Encounter Based Multi Robot Simultaneous Localization and Occupancy Grid Mapping

RB Choroszucha[1], C Hyman[2], and A Collier[3]

***Abstract*— In this paper we present and implement the algorithm in Howard's 2006 paper: "Multi-robot simultaneous localization and mapping using particle filters." While Howard's method is attractive for its speed, it combines the forward and reverse particle weights, which can result in undesired particle error. In this paper we propose a modification of the algorithm by using independent particle filters for each robot, and demonstrate that the modified algorithm provides a qualitatively better map, and a more accurate pose estimate.**

I. Introduction

The automated exploration of unknown environments has become one of the foremost challenges in mobile robotics. For a robot to explore an environment, it must map the environment and concurrently localize itself within said environment. The framework used to perform this task is known as simultaneous localization and mapping (SLAM) and has been well covered in the literature using a variety of techniques [4], [1].

While SLAM is well known and has a rich history of successes using a single robot, it can often be a slow process due to both constraints on the robot, such as speed and data processing, and lack of redundancy, i.e. robot failure [11], [3]. To address the speed of mapping and to add redundancy, coordinated or multi-robot SLAM (MRSLAM) was created.

MRSLAM is as it sounds, SLAM using multiple exploring/mapping robots. This approach allows for the partition of the physical search space using independent robots, typically decreasing the time it takes to map an area, and increasing the likelihood of full map coverage in the event of robot failure. This temporal exploration parallelism does, however, come at the cost of added complexity. The added complexity of MRSLAM comes in two major components: coordination of exploration using multiple search agents and merging the maps of these agents [5]. Coordinated exploration consists of planning the groups' search path, most often formulated to cover the most search space in minimum time or to optimize some other mapping cost criteria. The other major component, map merging, is the combination of individual robot's observations and maps into one cohesive global map estimate.

In this paper we use the map merging algorithm presented in Howard's 2006 paper: "Multi-Robot Simultaneous Localization and Mapping using Particle Filters" [7]. This paper is of particular interest as it presents a resource efficient, online solution to the MRSLAM map merging problem given unknown initial robot poses.

The algorithm is an encounter-based algorithm. Given that we start with one mapping robot (call this Robot 1), when it encounters a non-mapping robot (Robot 2), the relative pose between the two robots is computed and Robot 2's odometry and measurement data is transmitted in reverse order to Robot 1 and integrated to create a single map posterior.

Fig. 1 depicts the process with Robot 1's triple $(\mathbf{x}_t^1, u_{t-1}^1, z_t^1)$, and robot 2's triple $(\mathbf{x}_t^2, u_{t-1}^2, z_t^2)$, building a single global map posterior m. When $t = 2$, Robot 1 observes Robot 2 and determines the relative pose Δ, then the past odometry and scan data is integrated in reverse order.

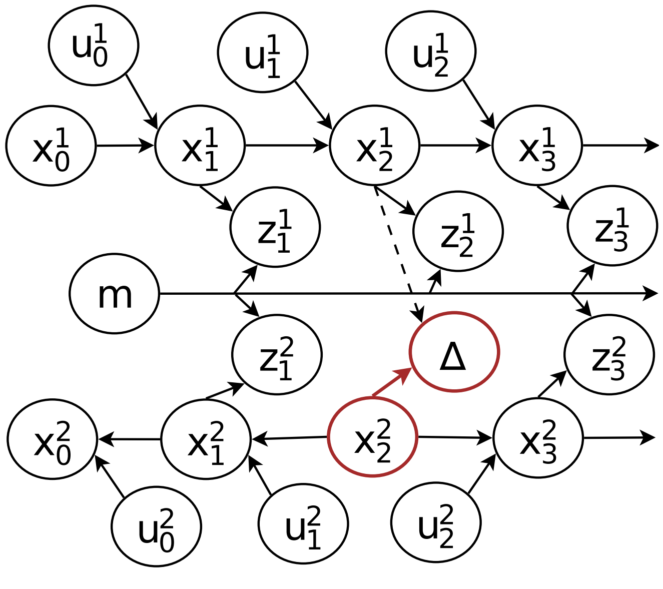


Fig. 1. Depiction of data integration. Taken from [7].

While Howard's method is attractive for its speed, it combines the forward and reverse particle weights, which can result in undesired particle error. In this paper we propose a modification of the algorithm by using independent particle filters for each robot, and demonstrate that the modified algorithm provides a qualitatively better map, and a more accurate pose estimate.

The paper is structured as follows: In §II some background about MRSLAM is provided, the contribution of [7] is discussed and a modified scheme is presented. In §III the modification of Howard's algorithm [7] is discussed. In §IV the generation of arbitrary data sets, the average time it takes to map 95% of the environment, and a comparison

[1] RB Choroszucha is with the School of Naval Architecture and Marine Engineering, University of Michigan. E-mail: riboch@umich.edu

[2] C Hyman is with the School of Electrical Engineering and Computer Science, University of Michigan. E-mail: hymanc@umich.edu

[3] A Collier is with the School of Electrical Engineering and Computer Science, University of Michigan. E-mail: amcollie@umich.edu

of Howard's algorithm and the modified algorithm are presented. §V rounds out the paper with conclusions and future work.

II. Background

A. State of the Art

In the literature there are two problems that are often addressed when dealing with MRSLAM:

1) robot coordination, i.e., how to cover the most area given an unknown environment [8], and
2) merging the data to create one global map posterior, which is what we will focus on.

