第二十课 DWA局部路径规划算法

1. 概述

机器人局部路径规划方法有很多，ROS中主要采用的是动态窗口法。动态窗口法主要是在速度(v,w)空间中 采样多组速度，并模拟机器人在这些速度下一定时间（sim_period)内的轨迹。 在得到多组轨迹以后，对这些轨迹进行评价，选取最优轨迹对应的速度来驱动机器人运动。该算法突出点在于动态窗口这个名词，它的含义是依据移动机器人的加减速性能限定速度采样空间在一个可行的动态范围内。

2. 机器人的运动模型（简化）

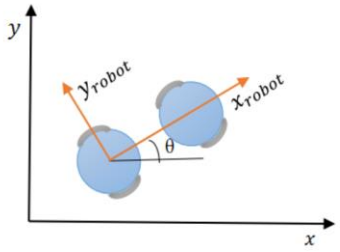
在动态窗口算法中，要模拟机器人的轨迹，需要知道机器人的运动模型。它采用的是两轮移动机器人，差分移动机器人的轨迹是一段一段的圆弧或者直线(旋转速度为0 时)，一对(v_t,w_t)就代表一个圆弧轨迹。

具体推导如下：假设机器人不是全向移动的，即不能纵向移动，只能前进和旋转。计算机器人轨迹时，先考虑两个相邻时刻，如下图所示。为简单起见，由于机器人相邻时刻（传感器融合周期内运动距离短，因此可以将两相邻点之间的运动轨迹看成直线，即沿机器人坐标系 x 轴移动了$v_t * \Delta t$。只需将该段距离分别投影在世界坐标系x 轴和 y 轴 上就能得到 t-1 时刻相对于 t 时刻机器人在世界坐标系中坐标移动的位 移Δx和Δy。

$$\Delta x = v\Delta t\cos(\theta_t)$$
$$\Delta y = v\Delta t\sin(\theta_t)$$

以此类推，如果你想推算一段时间内的轨迹，只需要将这段时间位移增量累计求和九可以了：

$$x = x + v\Delta t\cos(\theta_t)$$
$$y = y + v\Delta t\sin(\theta_t)$$
$$\theta_t = \theta_t + w\Delta t$$



如果机器人是全向运动的，有 y 轴速度，那轨迹如何推算呢？ 机器人坐标系的 y 轴有了速度，依葫芦画瓢，只需将机器人在机器人坐 标 y 轴移动的距离投影到世界坐标系即可：

$$\Delta x = v_y\Delta t\cos\left(\theta_t + \frac{\pi}{2}\right) = -v_y\Delta t\sin(\theta_t)$$
$$\Delta y = v_y\Delta t\sin\left(\theta_t + \frac{\pi}{2}\right) = v_y\Delta t\cos(\theta_t)$$

此时的轨迹推演只需要将 y 轴的移动的距离叠加在之前计算的公式上即可：

$$x = x + v\Delta t\cos(\theta_t) - v_y\Delta t\sin(\theta_t)$$
$$y = y + v\Delta t\sin(\theta_t) + v_y\Delta t\cos(\theta_t)$$
$$\theta_t = \theta_t + w\Delta t$$

在 HiBot 的启动节点包中，Huanyu_robot.cpp 中计算 odom 的轨迹推演过程就是以上公式的代码化。

```
this->dt = (current_time - last_time).toSec();
double delta_x = (vx * cos(th) - vy * sin(th)) * dt;
double delta_y = (vx * sin(th) + vy * cos(th)) * dt;
double delta_th = vth * dt;
x += delta_x;
y += delta_y;
th += delta_th;
```

机器人的运动模型（还原）

上面的计算中，假设相邻时间段内机器人的轨迹是直线，这是不准确的，更精确的做法是用圆弧来代替。正如论文原文中推导的那样，依然假设不是全向运动机器人，它做圆弧运动的半径为：

$$r = \frac{v}{w}$$

当旋转速度 w 不等于 0 时，机器人坐标为：

$$x = x - \frac{v}{w}\sin(\theta_t) + \frac{v}{w}\sin(\theta_t + w\Delta t)$$
$$y = y - \frac{v}{w}\cos(\theta_t) - \frac{v}{w}\cos(\theta_t + w\Delta t)$$
$$\theta_t = \theta_t + w\Delta t$$

还有其他一些推到算法如下：

$$\Delta\theta = w\Delta t \qquad (rad)$$
$$\Delta x = v\Delta t\cos\left(\theta + \frac{\Delta\theta}{2}\right)$$
$$\Delta y = v\Delta t\sin\left(\theta + \frac{\Delta\theta}{2}\right)$$

在以上三个模型中，代码中常见的就是简化模型 1 。

3. 速度采样

机器人的轨迹运动模型产生后，根据速度就可以推算出轨迹。因此只需采样很多速度，然后分别推算轨迹，然后评价产生轨迹的好坏即可。 速度如何采样是论文原著的第二个核心：在速度(v,w)的二维空间中，存在无穷多组速度。但是根据机器人本身的限制和环境限制可以将采样速度控制在一定范围内：

3.1 移动机器人受自身最大速度最小速度的限制：

$V_m = \{v \in [v_{min}, v_{max}], w \in [w_{min}, w_{max}]\}$

3.2 移动机器人受电机性能的影响：

由于电机力矩有限，存在最大的加减速限制，因此移动机器人轨迹前向模拟的周期 sim_period 内，存在一个动态窗口，在该窗口内的速度是机器人能够实际达到的速度：

$$V_d = \left\{ (v,\omega) \mid \begin{array}{l} v \in [v_c - \dot{v}_b \Delta t, v_c + \dot{v}_a \Delta t] \wedge \\ \omega \in [\omega_c - \dot{\omega}_b \Delta t, \omega_c + \dot{\omega}_a \Delta t] \end{array} \right\}$$

