机器人定位讲解(以 AMCL 为例)

唐宝芳

2020 年 11 月 16 日

主要内容

主要内容

算法原理

在 Local SLAM 中, 通过 Submap 中的 Scan-to-Map 匹配得到了一个比较理想的机器人位姿估计. 但是由于 Local SLAM 只使用了一段时间内的局部信息, 所以定位误差会随时间积累. 为了能够进一步降低局部累积误差的影响,Cartographer 通过 Pixel-accurate 扫描匹配来进行回环检测, 进一步优化机器人的位姿估计.

计 $H = \{h_1, \ldots, h_k, \ldots, h_K\}$ 为传感器扫描到 K 个 hit 点集合,h_k 是第 k 个 hit 点在机器人坐标系下的位置坐标. 那么 h_k 在地图坐标系下的坐标可以表示为:

$$T_\epsilon h_k = \begin{bmatrix} \cos\epsilon_\theta & -\sin\epsilon_\theta \\ \sin\epsilon_\theta & \cos\epsilon_\theta \end{bmatrix} h_k + \begin{bmatrix} \epsilon_x \\ \epsilon_y \end{bmatrix} \tag{1}$$

Pixel-accurate 扫描匹配问题可以用下式描述:

$$\epsilon^* = \underset{\epsilon \in W}{argmax} \sum_{k=1}^{K} M_{nearest}(T_\epsilon h_k) \tag{2}$$

式中,W 是一个搜索窗口,$M_{nearest}(T_\epsilon h_k)$ 是离 $T_\epsilon h_k$ 最近的栅格单元的占用概率. 可以解释为, 在搜索窗口 W 中找到一个最优的位姿, 使得 hit 点集合出现的概率最大化.

暴力搜索方法

有一种暴力搜索的方法, 如右图所示, 这是一种枚举的方法. 给定搜索步长 r 和 δ_θ, 搜索过程以 ϵ_0 为中心, 通过三层循环遍历所有的 Pixel 并选出得分最高的位姿作为输出 ϵ^*

暴力匹配的 algorithm 如下:

Algorithm 1 Naive algorithm for (BBS)

$best\_score \leftarrow -\infty$

for $j_x = -w_x$ to w_x do

  for $j_y = -w_y$ to w_y do

    for $j_\theta = -w_\theta$ to w_θ do

      $score \leftarrow \sum_{k=1}^{K} M_{nearset}(T_{\xi 0} + (rj_x, rj_y, \delta_\theta rj_\theta) h_k)$

      if $score > best\_score$ then

        $match \leftarrow \xi 0 + (rj_x, rj_y, \delta_\theta rj_\theta)$

        $best\_score \leftarrow score$

      end if

    end for

  end for

end for

return $best\_score$ and $match$ when set.

分支定界方法

- 暴力搜索方法中如果搜索窗口过大或者搜索步长太小, 都将导致整个搜索过程耗时过长.
- Cartographer 使用分支定界方法搜索, 该算法的基本思想是:
  - 用一颗树表示整个解空间.
  - 每一个节点的孩子都是对该节点所代表的搜索空间的一个划分.
  - 每个叶节点都对应着一个解.

整个搜索过程的基本思想:

- 不断地分割搜索空间, 这个过程称为分支.
- 为每次分支之后的孩子节点确定一个上界, 这个过程称为定界.
- 如果一个节点的定界超出了已知最优解的值, 这意味着该节点下的所有解都不可能比已知解更优, 将不在分支该节点.

branch and bound

分支定界 algorithm 如下:

Algorithm 2 DFS branch and bound scan matcher for (BBS)

$best\_score \leftarrow score\_threshold$

Compute and memorize a score for each element in $\mathcal{C}_0$. Initialize a stack $\mathcal{C}$ with $\mathcal{C}_0$ sorted by score, the maximum score at the top.

while $\mathcal{C}$ is not empty **do**

  Pop c from the stack $\mathcal{C}$.

  if $score(c) > best\_score$ **then**

    if c is a left node **then**

      $match \leftarrow \xi_c$

      $best\_score \leftarrow score(c)$

    else

      Branch: Split c int nodes $\mathcal{C}_c$.

      Compute and memorize a score for each element in $\mathcal{C}_c$.

