# An Incomplete Scan Matching Tutorial

Giorgio Grisetti

January 10, 2017

In this document we provide an explanation of the basic concepts for developing a simple maximum likelihood mapper for laser data. We first introduce the notation, then we describe how to compute an occupancy grid map, given a known set od poses. Subsequantly we present a simple gradient based scan matchgin algorithm.

## 1 Notation

- $x_t$: robot pose at time $t$.
- $z_t = \underbrace{(z_t^1, \cdots, z_t^n)}_{\text{beams}}$: scan taken at time $t$.
- $u_t$: odometry movement which brings the robot from $x_{t-1}$ to $x_t$, $t$.
- $m_t = f_t(x, y) \rightarrow [0, 1]$ occupancy grid map which can be seen as a function that maps each point in the probability of being occupied.

## 2 The Scan Matching Problem

A scan matching algorithm works in two steps:

- it determines the most likely pose, given the actual measurements and the previously build best map:

$$\hat{x}_t = \underset{x_t}{argmax}(p(x_t|z_t, \hat{x}_{t-1}, m_{t-1}, u_t)) \tag{1}$$

- it computes the next step map given the previously built one, the corrected pose and the range reading:

$$p(m_t|m_{t-1}, \hat{x}_t, \hat{z}_t) \tag{2}$$

In the following we present two simple approaches for performing these two steps.

# 3 Frequancy Based Occupancy grids

An occupancy grid is a discrete world representation. It describes the world as a matrix, whose cells represents the probability of being occupied. The cells are considered independent. Here we present a simple algorithm for updating an occupancy grid, based on a frequentist approach.

For each cell of the map $m^{x,y}$ we keep two numbers: the number of times that cell has been visited $v^{x,y}$, and the number of times that cell has been found occupied $b^{x,y}$.

Let $m_{t-1}$ be the map at the previous time step, $x_t$ the robot pose at time $t$ and $z_t = (z_t^1, \cdots, z_t^n)$ the reading. We first consider the range reading translated according to the current robot pose

$$\hat{z}_t = (\hat{z}_t^1, \cdots, \hat{z}_t^n) = \hat{x}_t \oplus z_t \tag{3}$$

Each beam $z_t^i$ can be described with its endpoint $\hat{p}_t^i$. The cells in the map spanned by such a beam lies on the segment $(\hat{x}_t, \hat{p}_t^i)$. All the cells wich are before th endpoint are detected as free, while the endpoint is occupied.

The we can incrementally update the cells of the map for each beam, in the following way:

$$m_t^{x,y} = (b_t^{x,y}, v_t^{x,y}) = \begin{cases} (b_t^{x,y} + 1, v_t^{x,y} + 1) & \text{if occupied} \\ (b_t^{x,y}, v_t^{x,y} + 1) & \text{if free} \end{cases} \tag{4}$$

The probability that a cell is pccupied is

$$p(m_t^{x,y}) = \frac{b_t^{x,y}}{v_t^{x,y}} \tag{5}$$

# 4 A simple gradient descent scan matcher

Here we describe the simple scamnmatcher you find in this software bundle. The basic principle of this algorithm is to perform a gradinent descent search, on a score function $s(x, z, m)$ of the pose, th reading and the map. The overall algorithm works as follows:

- $bestPose = x_{\text{init}}$
- $bestScore = s(bestPose, z, m)$
- $searchStep = initialSearchStep$
- $iterations = 0$
- while $(!iterations < maxIterations)$
  - $maxMoveScore = bestScore$
  - $bestMovePose = bestPose$
  - for $move$ in $(Backward, Forward, Left, Right, RotateLeft, RotatrRight)$
    - $testPose = computePose(bestPose, move)$
    - $score = s(testPose, z, m)$
    - if $(maxMoveScore < score)$
      - $maxMoveScore = score$
      - $bestMovePose = testPose$
  - if $(bestScore < maxMoveScore)$
    - $bestScore < maxMoveScore$
    - $bestPose = bestMovePose$
  - else
    - $searchStep = searchStep/2$
    - $iterations++$

Here a critical issue is how to compute the score function. A simple approach is to consider a score function as the sum of the score of the single beams score.

$$s(x, z, m) = \sum_i s(x, z^i, m) \tag{6}$$

Our choice is to use a simple endpoint based model. Given a pose $x$ and a reading we consider individually the single beams $z^i$.

- We compute the cell of the map in which the beam endpoint falls. This can be done by translating the endpoint according to $x$ as $\hat{z}^i = x \oplus z^i$
- We compute the cell of the map in which th endpoint falls.
- We found the busy cell $\hat{m}^{x,y}$ in the map which is closest to $\hat{z}^i$.
- We compute the score as a function which decreases with the increase of the distance among $\hat{z}^i$ and $(x, y)^T$. The squared distance $d^2$ can be computed as $d^2 = (\hat{z}^i - (x, y)^T))^T \cdot (\hat{z}^i - (x, y)^T))$

$$s(x, z^i, m) = e^{d^2/\sigma} \tag{7}$$

As an additional optimization for each map cell $m^{x,y}$ we can consider as its center the center of mass of the points falling in that cell.