Real-Time SLAM with Octree Evidence Grids for Exploration in Underwater Tunnels

Nathaniel Fairfield, George Kantor, David Wettergreen
The Robotics Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213
Email: {than, kantor, dsw}@cmu.edu

November 29, 2006

At 110m in diameter and over 350m in depth, the *cenote* Zacatón in central Mexico is a unique flooded sinkhole. A platform for conducting preliminary sonar tests is tethered in place.

Abstract

We describe a Simultaneous Localization and Mapping (SLAM) method for a hovering underwater vehicle that will explore underwater caves and tunnels, a true three dimensional (3D) environment. Our method consists of a Rao-Blackwellized particle filter with a 3D evidence grid map representation. We describe a procedure for dynamically adjusting the number of particles to provide real-time performance. We also describe how we adjust the particle filter prediction step to accommodate sensor degradation or failure. We present an efficient octree data structure which makes it feasible to maintain the hundreds of maps needed by the particle filter to accurately model large environments. This octree structure can exploit spatial locality and temporal shared ancestry between particles to reduce the processing and storage requirements. To test our SLAM method, we utilize data collected with manually-deployed sonar mapping vehicles in the Wakulla Springs cave system in Florida and the Sistema Zacatón in Mexico, as well as data collected by the DEPTHX vehicle in the test tank at the Austin Applied Research Laboratory. We demonstrate our mapping and localization approach with these real-world datasets.

1 Background

Zacatón is a flooded *cenote* (sinkhole) in Tamaulipas, Mexico, that has been measured at over 350m deep (Gary 2002). The depths of the *cenote* remain unexplored, but during a preliminary expedition in May 2005, we discovered that the upper 200m of Zacatón is roughly a cylinder 110m wide that tapers slightly with depth (Figure 1). Zacatón is the deepest of a series of similar water-filled formations, which are thought to have formed as hydrothermal groundwater dissolved through a layer of limestone (Gary 2002). Zacatón has a small river flowing out through a tunnel near the surface, which indicates that water is flowing in from somewhere below the mapped regions, perhaps through navigable tunnels.

The mineral-rich water in Zacatón supports colorful microbial mats in the photic zone and has exotic geochemical features which make it an excellent match

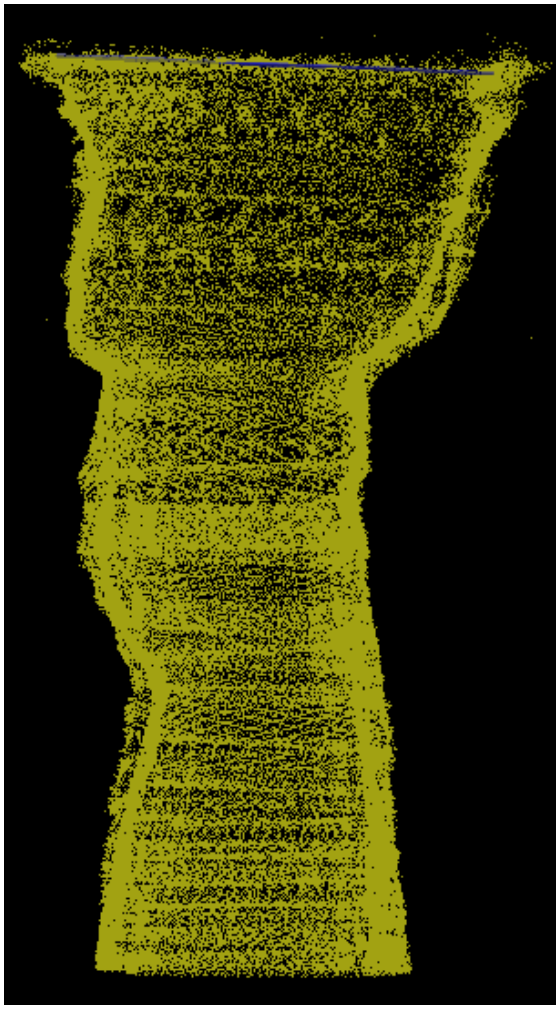

Figure 1: An north-facing side view of the first 200m of Zacatón. This is the raw sonar data from a single dive, plotted as a point cloud in orthographic projection.

for the exploration and sampling mission of the DEep Phreatic THermal eXplorer (DEPTHX). The goal of the DEPTHX project is to autonomously explore and map Zacatón, including any underlying tunnel systems, and then to use various environmental signatures (such as thermal plumes) to direct focused sample collection – with the goal of detecting and sampling unusual microbiota. In this paper, we address the mapping and localization capabilities required to fulfill the requirements implied by these goals. In particular, we emphasize robustness, navigation through tunnels, and precision localization in a large volume.

A typical Autonomous Underwater Vehicle (AUV) uses a combination of depth sensors, inertial sensors, and Doppler velocity sensors to compute a dead-reckoned estimate of its position while at depth (for an overview of underwater navigation methods, see (Leonard et al. 1998)). With high accuracy attitude and depth sensors the uncertainty in the AUV's 3D pose (roll, pitch, yaw, x, y, z) is primarily in x and y. Most underwater navigation systems are based on Kalman Filters which merge Doppler velocity and inertial measurements (Larsen 2000). Corrections to the unbounded drift error inherent in such systems have been achieved by using the global positioning system (GPS) while on the surface (Healey et al. 1998) or beacon-based long baseline (LBL) acoustic positioning systems (Whitcomb et al. 1999). But frequently surfacing for GPS fixes may not be possible or desirable, and LBL beacons, which are typically used in long-duration open-water operations, must be deployed and surveyed before use. Neither of these approaches is viable in Zacatón. Simultaneous localization and mapping (SLAM) offers an attractive method to bound dead-reckoning error because it allows the vehicle to be completely self-contained and unrestricted – and it yields a map of the environment.

In (Fairfield et al. 2005) we evaluated different sonar geometries for SLAM in 3D underwater tunnels. In (Fairfield et al. 2006), we demonstrated SLAM in a constrained scenario with real-world sonar data from Zacatón. In this work, we show that will be able to perform 3D SLAM in real-time on the DEPTHX vehicle. Our key innovation is our Deferred Reference Count Octree data structure, which makes real-time 3D SLAM possible. We also introduce simple methods for adjusting the particle count in order to maintain real-time performance in the face of varying world geometry, and adjusting the prediction model in response to degraded sensor quality.

The rest of the paper is laid out as follows: Section 2 describes related work in localization, estimation, and mapping. Section 3 outlines our approach to the problem. Section 4 describes the map representation and Section 5 describes the particle filter, including the adaptive particle count and sensor-based prediction. We finish with Experiments, Results, and Conclusions.

2 Related Work

Simultaneous Localization and Mapping SLAM is the process of building a map of the environment from sensor data, and then using that map to localize. SLAM methods usually depend on the detection of features from sensor data, and combine observations of these features with an Extended Kalman Filter (EKF) (Smith et al. 1990). In the underwater domain, sonar sensors are not capable of providing the resolution necessary to resolve and recognize features. There has been work on off-line SLAM methods using tunnel cross-sections, or slide im-

ages, which can be derived from sparse sonar ranges as long as the environment is tunnel-shaped (Bradley et al. 2004). In the case where there are free floating artificial features, scanning sonars have been shown to have high enough resolution to support feature-based SLAM (Williams et al. 2000). Alternatively, in clear water with good lighting, SLAM has been demonstrated via video mosaicing (Eustice et al. 2005) and also a combination of vision-based feature detection and sonar (Williams and Mahon 2004).

Underwater localization has also been demonstrated in cases where there is variation in the sea-floor and the vehicle has a prior map of the bathymetry (Williams 2003, Leonard et al. 1998). Many underwater environments are characterized by large monotonous regions where there has been promising work with Synthetic Aperture Sonar (SAS) to support range-and-bearing SLAM (Newman et al. 2003).

Particle Filters As an alternative SLAM method to EKFs, particle filters provide a proven implementation of Bayesian filtering for systems whose belief state, process noise, and sensor noise are modeled by non-parametric probability density functions (for a good summary of particle filters, see (Arulampalam et al. 2002)).