In the case where all relative robot poses are known, merging maps is a trivial problem using small modifications to existing SLAM techniques [11]. In the general MRSLAM problem, however, robots may start with unknown absolute and relative poses, and therefore merging of maps requires the discovery of relative relationships between different robot trajectories to build a single map. This is often a costly process, which in general can be solved for robots sharing a search space by estimating each robot's relative pose given a partial map, but this leads to exponential complexity with respect to the number of exploring robots [5]. Nevertheless, several practical algorithms exist to circumvent this naive and inefficient approach, including coarse topological matching and stitching techniques borrowed from computer vision [2].

In 2006 two schools of thought arouse about how to handle MRSLAM:

1) Let independent robots build individual maps, then at the end combine the maps together [2]. This approach is particularly attractive for post processing because it requires multiple trial and error to maximize a score function.
2) On the other hand, there was Howard's paper [7], which uses communication between robots, but requires precisely known relative poses to construct a single global map posterior.

Both had their benefits: [2] does not require knowledge of any global or relative poses, but [7] can be implemented in real-time. Because of the real-time nature, and the desire to map as fast as possible, there has also been a great deal of research using [7], but with the added constraint of limited communication [9].

In the following subsection we will propose yet another extension to [7].

B. Extension of [7]

Define the weights for a particle filter: w_t^i, and the sensor model, $p(z_t|\mathbf{x}_t, u_{t-1})$ for the forward and reverse model. In [7], Howard follows the typical RBPF formulation and defines the un-normalized weight update to be:

$$w_t^{(i)} = p(z_t^f|\mathbf{x}_t^{f,((i))}, m_{t-1}^{(i)})p(z_t^r|\mathbf{x}_t^{r,(i)}, m_{t-1}^{(i)})w_{t-1}^{(i)}, \quad (1)$$

where $\bullet^{f,(i)}$ and $\bullet^{r,(i)}$ denote the forward and reverse elements of particle (i), $\mathbf{x}_t$ denotes the pose, z_t denotes the measurement, and m_{t-1} denotes the map..

In this formulation it is no stretch of the imagination that the forward sensor model and previous weight $p(z_t^f|\mathbf{x}_t^{f,(i)}, m_{t-1}^{(i)})w_{t-1}^{(i)}$ could be large, denoting a good match to the data in the forward direction, yet the reverse sensor model, $p(z_t^r|\mathbf{x}_t^{r,(i)}, m_{t-1}^{(i)})$, could be small. Thereby reducing the probability that the best forward direction is resampled. See for example Fig. 2, particularly particle 1.

Because of this, we propose assuming that the noisy forward and reverse motions are independent, motivates using two particle filters with weights defined by:

$$w_t^{f,(i)} = p(z_t^f|\mathbf{x}_t^{f,(i)}, m_{t-1}^i)w_{t-1}^{f,(i)} \quad (2)$$
$$w_t^{r,(i)} = p(z_t^r|\mathbf{x}_t^{r,(i)}, m_{t-1}^i)w_{t-1}^{r,(i)} \quad (3)$$

and with $\{\mathbf{x}_t^{f,(i)}, m_{t-1}^i\}$ being resampled using $\{w_t^{f,(i)}\}$, and $\{\mathbf{x}_t^{r,(i)}\}$ being resampled using $\{w_t^{r,(i)}\}$.

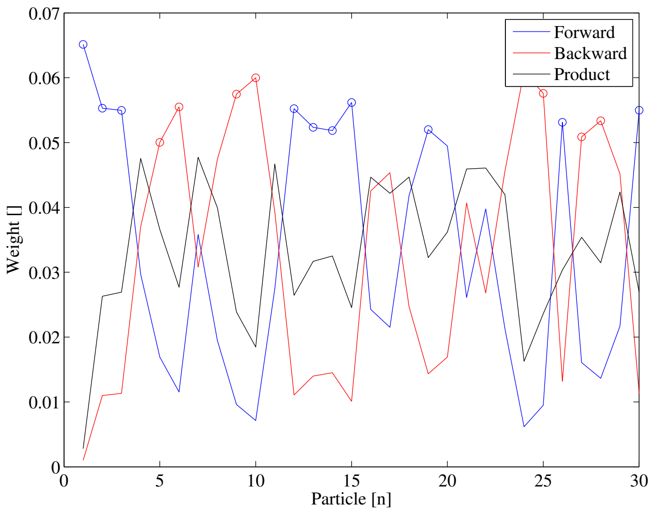


Fig. 2. Case where the best forward and reverse poses are not chosen. $w_{t,f}^i$ is the forward weight, $w_{t,r}^i$ is the reverse weight, and w_t^i is the product of the weights.

Our contribution is two-fold, verifying that Howard's algorithm works, and creating independently sampled particle filters for each mapping robot in the aim to build a more accurate occupancy grid, and obtain a better localization within the map while only marginally increasing the computational complexity.

III. The Algorithm

A. Overview

Howard's multi-robot SLAM algorithm generates a single map and pose posterior much in the same way an occupancy grid map SLAM algorithm based on a Rao-Blackwellized particle filter (RBPF) would, but with additions to accommodate multiple robots. Mapping begins with a set consisting of a single robot with known pose and an occupancy grid map of the environment is built as this robot traverses and measures it. The ultimate goal of the algorithm is to simultaneously compute for time t, the full SLAM posterior containing all robot pose trajectories $x_{1:t}^i$ and the global map

$m_{1:t}$, given only a single known initial pose and sets of measurements

$$p(x^1_{1:t}, x^2_{1:t}, ..., x^M_{1:t}, m_{1:t}| \\ x^1_0, z^1_{1:t}, u^1_{0:t}, \Delta^2_s, z^2_{1:t}, u^2_{0:t}, \Delta^3_s, ..., \Delta^M_s, z^M_{1:t}, u^M_{0:t})$$

The odometry and measurement data of all other robots is stored as it is collected, as it cannot contribute to the global map without a known relative pose linking it to the frame of the first robot. As the first robot encounters additional robots via a mutual pose observation, the newly observed robot is added to the set of mapping robots and its actions and the map is sequentially conditioned by its actions and measurements. From the point of observation, all future actions and measurements of the new robot are used to condition the map posterior, and likewise all of its previously stored actions and measurements are played back in reverse order from that point as a virtual robot. This information is then passed to a pair of new particle filters for the pose of these robots that contribute to the same global map. Figure 3 gives a graphical example of an encounter where a causal and acausal virtual robot are spawned from a robot encounter.

