**算法说明**

1. 语音识别模块

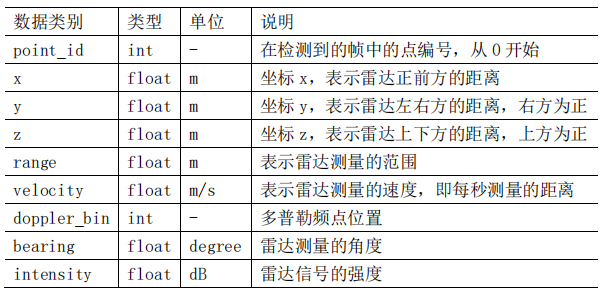
语音识别使用到了一个专门为嵌入式环境设计的离线语音库Maix-Speech，详情请见[GitHub - sipeed/Maix-Speech: Maix Speech AI lib, a fast and small speech lib running on embedded devices, including ASR, chat, TTS etc.](https://github.com/sipeed/Maix-Speech)。

其中，原算法提供了四种尺寸的声学模型、三种尺寸的语言模型和四种功能。我们选择了关键词识别的功能，并且经过性能测试，最终的声学模型为am\_7332，语言模型为lmS。同时，我们修改了部分核心代码，将main.cpp中的关键词修改为我们预设的关键词，并且设置了概率限值，增加了mqtt通信功能，即当识别概率超过概率限值程序就会根据识别的内容向mqtt服务器下对应话题发布相应内容。

1. 姿态检测模块

训练数据集的制作

毫米波雷达所采集的原始数据格式如下图所示，本神经网络采用雷达所检测到的坐标、速度信息组成的四元组作为动作特征。此外，考虑到人体姿态动作具有时序性，因此使用个四元组点云数据所组成的矩阵作为网络的固定输入，以引入具有时序特征的信息。

表1 原始数据格式

最终经过处理得到24404个数据，其中fall、stand、sit、walk四个标签各有个。划分数据集的80%作为训练集，20%作为测试集。

算法1：姿态检测神经网络训练过程

Input：预处理后的数据集，表示标签

Result：人体姿态检测网络

1 初始化

2 设置损失函数、优化器、学习率、衰减率，和

3 for  do

4 把随机切分成多个大小为的小批量

5 for  do

6 批量输入样本进行前向传播，得到预测输出

7 根据标签计算

8 反向传播并更新

9 返回训练好的网络模型

1. 机器人联动服务模块

机器人SLAM建图使用Gmapping算法；自主定位使用自适应蒙特卡洛定位（AMCL）算法；导航框架使用Navigation Stack，其中局部轨迹规划器使用时间弹性带（TEB）算法。以上算法均来源于ROS官网，详见<http://wiki.ros.org/>。

导航框架中的全局路径规划器使用改良A\*算法，是对ROS官网的A\*算法的改进。传统的A\*算法能够快速找到最短路径，但机器人的最优移动应该平滑且连续，这样才能在最短时间内到达目标点，算法没有考虑机器人的运动特性，因此规划出的全局路径对机器人而言并非最优。根据算法特性，若地图上有障碍物，生成的路径会产生许多角度较大的转折点，若路径存在过多转向需求，也会导致机器人相关传感器的累积误差增大，不利于自主定位的准确性。综上所述，改良的思路为：一、抑止多余转向；二、平滑路径。

抑止多余转向的方法为在评估函数中引入角度约束函数，使机器人应该为转向动作付出相应代价。角度约束函数如公式（3-1）所示，即当前栅格的角度约束代价为

（3-1）

当前栅格相对前一栅格的角度变化的绝对值乘上对应权重，权重越大，角度约束力就越强，机器人更不容易产生转向动作，但一味抑制所有转向并非改良目的，值需要通过实验来确定，以完成抑制多余转向的目的。此时评估函数变为公式（3-2）

