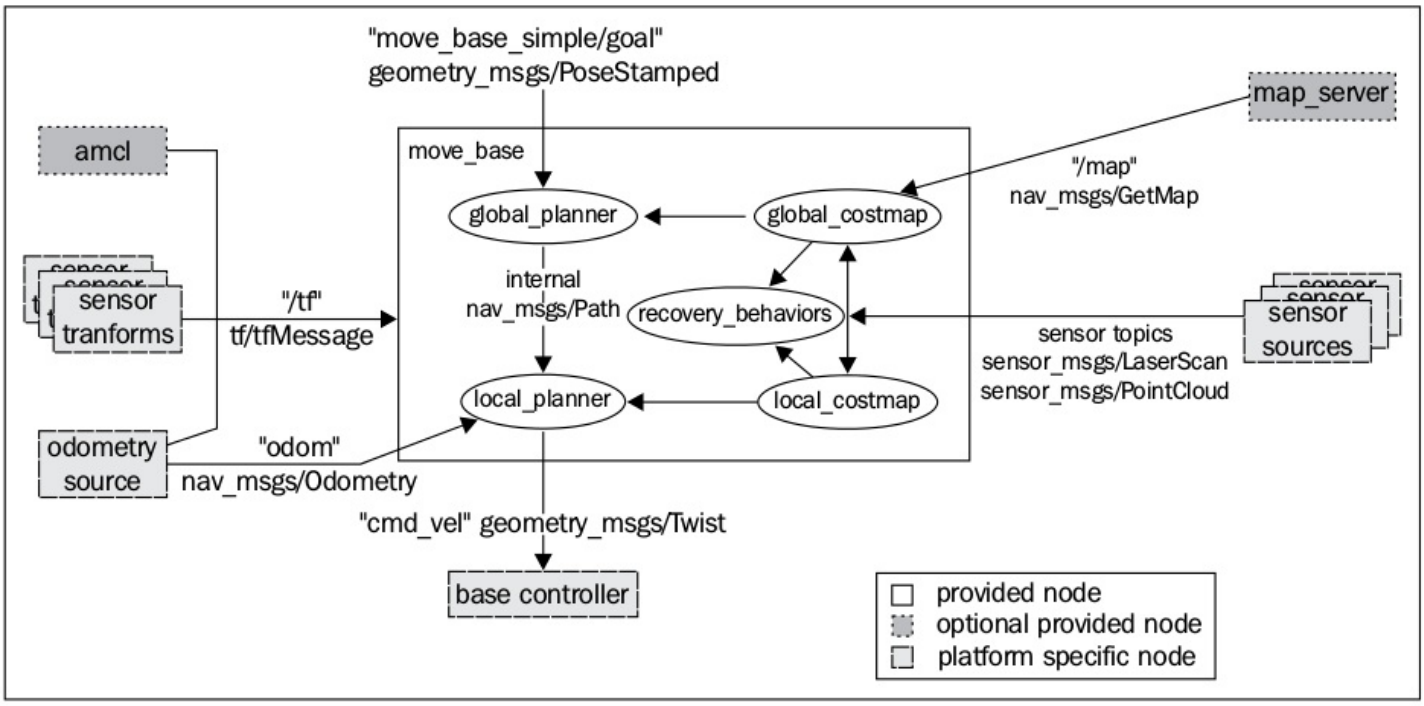
Ros navigation功能包集源码详解(一) amcl

Navigation stack总体结构

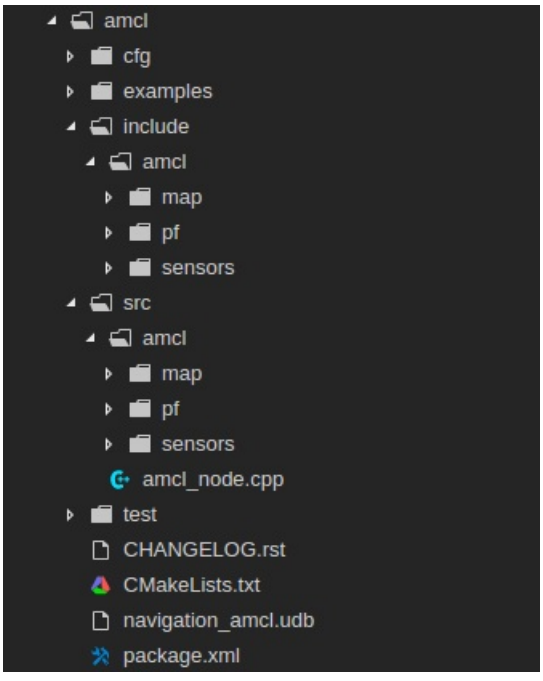


本文主要解读amcl包对蒙特卡洛定位算法的具体实现过程,主要关注源码中定义的数据类型,及程序的具体实现流程,对于详细的算法原理不作过多解释(主要是我不太懂,讲不好),想了解涉及到的理论知识推荐阅读<< Probabilistic Robotics >>