(Fox 2003) describes Kullback-Leibler distance (KLD) sampling, which estimates the number of samples needed at each iteration such that the error between the true density distribution and the discrete particle filter approximation is below some bound. This approach is applied to the bathymetry-map localization mentioned above by (Bachmann and Williams 2003). In short, they conclude that too few particles will poorly approximate the true posterior and too many particles take too long to process (so that measurements have to be thrown away). Our approach uses as many particles as possible in real-time *without* discarding any data.

Rao-Blackwellized particle filters (RBPFs), in which each particle contains both a position and a map, have proven an effective way to do SLAM with evidence grids (Murphy 1999, Doucet et al. 2000). This becomes important as we consider how to incrementally map a fully 3D environment.

3D Maps In the area of 3D maps, there has been work on land using laser range data. (Thrun et al. 2003) maps mine tunnels with a planar floor plan using scan matching to recover the 2D vehicle pose from which the 3D map is reconstructed in a post-processing step. Other terrestrial work builds maps from planes fitted to point clouds (Mahon and Williams 2003, Weingarten and Siegwart 2005, Hähnel et al. 2003). Unfortunately, lasers do not maintain coherence underwater, so they cannot be used to resolve fine features.

The 2D evidence grid is the classic featureless map (Martin and Moravec 1996), a uniform discretization of space with the value of each cell assigned the probability of occupancy. Since the entire space must be represented in memory, even two dimensional evidence grids are large and expensive to copy.

In the 3D evidence grid representation, space is divided into a grid of cubic volume elements, or voxels, which contain the occupancy evidence inferred from sensors. While 2D evidence grid-based SLAM is well established in the indoor mobile robot domain, it has limited applicability in truly 3D environments – largely because the 2D map simplification is only suitable in "two and a half"-dimensional environments, meaning those where only a single height needs to be associated with each 2D grid cell.

The latest version of Distributed Particle (DP) SLAM by Eliazar and Parr (2006) is a similar approach to ours, in that it uses evidence grids, a RBPF, and a sophisticated data structure that exploits the similarity between particles of common ancestry to reduce the cost of copying and storing particle maps. However, in order to get linear ray-tracing performance, they must repeatedly process their data structure to create a cache of uniform "local maps", a complex and memory intensive process, even for 2D maps. The Deferred Reference Count Octree (DCRO) we introduce below avoids this caching step and yields the additional advantages of full 3D, sparse spatial representation, and overlap between particles with common ancestry – all inherent properties of the relatively simple DRCO data structure.

3 System Description

The DEPTHX vehicle (Figures 2 and 3) has a full suite of underwater navigation sensors, including a Honeywell HG2001 Inertial Measurement Unit (IMU), two Paroscientific Digiquartz depth sensors, and an RDI Navigator 600 Doppler Velocity Log (DVL). There is also a Conductivity, Temperature, and Depth (CTD) sensor for measuring the speed of sound so that DVL velocity measurements can be corrected. Under certain circumstances, these sensors can provide excellent dead-reckoned navigation – on the order of 0.5% of distance traveled (Larsen 2000). Over the course of a $\sim$ 4 hour 2km mission, this would yield an error of around 10m, which would be perfectly acceptable in open water conditions but is a serious concern within confined tunnels. Additionally, we must anticipate that there may be times when the DVL will not provide velocity measurements, at which point the quality of the dead-reckoned solution will degrade significantly. In summary, we need to be

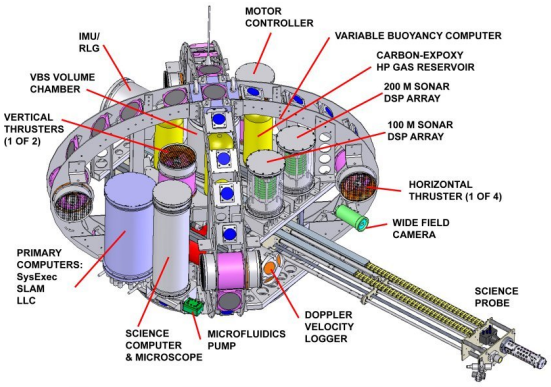

Figure 2: A model of the DEPTHX vehicle structure and components. Eleven pressure vessels house computing, batteries, sensors, and science instruments. Diameter is approximately 2m, weight 1.3 metric tons. ©Stone Aerospace, 2006.

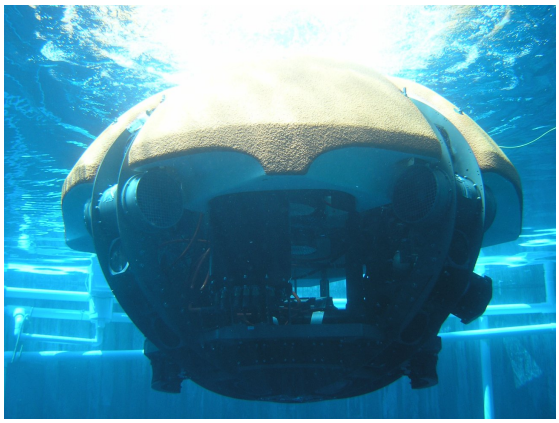

Figure 3: The DEPTHX vehicle in the test tank.

conservative about our expectations for dead-reckoning.

In addition to the standard dead-reckoning sensors, DEPTHX has a mapping system: an array of 54 pencil-beam sonars that provide a constellation of range measurements around the vehicle. The DEPTHX sonar array is in the shape of three great circles (Figure 3, and for a comparison of sonar geometries see Fairfield et al. (2005)). This means that DEPTHX will be able to observe previously mapped regions while exploring – which is vital for SLAM. The sonars have long ranges (some 100m and others 200m) but lack the resolution, update rate, and point density of a laser scanner, making feature recognition and association difficult. The depths of Zacatón are completely unexplored and have unknown geometries, which makes it even more difficult to design feature detectors. For these reasons we have selected a data-driven representation, 3D evidence grids, as our basic world representation. The difficulty in generalizing the 2D evidence grid approach to full 3D comes from the growth of the computational cost of accessing, modifying, and storing the map due to the third dimension. Likewise, there is an increase in the number of pose dimensions which must be estimated from three (heading, x, y) to six (roll, pitch, yaw, x, y, z). However, the IMU and depth sensors can provide excellent measurements for all but x and y).

A SLAM method applied to exploration must provide near real-time localization estimates for vehicle control and navigation. While onboard computational resources are increasing, we must deal with limited resources: 1 Gb of RAM, and a 1.8 GHz Pentium M processor for SLAM onboard the DEPTHX vehicle. The sonar array cycles at 1Hz, so SLAM must run at that rate as well.

For general navigation within Zacatón, the vehicle should remain more than 2m away from the walls. During sampling operations, the vehicle body will approach to within 1m when the sample arm makes contact with the wall. Our scientist team would like to be able to repeatably collect samples within 1m of a designated location, which defines the positioning precision requirement.

The system must cope with sensors that can degrade suddenly in performance, or fail entirely. It can do this by adjusting the particle distribution according to the current sensor error models or by switching to a localization-only mode in which the maps are not updated. It can also adjust the particle count so as to use as many particles as possible while maintaining real-time performance.

We use evidence grids because they can merge together the large number of noisy sonar measurements into a useful map. The Rao-Blackwellized particle filter is a natural match for evidence grids, and also allows the algorithm to represent non-Gaussian position distributions. Before discussing the particle filter itself, we describe the data structure that it will use to represent the map.