      Push $\mathcal{C}_c$ onto the stack $\mathcal{C}$, sorted by score, the maximum score last.

    end if

  end if

end while

return $best\_score$ and $match$ when set.

branch and bound

分支定界 algorithm 如下:

Algorithm 2 Generic branch and bound

$best\_score \leftarrow -\infty$

$\mathcal{C} \leftarrow \mathcal{C}_0$

while $\mathcal{C} \neq \emptyset$ **do**

  Select a node $c \in \mathcal{C}$ and remove it from the set.

  if c is a left node **then**

    if $score(c) > best\_score$ **then**

      $solution \leftarrow n$

      $best\_score \leftarrow score(c)$

    end if

  else

    if $score(c) > best\_score$ **then**

      Branch: Split c into nodes $\mathcal{C}_c$.

      $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}_c$

    else

      Bound.

    end if

  end if

end while

return $best\_score$ and $solution$ when set.

分支定界方法

- 搜索窗口的栅格索引集合 $\overline{\mathcal{W}}$ 可以通过笛卡尔积 $\{-w_x,\ldots,w_x\}\times\{-w_y,\ldots,w_y\}\times\{-w_\theta,\ldots,w_\theta\}$ 来枚举. 其中,w_x, w_y 分别是 x 和 y 方向上最大的索引, w_θ 是角度的最大索引.
- 搜索窗口 $\mathcal{W}$ 可以用集合 $\{\epsilon_0+(rj_x, rj_y, \delta_\theta j_\theta)\in\overline{\mathcal{W}}\}$ 来表示. 其中,ϵ_0 是搜索的中心, 也是机器人位姿的初始估计. r 和 δ_θ 分别是位移和角度的搜索步长.
- 搜索树中每个节点都可以用是个整数 $c=(c_x, c_y, c_\theta, c_h)\in\mathbb{Z}^4$ 来表示. 其中,c_x, c_y 分别是搜索空间 x, y 轴的起始索引, c_θ 是搜索角度, c_h 代表该搜索空间有 $2^{c_h}\times 2^{c_h}$ 个可能的解. 它们具有相同的角速度, 但位置坐标不同. 这些解的组合可以用如下的笛卡尔积来表示:

$$\overline{\mathcal{V}_c}=\left\{(j_x, j_y)\in\mathbb{Z}^2 \middle| \begin{array}{l} c_x\le j_x< c_x+2^{c_h} \\ c_y\le j_y< c_y+2^{c_h}\end{array}\right\} \tag{3}$$

分支定界方法

- 该节点对应搜索空间的栅格索引集合为 $\overline{W_c} = \overline{\mathcal{V}_c} \cap \overline{\mathcal{W}}$.
- 每当对节点进行分支，就相当于在空间坐标上将搜索空间划分为四个区域，如右图所示.
- 对于叶子节点而言,$c_h = 0$, 其搜索空间中只有一个索引对应着解 $\epsilon_c = \epsilon_0 + (rc_x, rc_y, \delta_\theta c_\theta)$.
- 如果指定搜索树的高度 h_0, 那么初始子空间节点集合中的节点 $c \in \{C_0\}$ 的四个整数可以表示为:

$$c = \begin{cases} c_x = -w_x + 2^{h_0} j_x & : j_x \in \mathbb{Z}, 0 \le 2^{h_0} j_x \le 2w_x \\ c_y = -w_y + 2^{h_0} j_y & : j_y \in \mathbb{Z}, 0 \le 2^{h_0} j_y \le 2w_y \\ c_\theta = j_\theta & : j_\theta \in \mathbb{Z}, -w_\theta \le j_\theta \le w_\theta \\ c_h = h_0 & \end{cases} \tag{4}$$

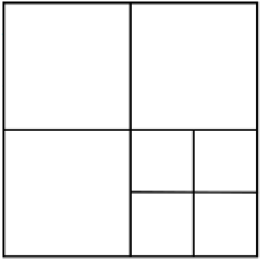

Figure 1: 四个区域搜索空间

分支定界方法

- 搜索树上的每个节点的上界可以通过下式计算得到:

$$\begin{aligned} score(c) &= \sum_{k=1}^{K} \max_{j\in\overline{\mathcal{V}_c}} M_{nearest}(T_{\epsilon_j}, h_k) \\ &\geq \sum_{k=1}^{K} \max_{j\in\overline{\mathcal{W}_c}} M_{nearest}(T_{\epsilon_j}, h_k) \\ &\geq \max_{j\in\overline{\mathcal{W}_c}} \sum_{k=1}^{K} M_{nearest}(T_{\epsilon_j}, h_k) \end{aligned} \tag{5}$$

- 如果对每一个节点都直接计算上界的话, 将是一个很大的计算量. Cartographer 采用一种类似图像金字塔的方法,预先计算出占用栅格地图在不同分支尺寸下的上界, 在实际计算上界时只需要根据 c_h 查询对应尺度下的占用栅格即可获得节点的上界. 右图是分支尺寸分别为 1,4,16,64 时的占用概率上界.