1.定位包amcl的 功能

输入:地图,激光数据,里程计
输出:位姿pose(x,y,θ),粒子集

2.源码文件结构

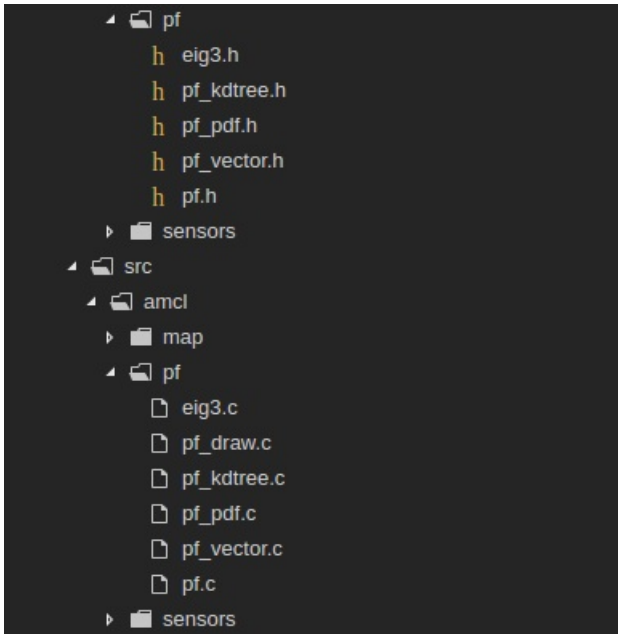


最为核心的是include目录和src目录,一共包含三个大类,map,pf,以及sensors,还有一个重要文件amcl_node.cpp

1). map处理地图，pf是算法的核心粒子滤波(particle filter),sensors就是处理雷达跟里程计传感器的
2). amcl_node.cpp，定义了一个可运行节点,定位过程主要在此文件中组织实现

3.主要数据类型与算法

3.1 pf



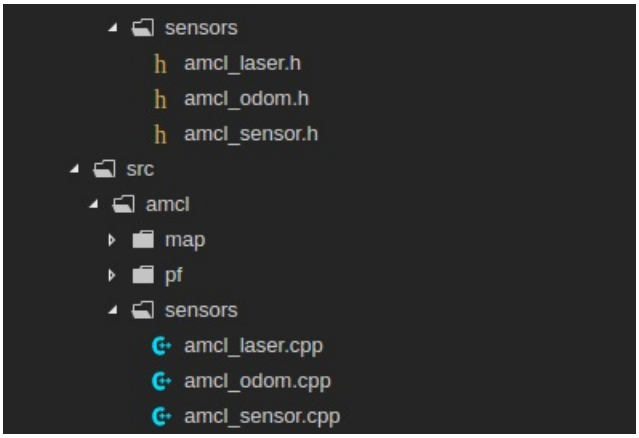
a).eig3(指头文件和源文件,下同)实现的是一个3x3对称矩阵的特征值与特征向量的计算,首先用Householder矩阵将矩阵变换为三对角矩阵,然后使用ql分解迭代计算
b).pf_kdtree定义了一个kdtree以及维护方法来管理所有粒子
c).pf_pdf主要定义了一个从给定pdf中采样粒子的方法
d).pf_vector定义了三维列向量和三维矩阵和基本的运算方法
e).pf定义了粒子单元pf_sample_t,粒子集pf_sample_set_t,粒子滤波pf_t的数据类型,还有一个pf_cluster_t表示粒子集的聚类信息,关键函数主要包含如下三个,分别对应粒子滤波中的运动更新,观测更新,重采样三个过程

```
// Update the filter with some new action
void pf_update_action(pf_t *pf, pf_action_model_fn_t action_fn, void *action_data);

// Update the filter with some new sensor observation
void pf_update_sensor(pf_t *pf, pf_sensor_model_fn_t sensor_fn, void *sensor_data);

// Resample the distribution
void pf_update_resample(pf_t *pf);
```

