Multi-robot Simultaneous Localization and Mapping using Particle Filters

Andrew Howard
NASA Jet Propulsion Laboratory
Pasadena, California 91109, U.S.A.
Email: abhoward@robotics.jpl.nasa.gov

Abstract— This paper describes an on-line algorithm for multi-robot simultaneous localization and mapping (SLAM). We take as our starting point the single-robot Rao-Blackwellized particle filter described in [1] and make two key generalizations. First, we extend the particle filter to handle multi-robot SLAM problems in which the initial pose of the robots is known (such as occurs when all robots start from the same location). Second, we introduce an approximation to solve the more general problem in which the initial pose of robots is not known a priori (such as occurs when the robots start from widely separated locations). In this latter case, we assume that pairs of robots will eventually 'bump into' one another, thereby determining their *relative* pose. We use this relative pose to initialize the filter, and combine the subsequent (and prior) observations from both robots into a common map. This algorithm has been experimentally validated using data from a team of four robots equipped with odometry and scanning laser range-finders.

I. INTRODUCTION

Simultaneous localization and mapping (SLAM) is a well studied problem for which there exists a number of good solutions [2]. The vast majority of these solutions, however, consider a single robot in a static environment, using either sparse 2D/3D feature points or dense 2D laser range-finder data. This paper addresses the less-well-studied problem of multi-robot SLAM, motivated by the fact that multiple robots can complete exploration and mapping tasks in less time than a single robot can.

The starting point of our analysis is the Rao-Blackwellized particle filter described in [1]. This algorithm uses a particle filter to approximate the posterior probability distribution over possible maps, and adds robot observations incrementally using a Bayesian update step. This algorithm can readily be generalized to solve the multi-robot SLAM problem, *assuming that the initial pose of all robots is known*. More precisely, one must know at least the initial relative pose of all but one of the robots, with the pose of the remaining robot being chosen arbitrarily (as in the case with single-robot SLAM). This assumption is extremely limiting in practice: it implies either that all robots have started from the same location, or that the initial locations of the robots have been surveyed by some external means.

In the more general case, the initial pose of the robots will not be known a priori: the robots may be placed at widely separated (and possibly random) initial locations. Our solution for this problem has two important elements. First, we assume that robots are to able detect, identify and measure the relative pose of other robots at some time during the exploration task (when those robots are both nearby and within line-of-sight, for example). Such encounters allow robots to fuse their subsequent observations into a common map, using the measured relative pose to initialize the filter (note, however, that only the first such encounter is used; subsequent encounters between robots are ignored). Second, we generalize the particle-filter-based SLAM algorithm such that the filter supports time-reversed updates; this generalization allows robots to incorporate observations that occured prior to the first encounter, by treating those observations as if they came from additional 'virtual' robots travelling backwards in time.

As an illustration, consider the following example. Two robots are exploring an environment from distant (and unknown) initial locations. At some time, the robots encounter one another and measure their relative pose. At this time, we construct a filter in which robot 1 has an initial (arbitrary) pose of zero, and robot 2 has the measured relative pose. Subsequent measurements from the two robots are fed to the filter, and thereby fused into a common map. At the same time, two *virtual* robots are added to the filter with poses initialized as above. Previously recorded measurements are fed to the filter in reverse time-order, such that these virtual robots appear to be driving backwards through the environment. Thus, the filter incrementally fuses data from both robots, recorded both before and after the encounter, into a single map.

This algorithm has number of attractive features. First, it is able to fuse all data from all robots into a single map, without knowing the initial robot poses. Second, it inherits the bounded-time, bounded-memory properties of the single-robot SLAM algorithm (CPU and memory requirements do not increase with path length, as is the case with some algorithms [3], [4]). Third and finally, the algorithm is fast: on a 2.4 GHz P4 processor, our implementation can fuse data from four robots in real time. Collectively, these features make the algorithm highly suitable for on-line, in-the-loop applications, such as multi-robot exploration and search tasks.

II. BAYES FILTERS, PARTICLE FILTERS AND SIMULTANEOUS LOCALIZATION AND MAPPING

In this section, we review the basic theory of Rao-Blackwellized particle filters, as applied to the problem of simultaneous localization and mapping [1], [5]–[7]. Following both the approach and notation of Hähnel [1], we first develop
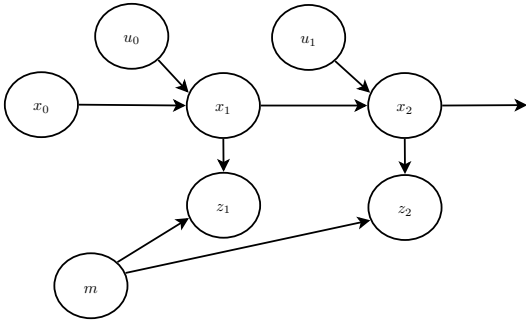
Fig. 1. Bayes net for single-robot SLAM. The robot trajectory is indicated by the sequence $x_0, x_1, x_2, x_3...$, observations by the sequence $z_1, z_2, ...$, and actions by the sequence $u_0, u_1, ...$

a mathematical formalism for the single robot case, then extend (and approximate) the formalism to handle multi-robot SLAM. This extended formalism provides the basis for the on-line SLAM algorithm presented in Section II-D.