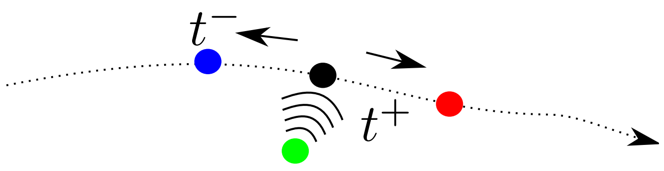


Fig. 3. Causal and Acausal Virtual Robots Added on Encounter

As further encounters occur between any robot, real or virtual, in the mapping set and a previously unseen robot, the previously unseen robot is added to the set of mapping robots and its measured relative pose is again used to establish a reference point with which its actions and measurements can condition the map posterior. Further mutual observations of two robots already in the mapping set are ignored for the sake of simplicity. This encounter-add process is continued recursively for all robots until all stored and future odometry and measurements are exhausted or all possible mapping robots have made an encounter with the mapping set. Beyond this point, the algorithm behaves as a normal RBPF with a stacked state containing pose posteriors of all robots in addition to the map.

This algorithm relies on a number of assumptions that are requisite for it to effectively solve the MRSLAM problem. Firstly, for all explored regions to count towards the map, each mapping robot must have encountered a robot that is an element of the mapping set in order to make a fully connected graph of robot poses, and therefore maximally complete map. Additionally, each robot pose trajectory is independent of all other trajectories such that motion or observation from one robot does not affect another outside of encounter events [7].

B. Algorithm Details

Our implementation of Howard's MRSLAM algorithm follows the basic structure of the below steps that perform data queueing, robot encounter management, and FastSLAM filter updates.

1) Queue measurement, odometry, and mutual observation (encounter) data
2) For all robots in the mapping set, take first element from queue and update SLAM filter
3) For all new encounters, split encountered queue at present time into causal and acausal queues and instantiate a new set of particles for the causal and acausal trajectory of the observed robot.
4) For all available acausal queues, remove last element and use it to update the SLAM filter for the corresponding acausal robot
5) If data includes a prior encounter with a robot not in the mapping set, instantiate a new set of particle for this robot and its acausal virtual self at its location and split the queue for that robot into a causal and acausal queues
6) Resample all particles to prevent weighting degeneracy
7) Repeat until all data is consumed

1) Data Queueing: As each robot R collects data, its current odometry u^R_t and measurements z^R_t are added to a running queue data structure before processing. An encounter flag E^R_t and relative pose Δ^R_t, if available, are also added to this queue.

$$queue^R_{0:t} = append(queue^R_{0:t-1}, \{u^R_t, z^R_t, E^R_t, \Delta^R_t\})$$

The storing of this data allows for either immediate use by particle filters for robots in the mapping set, or later joining or SLAM filtering after a future encounter with a mapping robot.

2) Encounters: An encounter in this sense is defined as the observation of one robot by another mapping robot that produces a relative pose estimate between the robots of Δ. For this implementation, it was assumed that the estimation of Δ was deterministic for the sake of demonstration and simplicity.

Upon an encounter at time s between robot A in the mapping set with a pose x^A_s and a robot B not in the set with a measured relative pose Δ^{AB}_s, a reference frame pose for each particle i of the newly observed robot is instantiated as causal particle posterior $x^{B(i)}_s$ and acausal particle pose posterior $\bar{x}^{B(i)}_s$ as:

$$x^{B(i)}_s = x^{A(i)}_s \oplus \Delta^{AB}_s$$
$$\bar{x}^{B(i)}_s = x^{A(i)}_s \oplus \Delta^{AB}_s$$

Where $\oplus$ is the pose composition operator [10]. At this time, the stored data queue for robot B is also split into a causal queue $queue^B$ and acausal queue $\overline{queue}^B$.

$$\overline{queue}^B_{t+1:2t} = reverse(queue^B_{0:t-1})$$
$$queue^B = queue^B_{t+1:}$$

If neither robot is within the mapping set and an encounter occurs, this encounter and the associated measured relative pose is stored for later use. In the event that one of these robots later encounters a robot in the mapping set, its virtual acausal robot will spawn another robot pair of the unencountered robot upon backtracking to the location of this initial encounter. At this acausal encounter, the same method of adding relative pose and splitting the queues used in the causal join is used to generate the causal and acausal virtual robots created by this encounter.

Figure 4 explains the joining process graphically. The scenario starts with three robots x^1, x^2, and x^3, with x^1 being the sole robot in the initial mapping set. The actual robots are represented as white circles, the causal particles as red circles, the acausal particles as blue circles, and encounters as purple arrows. At timestep T_1, an encounter occurs between x^2 and x^3, but neither is in the mapping set at this time, so this encounter is stored. At timestep 2, robots x^1 and x^2 encounter each other, spawning a causal and acausal pair of pose particles x^{2+} and x^{2-} respectively. x^{1+} and x^{2+} continue propagating forward with new data (not shown) while x^{2-} propagates backwards in time using stored odometry and measurements. The stored encounter between x^2 and x^3 that was previously stored is then incorporated at time T_3 when the acausal particles x^{2-} reach the location of the encounter. This spawns a set of particles for the previously encounter robot x^3. As the original encounter happened on the first step, no acausal particles are spawned at this time. As x^{2-} reaches the last remaining pose in the acausal queue, the particles are removed as the past data is exhausted.