3.2 sensors



a) amcl_sencor定义了基类,其成员函数都为虚函数,主要有如下两个,为两个派生类amcl_laser,amcl_odom提供统一接口

```
// Update the filter based on the action model.  Returns true if the filter
// has been updated.
public: virtual bool UpdateAction(pf_t *pf, AMCLSensorData *data);

// Update the filter based on the sensor model.  Returns true if the
```

```
// filter has been updated.
public: virtual bool UpdateSensor(pf_t *pf, AMCLSensorData *data);
```

b) amcl_laser定义了激光数据类型,三种观测更新模型(详细见<<概率机器人>>),具体实现了UpdateSensor,用于计算粒子权值

```
//amcl_laser.h
// Laser sensor data
class AMCLLaserData : public AMCLSensorData
{
  public:
    AMCLLaserData () {ranges=NULL;};
    virtual ~AMCLLaserData() {delete [] ranges;};
  // Laser range data (range, bearing tuples)
  public: int range_count;
  public: double range_max;
  public: double (*ranges)[2];
};
...
//amcl_laser.cpp
// Apply the laser sensor model
bool AMCLLaser::UpdateSensor(pf_t *pf, AMCLSensorData *data)
{
  if (this->max_beams < 2)
    return false;

  // Apply the laser sensor model
  if(this->model_type == LASER_MODEL_BEAM)
    pf_update_sensor(pf, (pf_sensor_model_fn_t) BeamModel, data);
  else if(this->model_type == LASER_MODEL_LIKELIHOOD_FIELD)
    pf_update_sensor(pf, (pf_sensor_model_fn_t) LikelihoodFieldModel, data);
  else if(this->model_type == LASER_MODEL_LIKELIHOOD_FIELD_PROB)
    pf_update_sensor(pf, (pf_sensor_model_fn_t) LikelihoodFieldModelProb, data);
  else
    pf_update_sensor(pf, (pf_sensor_model_fn_t) BeamModel, data);

  return true;
}
...
```

c)amcl_odom具体实现了基类定义的UpdateAction函数,用于根据运动更新粒子,定义了两种运动模型,差分和全向

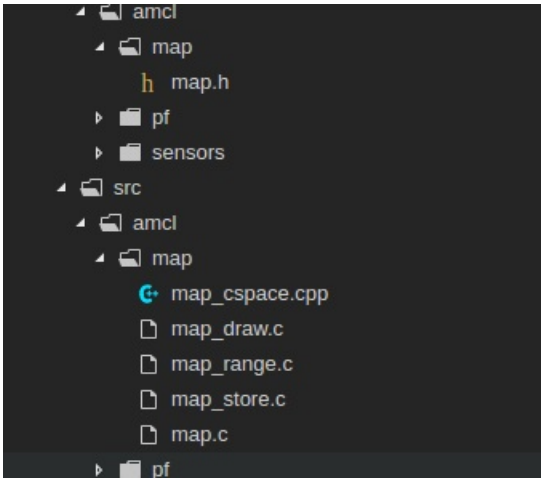
```
  public: void SetModelDiff(double alpha1,
                            double alpha2,
                            double alpha3,
                            double alpha4);

  public: void SetModelOmni(double alpha1,
                            double alpha2,
                            double alpha3,
                            double alpha4,
                            double alpha5);

  public: void SetModel( odom_model_t type,
                         double alpha1,
                         double alpha2,
                         double alpha3,
                         double alpha4,
                         double alpha5 = 0 );

  // Update the filter based on the action model.  Returns true if the filter
  // has been updated.
  public: virtual bool UpdateAction(pf_t *pf, AMCLSensorData *data);
```

3.3 map



a) map中主要定义了概率栅格地图的数据表示,如下所示

```
// Description for a single map cell.
typedef struct
{
  int occ_state;// Occupancy state (-1 = free, 0 = unknown, +1 = occ)
  double occ_dist;// Distance to the nearest occupied cell
} map_cell_t;

// Description for a map
typedef struct
{
  double origin_x, origin_y; // Map origin; the map is a viewport onto a conceptual larger map.
  double scale;      // Map scale (m/cell)
  int size_x, size_y;  // Map dimensions (number of cells)
  map_cell_t *cells;  // The map data, stored as a grid
  double max_occ_dist;  // Max distance at which we care about obstacles,
                        //for constructing likelihood field
} map_t;
```

b) 此外还有四个源文件map_cspace.cpp,map_draw.c,map_range.c,map_store.c 不是主要内容(还没仔细读),暂不深究

4.amcl_node.cpp解读

4.1 类AmckNode的主要内容

public:除构造函数和析构函数外,只有三个函数

```
  void runFromBag(const std::string &in_bag_fn);//根据信息记录包来运行amcl,
                                                 //实际此函 数是把信息记录不断的发布出去
  int process();  //这个好像只声明了但没定义
  void savePoseToServer();    //把位姿信息保存到参数服务器
```

private:定义了很多变量,tf ,激光,里程,粒子滤波参数和变量等等,订阅者和发布者以及一些回调函数

4.2 main函数

```
int main(int argc, char** argv)
{
  ros::init(argc, argv, "amcl");
  ros::NodeHandle nh;
  //监听到程序是否接收到退出命令,或者中断命令等处理
  // Override default sigint handler
  signal(SIGINT, sigintHandler);
  // Make our node available to sigintHandler
  amcl_node_ptr.reset(new AmclNode());//使boost::shared_ptr指向一个新的对象

  if (argc == 1)
  {
    // run using ROS input
    ros::spin();
```

```
  }
  else if ((argc == 3) && (std::string(argv[1]) == "--run-from-bag"))
  {
    amcl_node_ptr->runFromBag(argv[2]);
  }
  // Without this, our boost locks are not shut down nicely
  amcl_node_ptr.reset();//销毁boost::shared_ptr指向的对象
  // To quote Morgan, Hooray!
  return(0);
}
```