A. Single-robot SLAM

The SLAM problem for a single robot is treated as follows. Let $x_{1:t}$ denote a sequence of robot poses at times $1, 2, ...t$, let $z_{1:t}$ denote a corresponding sequence of observations, and let $u_{0:t-1}$ denote the sequence of actions executed by the robot. Our (intermediate) aim is to compute the posterior probability $p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1}, x_0)$ over the robot trajectory $x_{1:t}$ and map m, given some initial pose x_0. We write this as the product of two factors:

$$p(x_{1:t}, m \mid z_{1:t}, u_{0:t-1}, x_0) = \\ p(m \mid x_{1:t}, z_{1:t}, u_{0:t-1}, x_0) p(x_{1:t} \mid z_{1:t}, u_{0:t-1}, x_0) \quad (1)$$

where the first term is a distribution over possible maps and the second is a distribution over possible trajectories. The utility of this expression lies in the fact that the first term can be computed analytically once the robot trajectory $x_{1:t}$ is known. Thus, we may approximate the posterior over trajectories and maps using a particle filter in which each sample represents a complete robot trajectory, and a separate map is conditioned on each such sample.

Exploiting the conditional dependencies inherent in the single-robot SLAM problem (see Figure 1), we construct the Rao-Blackwellized particle filter as follows. Let each particle (i) be a tuple $\langle x_t^{(i)}, m_t^{(i)}, w_t^{(i)} \rangle$ such that $x_t^{(i)}$ is the robot pose at time t, $m_t^{(i)}$ is the map generated using observations recorded up to and including time t, and $w_t^{(i)}$ is the particle weight. Given some action/observation pair (u_{t-1}, z_t), the filter is updated using:

$$x_t^{(i)} = A(u_{t-1}, x_{t-1}^{(i)})$$
$$m_t^{(i)} = M(z_t, x_t^{(i)}) + m_{t-1}^{(i)}$$
$$w_t^{(i)} = S(z_t, x_t^{(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)} \quad (2)$$

where A, S and M are the action, sensor and map models, respectively. More specifically: A is an action model that returns a random pose drawn from the distribution $p(x_t^{(i)} \mid$
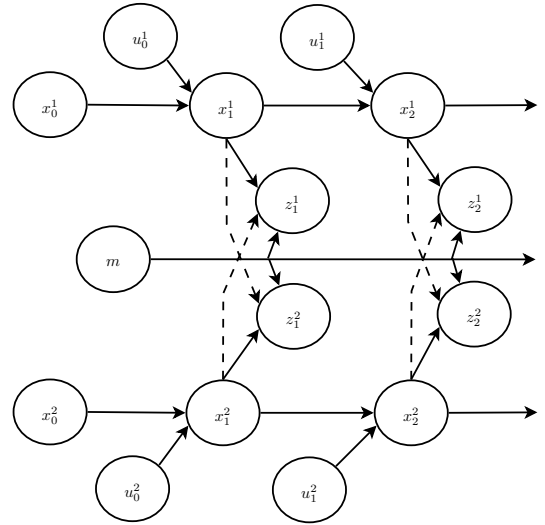

Fig. 2. Bayes net for multi-robot SLAM with known initial poses. The trajectories of robots 1 and 2 are indicated by the sequences $x_0^1, x_1^1, x_2^1, ...$ and $x_0^2, x_1^2, x_2^2, ...$, respectively. We ignore the dependencies (dashed-lines) between the observations z^2 made by robot 2 and the pose x^1 of robot 1 (and vice versa).

$x_{t-1}^{(i)}, u_{t-1})$; S is the sensor model $p(z_t \mid x_t^{(i)}, m_{t-1}^{(i)})$; and M is an incremental map generator that returns a partial occupancy grid (expressed in log-likelihood form to allow for linear superposition of grids). The basic intuition captured in these equations is that particles with self-consistent maps will be assigned larger weights than particles with inconsistent maps, and that the latter will ultimately be removed by resampling.

This approach has some valuable practical characteristics. It is a bounded-time, bounded-storage algorithm, in which processing effort and storage requirements scale linearly with particle count, but are independent of the elapsed time t. It is also very easy to implement: the sensor model S and action model A are identical to those used in the standard Monte-Carlo localization algorithm [8], and the map generator M is a simple ray-tracing algorithm. The approach does have one crucial limitation, however: the state space is extremely large (with hundreds or thousands of dimensions), while the number of particles is necessarily small (a few hundred to a few thousand at most). Thus, the filter is a very sparse sampling of the state space, and convergence is far from guaranteed; this under-sampling typically manifests itself during loop closure, when we may find that none of the particles generates a self-consistent map. In order to make this approach mananageable, one requires either very good action models [1] or very good proposal distributions [7]. In this paper, we adopt the former approach, combining odometry with laser data to produce "stabilized" odometric pose estimates (see Section III-A).

