

《ROS机器人开发技术》

**课程名称:ROS机器人开发技术**

**教师姓名:XXX**

**提交时间:2018年7月x日**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 课程信息 | 课程名称 | | 名称 | | 章数 | 节数 | | 课程类型 |
| 《ROS机器人开发技术》 | | Navigation知识学习 | | 九 | 一 | | 授课(√ )  实训( ) |
| 教师 |  | | | 时长 |  | |
| 参考文献 | 1. ROS机器人程序设计（原书第二版） [西班牙]恩里克.费尔南德斯等著 ，刘锦涛 等译 | | | | | | | |
| 教学  目的  要求 | 理解路径导航的基本框架与原理 | | | | | | | |
| 教学  重点  难点 | 重难点 | | | PPT页面 | | | 时间分配 | |
| 重点 | 1. Navigation介绍 | | 1~4页 | | | 15分钟 | |
| 1. tf转换 | | 5~6页 | | | 10分钟 | |
| 1. move\_base介绍 | | 7~9页 | | | 10分钟 | |
| 1. costmap理解 | | 10~14页 | | | 20分钟 | |
| 1. mapserver | | 15~17页 | | | 15分钟 | |
| 1. AMCL | | 18~20页 | | | 15分钟 | |
| 难点 | 1. costmap | | 10~14页 | | | 20分钟 | |
| 1. AMCL | | 18~20页 | | | 15分钟 | |
| 教学方法 | 本授课以课堂讲授为主，与课堂演示方式相结合 | | | | | | | |

|  |  |  |
| --- | --- | --- |
| 教学内容 | 操作演示 | 知识点 |
| **PPT第1页：**  同学们好，我们进行Navigation章节授课部分的学习 |  |  |
| **PPT第2页:**  我们知道机器人在未知环境中需要使用激光传感器（或者是深度传感器）进行地图建模，然后根据构建的地图进行导航与定位，在ROS中可以利用以下三个功能包集实现自主导航。  Gmapping:根据激光数据（或者是深度数据模拟的激光数据）构建地图  Moves\_base:根据参照消息进行路径规划，使机器人到达指定位置  Amcl:根据已有地图进行定位  根据这三个主体部分，我把Navigation分为了五个小节，分别是：Navigation Stack、Move\_base、costmap、map\_server、AMCL |  |  |
| **PPT第3页:**  首先，我们先进行Navigation stack部分的学习 |  |  |
| **PPT第4页:**  我们先来看机器人运动控制系统的架构  关于机器人运动控制系统架构，从低到高依次是：motor controllers anddrivers-> ROS base controller ->Frame-Base Motion（move\_base）->Frame-Base Motion（gmapping + amcl）->Semantic Goals。总结起来如图所示。  其实，可以理解为以下几个问题：电机如何控制？机器人怎么跟上位机连接，如何传送数据？如何进行路径规划？如何建立环境地图？如何设定运动的目标？我们可以把这些问题分为三个层次。  **1.最底层：**机器人本身的电机驱动和控制部分，驱动器接收的是电机左右轮期望速度，根据期望速度对左右轮分别进行 PID控速。同时，定时采样电机码盘值，并转化为左右轮速度值上传给电机控制器。控制器将左右轮的速度值，结合机器人的运动学模型，用航迹推演法推算出机器人当前的速度，包括线速度和角速度。控制器也可以接受机器人的控制指令（cmd\_vel），将速度解算为左右轮的期望速度，从而给电机的驱动部分，控制电机转动。它的输入是控制速度值，输出是电机的转速。  **2.中间通信层：**电脑端和机器人平台的控制通信。电脑将发出的速度指令传给平台的控制部分，以及电脑端接受控制部分的当前速度，然后发布/odom topic，让其他节点订阅，从而进行机器人位置估计。  **3.决策层：**与导航有关，建立地图和实现定位，然后用move\_base根据你发布的传感器信息做出路径规划，输出机器人的速度和转向速度。这部分主要与ROS中的Navigation stack有关，可以调用相应的package，如amcl，gmapping，move\_base，map\_server等。它的输入是各种传感器的信息和地图，输出是速度cmd\_vel。 |  |  |