3) FastSLAM Updates: At each time-step, the pose posterior is updated for all pose states within the current mapping set are updated according to a modified Rao-Blackwellized particle filter. Both information from the causal and acausal queues is used to update the SLAM posterior in an almost identical fashion, save for a time-reversed odometry model.

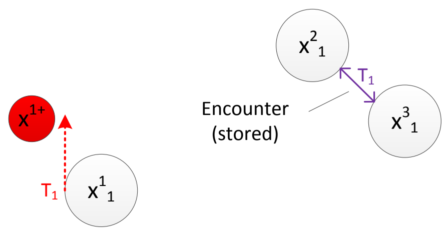
This process first applies an odometry prediction to each particle. This single update applies to each causal robot particle a forward odometry change given an arbitrary non-linear and noisy forward odometry model $g(x, u)$. For each virtual robot traversing its path in reverse, the next stored point is popped off the corresponding stack and used with a noisy inverse odometry model $\hat{g}_r(x, \bar{u})$

$$\begin{aligned} &\forall robots\ r \\ &\quad x_{t+1}^{r(i)} = \hat{g}(x_t^{r(i)}, u_{t+1}^{r(i)}) \quad \text{(Causal Prediction)} \\ &\quad \bar{x}_{t+1}^{r(i)} = \hat{g}_r(\bar{x}_t^{r(i)}, \bar{u}_{t+1}^{r(i)}) \quad \text{(Acausal Preditiction)} \end{aligned}$$

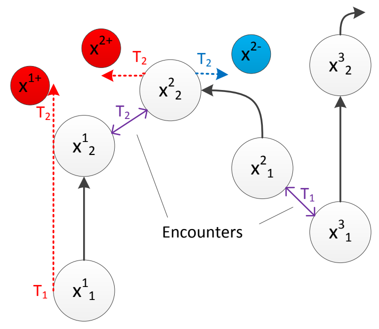
Where $\hat{g}$ and $\hat{g}_r$ consist of a deterministic odometry model $g(x, u)$ or $g_r(x, \bar{u})$ corrupted by model dependent noise η.

$$\hat{g}(x, u) = g(x, u) + \eta \tag{4}$$
$$\hat{g}(x, \bar{u}) = g_r(x, \bar{u}) + \eta \tag{5}$$

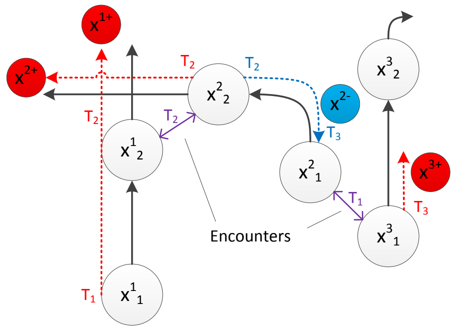
The chosen particle filter implementation allows for a wide choice of assumptions on the distribution of the noise model. Our choice of model used in testing is discussed in the later section on experimental validation.



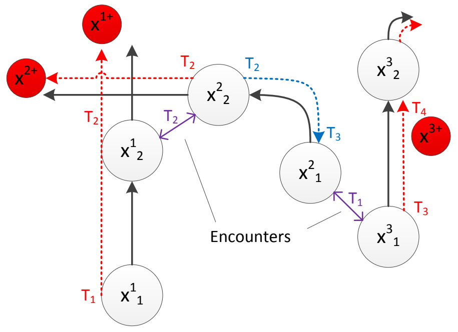
(a) Timstep 1



(b) Timestep 2



(c) Timestep 3



(d) Timestep 4

Fig. 4. Example of Robot Encounters in this MRSLAM Implementation

The odometry prediction step was followed by an occupancy grid map update for each real or virtual robot particle based on current or stored measurements. For each robot R, whether it represents causal or acausal particles, the next measurement in the queue, z_t^R is removed and used to update the map for each particle according to a simplified occupancy grid measurement model.

For each particle in x_t^R, the particle's associated map m_i, and associated measurement z_t^R is used with the simplified inverse sensor model to compute the log-odds of a particular cells occupancy. Our chosen model simulated a typical range,bearing LIDAR type sensor, with a fixed occupied (black) and free (white) probability p_{occ} and p_{free} respectively and simple sensor cone model similar to the one shown in Figure 5. This created the log-odds modifier at time t

$$l_{sensor,t,i} = \begin{cases} \log \frac{p_{free}}{1-p_{free}} & \text{Cell free} \\ \log \frac{p_{occ}}{1-p_{occ}} & \text{Cell Occupied} \end{cases}$$

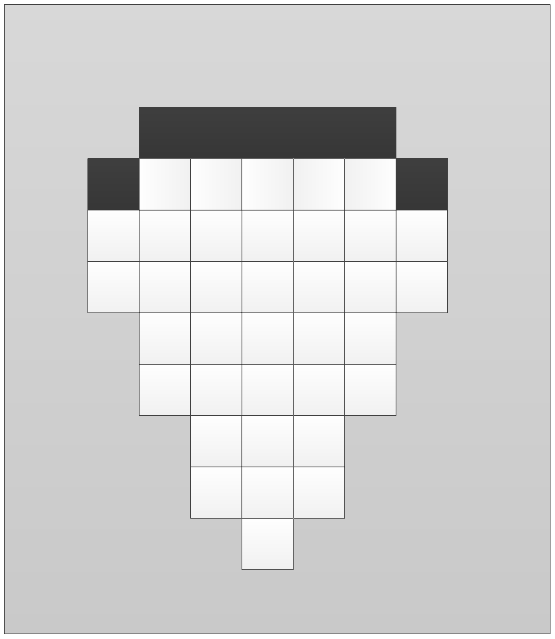


Fig. 5. Example Inverse Sensor Model Pattern

For each cell augmented by the inverse sensor model, the log-odds was updated as:

$$l_{t,i} = l_{t-1,i} + l_{sensor,t,i} - \log \frac{p(m_i)}{1-p(m_i)}$$

All cells uneffected were left at their previous log-odds

$$l_{t,i} = l_{t-1,i}$$

After incorporating the the next measurement for all particles to each map particle, the last portion of the SLAM update consists of updating each pose particles weight and resampling to prevent weighting degeneracy.