B. Multi-robot SLAM with known initial poses

The single-robot SLAM formalism can readily be generalized to handle multiple robots, provided that the initial robot poses are known. Consider a pair of robots, whose obser-

vations occur in lock-step time: $x^1_{1:t}$ denotes the trajectory for robot 1, and $x^2_{1:t}$ denotes the corresponding trajectory for robot 2. Our aim is to simultaneously estimate the posterior probability over two robot trajectories and one map. We write this down in factored form as:

$$
\begin{aligned}
p(x^1_{1:t}, x^2_{1:t}, m \mid z^1_{1:t}, u^1_{0:t-1}, x^1_0, z^2_{1:t}, u^2_{0:t-1}, x^2_0) = \\
p(m \mid x^1_{1:t}, z^1_{1:t}, x^2_{1:t}, z^2_{1:t}) \\
p(x^1_{1:t} \mid z^1_{1:t}, u^1_{0:t-1}, x^1_0) p(x^2_{1:t} \mid z^2_{1:t}, u^2_{0:t-1}, x^2_0)
\end{aligned} \tag{3}
$$

where the first term gives the distribution over maps, and the remaining terms give the distribution over possible robot trajectories. Crucially, this factorization assumes that these trajectories are independent, and that observations recorded by one robot do not depend on the pose of the other. While this assumption will not hold true in general (see Figure 2), we can *engineer* it to be true in the context of cooperative multi-robot mapping. When robots are far apart (outside sensor range or line-of-sight), there can be be no mutual dependency; when robots are nearby (within sensor range and line-of-sight), robots are able to detect one another and exclude those observations that are induced by other robots. If the robots are using laser range-finders, for example, mutual detection can be facilitated using retro-reflective targets [9], and the subset of laser rays striking other robots discarded.

The particle filter for multi-robot SLAM is constructed as follows. Each particle has properties $\langle x^{1(i)}_t, x^{2(i)}_t, m^{(i)}_t, w^{(i)}_t \rangle$, where $x^{1(i)}_t, x^{2(i)}_t$ are the instantaneous poses for robots 1 and 2, and $m^{(i)}_t$ is the common map. Given an observation tuple $(z^1_t, u^1_{t-1}, z^2_t, u^2_{t-1})$, the update step for the filter is given by:

$$
\begin{aligned}
x^{1(i)}_t = A(u^1_{t-1}, x^{1(i)}_{t-1}) \quad & x^{2(i)}_t = A(u^2_{t-1}, x^{2(i)}_{t-1}) \\
m^{(i)}_t = M(z^1_t, x^{1(i)}_t) &+ M(z^2_t, x^{2(i)}_t) + m^{(i)}_{t-1} \\
w^{(i)}_t = S(z^1_t, x^{1(i)}_t, m^{(i)}_{t-1}) &S(z^2_t, x^{2(i)}_t, m^{(i)}_{t-1}) w^{(i)}_{t-1}
\end{aligned} \tag{4}
$$

where the action model A, sensor model S and map generator M are identical to those used in the single robot case. This result will readily generalize to any number of robots.

From a practical stand-point, this approach has two important limitations. First, the size of the state space is necessarily larger than that seen in the single robot case, and the sampling is correspondingly sparse. Note, however, that dimensionality scales with total path length, such that generating a map with two robots, each of which travels 50m, is no more difficult that generating a map with one robot traveling 100m; the latter, moreover, can take half the time of the former. The second limitation is imposed by the filter resampling process: should one of the robots stop while the other continues moving, resampling may lead to particle impoverishment in the vicinity of the stationary robot. Since impoverishment can lead to filter divergence, best results are obtained when all robots are moving at comparable speeds.

C. Multi-robot SLAM with unknown initial poses

For robots deployed at widely separated locations, determining their initial poses may be impractical. In this case,
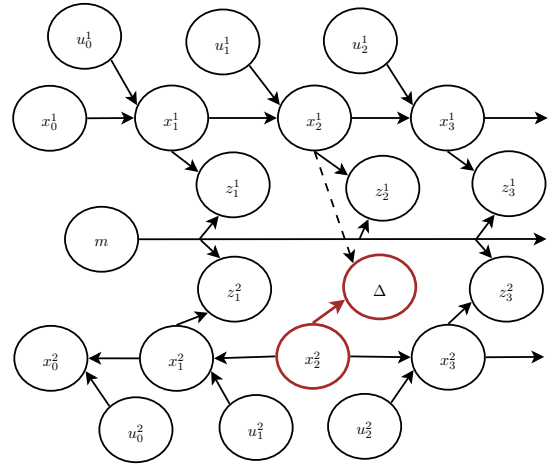

Fig. 3. Bayes net for multi-robot SLAM with unknown initial poses. The robots first encounter one another at time s, recording the relative pose Δ^2_s.

we adopt an alternative strategy, and exploit the notion of *encounters* to determine the relative pose of robots. Simply put, we start mapping with only one robot (whose initial pose is arbitrary) and wait until the first such encounter with each additional robot before incorporating data from that robot into the map. Subsequent encounters between robots are ignored.

Let Δ^2_s denote the relative pose of robot 2 as measured by robot 1 at time s. We wish to estimate the posterior over maps and trajectories given by:

$$
\begin{aligned}
p(x^1_{1:t}, x^2_{s+1:t}, m \mid z^1_{1:t}, u^1_{0:t-1}, x^1_0, z^2_{s+1:t}, u^2_{s:t-1}, \Delta^2_s) = \\
p(m \mid x^1_{1:t}, z^1_{1:t}, x^2_{s+1:t}, z^2_{s+1:t}) \\
p(x^1_{1:t} \mid z^1_{1:t}, u^1_{0:t-1}, x^1_0) p(x^2_{s+1:t} \mid z^2_{s+1:t}, u^2_{s:t-1}, x^1_s, \Delta^2_s)
\end{aligned} \tag{5}
$$

We now make two important approximations: first, we ignore the conditional dependency between the robot trajectories (see Figure 3) and treat these as independent variables; second, we assume that the uncertainty in the relative pose Δ^2_s is small. Under these approximations, the particle filter remains identical to that described in the previous section, with the exception that we initialize the robot pose $x^{2(i)}_s$ at time s using the relative pose Δ^2_s:

$$
x^{2(i)}_s = \Delta^2_s \oplus x^{1(i)}_s \tag{6}
$$

where the $\oplus$ operator indicates an appropriate 2D coordinate transform. The particle filter will incorporate all observations from robot 1, and all observations from robot 2 for times $t > s$.

