Lazy localization using the Frozen-Time Smoother

Andrea Censi and Gian Diego Tipaldi

***Abstract*— We present a new algorithm for solving the global localization problem called Frozen-Time Smoother (FTS). Time is 'frozen', in the sense that the belief always refers to the same time instant, instead of following a moving target, like Monte Carlo Localization does. This algorithm works in the case in which global localization is formulated as a smoothing problem, and a precise estimate of the incremental motion of the robot is usually available. These assumptions correspond to the case when global localization is used to solve the loop closing problem in SLAM. We compare FTS to two Monte Carlo methods designed with the same assumptions. The experiments suggest that a naive implementation of the FTS is more efficient than an extremely optimized equivalent Monte Carlo solution. Moreover, the FTS has an intrinsic laziness: it does not need frequent updates (scans can be integrated once every many meters) and it can process data in arbitrary order. The source code and datasets are available for download.**

I. Introduction

Simultaneous Localization And Mapping (SLAM) is the problem of learning maps under sensing uncertainty. In the literature, many approaches have been proposed: they differ by the underlying representation used (grids, features, etc.), and the estimation algorithm employed (Particle Filters, Extended Kalman Filters, Information Filters, etc.).

In order to make SLAM a reliable technology, the current view in the community is that it should be modularized by identifying independent subproblems [1], [2]. We think that a SLAM method should address at least these three different subproblems:

a) Incremental Mapping: The process of building a locally consistent incremental map.

b) Loop Closure: When a mapping robot returns in a previously mapped area, the error accumulated is typically such that the head and the tail of the map estimate are inconsistent. To ensure consistency, one needs a global constraint between the robot pose at closure time and a previous pose in the already mapped area. Establishing *if* a loop is being closed (*loop detection*) and finding the closure point (*loop proposal*) might be considered two different subproblems.

c) Map Optimization: The process of combining local constraints (provided by the incremental mapper) and global constraints (provided by the loop-closure algorithm) to obtain an overall consistent map. This phase should take into account that some loop-closure constraints might be false positives.

A. Censi is with the Control & Dynamical Systems department, California Institute of Technology, 1200 E. California Blvd., 91125, Pasadena, CA. andrea@cds.caltech.edu

G.D. Tipaldi is with Dipartimento di Informatica e Sistemistica "A. Ruberti", Università di Roma "La Sapienza", via Ariosto 25, I-00185 Rome, Italy. tipaldi@dis.uniroma1.it

Loop closure is a critical part of SLAM because it is the only way to correct long-term errors. For very large environments, it is similar to global localization in the partially built map, with one particularity: as part of the SLAM effort, a locally precise estimate of the robot motion is usually available. The algorithm described in this paper exploits this incremental estimate to solve global localization in an efficient way.

A. Related work

In Gutmann *et al.* [3] loop closure is reduced to global Markov localization in the partially built map. This is, of course, extremely expensive, therefore the likelihood computation is approximated by correlation of the local map over the occupancy grid for the global map, and such correlation is efficiently implemented using special-purpose vectorization operations (MMX).

In Fox *et al.* [4], an important problem of loop closure is pointed out: the robot might be *outside* the map built so far. They extended Bayesian localization by using a different way of computing the likelihood, when the robot is an unknown area. In this way, they reduced the number of false positive in the loop detection, which lead to enforcing wrong loop closing constraints.

In Neira *et al.* [5], the map is represented using features (segments and corners). They handle loop closure explicitly by trying to match sets of features in the local map to sets of features in the global map. For this, they use random samples consensus, followed by a joint compatibility test. Clearly, if one has a feature map, the loop closure problem can be solved easily and efficiently.

In grid-based Rao-Blackwellized particle filters (RBPF) [6], [7], in theory one does not need to explicitly handle loop closure. Each particle represents an hypothesis on the trajectory: when the robot revisits parts of the old map, only the particles with a good trajectory survive. However, in practice, there are some complications. In RBPF the number of particles is limited by the available memory and CPU. This means that there are typically as few as 50-100 particles representing all the hypotheses on the trajectory. When a loop is closed, only very few of these particles will have a non-zero likelihood, and therefore there will be a loss of diversity (particle depletion). Stachniss *et al.* [8] addressed the problem by actively detecting loops and then using techniques to restore particle diversity after closing the loop.

One of the problems to consider is that loop closure cannot be an instantaneous decision, because further data could disprove the identified closing point. Therefore, one should keep different hypotheses on the map topology. Haehnel

[9] extended FastSLAM with lazy data association; Grisetti *et al.* [1] extended the RBPF by using an hybrid map representation and tracking different topology hypotheses.

