
                  pointProj.x -= la * ld2;
                  pointProj.y -= lb * ld2;
                  pointProj.z -= lc * ld2;
                  float s = 1 - 0.9 * fabs(ld2);
                  coeff.x = s * la;
                  coeff.y = s * lb;
                  coeff.z = s * lc;
                  coeff.intensity = s * ld2;
                  if (s > 0.1) {
                    laserCloudOri->push_back(pointOri);
                    coeffSel->push_back(coeff);
                  }
                }
              }
            }
```

3.迭代计算步长

```
            float srx = sin(transformTobeMapped[0]);
            float crx = cos(transformTobeMapped[0]);
            float sry = sin(transformTobeMapped[1]);
            float cry = cos(transformTobeMapped[1]);
            float srz = sin(transformTobeMapped[2]);
            float crz = cos(transformTobeMapped[2]);
            int laserCloudSelNum = laserCloudOri->points.size();
            if (laserCloudSelNum < 50) {
              continue;
            }
            // matA 是雅克比矩阵,matAt*matA*matX = matAt*matB;
```

```
// 其中matX是步长,(roll , ptich ,yaw,x,y,z)
cv::Mat matA(laserCloudSelNum, 6, CV_32F, cv::Scalar::all(0));
cv::Mat matAt(6, laserCloudSelNum, CV_32F, cv::Scalar::all(0));
cv::Mat matAtA(6, 6, CV_32F, cv::Scalar::all(0));
cv::Mat matB(laserCloudSelNum, 1, CV_32F, cv::Scalar::all(0));
cv::Mat matAtB(6, 1, CV_32F, cv::Scalar::all(0));
cv::Mat matX(6, 1, CV_32F, cv::Scalar::all(0));
for (int i                i < laserCloudSelNum; i++) {
  pointOri                erCloudOri->points[i];
  coeff = coeffSel->points[i];

  float arx                rx*sry*srz*pointOri.x + crx*crz*sry*pointOri.y - srx*sry
                           srx*srz*pointOri.x - crz*srx*pointOri.y - crx*pointOri.z
                           rx*cry*srz*pointOri.x + crx*cry*crz*pointOri.y - cry*srx

  float ary                cry*srx*srz - crz*sry)*pointOri.x
           + (sry*srz + cry*crz*srx)*pointOri.y + crx*cry*pointOri.z) *
                           -cry*crz - srx*sry*srz)*pointOri.x
                           ry*srz - crz*srx*sry)*pointOri.y - crx*sry*pointOri.z) *

  float arz                crz*srx*sry - cry*srz)*pointOri.x + (-cry*crz-srx*sry*sr
                           rx*crz*pointOri.x - crx*srz*pointOri.y) * coeff.y
           + ((sry*srz + cry*crz*srx)*pointOri.x + (crz*sry-cry*srx*srz
  matA.at<float>(i, 0) = arx;
  matA.at<float>(i, 1) = ary;
  matA.at<float>(i, 2) = arz;
  matA.at<float>(i, 3) = coeff.x;
  matA.at<float>(i, 4) = coeff.y;
  matA.at<float>(i, 5) = coeff.z;
  matB.at<float>(i, 0) = -coeff.intensity;
}
cv::transpose(matA, matAt);
matAtA = matAt * matA;
matAtB = matAt * matB;
cv::solve(matAtA, matAtB, matX, cv::DECOMP_QR);
if (iterCount == 0) {
  cv::Mat matE(1, 6, CV_32F, cv::Scalar::all(0));
  cv::Mat matV(6, 6, CV_32F, cv::Scalar::all(0));
  cv::Mat matV2(6, 6, CV_32F, cv::Scalar::all(0));
  cv::eigen(matAtA, matE, matV);
  matV.copyTo(matV2);
  isDegenerate = false;
  float eignThre[6] = {100, 100, 100, 100, 100, 100};
  for (int i = 5; i >= 0; i--) {
    if (matE.at<float>(0, i) < eignThre[i]) {
      for (int j = 0; j < 6; j++) {
        matV2.at<float>(i, j) = 0;
      }
      isDegenerate = true;
    } else {
      break;
    }
  }
  matP = matV.inv() * matV2;
}
if (isDegenerate) {
  cv::Mat matX2(6, 1, CV_32F, cv::Scalar::all(0));
  matX.copyTo(matX2);
  matX = matP * matX2;
}
transformTobeMapped[0] += matX.at<float>(0, 0);
transformTobeMapped[1] += matX.at<float>(1, 0);
transformTobeMapped[2] += matX.at<float>(2, 0);
transformTobeMapped[3] += matX.at<float>(3, 0);
transformTobeMapped[4] += matX.at<float>(4, 0);
transformTobeMapped[5] += matX.at<float>(5, 0);