One can also take this approach one step further, and incorporate observations from robot 2 that occur *prior to* the encounter at time s. To do this, we break the robot trajectory into two parts $x^2_{1:s-1}$ and $x^2_{s+1:t}$, representing the trajectory

before and after time s. The new posterior is given by:

$$
\begin{aligned}
p(x_{1:t}^1, &x_{1:t}^2, m \mid z_{1:t}^1, u_{0:t-1}^1, x_0^1, z_{1:t}^2, u_{0:t-1}^2, \Delta_s^2) = \\
&p(m \mid x_{1:t}^1, z_{1:t}^1, x_{1:s-1}^2, z_{1:s-1}^2, x_{s+1:t}^2, z_{s+1:t}^2) \\
&p(x_{1:t}^1 \mid z_{1:t}^1, u_{0:t-1}^1, x_0^1) \\
&p(x_{1:s-1}^2 \mid z_{1:s-1}^2, u_{0:s-1}^2, x_s^1, \Delta_s^2) \\
&p(x_{s+1:t}^2 \mid z_{s+1:t}^2, u_{s:t-1}^2, x_s^1, \Delta_s^2)
\end{aligned} \tag{7}
$$

In the corresponding particle filter, robot 2 now has two instances: a *causal* instance corresponding to forward motion of the robot, and an *acausal* instance corresponding to time-reversed motion of the robot. That is, each particle has properties $\langle x_t^{1(i)}, x_t^{2(i)}, \bar{x}_t^{2(i)}, m_t^{(i)}, w_t^{(i)} \rangle$, where $x_t^{2(i)}$ is the pose of robot 2 at time t and $\bar{x}_t^{2(i)}$ is the pose of robot 2 at time $2s - t$. At the first encounter, these poses are initialized using:

$$
x_s^{2(i)} = \Delta_s^2 \oplus x_s^{1(i)} \quad \bar{x}_s^{2(i)} = \Delta_s^2 \oplus x_s^{1(i)} \tag{8}
$$

Two sets of observations are given to the filter: the forward observations $(z_t^1, u_{t-1}^1, z_t^2, u_{t-1}^2)$, and the time-reversed observations $(\bar{z}_t^2, \bar{u}_t^2) = (z_{2s-t}^2, u_{2s-t}^2)$. The filter update step is thus given by:

$$
\begin{aligned}
x_t^{1(i)} &= A(u_{t-1}^1, x_{t-1}^{1(i)}) \\
x_t^{2(i)} &= A(u_{t-1}^2, x_{t-1}^{2(i)}) \\
\bar{x}_t^{2(i)} &= \bar{A}(\bar{u}_{t-1}^2, \bar{x}_{t-1}^{2(i)}) \\
m_t^{(i)} &= M(z_t^1, x_t^{1(i)}) + M(z_t^2, x_t^{2(i)}) + M(\bar{z}_t^2, \bar{x}_t^{2(i)}) + m_{t-1}^{(i)} \\
w_t^{(i)} &= S(z_t^1, x_t^{1(i)}, m_{t-1}^{(i)}) S(z_t^2, x_t^{2(i)}, m_{t-1}^{(i)}) \\
&\qquad\qquad S(\bar{z}_t^2, \bar{x}_t^{2(i)}, m_{t-1}^{(i)}) w_{t-1}^{(i)}
\end{aligned} \tag{9}
$$

where we have introduced a new time-reversed action model $\bar{A}$ (the acausal instance of the robot drives backwards rather than forwards). Note that this filter has been explicitly constructed such that the causal and acausal instances are updated at the same rate; as such, we can conceptualize the acausal instance as an additional "virtual" robot travelling backwards in time.

D. An on-line algorithm for multi-robot SLAM

Figure 4 lists the algorithm for on-line multi-robot SLAM; we will walk through this algorithm using a concrete example. Consider a group of three robots, deployed into a closed environment at randomly chosen (and unknown) initial locations. We assume that the robots broadcast action/observation pairs to one another over a reliable wireless link, and that they are capable of mutual recognition and relative pose determination (for robots that are nearby and within line-of-sight). For simplicity, we also assume that the same mapping algorithm is running on all three robots.