4 Evidence Grids

An evidence grid is a uniform discretization of space into cells in which the value indicates the probability or degree of belief in some property within that cell. In 3D, the cells are cubic blocks of volume, or voxels. The most common property is occupancy, so evidence grids are often also called occupancy grids (Martin and Moravec 1996). The primary operations on a map are inserting new evidence, querying to simulate measurements, and copying the entire map. We call the process of updating all of the voxels which are affected by a particular

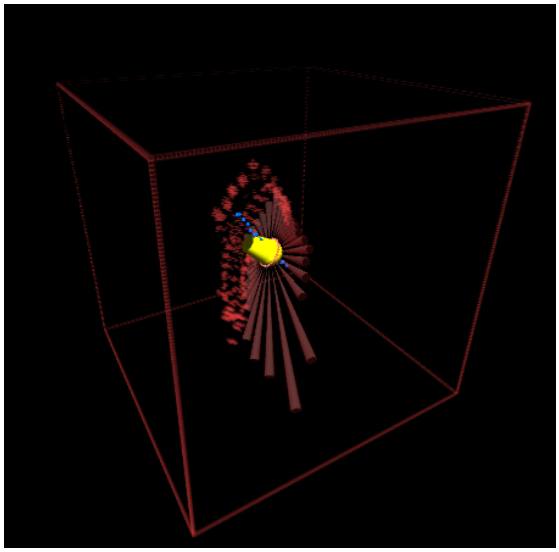

Figure 4: A cut-away view of sonar data being inserted into a 3D evidence grid. The vehicle is modeled as a yellow oblong, its sonar beams as red cones which leave traces of evidence behind in the grid.

measurement an "insertion", and likewise the process of casting a ray within a map until intersects with an occupied voxel we call a "query". Often, the log-odds value for each voxel θ

$$l(\theta) = \log\left(\frac{p(\theta)}{1-p(\theta)}\right)$$

is stored in the map rather than the raw probabilities because it behaves better numerically, and because the Bayesian update rule for a particular voxel according to the sensor model (like a conic beam-pattern) for some measurement z becomes a simple addition (Martin and Moravec 1996):

$$\log(\theta') = \overbrace{\log\left(\frac{p(\theta|z)}{1-p(\theta|z)}\right)}^{\text{beam model}} + \overbrace{\log\left(\frac{1-p(\theta)}{p(\theta)}\right)}^{\text{map prior}} + \log(\theta).$$

The first term on the right-hand-side is the sensor model, and the second is the map prior. If the prior $p(\theta) = 0.5$, the second term is zero and the initialization simply sets all voxels to zero. The update for each voxel can be reduced to simply summing the value of the sonar model with current voxel evidence. One important (and plainly false) assumption underlying the Bayesian insertion is that the cells are independent – that is that the occupancy of one cell is independent of the occupancy of any other cell. However, without this assumption evidence grids become intractable since the repercussions of updating a single cell could propagate through the entire map. The drawback is that maps tend to be more noisy in response to ambiguity in the measurements.

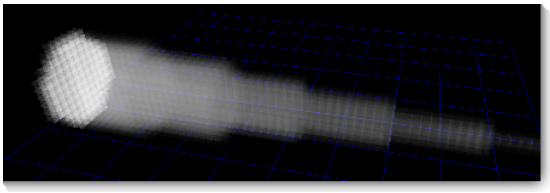

Figure 5: An example of the sonar beam model carved into an evidence grid.

4.1 Sonar Model

As measurements are collected, the evidence they provide about the occupancy of each voxel is entered into the map. A sonar beam model defines how a single range measurement can be inserted into the evidence grid. There are several methods which can be used to construct a beam model, including deriving it from physical first principles (Urick 1983) or learning it (Martin and Moravec 1996). We chose to use the simplest reasonable approximation – a cone with a cap that is loosely based on the beam-pattern of the sonar. The cone is drawn as a bundle of rays with constant negative value, with terminating voxels with constant positive values. These log-odds values were experimentally chosen to be -2 and 8. Likewise, the simplest method to query a sonar range from the 3D evidence grid is to trace a ray until some threshold (or other terminating condition) is met. Using matrix transformations for each voxel is too computationally expensive for operations such as filling in evidence cones or simulating ranges. These tasks can be decomposed into raster operations, which can be performed by a 3D variant of the classic 2D Bresenham line drawing algorithm, also called *ray-tracing* (Bresenham 1965).

4.2 Octree Data Structure

The main difficulties with 3D maps arise from the cost of copy operations and the storage requirements that increase with map size and resolution. If we store the evidence log-odds as single bytes (with values between -128 and 127), then an evidence grid 1024 cells on a side requires a megabyte of memory in 2D and a gigabyte in 3D. A typical memory bus can handle transfer rates of around 400 Mb/s, and so would require over two seconds to copy such a map. In the case of the Particle Filter (Section 5), we need to store and copy hundreds of maps

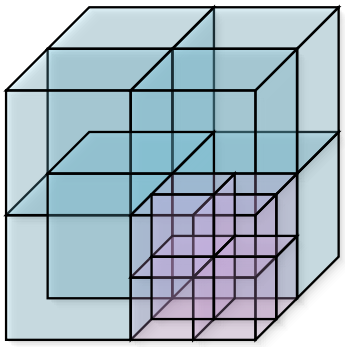

Figure 6: Each level of an octree divides the remaining volume into eight octants, but the tree does not have to be fully expanded.

per second. This requires a more efficient data structure than a uniform array; the octree is one such structure.

An octree is a tree structure composed of a node, or *octnode*, which has eight children that equally subdivide the node's volume into *octants* (Figure 6). The children are octnodes in turn, which recursively divide the volume as far as necessary to represent the finest resolution required. The depth of the octree determines the resolution of the leaf nodes. The main advantage of an octree is that the tree does not need to be fully instantiated if pointers are used for the links between octnodes and their children. Large contiguous portions of an evidence grid are either empty, occupied, or unknown, and can be efficiently represented by a single octnode – truncating all the children which would have the same value. As evidence accumulates, the octree can compact homogeneous regions that emerge, such as the large empty volume inside a cavern. Note that even with compaction the octree supports the insert, query, and copy operations, and is a drop-in replacement for the uniform array: it is possible to convert losslessly between the two representations. Insert and query can be done with a tree-traversing ray-tracing algorithm (see (Havran 1999) for an overview). We employ our own Bresenham 3D top-down approach, largely for its simplicity. Octrees have been widely used in ray-tracing as a way to sort polygons – not to store evidence, as we do here. Most applications also do not need to copy octrees, as we do for the particle filter (see below). The common approach to copying an octree is simply to traverse the entire tree copying every node (see the "Naïve" row of Figure 7).

To improve performance, we use our own custom memory management for the octnodes. Octnode memory is created as needed in large blocks (about 10k nodes) which are initialized as a FIFO linked list of free nodes. The first word of each node points to the next free node. As nodes (and entire octrees) are freed, nodes are pushed onto the front of this linked list. When a new node is needed, the first entry of the linked list is pulled off and initialized with default values.

Reference Counting Octree (RCO) The first real-time challenge when using octrees is not the ray-tracing operation, but rather copying the maps during the resampling step of the particle filter. Copying an octree is an expensive operation, but we can allow octrees to share subtrees by maintaining reference counts to the octnodes: each node keeps track of the number of references to itself. We refer to this number as refCount. This means that we replace naïve copying with copy-on-write (see the "Reference Counting" row of Figure 7). When an octree needs to be copied, we simply increment the reference count. Then when either copy is modified only subtrees which have a reference count greater than one need to be copied. And of course once a copy has been made, the reference count of the source is decremented. We call this "copy-on-write", and find it useful because in the particle filter application, queries to the map are much more common than insertions.

Deferred Reference Counting Octree (DRCO) However, maintaining the bookkeeping for reference counts requires a full traversal of the octree – or at least of the modified subtree – which is almost as expensive as copying. Our novel solution is to maintain deferred reference counts. Every octnode has a refCount and also a deferred reference count, which we will call the defCount (see the "Deferred Reference Counting" row of Figure 7). The defCount represents reference counts which have not yet been propagated to the children. This is similar to the work of (Baker 1994). The true reference count of the node is the sum of refCount and defCount – but changes in defCount do not usually trigger a recursive traversal through the subtree.

Copying a map now simply requires incrementing the defCount of the node pointed to by the source, freeing the node pointed to by the destination (if it is not new), and setting the destination pointer to the source node (see Procedure 1). If either of the two copies is modified then the copy-on-write code will automatically push the defCount down the tree, copy the portion of the tree which will be changed, and set the reference counts accordingly (see the "Deferred Reference Counting" row of Figure 7 and Procedure 3).