关键一句amcl_node_ptr.reset(new AmclNode());主要实现在于构造函数AmclNode()

4.3 构造函数AmclNode()

主要包含如下内容:

```
//设置粒子滤波参数,运动模型参数,观测模型参数等各种
//消息发布:
 1.pose_pub_ = nh_.advertise<geometry_msgs::PoseWithCovarianceStamped>("amcl_pose", 2, true);
       (位姿,均值方差表示的)
   2.particlecloud_pub_ = nh_.advertise<geometry_msgs::PoseArray>("particlecloud", 2, true);
        (所有的粒子)
//服务:
   1.global_localization,(globalLocalizationCallback)
       这里的globalLocalization就是在地图范围内产生均匀分布的粒子
   2.request_nomotion_update,(nomotionUpdateCallback)
        就是设置个标志,使amcl在无运动时也能够更新粒子
   3.set_map,(setMapCallback)调用如下两个函数
      handleMapMessage(req.map);//进行地图转换 ，记录free space ，
      以及初始化pf_t 结构体,实例化运动模型(odom)和观测模型(laser)
      handleInitialPoseMessage(req.initial_pose);
      根据接收的初始位姿消息,在该位姿附近高斯采样重新生成粒子集
//消息订阅:
   1.laser_scan_filter_->registerCallback(boost::bind(&AmclNode::laserReceived,this, _1));
       回调函数laserReceived是粒子滤波主要过程,根据激光扫描数据更新粒子
   2.initial_pose_sub_ = nh_.subscribe("initialpose", 2, &AmclNode::initialPoseReceived, this);
        回调函数initialPoseReceived中实际调用handleInitialPoseMessage,在接收到的初始位姿附近采样生成  粒子集
//动态调参
      dsrv_ = new dynamic_reconfigure::Server<amcl::AMCLConfig>(ros::NodeHandle("~"));
      ...CallbackType cb = boost::bind(&AmclNode::reconfigureCB, this, _1, _2);
      dsrv_->setCallback(cb);
      动态配置参数服务,重新设置粒子滤波参数,设置运动模型参数,设置观测模型参数,订阅消息等,
      就是AmclNode构造函数里做的事情
```

以上的最关键内容在回调函数laserReceived()

4.4 回调函数laserReceived()

- a： 获取laser对应于baselink的坐标
- b：获取baselink对应于odom的坐标
- c：根据里程计的变化值+高斯噪音 更新 pf_t中samples的内里程计值（运动模型） odom->updateAction()
- d：根据当前雷达数据更新各里程计对应的权值weights
  laser_[laser_index]->updateSensor（ ）
- e：得到滤波结果后,分别在话题/amcl_pose和/ particlecloud上发布位姿和粒子集

4.5 amcl_node的重要内容大体如上,总结一下它的几个主要过程

- a： 构造时初始化,从参数服务器中获取数据初始化各类参数;(接收地图设置,gui显示发布频率,保存位姿到参数服务器频率,laser测距范围及其概率模型参数,odom概率模型参数,粒子滤波及kld重采样参数,从参数服务器获取初始位姿,然后初始化了订阅者,发布者,服务)
- b：地图加载,两种方式(1.订阅/map话题2.请求服务得到地图),得到地图后也有个初始化过程(将消息类型的地图转换为定义的map类数据,统计free状态的栅格索引,从参数服务器获取位姿信息,并初始化粒子滤波器pf_,初始化odom模型参数,初始化laser模型参数)
- c：粒子滤波,订阅laser_scan的回调函数中处理,得到结果后发布位姿和粒子集
- d：initialpose的回调,接收到初始位姿消息后,融入最新的里程改变,然后在该位姿附近重新生成粒子集 剩下的是提供的服务及动态参数配置,提供给外部控制接口,如第三节所述

参考资料:

- http://wiki.ros.org/navigation
- https://zhuanlan.zhihu.com/p/28137335
- https://www.ncnynl.com/archives/201708/1911.html
- https://blog.csdn.net/wangqiang319670/article/category/7418854Probabilistic Robotics
- Learning ROS for Robotics Programming Second Edition
- Probabilistic Robotics