The robots execute individual exploration strategies, such that they are mapping their environment and occasionally encountering other robots. Let $t = a$ and $t = b$ denote the times at which robot 1 first encounters robot 2, and robot 2 first encounters robot 3. This sequence of encounters is illustrated in the *encounter diagram* in Figure 5: the diagrams show both the action/observation sequences for individual robots

```
    # Process a data from a single robot.
    # rob: id of robot generating data
    # (u, z): change in pose (odometry) and new laser scan
    # rob', Δ: id and relative pose of observed robot (if any)
0   def update(rob, u, z, rob', Δ):

        # Add data to the queue
1       append(queue[rob], (u, z, rob', Δ))

        # If the robot has joined the map...
2       if rob in joined:
3           if len(queue[rob]) > 0:

                # Do causal update
4               (t, u, z, rob', Δ) = remove(queue[rob])
5               update_filter(rob, u, z)
                # Check for causal joins
6               if rob' and rob' not in joined:

                    # Initialize filter for causal and acausal instances
7                   init_filter(rob, rob', Δ)
8                   init_filter(rob, rob', Δ)
                    # Split the queue into causal and acausal components
9                   queue[rob'] = reverse(queue[rob'][0 : t − 1])
10                  queue[rob'] = queue[rob'][t + 1 :]
                    # Add robot to joined list
11                  append(joined, rob')

12          if len(queue[rob]) > 0:

                # Do acausal update
13              (t, u, z, rob', Δ) = remove(queue[rob])
14              update_filter(rob, −u, z)
                # Check for acausal joins
15              if rob' and rob' not in joined:

                    # Initialize filter for causal and acausal instances
16                  init_filter(rob, rob', Δ)
17                  init_filter(rob, rob', Δ)
                    # Split the queue into causal and acausal components
18                  queue[rob'] = reverse(queue[rob'][0 : t − 1])
19                  queue[rob'] = queue[rob'][t + 1 :]
                    # Add robot to joined list
20                  append(joined, rob')

21  return
```

Fig. 4. Update algorithm for on-line multi-robot SLAM (expressed in Python pseudocode). In addition to the particle filter, there are two basic data structures: a set $joined$ listing the robots whose data is included in the map (initialized to contain exactly one robot), and a pair of queues $(queue[rob], \overline{queue}[rob])$ for each robot (all queues are initially empty). The function **update_filter()** is described by Equation 4.

(solid lines) and the encounters between robots (dotted lines); time increases from left to right. We will use this diagram to show how observations/action pairs from all three robots are ultimately incorporated into a single map.

Consider robot 1: since this robot does not know the initial pose of the other two robots, it performs single robot SLAM using its own action/observation sequence (the filter contains particles $\langle x^1, m, w \rangle$ describing the map and the pose of robot 1). Algorithmically, observations from robot 1 are appended to a queue (line 1), from which they are subsequently removed and used to update the filter (lines 4–5). Since the observations
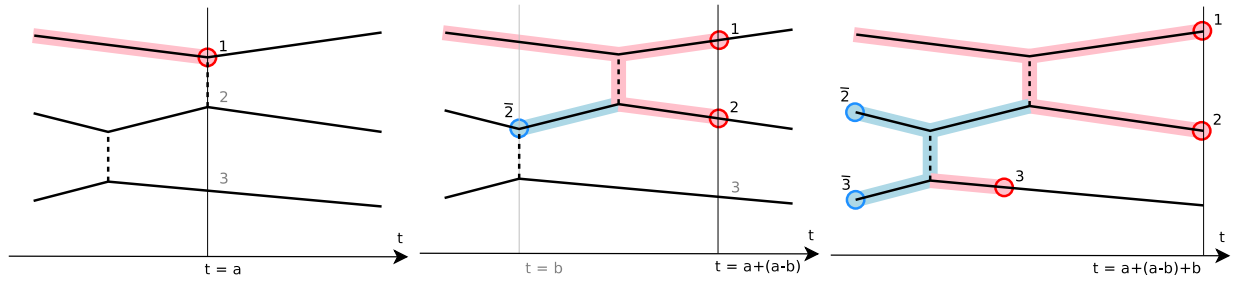
Fig. 5. *Encounter diagram* for an imaginary three robot experiment. The solid horizontal lines denote the sequence of observations recorded by each robot; the dotted vertical lines denote robot-robot observations at times a (robot 1 observes robot 2) and b (robot 2 observes robot 3). The shaded areas indicate observations that have been processed through the filter and incorporated into the map.

from robots 2 and 3 cannot yet be incorporated into the map (their relative initial pose is unknown), these observations are simply appended to individual queues (line 1). Any encounters between robots 2 and 3 (such as the one occurring at time $t = b$) are similarly queued.

At time $t = a$, robots 1 and 2 encounter and recognize one another, and robot 1 is able to determine the relative pose of robot 2 (line 6). The particle filter is augmented with two additional state variables $\langle x^2, \bar{x}^2 \rangle$ representing both causal and acausal instances of robot 2. The poses are initialized on a per-particle basis (lines 7–8) according to Equation 8. This initialization step ignores any uncertainty (assumed to be small) associated with the relative pose measurement. The queued data for robot 2 is now split into two (lines 9–10): the causal queue contains all of the data recorded after time $t = a$ (an empty queue, in this case), and the acausal queue contains all of the data recorded before time $t = a$ (all of the data to date). In subsequent calls to the update function, the causal queue (whose length remains fixed) is used to update the causal instance x^2 (lines 4–5), while the acausal queue (whose length decreases monotonically) is used to update the acausal instance $\bar{x}^2$ (lines 13–14). The first two encounter diagrams in Figure 5 illustrate this process, with the shaded areas indicating observations that have been incorporated into the map. The use of the acausal instance $\bar{x}^2$ allows us to include all of the observations occurring prior to the first encounter between robots 1 and 2.