In Ho and Newman [2] loop closure is completely decoupled from the incremental map estimation. They considered the environment as a collection of discrete scenes, for which a distance measure can be defined. The resulting scene-to-scene distance matrix is used to detect consistent *sequences* of scenes that indicate loop closure.

B. Filtering or smoothing?

SLAM methods that handle loop closing explicitly need a constraint between the current pose of the robot and the old part of the map to ensure consistency of the overall map. Several authors casted this problem as global localization on the map built so far [3],[4], that is computing the distribution

$$p(\boldsymbol{x}_n|\boldsymbol{y}_{0:n}, \boldsymbol{m}) \tag{1}$$

where 0 is assumed to be the time at which the localization procedure is activated, and $\boldsymbol{m}$ is the map built before that time. When the pose is disambiguated, a new constraint can be added to the global map graph. Note that (1) is a *filtering* distribution: it is the estimate of the last pose based on the past. We argue that in this context knowledge of the *smoothing* distribution

$$p(\boldsymbol{x}_0|\boldsymbol{y}_{0:n}, \boldsymbol{m}) \tag{2}$$

would be more useful. The reason is readily explained: in most cases, a single-shot relocalization is not reliable, and several data are needed to disambiguate the robot position. Given that one should consider an interval of time, it is better to have an estimate of the robot pose at the beginning of the interval, rather than at the end. To see why, consider the canonical example of a loop closure in Fig. 1. Using the filtering distribution means imposing a constraint later in time, while using the smoothing distribution imposes it earlier and better reduces the inconsistency in the map.

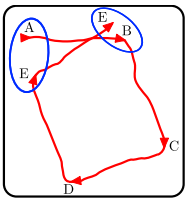
Not only the smoothing distribution is more useful, we will show it can also be computed efficiently if a reasonably precise estimate of the incremental robot motion is available. An example of this is the estimate given by a scan-matcher, which is already available at zero-cost if localization is done as a subprocess of SLAM.

More formally, we are set to compute the distribution

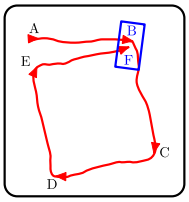
$$p(\boldsymbol{x}_0|\boldsymbol{y}_{0:n}, \boldsymbol{s}_{0:n}, \boldsymbol{m}) \tag{3}$$

where $\boldsymbol{s}$ is an estimate of the incremental motion of the robot: $\boldsymbol{s}_{i:j} \triangleq \boldsymbol{x}_j \ominus \boldsymbol{x}_i$. We will present three algorithms that compute (3): the first two are slight modifications of particle filters already studied in the literature, while the Frozen-Time Smoother is the novel contribution of this paper.

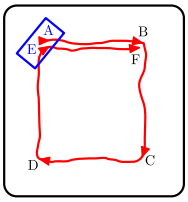
From now on, we omit the map $\boldsymbol{m}$ from the equations, and use the abbreviations $\boldsymbol{y}_{:} = \boldsymbol{y}_{0:n}$, $\boldsymbol{s}_{:} = \boldsymbol{s}_{0:n}$.



(a) Odometry data



(b) Enforcing B = F



(c) Enforcing A = E

Fig. 1. This picture illustrates the difference between modelling loop closing as smoothing or as filtering. Fig. 1(a) shows the the odometry data of a robot moving in a square environment. The two paths A–B and E–F correspond to the same corridor; assume that the robot can relocalize itself while traversing this corridor. If one models loop closing as filtering, one would get the constraint B = F which corresponds to the resulting map in (b). If one models loop closing as smoothing, the result is the constraint A = E, which results in the more consistent map in (c).

II. Particle filters approaches

In this section we develop two approximated solutions to the smoothing problem, as the cost of an exact implementation using Monte Carlo techniques (particle smoother) is quadratic in the number of particles.

The first algorithm is so simple that we had to try it before discarding it as *too* simple. One can use a vanilla particle filter and remember the first pose of the particles. Then the distribution of the first poses, weighted according to the distribution of the last poses, can be assumed as a crude approximation to (3). In the following, this is referred to as Monte Carlo Smoothing v1 (MCS-1).

A less crude approach uses the assumption that the incremental estimate of the pose is precise. The smoothing distribution can be factorized as:

$$p(\boldsymbol{x}_0|\boldsymbol{y}_{:}, \boldsymbol{s}_{:}) = \int p(\boldsymbol{x}_0|\boldsymbol{x}_n, \boldsymbol{y}_{:}, \boldsymbol{s}_{:}) p(\boldsymbol{x}_n|\boldsymbol{y}_{:}, \boldsymbol{s}_{:}) d\boldsymbol{x}_n \tag{4}$$

into the filtering distribution $p(\boldsymbol{x}_n|\boldsymbol{y}_{:}, \boldsymbol{s}_{:})$ and the inverse informed motion model $p(\boldsymbol{x}_0|\boldsymbol{x}_n, \boldsymbol{y}_{:}, \boldsymbol{s}_{:})$. Intuitively, the question "Where was I at time 0?", is split into "Where am I now?" and "What was my incremental motion?".