LazyFreeNode applies the same deferred (or lazy) principles of recursively freeing octnodes: if the octnode has defCount > 0, then it can just decrement the defCount (see Procedure 2): what the children don't know won't hurt them. If the defCount $= 0$, then LazyFreeNode must call itself recursively on the children (who

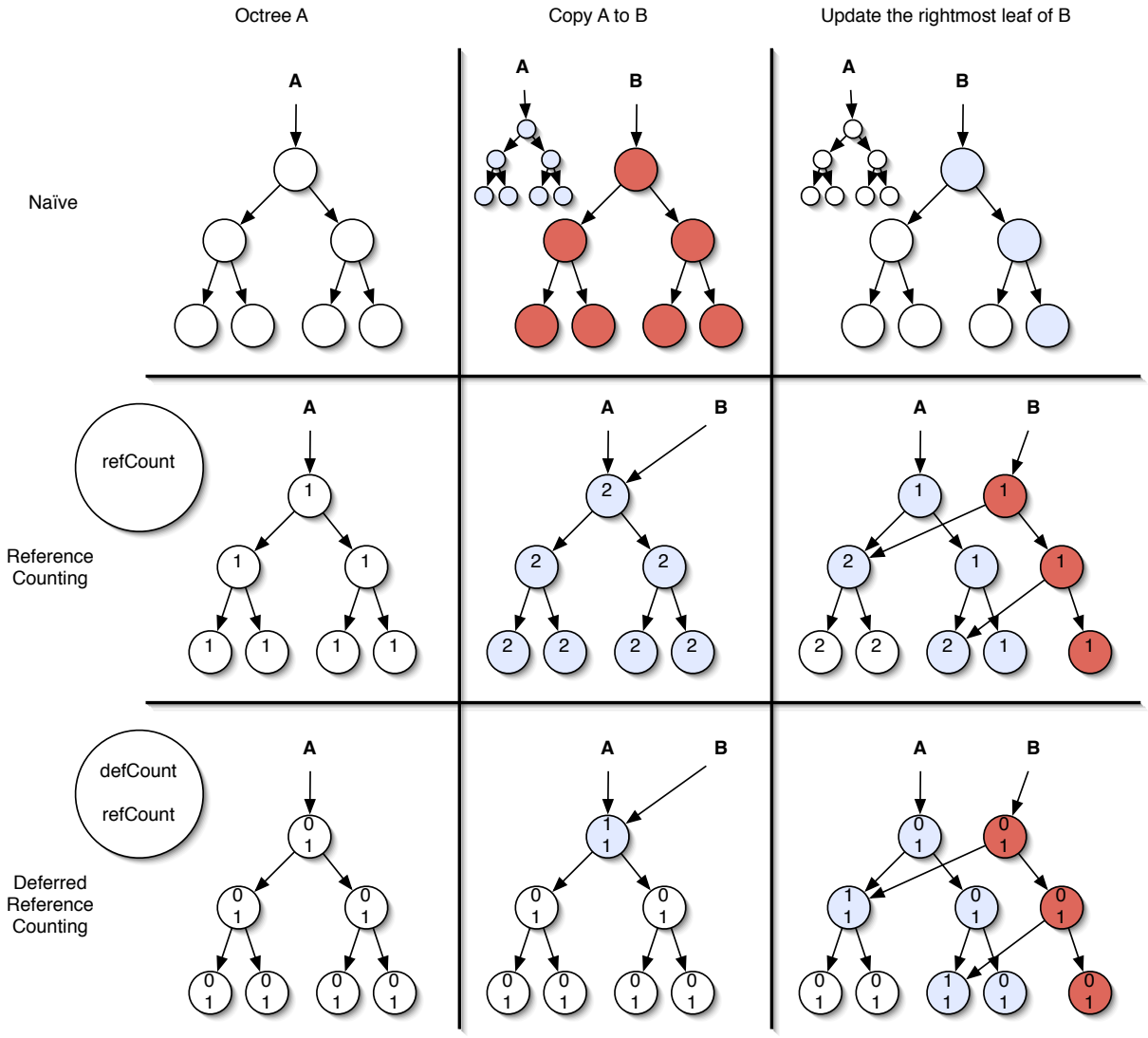

Figure 7: Octree diagram demonstrating the copy and write operations for three octree implementations: Naïve, Reference Counting, and Deferred Reference Counting. Blue (light gray) nodes are notes that must be accessed, and red (dark gray) nodes are new nodes.

Procedure 1 DRCO COPY(A to B)

Require: A and B are pointers to octnodes

1: A→defC++;
2: LAZYFREENODE(B); // See Procedure 2
3: B = A;

may decide they don't need to tell their children, etc).

The ray-trace write, or "insert", operation in the DRCO must deal with properly updating all these counts – the ray-trace read, or "query", operation is unchanged. When we want to insert updates into a map, copy-on-write propagates the node's defCount down to its children, sets the refCount = refCount + defCount, and sets defCount = 0 (since defCount represents the reference counts that the node hasn't told its children about) (see Procedure 3). If the new refCount is > 1, the node must be copied (maintaining the old connections to its children). The deferred updating works because the insert procedure always starts at the top of the octree – this top-down property is part of our ray-trace implementation. DRCO's yield a significant performance boost, and allow us to represent maps that would not even fit into memory as a uniform array (see the Results Section).

Compaction Compaction is the process of recursively traversing the tree, replacing groups of homogeneous

Procedure 2 DRCO LAZYFREENODE(N)

Require: N is a pointer to an octnode

```
if N→defC > 0 then
    N→defC--;
    return
end if
N→refC--;
for i = 1...8 do // Recursively free children
    LAZYFREENODE(N→child[i]);
end for
if N→refC == 0 then // Free the node
    N→nextNode = globalFreeNode;
    globalFreeNode = N;
end if
```

children with a single parent with the same value. As regions of the map become well explored, fuzzy compaction can simplify regions with small amounts of noise. Periodically compacting the octrees can yield significant space savings, especially when the map will only be used for querying – in which case the values can be lossily thresholded to {0=empty, 1=occupied} (see Table 1). Ray-tracing is also accelerated if the ray-tracing algorithm takes advantage of these compacted regions.

Now that we have developed a data structure to maintain and duplicate large 3D evidence grids efficiently, we can consider how to build maps and localize using a particle filter.

Original 512^3 at 1m map	100%
Lossless compaction	99.9%
{Empty, occupied} compaction	66%

Table 1: Example compaction results on a test map, as percentages of the original size of the map. Due to the amount of noise in the real data used to construct the map, the lossless compaction doesn't do much.

5 Particle Filtering

The goal of SLAM is to estimate the probability distribution at time t over all possible vehicle states s and world maps Θ using all previous sensor measurements Z_t and control commands U_t (for a complete list of notation, see Table 2):

$$p(s, \Theta | Z_t, U_t).$$

This distribution is called the *SLAM posterior*. The recursive Bayesian filter formulation of the SLAM problem is straightforward (see Montemerlo et al. (2002) for a derivation) but the integral is usually computationally

Procedure 3 DRCO SETNODE(N, value)

Require: N is a pointer to an octnode, value is some constant

```
if N→defC + N→refC > 1 then
    if N→defC > 0 then
        for i = 1...8 do // Propagate defC to children
            if N→child[i] != 0 then
                N→child[i]→defC += N→defC
            end if
        end for
    end if
    newN= NEWNODE();
    COPYNODE(newN, N);
    newN→refC = 1;
    newN→defC = 0;
    newN→value = value;
    N→refC = N→defC + N→refC - 1;
    N→defC = 0;
else
    N→value = value;
end if
N = NEXTNODEINTOPDOWNTRAVERSE()
if N != 0 then
    SETNODE(N, value)
end if
```

intractable to solve in closed form:

$$\overbrace{p(s_t, \Theta | Z_t, U_t)}^{\text{new posterior}} = \eta \times \overbrace{p(z_t | s_t, \Theta)}^{\text{measurement model}} \times \int \underbrace{p(s_t | s_{t-1}, u_t)}_{\text{motion model}} \underbrace{p(s_{t-1}, \Theta | Z_{t-1}, U_{t-1})}_{\text{old posterior}} ds_{t-1},$$

where η is a constant scale factor from Bayes' rule.