At time $t = a + (b - a)$, an encounter between robots 2 and 3 is popped from robot 2's acausal queue (this is an encounter that occurred *in the past* at time $t = b$). The particle filter is augmented with the state variables for robot 3 $\langle x^3, \bar{x}^3 \rangle$, which are initialized on a per-particle basis using the pose of the acausal instance of robot 2 (lines 16–17). The queue for robot 3 is split into causal and acausal components (lines 18–19), and subsequent updates incorporate data from both queues into the map. This process is illustrated in the last pair of diagrams in 5; note that, for robot 3, the causal queue is always non-empty, and causal updates for this robot lag behind those for robots 1 and 2.

It should be apparent from this example that robot 1 can fuse data from all three robots into a single map. Similar computations on the other two robots will produce comparable

(but not necessarily identical) maps.

E. Discussion: On-line algorithms, complexity and latency

Two questions naturally arise regarding this algorithm: first, in what sense is this an 'on-line' algorithm (there may be significant latency), and second, why are old observations (stored in the queues) processed at the same rate as new observations (gathered from sensors or received from other robots)? The answers to these two questions are closely intertwined.

As constructed, the algorithm has bounded time update steps: the worst-case effort required in each cycle is $O(nm)$, were n is the number of robots and m is the number of particles. This is a direct consequence of the fact that observations (both new and old) are always processed at the same rate. The drawback of this approach, however, is latency: in the encounter diagram shown in Figure 5, for example, the observations for robot 3 lag $2(a - b)$ seconds behind those for robots 1 and 2. In the worst case, the latency can be as high as $(n - 1)t$ seconds, where n is the number of robots.

To illustrate the alternative approach, consider again the encounter between robots 2 and 3 shown in Figure 5. At this time, one could step through all of the stored observations for robot 3, updating the filter with each and every observation, and entirely eliminating the latency. We choose not to do this for two reasons. First, this alternative approach no longer has bounded-time update steps (worst-case effort is now $O(nmt)$) and a single robot encounter may cause the algorithm to "pause and think" for a very long time. Second, the re-sampling induced by multiple filter updates tends to impoverish the sample set in the vicinity of the remaining robots (recall the discussion in Section II-B); as a consequence, more particles must be used to ensure filter convergence.

The algorithm, as presented, is well suited for use in the robot control loop (so long as one is careful not to confuse real robots with their virtual twins). We can, for example, use the incomplete map to navigate, find frontiers for exploration [10], [11], or plan a rendezvous with another robot [12], [13]. It is in this sense that we use the term *on-line* algorithm.

III. IMPLEMENTATION AND EXPERIMENTATION

We turn now to matters of practical implementation, and discuss three key system components: laser-stabilized odometry,

single-robot particle-filter-based SLAM, and its generalization to multiple robots.

A. Laser-stabilized odometry

As noted in Section II-A, PF-based mapping algorithms require very accurate action models, and experience demonstrates that raw odometry is not sufficient for this purpose. Instead, we make use of *laser-stabilized odometry*, in which laser range data is used to correct the raw odometry estimate; this combined *lodometric* pose estimate exhibits a drift rate that is an order of magnitude less than that observed using odometry alone.

The basic algorithm for laser-stabilized odometry is straight-forward: the robot maintains a local map generated from recent laser scans, and each new scan is first fitted against and then added to this map; the lodometric pose estimate is derived from the fitted scan pose. Since the map is strictly local (both spatially and temporally), these estimates have the same basic properties as odometry – an arbitrary origin and cumulative drift – but are much more accurate. Figure 6, for example, shows an error scatter plot for odometric and lodometric pose estimates; each point shows the difference between the estimated change in pose (over some interval) and the ground-truth change in pose (as determined by the global SLAM algorithm described in the next section). The errors in the lodometric estimates are significantly less than those for raw odometry, particularly for changes in robot orientation. In the experiment used to generate this plot, the robot travelled 125 m and executed 19 complete rotations before returning to its starting location; using laser-stabilized odometry, the final orientation error was less than 5 degrees, versus 110 degrees for pure odometry.
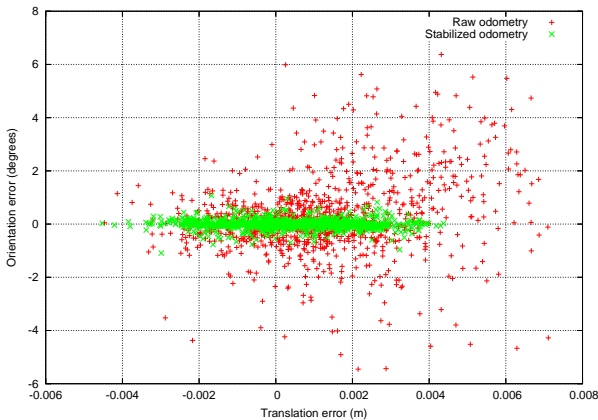

Fig. 6. Scatter plot of odometry and laser-stabilized odometry; each point shows the difference between the estimated change in pose (over a short interval) and the true change in pose (as determined by the single-robot SLAM algorithm).