If we make the assumption that the error of the scan matcher is very small, we can approximate $p(\boldsymbol{x}_0|\boldsymbol{x}_n, \boldsymbol{y}_{:}, \boldsymbol{s}_{:})$ as a Dirac distribution. Therefore, the integral (4) is greatly simplified, and the target distribution is approximated by translating the estimate at time n back in time according to the incremental estimate $\boldsymbol{s}_{0:n}$.

This approach, which we call Monte Carlo Smoothing v2(MCS-2), is sketched as Algorithm 1. The modifications with respect to a Monte Carlo Localization is that the incremental estimate is used, both for evolving the particles, and to translate them back to time 0.

Algorithm 1: Monte Carlo Smoothing v2

Input:
- a "freezing time" (0)
- a set of sensor scans $\boldsymbol{y}_0$, $\boldsymbol{y}_1$, ...
- an incremental estimate of the pose $\boldsymbol{s}_{0:k}$
- a map $\boldsymbol{m}$

Output: weighted samples $\{\langle \boldsymbol{x}_{0|k}^{(i)}, w_{0|k}^{(i)} \rangle\}$

for *each instant* k **do**
  for *each particle* i **do**
    sample $\boldsymbol{x}_k^{(i)}$ from $p(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}^{(i)}, \boldsymbol{s}_{k-1:k})$
    $w_k^{(i)} \leftarrow p(\boldsymbol{y}_k | \boldsymbol{x}_k^{(i)}, \boldsymbol{m})$
    // Translate back in time:
    $\boldsymbol{x}_{0|k}^{(i)} \leftarrow \boldsymbol{x}_k^{(i)} \ominus \boldsymbol{s}_{0:k}$
    $w_{0|k}^{(i)} \leftarrow w_k^{(i)}$
  end
  Resample the particles according to w_k
end

Algorithm 2: Frozen-Time Smoother

Input:
- a "freezing time" (0)
- a set of sensor scans $\boldsymbol{y}_0$, $\boldsymbol{y}_1$, ...
- an incremental estimate of the pose $\boldsymbol{s}_{0:k}$
- (optional) a prior for $\boldsymbol{x}_0$

Output: a grid estimate of $p(\boldsymbol{x}_0)$

Create the reference normal-map refMap
Initialize the belief *bel* to the prior $p(\boldsymbol{x}_0)$
for some $\boldsymbol{y}_k$, in arbitrary order **do**
  Create a local normal-map localMap from scan $\boldsymbol{y}_k$
  $p(\boldsymbol{x}_0 | \boldsymbol{y}_k) \leftarrow$ `ght` (refMap, localMap, $\hat{\boldsymbol{s}}_{0:k}$)
  bel $*=$ $p(\boldsymbol{x}_0 | \boldsymbol{y}_k) / p(\boldsymbol{x}_0)$
end

Function `ght` (refMap, localMap, $\hat{\boldsymbol{s}}$)

for $\langle p_i, \alpha_i \rangle \in$ localMap **do**
  for $\langle p_j, \alpha_j \rangle \in$ refMap **do**
    // Compute an estimate of the pose at time k
    $\hat{\theta}_k \leftarrow \alpha_j - \alpha_i$
    $\hat{\boldsymbol{t}}_k \leftarrow p_j - R(\hat{\theta}_k) \cdot p_i$
    $\hat{\boldsymbol{x}}_k \leftarrow \langle \hat{\boldsymbol{t}}_k, \hat{\theta}_k \rangle$
    // Use the displacement $\hat{\boldsymbol{s}}$ *and the estimate* $\hat{\boldsymbol{x}}_k$
    // to compute the pose at time 0
    $\hat{\boldsymbol{x}}_0 \leftarrow \hat{\boldsymbol{x}}_k \ominus \hat{\boldsymbol{s}}$
    // Add a vote for $\hat{\boldsymbol{x}}_0$
    buffer $[\hat{\boldsymbol{x}}_0]$ ++
  end
end
return buffer

III. The Frozen-Time Smoother

For this algorithm to work, we need two assumptions. The first is that an estimate of the incremental robot motion is available, roughly precise during the time it takes to localize.

The second assumption is that a fast way to compute a 'translated' likelihood is available. In this paper, we use a Generic Hough Transform (GHT)-like [10] algorithm, whose input is raw data. If one wants to use features, a completely equivalent algorithm that could be plugged here is the one described by Paz *et al.* [11].