The weighting of each particle in this step was adjusted using the measurement likelihood given the robot's estimated pose and the current map

$$w^{(i)} = p(z_t^{1(i)}|x_t^{1(i)}, m_t)p(z_t^{2(i)}|x_t^{2(i)}, m_t)... \quad (6)$$

$$p(z_t^{M(i)}|x_t^{M(i)}, m_t)w^{(i)} \quad (7)$$

We also implemented an algorithm that assigns an independent particle weighting for each independent pose particle set. For robot R, this weight was computed as:

$$w^{R(i)} = p(z_t^{R(i)}|x_t^{R(i)}, m_t)w^{R(i)}$$

After the propagation and update, the FastSLAM algorithm performs stochastic universal resampling to generate an update set of uniform weight samples. All weights are normalized such that they sum to 1, a cumulative mass function is generated, and a random initial offset is chosen. From this initial offset, an identical number of samples equally spaced across the CMF are taken and the inverse CMF at each of those samples is used to create a new particle. The resulting weights of each particle are then set to a uniform value.

For our addition where each set of pose particles has independent weights, these weights are used to independently resample each pose set while leaving the map unchanged. Stochastic universal resampling is also used for this process.

4) Commentary on Sequential Updating: By only incorporating a fixed number of odometry and measurement updates in a given timestep, Howard's MRSLAM algorithm places a bound on computation time[7]. Each particle propagation, and map update, and measurement model weighting computation are constant time operations for a fixed map size. This results in bounds on computational complexity for a fixed map size using m robots each with n particles of $\mathcal{O}(mn)$. Incorporating all past measurements simultaneously would require computing a possibly large number of particle filter updates [7]. Using Howard's algorithm, where particle weights are shared across all poses, would also be at risk of resampling impoverishment where existing partcles would die out as a result of a simultaneous incorporation of all acausal data at an encounter.

IV. Experimental Validation

In this section we present how we made our data set, the time it takes to map 95% of the environment, and the results using Howard's algorithm and the proposed algorithm.

A. The Data Set

The data set consists of PAS triple $(\mathbf{x}_t^i, u_t^i, z_t^i)$, where i denotes the robot ID, and t denotes the time. The pose, x_t^i, is comprised of the x_t^i-y_t^i position, as well as the orientation θ_t^i. The input, u_t^i, is composed of the position deflection, δ_t^i, and the angular deflection, ω_t. Finally, the measurements, z_t^i, contains scan data for rays cast out at $1°$ intervals from $[-90°, 90°]$, to simulate a laser scan.

To obtain the measurements we use a ray-circle intersection algorithm.

1) Ray-Circle Intersection: The scan data is constructed using ray-circle intersection of a binary image, I (like that of Fig. 7(b) without the paths). The idea of ray-circle intersection is to find all the object pixels within a region of interest (ROI), in this case the shaded pixels in the semi-circle, I_{semi}, as seen in Fig. 6(a),

$$I_{semi} = \left\{ I_{xy} \in I | \; ||I_x y - I_{\mathbf{x}_t^i}||_2^2 < r^2 \right\}, \tag{8}$$

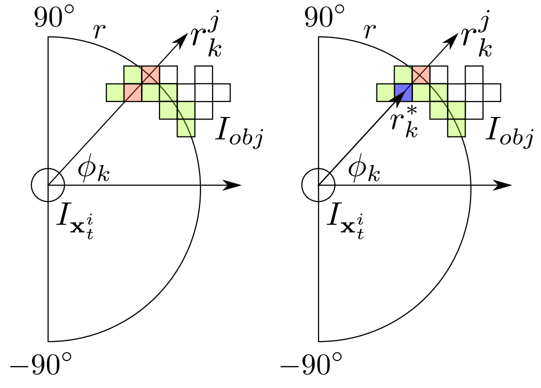
where $I_{\mathbf{x}_t^i}$ is the position of the robot i in the map.

Then I_{semi} is intersected with the object, $I_{obj} = I_{xy} \in I | \; I_{xy} = 1$ (the filled squares in Fig. 6(a), to get I_{filled}. A ray is then defined by

$$\mathbf{v}_k = \begin{bmatrix} I_{x_t^i} + r_k^j \cos(\phi_k) \\ I_{y_t^i} + r_k^j \sin(\phi_k) \end{bmatrix}, \tag{9}$$

where $r_k^j > 0$ is some partitioning of the length of the ray up to the maximum range of the sensor. This ray is intersected with I_{filled} to get I_{ray} (colored in orange in Fig. 6(a)). Finally, the minimum distance to the object, r_k^* , is selected (the r_k^j corresponding the blue square in Fig. 6(b)) to get the distance measurement $z_t^{i,k}$ of the object to the robot.

$$r_k^* = \min_j r_k^j \tag{10}$$



(a) I_{filled} and the intersection with the ray $\mathbf{v}_k$. (b) The selection of r_k^*.