| **PPT第5页:**  我们这节要讲的就是决策层中的导航问题，先来看一下一张很经典的图：  整体导航包框架如图所示：  其中白色框内的是ROS已经为我们准备好的必须使用的组件，灰色框内的是ROS中可选的组件，蓝色的是用户需要提供的机器人平台上的组件。  另外，我们可以清楚得看见navigation的输入：里程计odometry， 激光雷达或者rgbd-camera的信息sensor\_topics，还有已知的先验地图（map\_server）（可选），坐标系变换信息，输出就是cmd\_vel速度。 |  |  |
| **PPT第6页:**  下面对这张图的每一部分进行逐一介绍：  传感器坐标转换sensor transform（tf）  因为我们使用的机器人的控制中心不一定是在传感器上，所以要把传感器的数据转换成在控制中心上的坐标信息。如图所示，传感器获取的数据是在base\_laser的坐标系统中的，但是我们控制的时候是以base\_link为中心，所以要根据两者的位置关系进行坐标的变换。变换的过程不需要我们自己处理，只需要将base\_laser和base\_link两者之间的位置关系告诉tf，就可以自动转换了。 |  |  |
| **PPT第7页：**  我们来看一下传感器数据输入与里程计数据输入。  里程计信息如图所示，下面我们来解读一下：  图中：nav\_msgs/Odometry提供了机器人从frame\_id坐标系到child\_frame\_id坐标系的相对位置，它还通过geometry\_msgs/Pose消息提供机器人位姿信息，geometry\_msgs/Twists消息表示速度信息。位姿信息中包含着两个数据结构，一个表示欧拉坐标系中的位置，另一个则使用四元向量描述机器人的方向，也就是机器人的角位移。速度信息包含了线速度和角速度。  对于传感器数据输入来说，move\_base模块只接受激光传感器数据与点云数据。 |  |  |
| **PPT第8页：**  有了这些传感器数据后，我们利用这些数据进行机器人导航呢？这就需要最重要的模块：move\_base。 |  |  |
| **PPT第9页:**  如图所示，表示了move\_base的组成部分。  中间这个node就是move\_base，他要正常工作，首先需要设置好三个接口，分别叫做….， 在ROS里，实际的路径规划一般就是分为这么三部分，一部分是全局规划，负责轨迹的大方向，用户来躲避静态障碍物，一部分是局部路径规划，负责具体的运动细节，用来躲避移动中的障碍物，也就是动态避障。  做好了这两部分路径规划，机器人就可以正常的运动了，但有些时候，机器人也会陷入一些运动情况，比如碰撞到障碍物，那它就会有一些恢复动作，recovery behavior这些恢复动作就属于异常动作处理的方法。  那针对这三个接口，每一个ROS都给我们提供了一些插件，供用户来选择比如局部路径规划，ROS提供了两个插件，  这每一个插件，都是不同算法的实现，其实就是不同的package，在navigation里你都能找到这些package  比如说move\_base的插件Dwa\_local\_planner就是base\_local\_planner的重写版本，对动态窗口的速度采样更好  我们可以看出，move\_base是navigation的逻辑核心(nav\_core)  nav\_core这个package定义了这三个接口，其实就是定义了三个接口类，然后各自继承和实现了这些功能。 |  |  |
| **PPT第10页:**  我们进一步理解一下这个nav\_core。  要理解ROS navigation 最重要的部分是nav\_core:  这个包里面就包含了global\_planner ， local\_planner 与 recovery\_behavior的基类的头文件。  在ROS navigation中，move\_base提供的是框架，在move\_base 中是通过nav\_core中规定的planner与 recovery\_behavior的基类的接口进行调用。与具体的实现方法隔离开来。而具体采用的方法由pluginlib 根据不同参数导入。这样的实现方法使得navigation的可定制性大大增加。像base\_local\_planner 中就实现了两种局部路径规划方法，global\_planner 实现了A\* 与Dijkstra 两种方法，在navigation\_experimental 中还有更多这样的实现。这赋予了这个框架很大的灵活性。通过不同的配置方法可以让navigation适应很多不同的任务。 |  |  |