If the two assumptions hold, we can find a particularly simple factorization of the target distribution. The sensor data $\boldsymbol{y}_k$ can be considered independent when conditioning on both the initial state $\boldsymbol{x}_0$ and the incremental motion $\boldsymbol{s}_{k:0}$:

$$p(\boldsymbol{x}_0 | \boldsymbol{y}_{:}, \boldsymbol{s}_{:}) \propto p(\boldsymbol{x}_0) \prod_k p(\boldsymbol{y}_k | \boldsymbol{x}_0, \boldsymbol{s}_{0:k}) \tag{5}$$

Dis-integrate $p(\boldsymbol{y}_k | \boldsymbol{x}_0, \boldsymbol{s}_{0:k})$ with respect to $\boldsymbol{x}_k$ to obtain

$$p(\boldsymbol{x}_0 | \boldsymbol{y}_{:}, \boldsymbol{s}_{:}) \propto p(\boldsymbol{x}_0) \prod_k \int p(\boldsymbol{y}_k | \boldsymbol{x}_k) p(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{s}_{0:k}) d\boldsymbol{x}_k \tag{6}$$

If one approximates $p(\boldsymbol{x}_k | \boldsymbol{x}_0, \boldsymbol{s}_{k:0})$ by a Dirac distribution $p(\boldsymbol{x}_k = a | \boldsymbol{x}_0 = b) \simeq \delta(b \ominus a - \boldsymbol{s}_{k:0})$, then one can solve the integral in (6) and obtain

$$p(\boldsymbol{x}_0 = a | \boldsymbol{y}_{:}, \boldsymbol{s}_{:}) \propto p(\boldsymbol{x}_0 = a) \prod_k p(\boldsymbol{y}_k | \boldsymbol{x}_k = a \oplus \boldsymbol{s}_{0:k}) \tag{7}$$

The right hand side is a product of 'translated likelihoods': each likelihood at time k is translated by the motion $\boldsymbol{s}_{0:k}$. In short, because the incremental estimate is available and is precise, we can compute the likelihood at time 0 very efficiently.

FTS uses a three-dimensional (x, y, θ) grid as the representation of the belief, at the *freezing time* 0. The grid is used essentially as a voting space, therefore its resolution is relatively unimportant (in the experiments, we set the resolution to 1m, 30deg). The grid will represent in turn $p(\boldsymbol{x}_0)$ (prior), $p(\boldsymbol{x}_0 | \boldsymbol{y}_0)$, $p(\boldsymbol{x}_0 | \boldsymbol{y}_{0:1})$, etc. Note that the scans can be integrated in arbitrary order, and some can be postponed or skipped altogether.

A. Translated likelihood computation by GHT

The map representation is contingent on using the GHT for computing the likelihood. Points are sampled from the surfaces in the environment, and for these points the surface orientation is estimated. The result is a 'normal map': a series of tuples $\langle \boldsymbol{p}_j, \alpha_j \rangle$ where $\boldsymbol{p} \in \mathbb{R}^2$ is a point on the environment surfaces and α_j is the direction of its normal. An example of such a map is shown in Fig. 2. We remark that this is an extremely compact representation. Every scan is converted in the same way to a local normal map. The local and the global normal map are passed to the GHT algorithm.

The GHT creates hypotheses by considering all possible correspondences between the local and normal map points. These hypotheses (can also be thought as 'votes') are accumulated in a grid. At the end of the computation, one can assume that this is an approximation to $p(\boldsymbol{x}_k | \boldsymbol{y}_k)$.

More in detail, if the i-th point of the local map corresponds to the j-th point on the global map, then the pose of the robot must be $\hat{\boldsymbol{x}}_k = (\hat{\boldsymbol{t}}_k, \hat{\theta}_k)$, where $\hat{\boldsymbol{t}}_k$ and $\hat{\theta}_k$ are given by:

$$\hat{\theta}_k \leftarrow \alpha_j - \alpha_i \tag{8}$$

$$\hat{\boldsymbol{t}}_k \leftarrow \boldsymbol{p}_j - \mathbf{R}(\hat{\theta}_k)\, \boldsymbol{p}_i \tag{9}$$

And now to the trick that makes everything work: because we know the motion the robot did from $\boldsymbol{x}_0$ to $\boldsymbol{x}_k$, we can compute directly $p(\boldsymbol{x}_0|\boldsymbol{y}_k)$. In the GHT loop, we simply translate $\hat{\boldsymbol{x}}_k$ back by $\boldsymbol{s}_{0:k}$, to obtain the hypothesis for $\hat{\boldsymbol{x}}_k$:

$$\hat{\boldsymbol{x}}_0 \leftarrow \hat{\boldsymbol{x}}_k \ominus \hat{\boldsymbol{s}}_{0:k} \tag{10}$$

After the distribution $p(\boldsymbol{x}_k|\boldsymbol{y}_0)$ has been computed, this new information is integrated in the belief by a simple multiplication, according to (7). That's it: FTS can be described in a few lines of text (Algorithm 2).