Note that the use of lodometric pose estimates for mapping does lead to some double counting of data: laser scans are used once to generate lodometric estimates, and a second time to generate the map. In practice, this double counting has

minimal impact, as the mapping algorithm uses only a small fraction of the available laser scans.
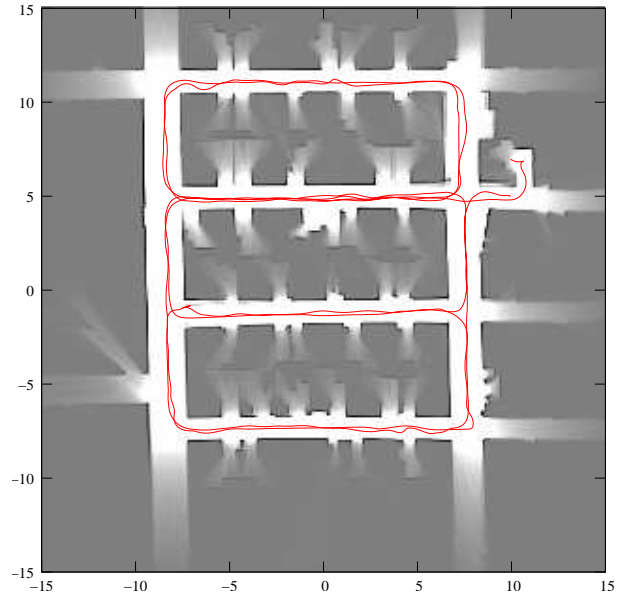

Fig. 7. Map generated using the single-robot algorithm; the map is 30 m by 30 m with a resolution of 0.10 m.

B. Single-robot SLAM

The single-robot SLAM algorithm contains many elements of the standard Monte-Carlo localization algorithm [8] and is relatively easy to implement. The key challenge lies in maximizing the per-particle update speed while minimizing the corresponding storage requirements, so that the filter may run in real time and in bounded memory with a relatively large number of particles. As always, the speed and storage demands tend to conflict, and our implementation favors the former over the latter.

For each particle, we maintain a complete occupancy grid map, generally with a resolution of 0.10 m and covering an area of between 400 and 2500 m^2. With an 8-bits-per-cell representation, this implies that each particle requires up to 0.25 Mb of storage, or 25 Mb for a filter with 100 particles. Such storage capacities are readily available on contemporary workstations and high-end embedded systems. The filter update step requires two ray-tracing operations on the occupancy grid for each and every particle: one to evaluate the sensor model and another to update the map. Since these operations are expensive, we approximate the ray-tracing step by considering only the ray end-points, and decimate the laser scans by using only one scan for every 0.10 m of distance travelled. These approximations improve processing speed by an order of magnitude or more, thereby allowing real-time operation.

Figure 7 shows a typical map generated by the single-robot algorithm, with all three loops correctly closed. Processing time for this map is 173 s on a 2.4GHz P4 workstation using 200 particles (versus 865 s to acquire the data). The top speed of the robot was 0.20 m/s.

Fig. 8. Combined map for all four robots using the multi-robot SLAM algorithm. One pair of robots starts from the right-top room, the other from the right-bottom corridor. The map is shown overlaid on the 'ground-truth' blueprint.
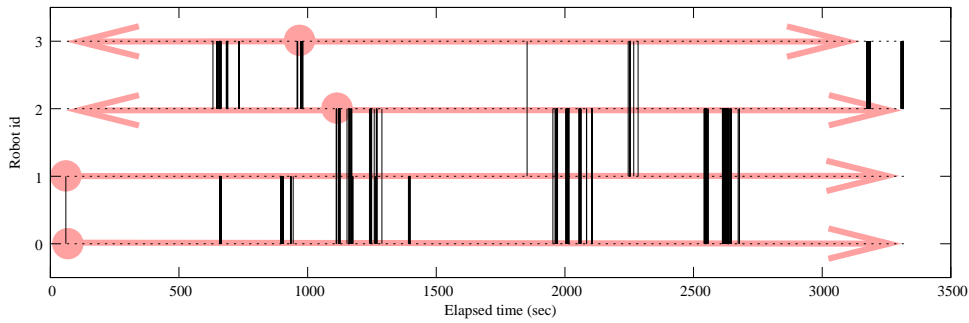

Fig. 9. Encounter diagram: each horizontal line represents a robot, each vertical line denotes an encounter between robots (e.g., robot 0 encounters robot 2 at time $t = 1100$ sec). The 'bunching' of vertical lines indicates multiple encounters occurring over a short period of time. Shaded circles indicate the set of encounters used to merge data from all four robots.

Fig. 10. Individual maps for two of the four robots, generated using the single-robot algorithm.

C. Multi-robot SLAM

The multi-robot SLAM implementation has a great deal in common with the single-robot implementation described in Section III-B. Each particle stores an occupancy grid and a set of pose estimates: for a team of n robots, there are $2n$ such estimates, representing both causal and acausal instances of each robot. Importantly, the filter does not distinguish between these two, and acausal instances are simply treated as addi-

tional robots that happen to be driving backwards. The multi-robot algorithm also uses a number of queues for managing data from the robots; these are implemented using a set of files (one for each robot) to which observations are appended. Causal updates are generated by seeking forwards through the file; acausal updates are generated by seeking backwards. Compared with the single-robot algorithm, the multi-robot algorithm has identical in-memory storage requirements (since

all robots share the same map) and requires worst-case effort that is $2n$ times that seen in the single-robot case. It is likely that the filter also requires more particles to ensure convergence, but we have not yet quantified this result.

Figure 8 shows the results produced by the multi-robot algorithm for an autonomous exploration task. Four robots were deployed into this environment at two distant locations, from which they executed a cooperative, but largely reactive, exploration strategy. All four robots were identically equipped, with odometry, scanning laser range-finders and retro-reflective markers (for mutual recognition).