（3-2）

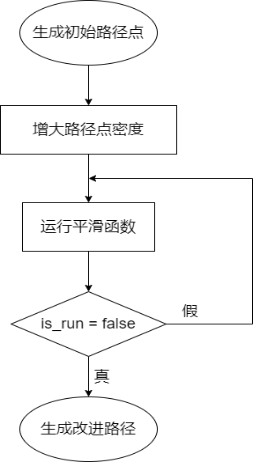
平滑路径的思路是添加点调整折线，使其接近平滑，方法为增大路径点密度并不断调整超出角度限制的转向路径点位置，具体流程如图3-1所示。

图3-1 平滑路径流程

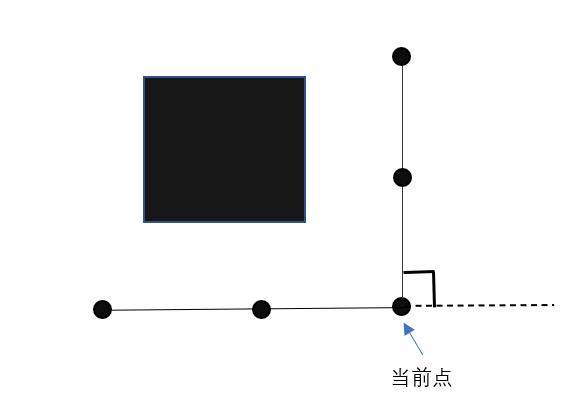
在A\*算法生成初始节点后，首先以每两点之间插入一点的方式增加点密度，然后运行平滑函数。平滑函数的逻辑为逐点检查转向，当前点的转向为前一点与当前点的连线的延长线和当前点与后一点的连线的夹角，如图3-2，若转向大于设定角度阈值，则将该点坐标改为，直到检查到倒数第二个路径点，若该过程出现了路径点坐标改动，则令标记is\_run = true。