| **PPT第11页:**  下面讲解一下move\_base中的costmap。  在讲解costmap之前先介绍一下costmap的基本概念，只要大家把基本概念弄清楚之后，才能根据自己的实际应用环境对自己的代码做出修改。  无论是激光雷达还是如kinect 或xtion pro深度相机作为传感器跑出的2D或3D SLAM地图，都不能直接用于实际的导航，必须将地图转化为costmap(代价地图),ROS的costmap通常采用grid(网格)形式。Costmap是机器人收集传感器信息建立和更新的二维或三维地图，可以从下图简要了解。 |  |  |
| **PPT第12页:**  红色cell（图中红色蓝色区域都是一系列cell堆叠出来的）代表的是代价地图中的障碍，  蓝色cell代表的是通过机器人内切圆半径计算的障碍物膨胀红色多边形代表的是机器人footprint(机器人轮廓的垂直投影)。  为了使机器人不碰到障碍物，机器人的footprint绝对不允许与红色cell相交，机器人的中心绝对不允许与蓝色cell相交。  栅格地图一个栅格占1个字节，也就是八位，可以存0-255中数据，也就是每个cell cost（网格的值）从0-255我们只需要三种情况：Occupied被占用（有障碍）, Free自由区域（无障碍）, Unknown Space未知区域。  当投影到代价地图时候，每种状态被赋一个特定的代价值，也就是说每个cell的cost值是由这个cell对应的各层中对应的cell的状态进行加权得到的。 |  |  |
| PPT第13页: 我们看一下机器人在costmap中的2d模型：  在图里可以看到左下角有两个淡蓝色同心圆，一个机器人的轮廓外切圆和一个机器人内切圆，机器人在costmap里就能够简化成为这两个圆。根据机器人中心至边界或者障碍物的距离和两个同心圆半径比较来判断是否碰撞。  lethal：值为254，传感器扫描到的实际障碍转化成map中的lethal网格。  inscribed ：内部半径，如果网格和lethal网格距离小于等于该半径，则为inscribed网格，值为253。取robot几何型所能容纳最大的圆的半径。  被标记为253～254的网格是障碍网格，也就是lethal和inscribed网格是障碍网格。当robot的边界覆盖到这两种网格时，一定会发生碰撞。  possibly circumscribed：膨胀半径，以robot的中心为园点旋转一周所能覆盖的最大圆半径，如果所在网格距离lethal小于等于膨胀半径，大于内部半径，则为possibly circumscribed网格。值为128～252，取决与robot的运动方向和轨迹，还有算法。当robot的边界覆盖到这种网格时，可能会发生碰撞。  free：值为0,说明没有任何信息可以阻碍机器人运行到这里。  unkonw：没有任何信息，传感器还没有扫描到该区域，都是障碍网格  1～127：不会发生碰撞的区域，这里的值赋值取决于与lethal的距离，还有用户自定义的算法 |  |  |
| PPT第14页: 下面根据这个图来介绍一下costmap\_2d中计算cost的方法。假设，机器人内切半径为0.5m，外切半径为0.7m，当激光返回障碍距离在机器人中心附近，叫致命障碍，机器人一定能碰到障碍物，比如说0m,直接贴着机器人，或会取小于栅格的边长，比如小于0.1m范围内,则这个栅格值就设为254；当返回来的数值在0.1-0.5m之间，就设253；当在0.5-0.7之间，则可以设128，或者在252-128找个比例值（程序中可以控制），属于受限区域，可能发生碰撞，是否碰撞，取决于机器人的姿态；当0.7-膨胀半径之间，设1-127之间的映射值，不会发生碰撞；当大于膨胀距离，则设为0，称为freespace。Unknown -- 意味着给定的单元没有相应的信息。我们看坐标系中较细的红色光滑曲线就是cost曲线，x是距离机器人footprint的圆心距离，而y是cost值，cost随着x的增大而减小距离，当x>=内切圆半径时开始有值；当x=0时，y=254；当x=resolution/2时，cost=253；（图中右上角的较粗的台阶状红线是单元格的边线，或者认为是障碍物（单元格化后）的边线） |  |  |
| PPT第15页: Costmap中的cost设置完成后，我们如何根据实时构建的地图来更新costmap呢？  从Hydro发布版本开始， 用来写数据到代价地图的底层方法已经完全可配置了。 Costmap由多层组成，每种功能放置一层中。 例如左所示，静态地图是一层，障碍物是另一层。 缺省情况下，障碍物层维护的是3D信息，3D障碍物数据可以让层更加灵活的标记和清除障碍物。例如在costmap\_2d包中，StaticLayer（静态地图层）是第一层，ObstacleLayer（障碍物层）是第二层，InflationLayer（膨胀层）是第三层。这三层组合成了master map（最终的costmap），供给路线规划模块使用。  Costmap的更新在mapUpdateLoop线程中实现，此线程分为两个阶段：  （阶段一）UpdateBounds：这个阶段会更新每个Layer的更新区域，这样在每个运行周期内减少了数据拷贝的操作时间。