B. On the use of a grid

FTS uses a three-dimensional grid, but no expensive operation is performed on the grid. Compare with using plain Markov localization, in which one has to perform a convolution on the grid (prediction step), and compute the likelihood for every cell (update step). Also note that Markov Localization would need small cells, or else it would make little sense to compute the likelihood for a 1m $\times$ 1m cell.

Instead, FTS's grid can be as coarse as the filter designer wants. The cell size does not impact the speed of the GHT step[1]. The cost of weighting the grid by the likelihood *does* depend on the grid resolution, but this cost is, in practice, negligible with respect to the GHT step.

Moreover, the grid representation is easier to handle than particle distributions. The problem with particles is that they tend to concentrate on small parts of the state space after few observations. If one uses too few particles to bootstrap the filter, it is likely that no particle will be near the true solution: in few steps the distribution will suffer from particle depletion. In actual implementations, this problem must be mitigated by either using a particularly relaxed likelihood, or by injecting new particles in the distribution.

C. On the laziness of FTS

Both the PFs algorithm rely on a filtering stage which implies frequent integration of the observations. On the contrary, FTS can integrate observation "distant" in time, as long the scan-matcher is sufficiently precise.

Moreover, in FTS the scans can be processed in arbitrary order: the factorization in (7) is, of course, commutative. While the typical case would be to integrate scans as they are available, there is much freedom here. One can skip scans, or procrastinate and postpone some, if there are not enough computational resources at the moment. For the experiments, we integrated one scan every 5m and simply discarded the others.

[1]If one assume the software is running on an ideal Von Neumann machine with $O(1)$ memory access.

Thhe integration order being arbitrary hints to the fact that FTS has the same "consideration" for every scan. This in contrast with any PF, where the initial observations are more important than the last, as the first scans essentially choose which part of the state space will be explored.

IV. Experiments

We chose two logs from the Radish[2] repository. The first was collected in an Intel building in Seattle. This is a building, of size 30m$\times$30m, with office rooms, many of which are very similar. Some are cluttered with object/people, and there are also curved surfaces. In such environment, the typical situation is that there are many initial hypotheses that gets quickly disambiguated.

The second log was collected in the Aces building at the University of Texas in Austin. This is the typical ambiguous situation: in this 60m$\times$60m environment, there is a symmetry – a central room, from which four similar corridors depart, with other feature-less corridors around. In this case, the typical situation is that a filter must keep a relatively small number of hypotheses (2 or 4) for a long time, until there is enough data to completely disambiguate the pose.

We used GMapping[3] to process the logs, and we obtained two sets of data: the final SLAM result, and an incremental scan-matching estimate. The SLAM result was used as the map input $\boldsymbol{m}$ to the algorithms, from which the PFs would create an occupancy grid, and FTS creates the normal map. The scan-matching result was split into multiple chunks, with each being an independent experiment. In each chunk, the robot traveled for about 30m. In total, we obtained 20 chunks for Aces and 40 for Intel. Each method was given the global map and one scan-matching chunk, and had to guess where the robot started.

FTS's parameters were chosen as follows. The normal map has a resolution of one point every 0.2m – this results in about 4000 points in the aces environment. The belief grid has a resolution of 1m x 1m x 30deg. We integrate a scan every 5m, and discard the others.

As for the PFs, the occupancy grid has a resolution of 0.05m, the number of particles is 10000 (sufficient but not excessive for the environments considered. The scans are integrated every 0.5m or 0.5rad, whichever comes first.

The most natural performance measure is the **distance** between the maximum-likelihood pose estimated by the method, and the true pose. This is a quick measure that indicates whether the method has converged near the true solution. Note, however, that this distance measure could be misleading in ambiguous situations. For example, as can be seen in Fig. 3, there are three cases in which FTS cannot disambiguate the pose by considering only one scan every 5m, simply because there is not enough information. In those cases, the 'true' pose is actually the second highest peak of the distribution, but this cannot be seen by considering only the distance measure. Therefore other statistics are needed.