Fig. 6. Visualization of Ray-Circle Intersection

We introduce zero mean, additive Gaussian noise with variance Q to $z_t^{i,k}$ to get noisy measurements $\hat{z}_t^{i,k}$:

$$\hat{z}_t^{i,k} = z_t^{i,k} + \mathcal{N}(0, Q). \tag{11}$$

2) Wall Following: In this paper we will use wall following to search the environment. The specifics of this implementation are that it is turned on when $t \in [50, 200]$, each robot is equipped with a parity bit to ensure that some robots move in opposite directions, and if the robot happens to get stuck in a loop, wall following will temporarily turn off.

Using wall following to search the environment led to the choice of the odometry motion model, because wall following can be easily implemented with scan data obtained using ray-circle intersection.

3) The Odometry Motion Model: The motion model used was the classic odometry motion model.

$$\begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} + \begin{bmatrix} \hat{\delta}_{t-1} \cos(\theta_t + \hat{\omega}_t) \\ \hat{\delta}_{t-1} \sin(\theta_t + \hat{\omega}_t) \\ \hat{\omega}_{t-1} \end{bmatrix}, \tag{12}$$

where

$$\hat{\delta}_{t-1} = \delta_{t-1} - \mathcal{N}(0, \alpha_1 \delta_{t-1}^2 + \alpha_2 \omega_{t-1}^2) \tag{13}$$
$$\hat{\omega}_{t-1} = \omega_{t-1} - \mathcal{N}(0, \alpha_3 \delta_{t-1}^2 + \alpha_4 \omega_{t-1}^2). \tag{14}$$

The reverse odometry model is

$$\begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \theta_{t-1} \end{bmatrix} = \begin{bmatrix} x_t \\ y_t \\ \theta_t \end{bmatrix} - \begin{bmatrix} \hat{\delta}_t \cos(\theta_t - \hat{\omega}_{t-1}) \\ \hat{\delta}_t \sin(\theta_t - \hat{\omega}_{t-1}) \\ \hat{\omega}_{t-1} \end{bmatrix} \tag{15}$$

4) Encounter Detection and Relative Pose: In place of using a hypothetical camera to determine if robot encounters occur, we use the ray-circle intersection algorithm. We treat each robot as an object in the binary image, I, and if a (r_k^*, ϕ_k) is found to coincide with an added robot, and encounter is declared and a relative pose is calculated.

5) The Environment, Robot Trajectories, and Encounters: Fig. 7 contains the test environment and the time when each encounter occurred. Fig. 7(a) shows at what time a robot encountered another, note: only the first encounter is used.

Fig. 7(b) shows the binary map of the environment, the paths each robot took (colored lines), the point when a robot encounter occurred (dashed gray lines), and where the encounter occurred (thick grey line).

For ease of use, we will set Robot 1's initial pose as the global frame's origin.

B. Timing

One of the primary motivations for using MRSLAM is to decrease the amount of time it takes to complete the mapping objective. In this section we demonstrate that generally speaking the time it takes to map an environment decreases with time.

To present the time it takes to map the environment, we must first define what we mean by mapping the environment. When we say that we have mapped an environment, we mean that we have created an occupancy grid, J, that matches the black edges in Fig. 8(b) using a perfect sensor/actuator.

To obtain the edges of the binary map, I, we use morphological erosion:

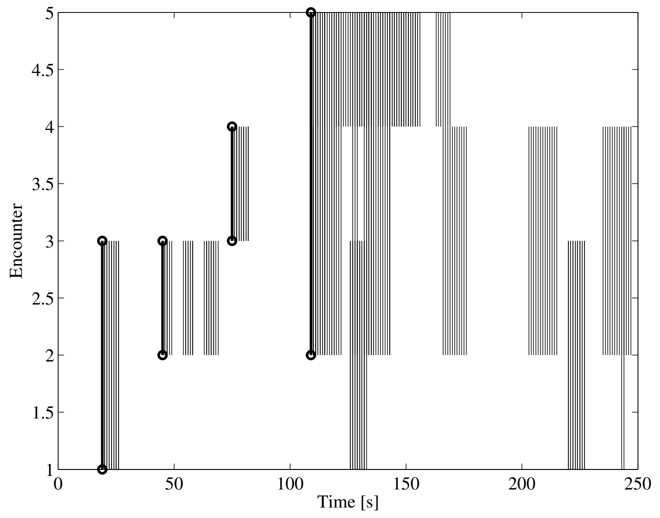
$$\tilde{I} = I \ominus \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}, \tag{16}$$

then padding $\tilde{I}$ with zeroes, we take the exclusive or of I and $\tilde{I}$ to get I_{edge}:

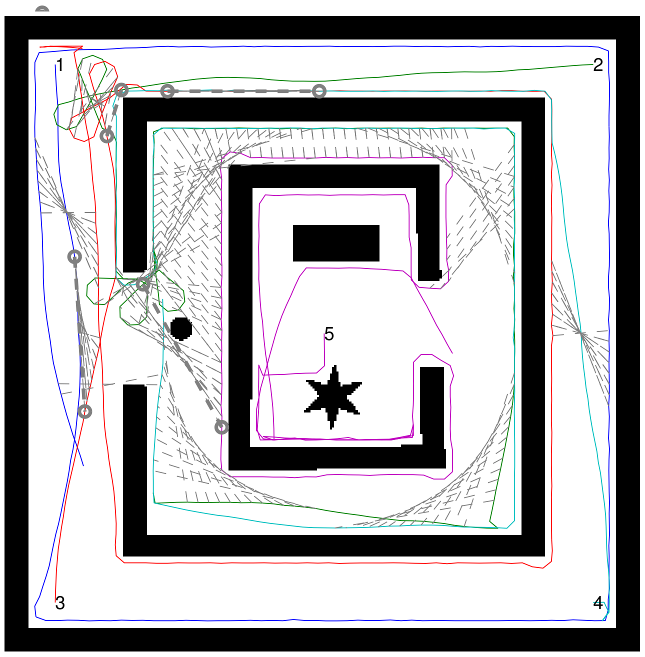
$$I_{edge} = (\tilde{I} \wedge \neg I) \vee (\neg \tilde{I} \wedge I). \tag{17}$$

Then we define the percent mapped as

$$\zeta = \frac{\sum_{x,y} I_{edge} \wedge J}{\sum_{x,y} I_{edge}}, \tag{18}$$



(a) Robot encounters versus time.