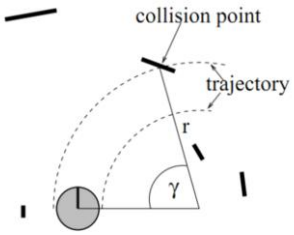
3.3 基于移动机器人安全的考虑：

为了能够在碰到障碍物前停下来，因此在最大减速度条件下，速度有一 个范围：

$$V_a = \left\{ (v,\omega) \mid v \le \sqrt{2 \cdot \mathrm{dist}(v,\omega) \cdot \dot{v}_b} \ \wedge \ \omega \le \sqrt{2 \cdot \mathrm{dist}(v,\omega) \cdot \dot{\omega}_b} \right\}$$

其中 dist(v,w)为速度(v，w)对应轨迹上离障碍物最近的距离，如下图弧线所示：

其中 dist(v,w)为速度(v，w)对应轨迹上离障碍物最近的距离。如下圆弧线所示：

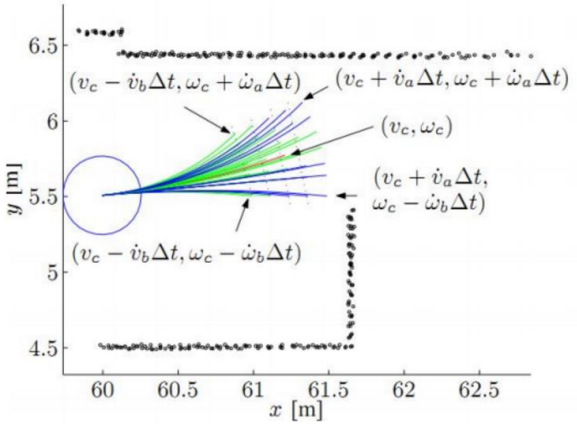


注意：这个条件并不是在采样一开始就能得到的。需要我们模拟出机器人轨迹以后，找到障碍物位置，计算出机器人到障碍物之间的距离， 然后看当前采样的这对速度能否在碰到障碍物之前停下来，如果能够停下来，那这对速度就是可接收的。如果不能停下来，这对速度就得抛弃掉。

为了更直观的感受速度如何采样以及如何排除会碰到障碍的速度，将速度采样的伪代码列出如下：

```
//首先在V_mnV_d的范围内采样速度：
allowable_v = generateWindow(robotV, robotModel)
allowable_w  = generateWindow(robotW, robotModel)
//然后根据能否及时刹车剔除不安全的速度：
    for each v in allowable_v
        for each w in allowable_w
        dist = find_dist(v,w,laserscan,robotModel)
        breakDist = calculateBreakingDistance(v)//刹车距离
        if (dist > breakDist)  //如果能够及时刹车，该对速度可接收
// 如果这组速度可接受，接下来利用评价函数对其评价，找到最优的速度组
```

为了简化每组速度对应轨迹的计算，该算法假设机器人在往前模拟轨迹的时间段内速度不变，直到下一刻采样给定新的速度命令，动态窗口的采样轨迹如下图所示：



4. 评价函数

在采样的速度组中，由若干组轨迹是可行的，因此采用评价函数的方式对每条轨迹进行评价。采用的评价函数如下：

$$G(v,\omega) = \sigma(\alpha \cdot \text{heading}(v,\omega) + \beta \cdot \text{dist}(v,\omega) + \gamma \cdot \text{velocity}(v,\omega))$$

4.1 方位角评价函数：

heading(v,w)是用来评价机器人在当前设定的采样速度下，达到模拟轨迹末端时的朝向和目标之间的角度差距。

4.2 空隙

dist(v,w)代表机器人在当前轨迹上与最近障碍物之间的距离，如果这条轨迹上没有障碍物，那就将其设为常数。

4.3 速度

velocity(v,w)用来评价当前轨迹的速度大小。

4.4 平滑处理

也就是归一化，上面三个部分计算出来后不是直接相加，而是每个部分归一化以后，在相加。如何归一化呢？每一项处以每一项的总和。

$$normal\_head(i) = \frac{head(i)}{\sum_{i=1}^{n} head(i)}$$

$$normal\_dist(i) = \frac{dist(i)}{\sum_{i=1}^{n} dist(i)}$$

$$normal\_volocity(i) = \frac{velocity(i)}{\sum_{i=1}^{n} velocity(i)}$$

其中，n 为采样的所有轨迹，i 为待评价的当前轨迹。

5. 算法总结

梳理清除以上的算法处理流程后，算法的伪代码如下：

```
BEGIN DWA(robotPose,robotGoal,robotModel)
   laserscan = readScanner()
   allowable_v = generateWindow(robotV, robotModel)
   allowable_w  = generateWindow(robotW, robotModel)
   for each v in allowable_v
      for each w in allowable_w
      dist = find_dist(v,w,laserscan,robotModel)
      breakDist = calculateBreakingDistance(v)
      if (dist > breakDist)  //can stop in time
         heading = hDiff(robotPose,goalPose, v,w)
          //clearance与原论文稍不一样
         clearance = (dist-breakDist)/(dmax - breakDist)
         cost = costFunction(heading,clearance, abs(desired_v - v))
         if (cost > optimal)
            best_v = v
            best_w = w
            optimal = cost
    set robot trajectory to best_v, best_w
END
```