The sequence of events in this experiment can be read from the encounter diagram in Figure 9.

- The filter is 'seeded' with a causal instance for robot 0, which immediately encounters, and generates a casual instance for, robot 1. These two robots start from adjacent locations in the bottom-right corridor. The map is subsequently updated using causal data from both robots.
- At time $t = 1100$, robot 0 encounters robot 2, generating both causal and acausal instances for the latter. Subsequent map updates combine data from four sources: the causal instances of robots 0 and 1, and the causal and acausal instances of robot 2.
- At time $t = 1250$, the acausal instance of robot 2 encounters robot 3; i.e., we take note of an event that actually occurred 300 seconds *in the past*, at time $t = 950$. This encounter generates both causal and acausal instances for robot 3.
- The experiment proceeds until time $t = 3600$, at which time all of the data from all of the robots has been captured into a single map, and robots 0, 1 and 2 have returned to their initial locations (robot 3 failed to return due to a communication failure).

It should be noted that robot 0 never encounters robot 3 during the course of this experiment. The data from the latter robot is included entirely as a result of a transitive encounter: robot 0 encounters robot 2, and the acausal instance of robot 2 encounters robot 3.

In the final map, all of the major topological features have been correctly extracted, and, with the exception of a single scan mis-alignment on the bottom right corridor, the quality of the map is uniformly high. The minor differences between the map and the blueprints are primarily due to errors in latter. Processing time for this map is 2940 s on a 2.4GHz P4 workstation using 500 particles (versus 3600 s to acquire the data). For comparison purposes, Figure 10 shows the individual maps generated using the single-robot SLAM algorithm for two of the four robots used in this experiment; both maps are incomplete, which is the expected result for a cooperative exploration task.

IV. DISCUSSION AND CONCLUSION

One of the attractive features of this multi-robot SLAM algorithm is the ease with which it may be implemented. The basic elements of the algorithm – sensor and action models, occupancy grids and ray-tracing – are easily adapted from Monte-Carlo localization, and while acausal filtering does require some additional book-keeping, the overall complexity is low. If desired, one can further improve the quality of the map by post-processing with a local constraint algorithm in the style of Lu and Milios [14].

There are several possible enhancements to the algorithm as presented, of which we will mention just one. The current sensor model assumes that each robot's observations are independent of the pose of all the other robots, and we take care in the implementation to ensure that this assumption is always satisfied (by selectively discarding range data, for example). For those robots that are already in the map, however, we may be able to relax this assumption, and condition the observations of one robot on the pose of the others. This dependency could be usefully exploited during loop closure, as the robots themselves would form a unique and unambiguous set of landmarks.

REFERENCES

[1] D. Hähnel, W. Burgard, D. Fox, *et al.*, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, U.S.A, Oct 2003.

[2] S. Thrun, "Robotic mapping: A survey," in *Exploring Artificial Intelligence in the New Millenium*, G. Lakemeyer and B. Nebel, Eds. Morgan Kaufmann, 2002.

[3] A. Howard, "Multi-robot mapping using manifold representations," in *IEEE International Conference on Robotics and Automation*, New Orleans, Louisiana, Apr 2004, pp. 4198–4203.

[4] K. Konolige, D. Fox, C. Ortiz, *et al.*, "Centibots: Very large scale distributed robotic teams," in *9th International Symposium on Experimental Robotics (ISER-04)*, Singapore, Jun 2004.

[5] K. Murphy, "Bayesian map learning in dynamic environments," in *Neural Info. Proc. Systems (NIPS '99)*, 1999, pp. 1015–1021.

[6] M. Montemerlo, S. Thrun, D. Koller, *et al.*, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *Proceedings of the AAAI National Conference on Artificial Intelligence*. Edmonton, Canada: AAAI, 2002.

[7] G. Grisetti, C. Stachniss, and W. Burgard, "Improving grid-based SLAM with Rao-Blackwellized particle filters by adaptive proposals and selective resampling," in *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2005.

[8] S. Thrun, D. Fox, W. Burgard, *et al.*, "Robust Monte Carlo localization for mobile robots," *Artificial Intelligence Journal*, vol. 128, no. 1–2, pp. 99–141, 2001.

[9] A. Howard, L. E. Parker, and G. S. Sukhatme, "The SDR experience: Experiments with a large-scale heterogeneous mobile robot team," in *9th International Symposium on Experimental Robotics 2004*, Singapore, Jun 2004.

[10] B. Yamauchi, A. Shultz, and W. Adams, "Mobile robot exploration and map-building with continuous localization," in *Proceedings of the 1998 IEEE/RSJ International Conference on Robotics and Automation*, vol. 4, San Francisco, U.S.A., 1998, pp. 3175–3720.

[11] W. Burgard, M. Moors, D. Fox, *et al.*, "Collaborative multi-robot exploration," in *Proc. of IEEE International Conferenceon Robotics and Automation (ICRA)*, vol. 1, 2000, pp. 476–81.

[12] N. Roy and G. Dudek, "Collaborative robot exploration and rendezvous: Algorithms, performance bounds and observations," *Autonomous Robots*, vol. 11, no. 2, pp. 117–136, 2001.

[13] J. Ko, B. Stewart, D. Fox, *et al.*, "A practical, decision-theoretic approach to multi-robot mapping and exploration," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, Las Vegas, Nevada, U.S.A, Oct 2003.

[14] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, pp. 333–349, 1997.