(b) The test geometry, robot paths (colored lines), and encounters (dashed gray lines).

Fig. 7. Environment geometry, robot paths, and encounters.

and when $\zeta > 0.95$, we say that the environment has been mapped.

Table I provides the average amount of time it took to map 95% of the environment, for 100 samples, using varying number of robots. Note: the averages do tend to decrease as expected; however, there is a large degree of uncertainty, at least until we get beyond 7 robots.

C. The Results

From Table I we see that there is a large drop-off when we increase the number of robots from 2→3, 4→5, and 7→8. Despite it having the highest average time, mapping with 5 robots adequately conveys the complexity that can be

TABLE I

Time to map 95% of the environment.

# Robots	Mean	Std	Max	Min
1	501.5	99.4	684	341
2	357.3	170.8	654	162
3	201.9	92.4	549	122
4	332	179.2	681	140
5	212.8	113.4	523	117
6	242.5	120.7	594	100
7	262.9	134.3	681	79
8	74.2	20.1	150	55
9	72.0	27.4	257	54
10	57.0	6.3	91	48

achieved; as such, 5 robots, each with 30 particles, will be used to highlight the improvement gained using the proposed algorithm.

To compare the two algorithms, we use two sets of sensor/actuator data: a low noise set and a high noise set.

Low The low noise set perturbs the actuator input by $\mathcal{N}(0, 0.1)$ $[m]$ on δ_t^i, $\mathcal{N}(0, 0.01)$ $[m]$ on ω_t^i, and $\mathcal{N}(0, 0.5)$ $[m]$ on $z_t^{i,k}$.

High The high noise set perturbs the actuator input by $\mathcal{N}(0, 1)$ $[m]$ on δ_t^i, $\mathcal{N}(0, 0.1)$ $[m]$ on ω_t^i, and $\mathcal{N}(0, 1.5)$ $[m]$ on $z_t^{i,k}$.

The qualitative results are presented in Fig. 8. Fig. 8(a) provides the occupancy grid if pure odometry was used, to contrast this the occupancy grid using known poses is given in Fig. 8(b).

For Fig. 8(c) and 8(e), the solid and dashed line represent the best estimated trajectory in the causal and acausal direction. Whereas, for Fig. 8(d) and 8(f), the solid and dashed lines represent the best estimated trajectory for the causal data and acausal data, resp.

Fig. 8(c)-8(f) provide the comparison between Howard's and the proposed algorithm. Comparing Fig. 8(c) and 8(d), it is easy to see that the proposed implementation is almost as good as the occupancy grid with known poses (Fig. 8(b)). Even in the high noise case, Fig. 8(e) and 8(f), we see less error with the proposed algorithm.

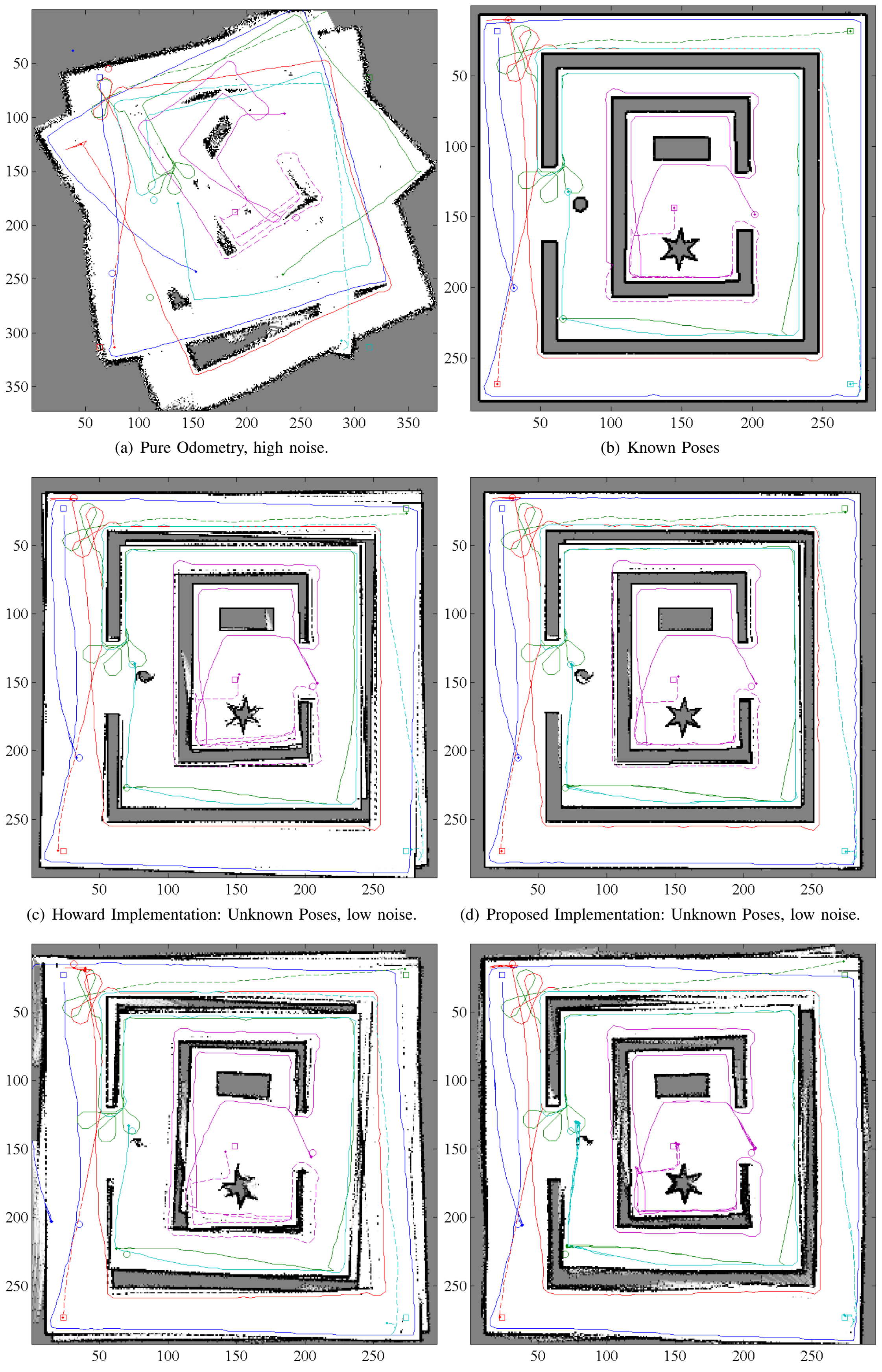
Over the number of particles, we define the minimum sum of squares error (or the least absolute deviations error):