The key insight of (Murphy 1999) is that the SLAM posterior distribution can be factored into two parts, or marginals: the path distribution and the map distribution. Furthermore, knowing the vehicle's trajectory S_t makes the observations U_t conditionally independent, so that the map sample Θ can be computed in a closed form. The process of factoring a distribution such that one part can be computed analytically is known as Rao-Blackwell factorization (Doucet et al. 2000). As a result, following (Montemerlo et al. 2002) we compute the posterior over *trajectories*, we can factor the distribution as

$$p(S_t, \Theta | Z_t, U_t) = p(S_t | Z_t, U_t) p(\Theta | S_t, Z_t).$$

Particle filters are a Monte Carlo approximation to the Bayesian filter. The particle filter maintains a discrete approximation of the SLAM posterior using a (large) set

$\#_{par}$	number of particles
$\#_{son}$	number of sonars
$s_t^{(m)}$	vehicle pose of the m-th particle at time t $= (roll, pitch, yaw, x, y, z)^T$
$S_t^{(m)}$	trajectory of m-th particle from time 0 to t $= \{s_0^{(m)}, s_1^{(m)}, s_2^{(m)}, \ldots, s_t^{(m)}\}$
z_t	sonar measurements at time t
Z_t	history of measurements from time 0 to t $= \{z_0, z_1, z_2, \ldots, z_t\}$
${}^n z_t$	n-th sonar measurement at time t
u_t	vehicle dead-reckoned innovation at time t
U_t	history of dead-reckoning from time 0 to t $= \{u_0, u_1, u_2, \ldots, u_t\}$
$\Theta^{(m)}$	map of m-th particle
θ	a particular voxel $= {}^{ijk}\Theta$
$w_t^{(m)}$	m-th particle weight at time t

Table 2: Particle filter notation.

$N(\mu, \sigma)$	normal distribution with mean μ and std dev σ
$h(s_{t-1}, u_t, N_h)$	vehicle motion model with noise model N_h $= p(s_t \mid u_t, s_{t-1})$
$g(s_t, \Theta, N(0, \sigma_z))$	sonar measurement model $= p(z_t \mid s_t, \Theta)$

Table 3: Model notation.

of samples, or *particles*. The m-th instance of the $\#_{par}$ particles represents both a sample pose $S_t^{(m)}$ from the distribution of vehicle trajectories, and the sample map $\Theta^{(m)}$ which results from that trajectory combined with the sensor measurements Z_t. Since we update the particle maps at every time-step, they represent the combination of sensor measurements and vehicle trajectory – so each particle only needs to store the current map $\Theta^{(m)}$ and pose $s_t^{(m)}$ (rather than the whole trajectory $S_t^{(m)}$).

For practical purposes, when SLAM is being used to provide a pose for the rest of the vehicle control software, we usually want to turn the set particles into a single point estimate. If the posterior distribution is Gaussian, then the mean is a good estimator, but other estimators may be better if the distribution becomes non-Gaussian.

The particle filter algorithm has the following steps:

Initialize The particles start with their poses s_0 initialized according to some initial distribution and their maps Θ (possibly) containing some prior information about the world. This is called the *prior distribution*.

Predict The dead-reckoned position innovation u_t is computed using the navigation sensors (IMU, DVL and depth sensor). A new position s_t is predicted for each particle using the vehicle motion model (see Table 3):

$$s_t = h(s_{t-1}, u_t, N).$$

This new distribution of the particles is called the *proposal distribution*.

Weight The weight w for each particle is computed using the measurement model and the sonar range measurements (from the $\#_{son}$ different sonars):

$$w = \eta \prod_{n=1}^{\#_{son}} p({}^n z_t \mid s_t, \Theta),$$

where η is some constant normalizing factor (different than the one used in the expression for the Bayesian filter). In our implementation, the real range measurements z are compared to ray-traced ranges $\hat{z}$ using the particle pose and map. We compare the simulated and real ranges using the measurement model

$$z = g(s_t, u_t, N(0, \sigma_z)),$$

which is assumed to have a normal noise model, so

$$p(z \mid s, \Theta) = \frac{1}{\sqrt{2\pi\sigma_z^2}} e^{\frac{-(\hat{z}-z)^2}{2\sigma_z^2}}.$$

Substituting into the expression for particle weight and taking the logarithm of both sides shows that maximizing this weight metric is very close to minimizing the intuitive sum squared error metric:

$$\log w = C - \frac{1}{2\sigma^2} \sum_{i=1}^{\#_{son}} ({}^i\hat{z} - {}^i z)^2,$$

where $C = \#_{son} \times \log\left(\sqrt{2\pi\sigma^2}\right)$. An alternative weighting method, called "point correlation" was found to be slightly less informative (Fairfield et al. 2005).

Resample The $O(n)$ algorithm described in (Arulampalam et al. 2002) is used to resample the set of particles according to the weights w such that particles with low weights are likely to be discarded and particles with high weights are likely to be duplicated. The set of particles is now our new estimate of the new SLAM posterior.

Update The measurements z are inserted into the particle maps $\Theta^{(m)}$ (as described in Section 4) to update the evidence of all the voxels θ which lie in the conic sonar beam model of each measurement relative to the particle position. This is when maps must be copied and updated. We save duplicate insertions by inserting *before* copying successfully resampled particles.

Estimate Generate a position estimate from the particles.

Repeat from Predict

5.1 Modifications

Sensor-based prediction The particle filter algorithm uses the current sensor models during the prediction phase. For example, the IMU provides excellent heading ($\pm 0.1^\circ$), while the DVL provides much worse heading ($\pm 2^\circ$). If the IMU is available, the predict step takes the heading value, adds Gaussian noise $N(0, 0.1^\circ)$, and uses the new value to dead-reckon the particle's new position. If the IMU fails, then the filter will fall back to the DVL data, and generate more noisy predictions. This is vital to the robustness of the SLAM method. In the case where there are redundant sensors, it would make sense to use a Kalman filter to combine the information and perform the prediction step using the covariance estimates as the prediction noise model.

An important observation is that when all DEPTHX sensors are functioning, the particle filter only really estimates x and y as the particles are distributed in the xy plane. However when sensors fail or degrade the particle filter distributes particles over the newly uncertain dimensions. This will dramatically increase the risk of under-sampling, but since the failure or degradation of a sensor will also cause the vehicle to terminate the mission and return to the surface, SLAM can switch to a localization-only mode (without map updates) in which it can support many more particles, which may ameliorate the situation.

Adaptive particle count The processing time of a single iteration of the SLAM algorithm varies significantly depending on local world geometry. This is because of the range-dependent ray-tracing time, but the implication is that the particle filter could be using more particles and still maintain real-time performance. Furthermore, the Weighting and Resampling steps take approximately the same amount of time given a fixed number of particles. How to best spend the available computational resources: weighting more particles in the hopes of finding good ones, or resampling more particles in order to maintain particle diversity? Our approach is to weight as many particles as possible, but then to fully resample according to those weights. With DRCO's, resampling particles is fast. We modify the particle filter algorithm as follows:

1. Start with a huge number of particles (in our case 5000: more than could ever be supported computationally)
2. During the Weighting step, set all particle weights to zero and then randomly pick particles and weight them until the time limit is reached
3. Fully resample the particle set – particles which were not weighted will always be replaced with weighted particles

Note that it is important to randomly pick particles during the weighting phase to avoid sorting effects, such that the distribution is accurately sub-sampled.

5.2 Discussion

The idea of the adaptive particle count is to use as many particles as possible in real-time *without* discarding any data, which is the dual of the Kullback-Liebler distance (KLD) sampling technique mentioned above. The adaptive particle count method appears to provide an improvement over an equivalent (from a real-time performance perspective) fixed particle count, particularly while in enclosed areas (see Figure 19). The most important contribution of adaptive particle count is the reliable real-time performance in the face of unknown world geometry. However, it does not provide any guarantees about avoiding under-sampling. We believe that a better system would incorporate both KLD sampling and temporal constraints to determine the particle count. In the case where KLD sampling cannot be satisfied within the given time constraints the vehicle should probably terminate the mission and resurface, but it still must provide a SLAM solution.

The particle filter/octree combination is stable with regard to the fraction of particles that are resampled at each time step (which can vary between 0 and $N_{part} - 1$). Although discarding a particle is costly because of the potential for a recursive freeNode, the node does not have to be updated – which saves many ray insertions. What does seem to be a good indicator of the duration of an iteration is the average sonar beam length, which makes sense.