float deltaR = sqrt(
                    pow(rad2deg(matX.at<float>(0, 0)), 2) +
```

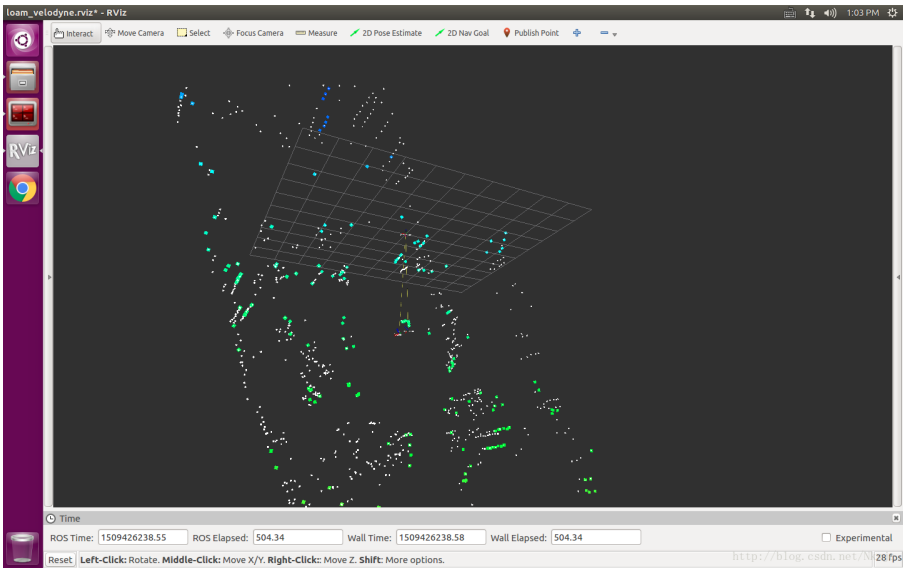
```
                                pow(rad2deg(matX.at<float>(1, 0)), 2) +
                                pow(rad2deg(matX.at<float>(2, 0)), 2));
            float deltaT = sqrt(
                                pow(matX.at<float>(3, 0) * 100, 2) +
                                pow(matX.at<float>(4, 0) * 100, 2) +
                                pow(matX.at<float>(5, 0) * 100, 2));

            if (deltaR < 0.05 && deltaT < 0.05) {
              break;
            }
          }

        // jc : in this f      on ,update the transformAftMapped
          transformUpda
```

4.发布点云

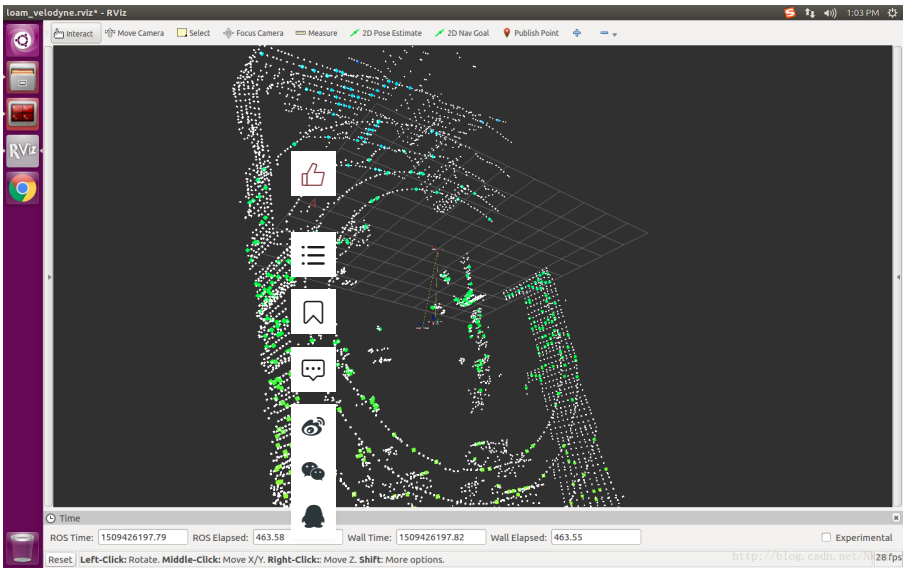
```
        if (mapFrameCou      mapFrameNum) {
          mapFrameCount