StaticLayer的Static map只在第一次做更新，Bounds 范围是整张Map的大小，而且在UpdateBounds过程中没有对Static Map层的数据做过任何的更新）。ObstacleLayer在这个阶段主要的操作是更新Obstacles Map层的数据，然后更新Bounds）。InflationLayer则保持上一次的Bounds。  （阶段二）UpdateCosts：这个阶段将各层数据逐一拷贝到Master Map，可以通过上图观察Master Map的生成流程。  在（a）中，初始有三个Layer和Master costmap,Static Layer和Obstacles Layer维护它们自己的地图，而inflation Layer并没有。为了更新costmap,算法首先在各层上调用自己的UpdateBounds方法（b）。为了决定新的bounds,Obstacles Layer利用新的传感器数据更新它的costmap。然后每个层轮流用UpdateCosts方法更新Master costmap的某个区域,从Static Layer开始（c），然后是Obstacles Layer(d)，最后是inflation Layer(e)。 |  |  |
| PPT第16页: 在很多情况下，我们已经建立好了需要的地图，不再需要SLAM来建图，  我们直接从一组地图的数据中发布出来，就可以。  这时候要用到的工具就是mapserver  Mapserver：提供地图 直接发布/map topic  map\_server package有两个节点：  1.map\_server node：读取地图信息，并作为ROS service 为其余节点提供地图数据  2.map\_saver node：保存现有扫描到的地图信息 |  |  |
| PPT第17页: Map\_server这个node提供已知的地图信息，或者叫静态地图信息发布的topic一个是具体的地图的图像，一个是地图的描述信息。  这个service的类型，它的request不用填，直接会response给你一张当前的地图。  然后需要设置一个参数，就是这个map topic它的header里面的frame，也就是他所在tf里面哪个位置，并设置坐标系的名称。  这个一般默认就是map，也不用改。  Map\_saver获取地图数据，并把它写到map.pgm和map.yaml |  |  |
| **PPT第18页：**  mapserver发布的是两张地图：  一个是一张地图的照片，  一个是地图的描述文件，是带有障碍物信息的OccupancyGrid  occ = (255 - color\_avg) / 255.0 color avg表示RGB（红绿蓝）三个通道的平均值 |  |  |
| PPT第19页: 我们来看一下AMCL wiki上的官方解释：    Amcl is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach (as described by Dieter Fox), which uses a particle filter to track the pose of a robot against a known map.  以上是官网的介绍，说白了就是2D的概率定位系统，输入激光雷达数据、里程计数据，输出机器人在地图中的位姿。用的是自适应蒙特卡洛定位方法，这个方法是在已知地图中使用粒子滤波方法得到位姿的 |  |  |
| PPT第20页: 在有了实际地图之后，机器人进行动作首先需要确定自己在地图中的位置。定位的方法有很多，比如马尔可夫定位、EKF定位、栅格定位等，本课程着重讲一下AMCL定位方法。首先理解蒙特卡洛方法:所要求解的问题是某种事件出现的概率，或者是某个随机变量的期望值时，它们可以通过某种“试验”的方法，得到这种事件出现的频率，或者这个随机变数的平均值，并用它们作为问题的解。 |  |  |
| **PPT第21页:**  蒙特卡罗方法的基本原理及思想如下  所要求解的问题是某种事件出现的 概率，或者是某个随机变量的期望值时，它们可以通过某种“试验”的方法，得到这种事件出现的频率，或者这个随机变数的平均值，并用它们作为问题的解.这就是蒙特卡罗方法的基本思想。  我们使用激光传感器，它能够测量机器人各个方向和最近障碍物之间的距离。在每一个时间点，机器人都会获得激光传感器的测量值。如图a，绿色三角形是机器人，红色的线是激光束，黄色的格子是机器人在该激光方向上检测到的最近的障碍物。  图b是占据栅格地图（Occupancy Grid Map）。比如，下面的地图中，浅色（白色）的格子表示障碍物，深色（黑色）的格子表示空白位置。  那么，在这个时间点，我们要做的就是把机器人放到图c中去，使得激光传感器的读数尽可能符合地图信息，这样，对于一个时间点的定位问题就变成了求解最优函数的问题了。