The performance of the DRCO depends heavily on the circumstances. It will be most efficient when the environment and particle filter are amenable to the exploitation of spatial locality (particles share most of the map

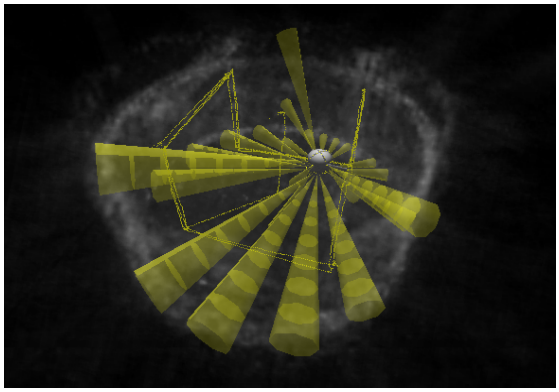

Figure 8: This figure shows the 3D trajectory of the DEPTHX vehicle in the ARL test tank, as well as a rendering of the vehicle and its sonar beams. The vehicle is surrounded by the cloudy evidence map constructed by SLAM, where opacity indicates occupancy.

in common when the vehicle is only modifying small regions) and volumetric sparsity (octrees compactly represent a map that is mostly empty or full). Most large-scale outdoor environments seem to be well suited for this type of exploitation.

6 Experiments

We present two experiments. The first is a basic demonstration of SLAM in cylindrical test tank using data by the actual DEPTHX vehicle. The second is demonstration of SLAM in a synthetic (partially simulated) environment that closely models the challenging features we expect to encounter in Zacatón.

6.1 ARL Tank Test

The large wooden test tank at the University of Texas at Austin Applied Research Lab (ARL) is a cylinder 38 feet (11.6m) deep and 55 feet (16.8m) in diameter. To test SLAM in this environment, we had the DEPTHX vehicle drive three cycles around a 3D box pattern (Figure 8), using dead-reckoning for localization and navigation. The box pattern was 8m on a side and 5m deep, and each cycle took about 13 minutes for a total run time of 40 minutes. The vehicle rotated $\sim 5^\circ/s$ during ascent and descent in order to obtain better sonar coverage of the walls.

6.1.1 Test tank results

We are still in the testing phase for SLAM and did not use it to guide the DEPTHX vehicle. However, we did

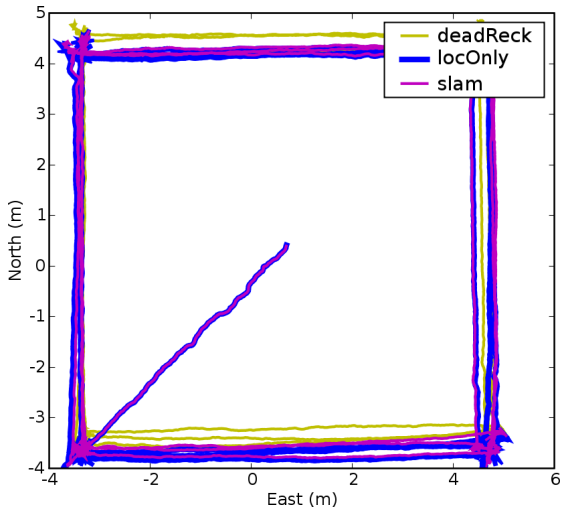

Figure 9: Planar XY view of the trajectories of the various localization solutions in the ARL test tank. The deadReck solution looks quite square as it was used to navigate during the test, but the true vehicle trajectory is shown by locOnly (localization-only SLAM with 3000 particles).

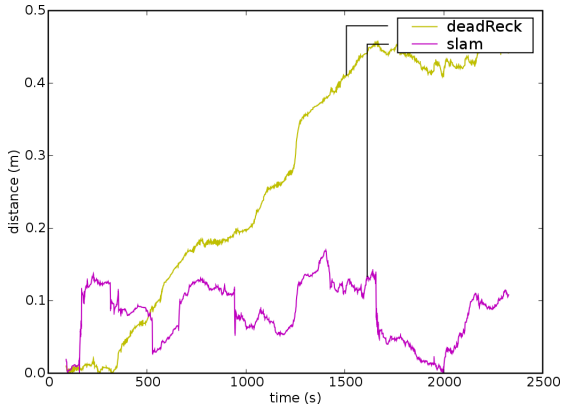

Figure 10: Distance between various localization solutions in the ARL test tank. The ground truth was established using localization-only SLAM with 3000 particles – dead reckoning drifts away while SLAM error is bounded.

run SLAM in a diagnostic mode onboard DEPTHX and demonstrated convergence of localization-only SLAM with 500 particles and a prior map, which consumed less than 20% of the onboard CPU.

To establish the ground truth trajectory of the vehicle, we ran SLAM with 3000 particles in localization only mode with a manually constructed 0.25m resolution map of the ARL tank. The dead-reckoned trajectory drifted from the ground-truth by $\sim$0.5m, which agreed with our observations during the test. We then ran SLAM using 500 particles (with no prior map), which yielded a bounded localization error of $\sim$0.1m (Figures 9 and 10).

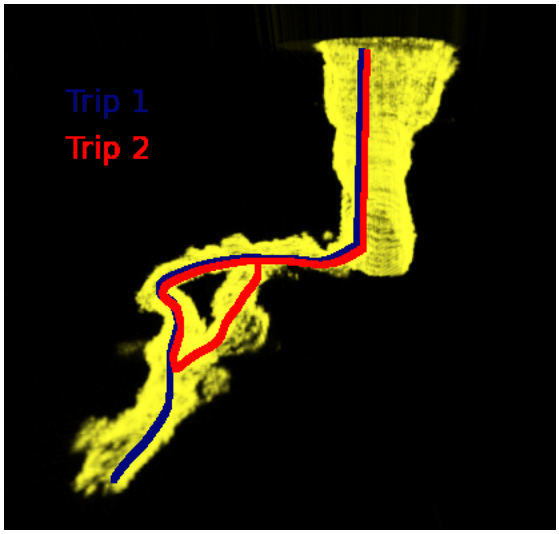

Figure 11: This figure shows the two test dives superimposed on the synthetic Wakatón map, which was constructed by merging portions of the Zacatón and Wakulla Springs datasets. Dive 1 went out to the furthest extremity of the tunnel and back, while Dive 2 went around the loop.

To demonstrate SLAM in a more challenging scenario, we turned to the synthetic Wakatón environment.

6.2 Wakatón

In May 2005, the DEPTHX team lowered a 32-sonar probe, called the DropSonde, into Zacatón to a depth of 200m (Figure 1) (Fairfield et al. 2006). A similar probe, called the Digital Wall Mapper (DWM) was used in 1998-1999 to map several kilometers of the Wakulla Springs cave system in Florida (Stone et al. 2000). The DWM was a diver-driven sensor sled, and the DropSonde was simply lowered on a cable from a barge. Both probes were based around a ring laser gyro IMU, two depth sensors, and a ring of 32 pencil beam sonars arrayed radially.

The problem with using either of the DWM or DropSonde datasets to test our SLAM method was that neither dataset contained sonar data with a reasonable geometry for SLAM. In both cases, the data was collected by driving a ring of sonars along the axis of a cylindrical tunnel. In this orientation, the sonars provide no "look-back" to previously mapped regions, which is essential for SLAM.

Our solution was to create a synthetic world by building partial maps from the datasets and merging them together. In the case of the DropSonde data, we recorded all six degrees of freedom: the xy position of the barge and the depth, attitude, and heading of the probe. Together with the sonar data, this DropSonde pose data provided an excellent map of the first 200m of Zacatón. The Wakulla Springs data also contained excellent attitude, heading, and depth, but there was no ground truth for x and y position except for a few widely spaced waypoints. Using these waypoints to estimate the IMU drift rates, we generated a reasonable trajectory which was consistent with the sonar data, and used this to construct maps of two small tunnels, which we grafted onto the base of the partial Zacatón map (Figure 11). By combining the two datasets, we created a a high-fidelity model of Zacatón including challenging hypothesized features, such as small tunnels, loops, and bell domes. From this combined model, which we called "Wakatón", we could simulate sonar ranges for any desired sonar geometry with any ground truth path, including loop closure. We generated virtual vehicle trajectories and then used sensor noise models to simulate sensor readings for that trajectory.