          laserCloudSur      2->clear();
          for (int i =      < laserCloudSurroundNum; i++) {
            int ind = laserCloudSurroundInd[i];
            *laserCloudSurround2 += *laserCloudCornerArray[ind];
            *laserCloudSurround2 += *laserCloudSurfArray[ind];
          }

          laserCloudSurround->clear();
          downSizeFilterCorner.setInputCloud(laserCloudSurround2);
          downSizeFilterCorner.filter(*laserCloudSurround);

          sensor_msgs::PointCloud2 laserCloudSurround3;
          pcl::toROSMsg(*laserCloudSurround, laserCloudSurround3);
          laserCloudSurround3.header.stamp = ros::Time().fromSec(timeLaserOdometry);
          laserCloudSurround3.header.frame_id = "/camera_init";
          pubLaserCloudSurround.publish(laserCloudSurround3);
        }
```

5.特征点示意图

cornerPoints, 彩色的是sharp，白色lessSharp



同样是平面点



写下你的评论...

pingdifengsheng 2018-04-10 15:41 #8楼 回复

感谢楼主，分析的很彻底，学习了

wzy_2681 2018-04-01 15:08 #7楼 回复

感谢楼主的分享！请问laserOdometry里得到的旋转和平移是否是相对于雷达坐标系的?

pmouselor 2018-03-26 15:09 #6楼 回复

博主您好，看了您的分析非常受益，我正准备研究zhang ji的loam 代码，但github上下不到了，能否麻烦您发一份到我的邮箱 56392987@qq.com, 非常感谢!

查看 13 条热评

如何修改CJlibrary608在VC.net环境下运行 objectman 2003-04-09 09:56:00 806

CJlibrary 6.08是一套非常漂亮的用户界面类。为广大的VC用户所欢迎。但是在VC.net下编译的时候报错，需要修改方能运行通过。我已把我修改并编译通过的过程记录下来，供大家参考。下面列出每个...

基于loam_velodyne的64线激光数据可视化 weixin_39837709 2018-01-24 11:33:21 530

0.背景因为课题组任务以及自己课题的需要，需要对velodyne64线激光数据进行可视化处理，使用LOAM算法，参考了网上已有的教程和方法，复现还是有些吃力，并且发现这些教程讲述的并不是很清晰明，因此...

cartogarpher *slam* 4 - CSDN博客

由于小车搭载处理器的性能*瓶颈*,另一方面,室外场景确实...cartographer与karto的都是2D*激光SLAM*算法,而且不是基于...*loam*_velodyne论文与代... zhugeaming2018: 谢谢分享...

2018-4-17

Cartographer_ROS初探 - CSDN博客

3D *激光SLAM ->loam*_velodyne论文与代码解析Lidar Odometry and Mapping 3D *SLAM*...遇到了莫名的卡顿——一方面是由于小车搭载处理器的性能*瓶颈*,另一方面,室外场景...

2018-4-16

4

【SLAM】之Velodyne VLP1 光雷达使用

littlethunder 2016-07-15 20:48:42 21830

Velodyne VLP16型激光雷达横向视角360°，纵向视角30°，如下图：实验机器是ubuntu 14.04 x64， ROS版本Indigo，目前ROS支持的Velodyne型号是：HD...

【SLAM】之建图Bag->Pcd-> oMap

littlethunder 2016-07-19 15:55:15 9992

上篇中我们得到了3D激光雷达获得的点云 在.bag文件中，接下来我们再用上上篇末尾的做法跑loam_velodyne算法，在RVIZ中的显示效果如下：这时我们用rqt_grap 看到： ...

【SLAM】之ROS安装，配置 试

littlethunder 2016-07-15 18:39:11 6759