$$\varepsilon_t = \min_m \sqrt{\sum_{i=1}^{n_{robots}} ||\mathbf{x}_t^i - \hat{\mathbf{x}}_t^{i,m}||_2^2}, \qquad (19)$$

where m is the number of particles. Using this measure, quantitative results demonstrating that the proposed algorithm has a smaller error are provided in Fig. r9. To note, for the first 100 seconds both techniques have about the same error, but for the high noise case the proposed algorithm dominates with less than half the error of Howard's algorithm.

V. Conclusion

In this paper we have presented Howards algorithm from [7] for the integration of data from several independent



(a) Pure Odometry, high noise.

(b) Known Poses

(c) Howard Implementation: Unknown Poses, low noise.

(d) Proposed Implementation: Unknown Poses, low noise.

(e) Howard Implementation: Unknown Poses, High noise.

(f) Proposed Implementation: Unknown Poses, High noise.

Fig. 8. Comparison between Howard's implementation and the proposed implementation.

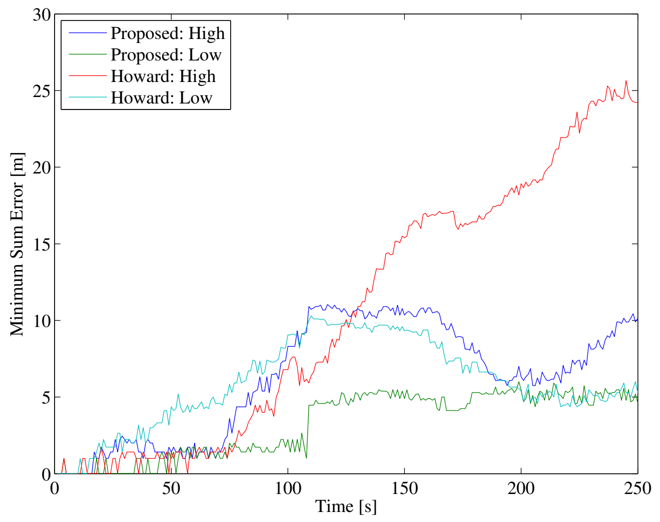


Fig. 9. Minimum sum error.

robots, and demonstrated a successful implementation. Further, we have modified the algorithm to use a particle filter for each robot and demonstrated that it can qualitatively

References

[1] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[2] Andreas Birk and Stefano Carpin. Merging occupancy grid maps from multiple robots. *Proceedings of the IEEE*, 94(7):1384–1397, 2006.

[3] Wolfram Burgard, Mark Moors, Cyrill Stachniss, and Frank E Schneider. Coordinated multi-robot exploration. *Robotics, IEEE Transactions on*, 21(3):376–386, 2005.

[4] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

[5] Dieter Fox, Jonathan Ko, Kurt Konolige, Benson Limketkai, Dirk Schulz, and Benjamin Stewart. Distributed multirobot exploration and mapping. *Proceedings of the IEEE*, 94(7):1325–1339, 2006.

[6] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *Robotics, IEEE Transactions on*, 23(1):34–46, Feb 2007.

[7] Andrew Howard. Multi-robot simultaneous localization and mapping using particle filters. *The International Journal of Robotics Research*, 25(12):1243–1256, 2006.

[8] Miguel Juliá, Arturo Gil, and Oscar Reinoso. A comparison of path planning strategies for autonomous exploration and mapping of unknown environments. *Autonomous Robots*, 33(4):427–444, 2012.

[9] Maria Teresa Lazaro, Lina María Paz, Pedro Pinies, José A Castellanos, and G Grisetti. Multi-robot slam using condensed measurements. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 1069–1076. IEEE, 2013.

[10] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In Ingemar J. Cox and Gordon T. Wilfong, editors, *Autonomous Robot Vehicles*, pages 167–193. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[11] Sebastian Thrun. A probabilistic on-line mapping algorithm for teams of mobile robots. *The International Journal of Robotics Research*, 20(5):335–363, 2001.