The sensor noise was generated from zero-mean normal distributions. We elected to use conservative (high) noise values for three reasons: we have little information about the performance of the various sensors on the integrated DEPTHX vehicle in Zacatón, we wanted to encourage particle diversity, and we wanted to cause a clear distinction between dead-reckoning and SLAM on the same dataset. Accordingly, the DVL velocity noise was $0.2m/s$ 1σ, the IMU yaw noise was 1° 1σ, the depth sensor depth noise was $0.01m$ 1σ, and the sonar range noise was $1m$ 1σ. We compared the simulated sonar ranges with real sonar data to verify that the synthesized map generated realistic data. The two trajectories shown in Figure 11 were used in the results section.

This method differs from pure simulation in several important ways. The complex real-world geometry of the tunnels is preserved, as are some of the noise characteristics of the sonars. We were also able to use the real attitude and depth data (from the IMU and depth sensors, respectively), although we had to simulate the DVL data.

6.2.1 Wakatón results

Figure 12 shows that as the number of particles increases beyond 100 the localization error decreases, although the error does not go to zero. Below 100 particles the filter diverges, and so it actually performs worse than just dead-reckoning (which is the same as 1 particle). This is due to particle depletion, when the filter is unable to adequately sample the posterior distribution with so few particles.

Figure 13 shows that SLAM with 200 particles has a relative accuracy of about 1m. The first dive has significant *absolute* error (as shown in Figure 12), the second dive (which used the map constructed during the first

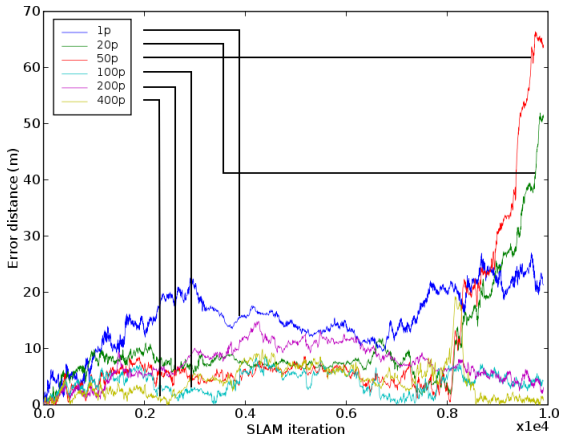

Figure 12: SLAM localization error during Dive 1 through Wakatón with 1, 20, 50, 100, 200, and 400 particles. One particle is equivalent to dead-reckoning. With 20 and 50 particles, the filter appears to diverge near the end of the dive. The performance clearly improves as the number of particles increases beyond 100 particles.

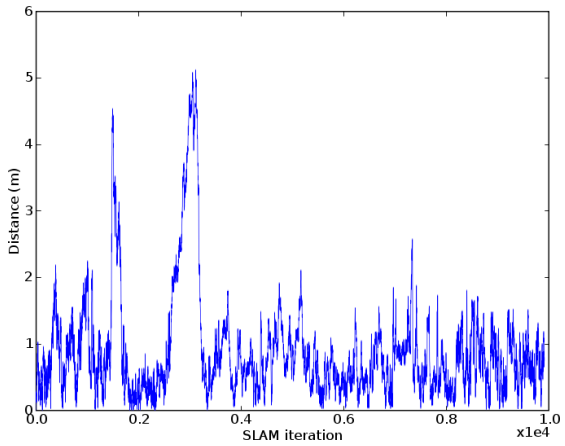

Figure 13: SLAM repeatability in Wakatón shows that with 200 particles the vehicle re-localized to within about 1m of its first estimated trajectory. This figure shows the distance between the first estimated trajectory, and the second estimated trajectory that used the map constructed during the first dive.

dive) is close enough that we will be able to consistently return to within about a meter of the same location for sampling.

Since the Wakatón environment is closely modeled on what we expect to find in Zacatón and we used pessimistic sensor error models, we have a reasonable expectation that SLAM will succeed in Zacatón with a similar number of particles (~ 500), although we certainly expect to support more. We will be able to make stronger predictions as we proceed with testing the DEPTHX vehicle.

Deferred Reference Count Octree The deferred reference count method rolls the cost of propagating reference counts into the insert operation, but the overall performance improvement is substantial. Without deferred reference counts, 70% of the SLAM processing time during a Wakatón dive is consumed by maintaining reference counts (for map copies) while only 30% is being used for ray-tracing (for weighting and updating the maps). Deferred reference counting reduces the map management overhead by a factor of ten, to 7%. But this under-represents the actual performance gain in the context of the particle filter: a major cost for the particle filter comes from particles which are resampled (copying the pose and map from a more highly weighted particle) in one time-step, and then are resampled *again* in the next time-step – without ever inserting anything into their maps. DRCO's give us this for free, simply incrementing and decrementing the defCount of the root node.

Octree ray-tracing is still three times slower than uniform ray-tracing in this implementation. However, due to the enormous gains in map efficiency, octree maps allow the SLAM system to support hundreds of particles with high-resolution maps; something that isn't even possible with uniform maps (for example, the 400 particles with 0.5m resolution 512^3m maps used in the experimental scenario).

Figure 14 shows that the DRCO storage efficiency of the particle filter increases with the number of particles (due to increased amounts of overlap between particles). Each particle takes about 1 Mb of memory, and about ten seconds of computation time. Memory usage and computation time increases cubically with map resolution, as expected.

Adaptive Particle Count Figure 15 shows that SLAM time is roughly split between the weighting and resampling steps, and Figure 17 demonstrates that the amount of processor time for a single SLAM iteration varies significantly between when the vehicle is in the main Wakatón cylinder and when it is in the much more constricted tunnels. This encourages the idea that the particle filter could do better if it could vary its particle count: Figure 19 shows the number of particles that were evaluated for different values of the weighting time limit. Figure 16 shows the improved timing consistency for adaptive particle count version of the particle filter (although there is still variation in the amount of time necessary for the weighting step).

Real-Time Performance Figure 19 shows that for the desired 2Hz SLAM timing, which is achieved with a weighting time of 0.1s, the DEPTHX SLAM processor (a 1.8 GHz Pentium M) is capable of supporting between

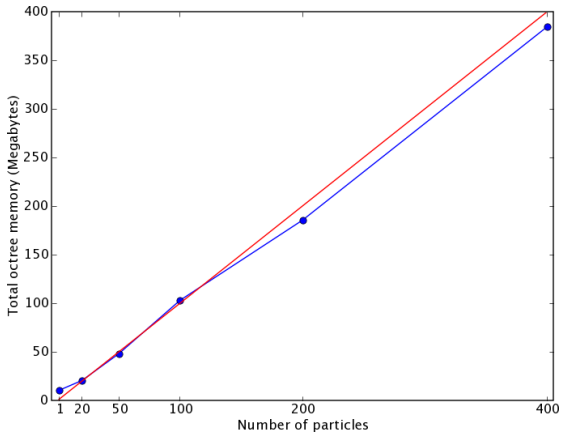

Figure 14: There is a slightly sub-linear relationship between number of particles and octree map memory in Wakatón (each particle taking about 1 Mb of memory). The red line shows what a linear relationship would be.

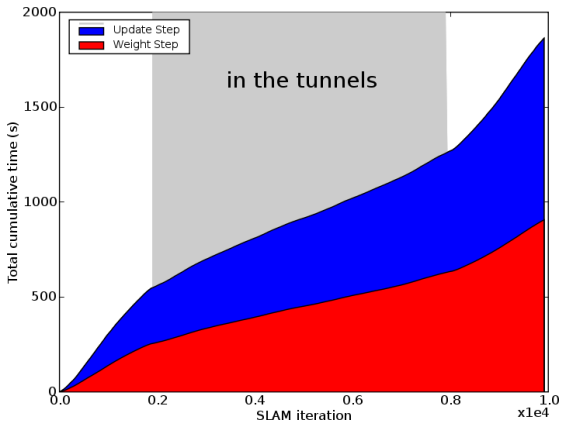

Figure 15: Cumulative processing time of the SLAM loop with 200 particles is shared almost equally by two of the steps: weighting and updating. It also shows that while the vehicle is in the tunnels at the bottom of Wakatón, the iteration duration is shorter, as we would expect since the sonar ranges are shorter.