[2]`http://radish.sourceforge.net/`
[3]`http://openslam.org/`

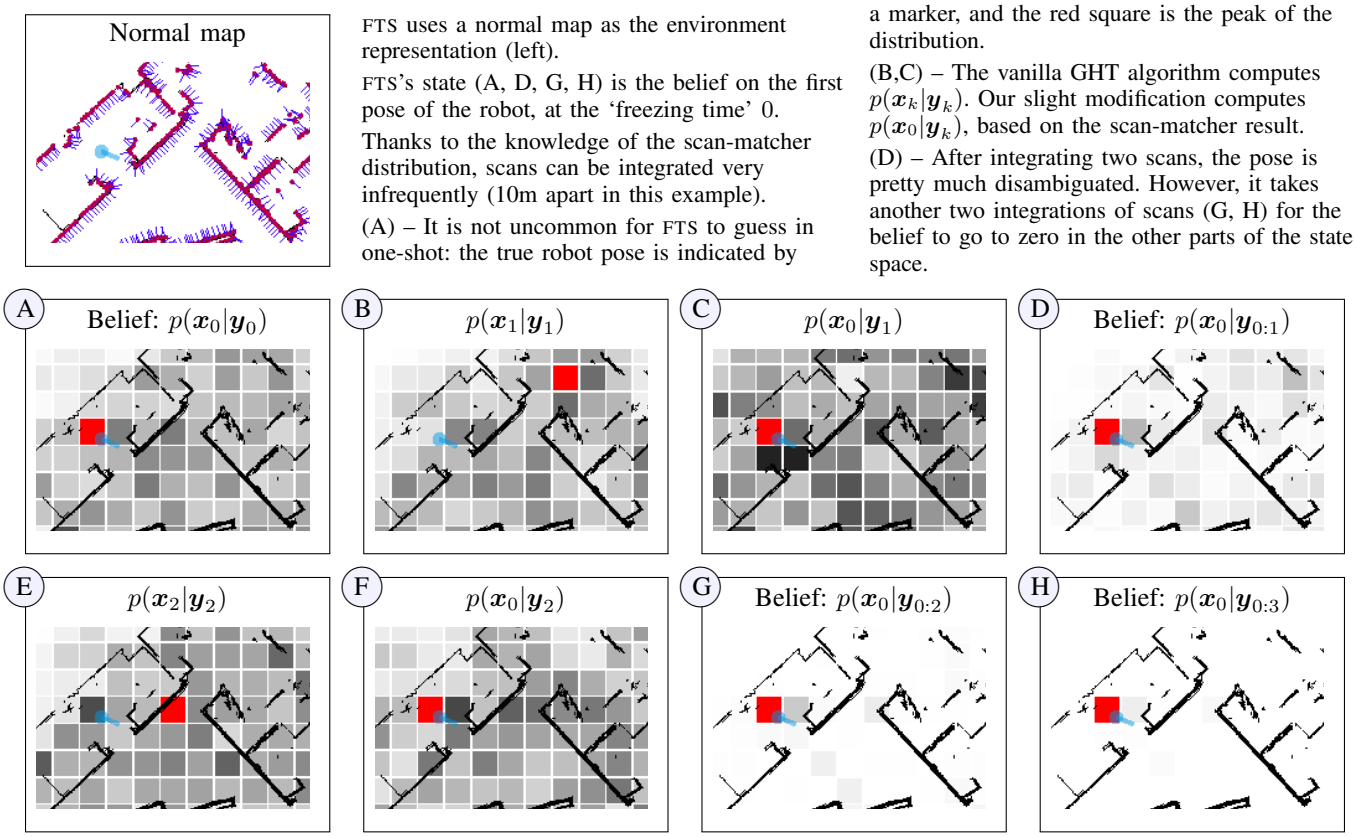


Fig. 2. Example of Frozen-Time Smoother behavior.

FTS uses a normal map as the environment representation (left).

FTS's state (A, D, G, H) is the belief on the first pose of the robot, at the 'freezing time' 0.

Thanks to the knowledge of the scan-matcher distribution, scans can be integrated very infrequently (10m apart in this example).

(A) – It is not uncommon for FTS to guess in one-shot: the true robot pose is indicated by a marker, and the red square is the peak of the distribution.

(B,C) – The vanilla GHT algorithm computes $p(\boldsymbol{x}_k|\boldsymbol{y}_k)$. Our slight modification computes $p(\boldsymbol{x}_0|\boldsymbol{y}_k)$, based on the scan-matcher result.

(D) – After integrating two scans, the pose is pretty much disambiguated. However, it takes another two integrations of scans (G, H) for the belief to go to zero in the other parts of the state space.

To compute the following two measures in a consistent way, we converted the PF distribution to a grid distribution with the same resolution as the grid used for FTS.