图3-2 路径点的转向角

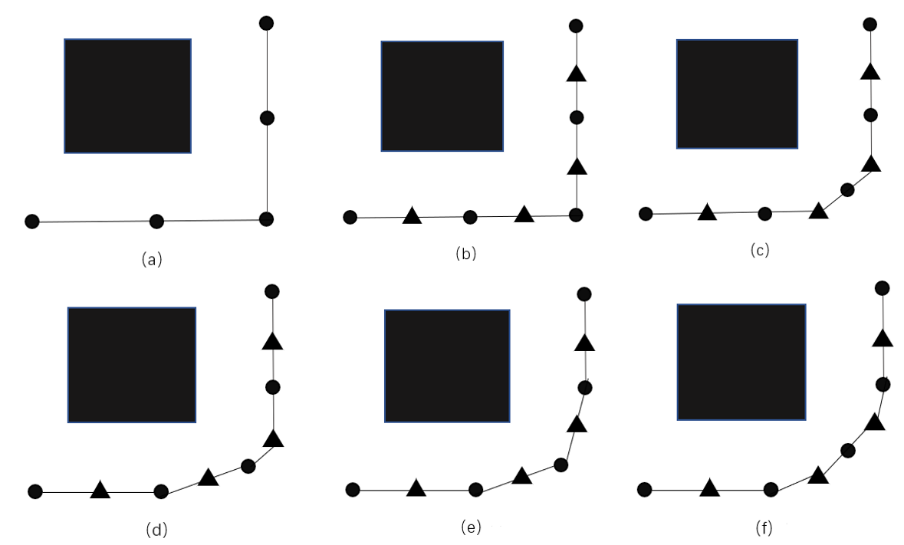
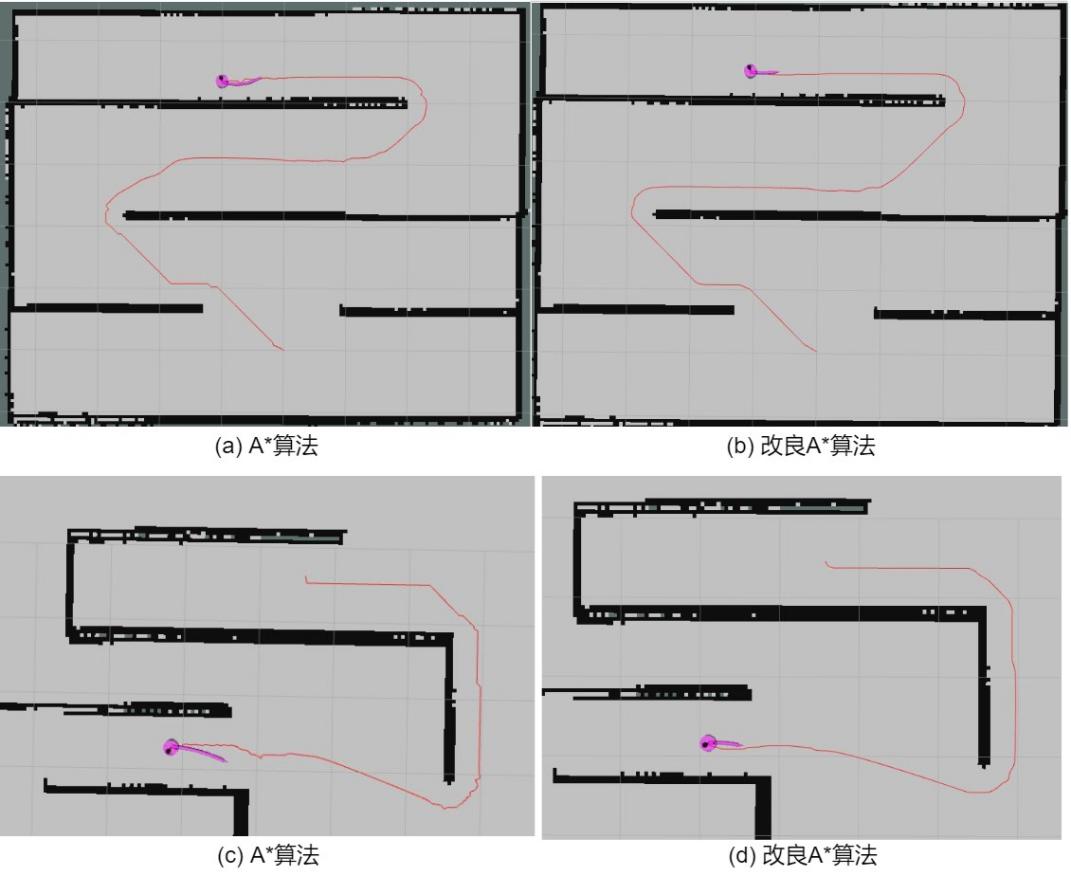
平滑函数运行完后，若标记is\_run为false，代表路径点的转向均在角度阈值以下，路径调整完成，否则需再次运行平滑函数。该平滑路径的效果示例如图3-3，黑色矩形为障碍物，黑色圆点表示初始路径点，黑色三角形表示增加的路径点，图3-3(a)是初始路径，(b)~(e)是平滑过程，(f)是最终平滑的路径，可以看出，该方法具备良好的路径平滑效果。

图3-3 平滑路径效果

我们搭建了两个不同的实验场景对A\*算法和改良A\*算法进行测试对比，图3-4是效果对比图。可以看到，改良A\*算法规划的全局路径相较于A\*算法更加平滑且连续。

图3-4 A\*算法和改良A\*算法效果对比

针对这两个实验场景，我们分别从相同起点出发做了5次对比实验，统计其到达相同目标点的平均时间耗费如表3-1。可以看出，改良A\*算法导航到目标点所需的时间有所减少。

表3-1 A\*算法和改良A\*算法导航平均时间耗费

|  |  |  |
| --- | --- | --- |
|  | 实验场景1 | 实验场景2 |
| A\*算法 | 1:58.47 | 1:34.92 |
| 改良A\*算法 | 1:54.58 | 1:27.22 |