然而这个最优化函数太难求解了（坐标和角度都是连续变化的，而地图是一个一个格子的数值）。  我们需要注意到两点。第一，对于给定的机器人位置信息，我们可以很轻松地计算出和地图的吻合程度；第二，相邻两个时间点机器人位置的变化不会太大。基于这两点，我们引出蒙特卡罗定位法。这个算法的核心思想是用高斯分布描述机器人位置信息的噪音，用大量的粒子来描述机器人可能的位置。  具体来说，假如估测的机器人位置信息为[x,y,θ]（x和y表示坐标，θ表示机器人朝向），我们会记录机器人的位置信息符合μ=[x,y,θ]的多元高斯分布。在算法实现中，我们用高斯分布采样出的个粒子来表示机器人的位置。如下左图所示的单元高斯模型，下面蓝色的点是采样的粒子，上面是对应的高斯分布的模型；如图d所示是二元高斯模型采样的情况。 |  |  |
| **PPT第22页:**  下面我们来研究一下AMCL算法。  AMCL(adaptive Monte Carlo Localization)自适应蒙特卡洛定位 ，是源于MCL算法的一种升华，一种提高。那么为什么要从MCL上升至AMCL呢？  MCL算法，蒙特卡洛定位适用于局部定位和全局定位两类问题。它解决了全局定位问题，但无法从机器人绑架或全局定位失败中恢复过来。当位置被获取时，其他地方的不正确粒子会逐渐消失。在某种程度上，粒子只能“生存”在一个单一的姿势附近，如果这个姿势恰好不正确，算法就无法恢复。  MCL算法和AMCL算法的区别:AMCL算法增加了短期和长期的指数滤波器衰减率αslow，αfast，换句话说MCL中αslow，αfast为0，AMCL中的不为0。  具体公式可参考wiki百科上AMCL的词条。  我们来看一下简化的amcl定位原理：  （读PPT） |  |  |
| PPT第23页: 那么，AMCL是如何运行的呢，我们来看一下定位过程。  过程可分为两个步骤：全局定位和重定位。  a：  是在进行第一次标记检测时，所有粒子随机撒点  b-d:  多次检测后，粒子紧紧环绕早真实的机器人周围，就像d所示，并且短期和长期测量似然平均都将增加。在这个定位阶段，机器人只是跟踪其位置，观察似然相当高，并且只偶尔增加小数量的随机粒子。  e  当将机器人放置在其他位置时，测量概率下降。  在新的位置，第一次标记检测还没有触发任何附加粒子，因为平滑估计Wfast仍然很高  f、g  在新位置进行了几次标记检测后，Wfast比Wslow下降的快，并有更多的随机粒子被加进来  h  最后机器人定位成功 |  |  |
| PPT第24页: 具体到ROS 导航包里，解读一下AMCL的输入输出。  输入信息  （1）scan （sensor\_msgs / LaserScan） 激光扫描数据。  （2））tf（tf / tfMessage） tf转换。  （3）initialpose（geometry\_msgs / PoseWithCovarianceStamped） 用于（重新）初始化粒子滤波器的平均值和协方差。  （4）map（nav\_msgs / OccupancyGrid） 当设置了use\_map\_topic参数时，AMCL订阅此主题以检索用于基于激光的本地化的映射。  输出信息：  （1）amcl\_pose（geometry\_msgs / PoseWithCovarianceStamped） 机器人在地图上的估计姿态，是以协方差的形式表示。  （2）particlecloud（geometry\_msgs / PoseArray） 由过滤器维护的姿态估计集合。  （3）tf（tf / tfMessage） 从odom发布的tf变换（可通过〜 odom\_frame\_id参数被重新映射）,以映射。 |  |  |
| PPT第25页: 那么，在导航中，我们既然有了里程计，那么我们为什么要使用AMCL进行定位呢？  如图所示，如果里程计没有误差的情况下，我们可以直接使用里程计信息（上半图）推算出机器人（base\_frame）相对里程计坐标系的位置。但现实情况，里程计存在漂移以及无法忽略的累计误差，所以AMCL采用下半图的方法，即先根据里程计信息初步定位base\_frame，然后通过测量模型得到base\_frame相对于map\_frame（全局地图坐标系），也就知道了机器人在地图中的位姿。（注意，这里虽然估计的是base到map的转换，但最后发布的是map到odom的转换，可以理解为里程计的漂移。）  具体算法通过实训例子来体现定位效果，这里不展开叙述。  到这里，navigation授课的部分讲解结束了，请大家接下来根据实训部分的实例对导航包做进一步的理解，谢谢。 |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  | | |
|  | | |