Another measure – less intuitive than the distance, but more correct – is the estimated likelihood for the true pose. That is, if each method estimates $p(\boldsymbol{x}_0 = x|\boldsymbol{y}_:, \boldsymbol{s}_:) = f(x)$, we consider the **score** $s = \log f(\overline{\boldsymbol{x}}_0)$, where $\overline{\boldsymbol{x}}_0$ is the true pose. It can be shown that the score is an approximation to the KLD distance between the belief and the true distribution. In Fig. 3, we see that the score is high, even in the cases for which the first peak is not the true pose. In Fig. 4, there is one case in which FTS's distance is high and the score is low: this is a genuine failure, caused by a corresponding failure of the scan-matcher during the chunk (GMapping, employing a RBPF, can afford to have a non particularly robust scan matcher).

It is interesting to note that the score values are similar among the three methods: this reinforces our belief that they are computing the same distribution, albeit in completely different ways.

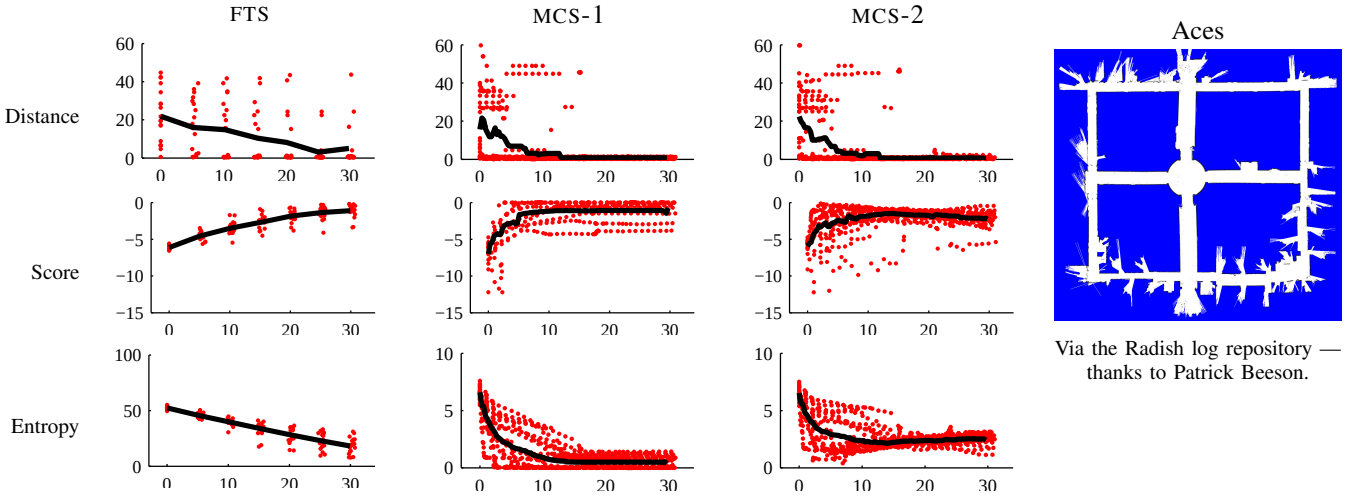
A measure that describes the character of the method is the **entropy** of the estimated distribution. The entropy for the PFs quickly goes to zero, as particles tend to concentrate in small areas of the state space. MCS-1 has a lower entropy than MCS-2, as the particles do not "move" within the state space. Instead, for FTS, the distribution is very smooth and non-zero practically everywhere. This gives a very high value of entropy. Note, however, that large values of entropy do not imply bad precision. In fact, as can be seen by comparing the distance and entropy graphs, for the first step, the entropy is high even when FTS did a one-shot localization.

Finally, we do some consideration on the efficiency. On an Intel Core 2 Duo, with 2.0GHz and 4Mb of cache, an iteration of FTS needed about 1.2s, while the PFs took about 2.1s. In these experiments, FTS was fed about one fifth of the data fed to the PFs. Therefore, we observed about an order of magnitude gain in efficiency. Note, however, that this kind of benchmark is highly dependent on the implementation and the parameters used. For the PFs, we used some old, well-honed source code with all the tricks we know for a particularly efficient and robust implementation. For FTS, we did the straightforward naive implementation of Algorithm 2. We are particularly happy about these numbers, as FTS is already quite efficient, and there is a lot of space for improvement.