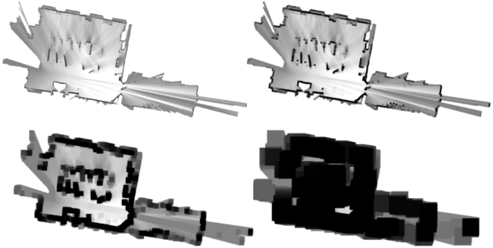

Figure 2: 分支尺寸分别为 1/4/16/64 时的占用概率上界

分支定界方法

- 我们称这个图为预算图，对于第 h 层的节点，在预算图中 x, y 的占用上界可以表示成下式，即以考察点 x, y 为中心，尺寸为 $2^h \times 2^h$ 的窗口内栅格的最高占用概率为上界.

$$M_{precomp}^{h}(x, y) = \max_{\substack{x' \in [x, x + r(2^h - 1)] \\ y' \in [y, y + r(2^h - 1)]}} M_{nearest}(x', y') \tag{6}$$

- 节点 c 的上界可以直接查表得到:

$$score(c) = \sum_{k=1}^{K} M_{precomp}^{c_h}(T_{\epsilon} h_k) \tag{7}$$

- 整个闭环检测的业务逻辑是: 根据当前的子图构建一个占用栅格地图，然后为该地图计算预算图，接着通过深度优先的分支定界搜索算法估计机器人的位姿，最后建立机器人位姿与子图之间的约束关系.

分支定界方法

- 把全部可行解空间反复分割为越来越小的子集，称为分支.
- 对每个子集内的解集计算一个目标下界 (对于最小值问题)，称为定界.
- 在每次分支后，凡是界限超出已知可行解集目标值的那些子集不再进一步分支，这样，许多子集可不予考虑，这称为减枝.
- 分支定界法中，通过分支/定界/剪枝操作，使我们仅在一部分可行解中寻找最优解，而不是全部穷举出来在寻找，求解效率更高.

以整数规划为例，首先规定求解的整数规划问题为 A，相应的线性规划问题为 B(松弛问题).

1. 对问题 B 进行求解:
   1. 若 B 无可行解，则 A 也无可行解，停止计算.
   2. 若 B 有最优解，且符合整数条件，该最优解为 A 的最优解，停止计算.
   3. 若 B 有最优解，但不符合整数条件，计它的目标函数值为 z^* 作为最优解的下界.
2. 找出问题 A 的一个可行解，其目标函数值作为最优解的上界.
3. 迭代:
   1. 分支，在 B 中的最优解中人选一个不符合整数条件的变量 x_j，其值为 b_j，构造两个约束条件,$x_j \leq [b_j], x_j \geq [b_j] + 1$，分别加入到问题 B 中，形成两个子问题 B1 和 B2. 不考虑整数条件求解这两个子问题.
   2. 定界，对每个后续问题表明其求解结果，与其他问题进行比较，将最优目标函数值最小者 (不包括问题 B) 作为新的下界，在已符合整数条件的各分支中，找出目标函数值最小者作为新的上界.
   3. 剪枝，将目标函数不在下界和上界中的分支剪去.
   4. 重复 123，直到得到最优解.

分支定界方法

- Cartographer 使用类 FastCorrelativeScanMatcher2D 具体实现了深度优先的分支定界搜索算法, 该算法能够高效地确定激光点云与子图的匹配度, 估计采样点云时机器人相对于子图的位姿. 为了能够高效的对搜索空间进行分割并计算上界,Cartographer 还为每个子图计算了不同尺度下的占用概率, 以后的搜索过程只需要简单的查表就可以完成.

分支定界方法

branch and bound

branch and bound

回环检测是一种匹配过程，即当获得新的 scan 时，再其附近一定范围内搜索最优匹配帧，若该最优匹配帧符合要求，则认为是一个回环. 该匹配问题可以描述为以下式子.

$$\xi^* = \underset{\xi \in W}{argmax} \sum_{k=1}^{K} M_{nearest}(T_\xi h_k) \qquad (BBS) \tag{8}$$

其中 W 是搜索空间,$M_{nearest}$ 是该点对应栅格点的 M 值. 该式子可以理解为对于 scan 中的每一个光束映射到该地图中某个 submap 的某个 laser scan 上时的置信度和，置信度越高则认为越相似，我们需要在 W 空间中寻找出该置信度和最大的 submap.

欢迎批评指正！