我的系统是Ubuntu 15.04 Vivid， 安装ROS Jade。 sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb...

Loam_velodyne(二)

zetan 2017-10-20 19:31:15 658

draft

查看loam的三维点云地图

u013630299 2018-01-27 10:48:47 219

roslaunch loam_velodyne loam_velodyne.launch rosbag record -o out /velodyne_points rosbag play nsh...

Loam_velodyne(一)

zetan 2017-10-16 19:21:31 361

draft

loam_velodyne论文与代码解析

datase 2018-03-22 09:26:26 145

3D SLAM ->loam_velodyne论文与代码解析一直做2D 的激光SLAM，最近终于接触到3D的了，想彻底的看透一个开源的3D 激光SLAM，选择了Loam_velodyne...

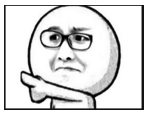
Hector SLAM算法学习与代码解析

Nksjc 2017-05-27 11:03:29 2432

Hector SLAM算法学习与代码解析最初了解到Hector SLAM 是从https://www.youtube.com/embed/F8pdObV_df4看到手持激光建图，被吸引到了也就想看一下...

50万码农评论：英语对于程序员有多重要?

不背单词和语法，一个公式学好英语



机器人SLAM算法漫谈

u010368556 2017-03-15 19:26:20 4541

本文转载微信公众号 “智能算法” 完整的干货，拿来大家分享！ http://mp.weixin.qq.com/s/pBpTH0B5AKRGMZ_8rrO4zg 1. 前言 开始做SLAM（...

Gmapping hector cartogarpher 三种激光雷达算法的比对

一、Gmapping是基于粒子滤波的算法。缺点：严重依赖里程计，无法适应无人机及地面不平坦的区域，无回环（激光SLAM很难做回环检测），大的场景，粒子较多的情况下，特别消耗资源。源码的...

datase 2017-11-23 20:13:21 506

经典ASP.NET MVC3.0入门详解

csh624366188 2011-12-12 17:16:22 40763

注：由于本文原在word文档里编写，写本文章时运用了大量截图，直接复制到博客里，没有显示图片，图片只是一些简单的运行结果截图，不影响大家学习 p.Net MVC已经到第三版了，相信大家也都...

SLAM的前世今生 终于有人说 了

jaccen 2017-02-17 10:56:07 1717

SLAM的前世我之前从本科到研究生，一直在研究与定位领域学习，一开始偏重于高精度的惯性导航、卫星导航、星光制导及其组合导航。出于对实现无源导航的执念，我慢慢开始研究视觉导航中的SLAM方向，并...

【SLAM】（一）Google Car rapher的初步尝试

jsgaobiao 2016-11-10 15:14:22 10246

最近刚定了毕业设计的题目，大概是做SLAM方向，恰好Google不久前开源了他的室内SLAM库Cartographer，我就拿来用实验室的Robot试了一下，总体效果还是很好的。

3D激光slam，cartographer 用，第一视角点云

Nksjc 2017-08-18 09:37:18 967

3D激光slam，cartographer的使用，第一 云

即时定位与地图构建（SLAM）的相关研究

lcj_cjfykx 2015-06-08 08:43:04 10336

即时定位与地图构建（SimultaneousLocalization AndMapping）指的是机器人在自身位置不确定的条件下,在完全未知环境中创建地图,同时利用地图进行自主定位和导航。 SL...

octomap

linuxarmsummary 2016-03-18 16:50:50 3930

大家好，时隔多年之后，我又开始了博客旅程。经历了很多事情之后呢，我发现自己的想法真的很简单：好好读书做课题，闲下来时写写博客，服务大家。所以我会继续写SLAM相关的博客。如果你觉得它对你有帮助，那是最...

【SLAM】（二）Cartographer的原理探究——GraphSLAM理论基础

Cartographer的原理探究——GraphSLAM理论基础

jsgaobiao 2017-03-25 01:35:00 5663

Gmapping、hector、Cartographer三种激光SLAM算法简单对比

一、Gmapping是基于粒子滤波的算法。缺点：严重依赖里程计，无法适应无人机及地面不平坦的区域，无回环（激光SLAM很难做回环检测），大的场景，粒子较多的情况下，特别消耗资源。二、Hector S...

Jeff_Lee_ 2017-09-06 18:55:54 5606