V. Conclusions and Future Work

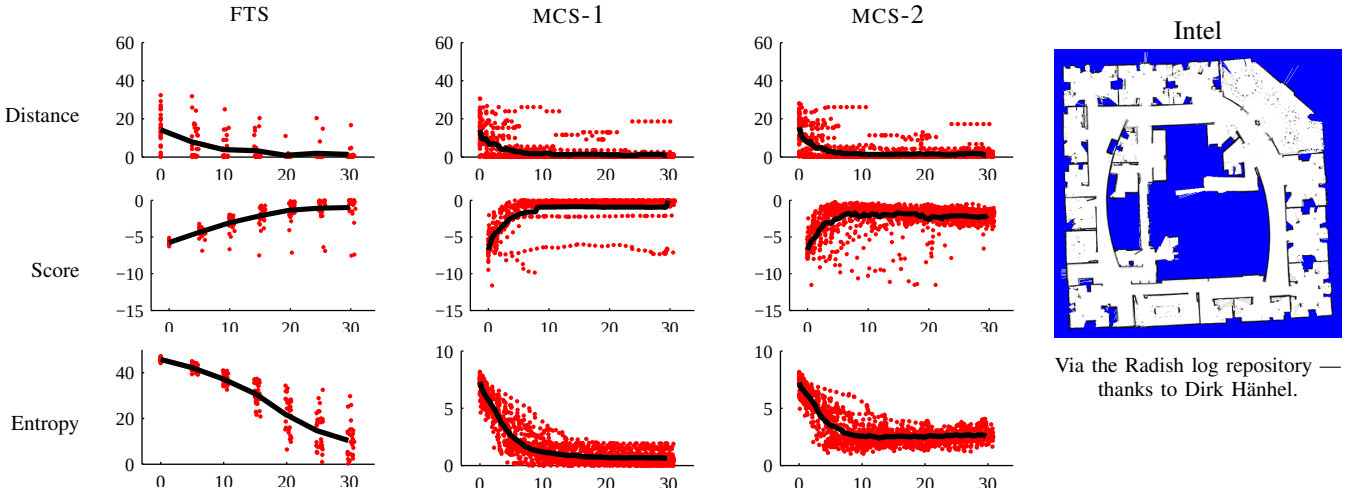
We presented an algorithm, called Frozen-Time Smoother, which can efficiently solve the global localization problem when it is formulated as a smoothing problem, and a precise incremental estimate of the robot motion is available. These assumptions hold when when global localization is used for loop closing in SLAM,

We compared the FTS with the closest technique available in the literature. The experiments suggest that a naive implementation of FTS is more efficient than an extremely



Via the Radish log repository — thanks to Patrick Beeson.

Fig. 3. Results for the Aces environment. In all the plots, the x axis measures the linear distance along the chunck in meters; each chunk is about 30 meters long. The red dots are individual samples; the black bold line is the samples average.



Via the Radish log repository — thanks to Dirk Hähnel.

Fig. 4. Results in the Intel environment.

optimized state-of-the-art Monte Carlo filter working under the same assumptions.

Moreover, it has several other nice properties. The grid representation is efficient and does not have the common problems of particle methods, like overconfidence and the need of a frequent update. Moreover, FTS has an intrinsic laziness, as it does not need frequent updates and it can process data in arbitrary order.

Source code, datasets and animations are available at the website `http://purl.org/censi/2007/fts`.

References

[1] G. Grisetti, G. Tipaldi, C. Stachniss, W. Burgard, and D. Nardi, "Fast and accurate SLAM with Rao-Blackwellized particle filters," *Robots and Autonomous Systems*, 2007.

[2] P. Newman, D. Cole, and K. Ho, "Outdoor SLAM using visual appearance and laser ranging," in *Proc. of the IEEE Int. Conf. on Robotics & Automation*, Orlando, FL, USA, 2006.

[3] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Proc. of the IEEE Int. Symposium on Computational Intelligence in Robotics and Automation (CIRA)*, Monterey, CA, USA, 1999, pp. 318–325.

[4] D. Fox, J. Ko, K. Konolige, and B. Stewart, "A hierarchical bayesian approach to the revisiting problem in mobile robot map building," in *Proc. of the Int. Symposium of Robotics Research*, Siena, Italy, 2003.

[5] J. Neira, J. Tardos, and J. Castellanos, "Linear time vehicle relocation in SLAM," in *Proc. of the IEEE Int. Conf. on Robotics & Automation*, 2003, pp. 427–433.

[6] D. Hähnel, W. Burgard, D. Fox, and S. Thrun, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Las Vegas, NV, USA, 2003.

[7] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with Rao-Blackwellized particle filters," *IEEE Transactions on Robotics*, 2006.

[8] C. Stachniss, D. Hähnel, W. Burgard, and G. Grisetti, "On actively closing loops in grid-based FastSLAM," *Advanced Robotics*, vol. 19, no. 10, pp. 1059–1080, 2005.

[9] D. Hähnel, W. Burgard, B. Wegbreit, and S. Thrun., "Towards lazy data association in SLAM," in *Proc. of the Int. Symposium of Robotics Research*, Siena, Italy, 2003, pp. 421–431.

[10] D. H. Ballard, "Generalizing the Hough transform to detect arbitrary shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111–122, 1981.

[11] L. Paz, P. Piníes, J. Neira, and J. Tardós, "Global localization in SLAM in bilinear time," in *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Edmonton, Canada, 2005.