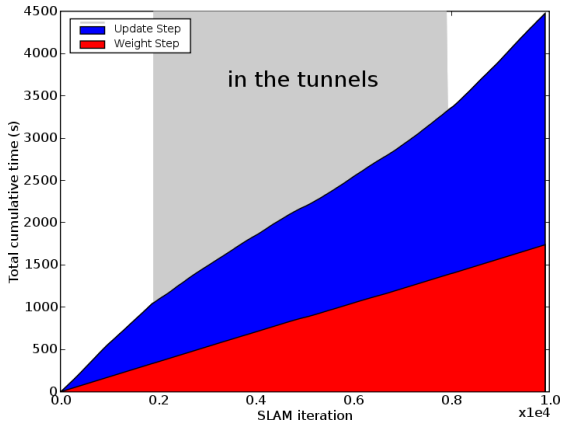

Figure 16: Cumulative processing time of the time limited variant of the particle filter. For this run, the weighting step was limited to 0.18s. Compare the improved consistency versus a fixed number of particles (Figure 15).

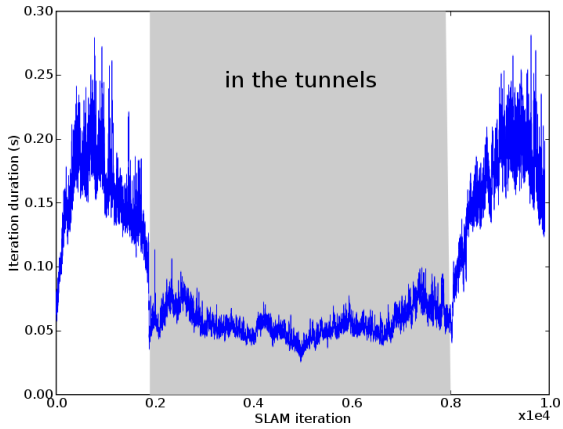

Figure 17: Processing time of the SLAM loop drops significantly while the vehicle is in the tunnels at the bottom of Wakatón, as we would expect since the sonar ranges are shorter. This indicates that the particle filter could use more particles while in the tunnels.

100 and 600 particles, with an average of around 300.

We expect that map quality will degrade as the sonar ranges increase as a natural consequence of the spreading of the sonar beams, and also the fact that coverage will tend to be more sparse (per unit area). At the same time, localization quality may well be better with long ranges because the vehicle will be sensing over a large area, which should provide strong localization information. It seems that as sonar ranges increase, precision degrades but overall accuracy improves. The is true in small tunnels. This phenomenon explains we see improved performance with a varying number of particles (Figure 20): the filter maintains accuracy in enclosed areas because of the increased number of particles which it can support.

7 Conclusions

The core result is that our method accurately localizes the vehicle in the ARL tank test dive and over the course of several synthesized test dives in the Wakatón environment. This demonstrates improvement over pure dead-reckoned navigation in both convergence in vehicle trajectory (returning to a previously mapped path) and map quality.

We have developed an efficient octree data structure

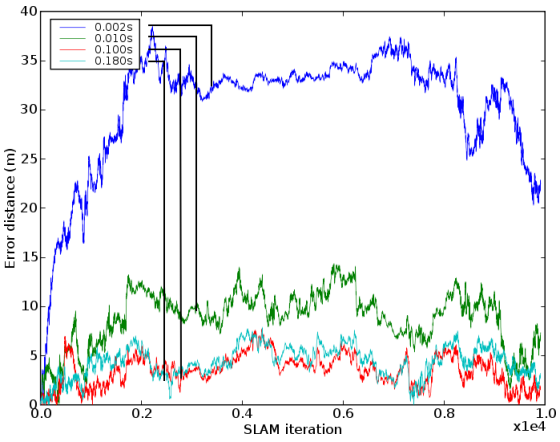

Figure 18: Localization error over time for different time limits for the weighting function. As expected, the particle filter performance improves as the amount of time it is allowed to spend weighting particles increases.

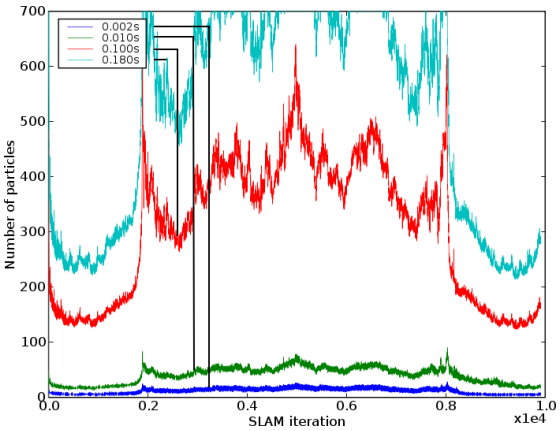

Figure 19: Number of particles evaluated (weighted) over time for different time limits for the weighting function (on the DEPTHX SLAM processor). As expected, the particle filter is able to evaluate more particles as the amount of time it is allowed to spend weighting particles increases.

for manipulating 3D evidence grids. This is the key innovation which allows the efficient implementation of a Rao-Blackwellized particle filter for 3D SLAM. Using the ARL tank test and synthetic Wakatón environment, we have demonstrated that the target computing platform can support hundreds of particles – enough to provide an accurate SLAM solution – in real-time. We also demonstrated a robust real-time particle filter implementation that adjusts the particle count in order to achieve accurate timing in the face of varying environmental geometry. This allows the algorithm to satisfy real-time constraints while using as many particles as possible, which has demonstrably improved performance over the conservative static particle count.

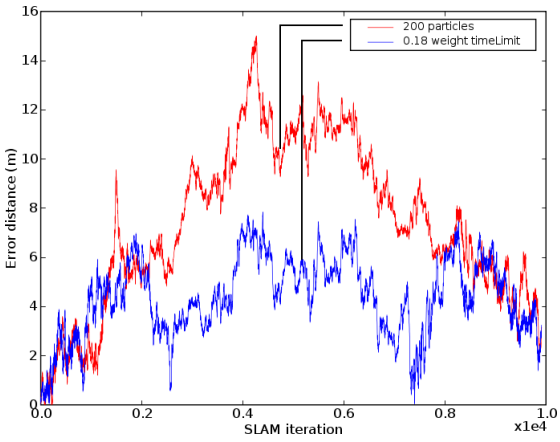

Figure 20: Comparison of localization performance of the two variants of the particle filter. In blue, the particle count was allowed to vary, but the weighting time limit was set to 0.18s. In red, the particle filter used a fixed particle count of 200, which is the maximum static number of particles that could be safely specified to achieve the same 0.18s real-time performance (Figure 19).

The next phase of this research involves intensive integration and testing of SLAM on the DEPTHX vehicle. This will include definitive characterization of the SLAM algorithm, in particular the performance under various failure conditions. We plan to conduct field experiments in Zacatón in 2007.

Acknowledgments

The DEPTHX project is managed by Stone Aerospace, William Stone Principal Investigator, and supported by the NASA ASTEP program, Carl Pilcher, Program Scientist, and David Lavery, Program Executive. We would like to thank Bill Stone, Barbara am Ende, and the U.S. Deep Cave Diving Team for providing the Wakulla Springs data set. We also thank Bill Stone for the illustration of the DEPTHX vehicle. Nathaniel Fairfield would like to thank Matt Zucker and John Fairfield for their technical discussions of octrees.

References

Arulampalam, S., Maskell, S., Gordon, N., and Clapp, T. (2002). A tutorial on particle filters for on-line non-linear/non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188.

Bachmann, A. and Williams, S. (2003). Terrain aided underwater navigation - a deeper insight into generic

monte carlo localization. In *Proc. of the Australasian Conf. on Robotics and Automation*.

Baker, H. (1994). Minimizing reference count updating with deferred and anchored pointers for functional data structures. *ACM SIGPLAN Notices*, 29(9):38–43.

Bradley, D., Silver, D., and Thayer, S. (2004). A regional point descriptor for global localization in subterranean environments. In *IEEE Conf. on Robotics Automation and Mechatronics*.

Bresenham, J. (1965). Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30.

Doucet, A., de Freitas, N., Murphy, K., and Russell, S. (2000). Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proc. of the Sixteenth Conf. on Uncertainty in AI*, pages 176–183.

Eliazar, A. and Parr, R. (2006). Hierarchical linear/constant time slam using particle filters for dense maps. In Weiss, Y., Schölkopf, B., and Platt, J., editors, *Advances in Neural Information Processing Systems 18*, pages 339–346.

Eustice, R., Singh, H., Leonard, J., Walter, M., and Ballard, R. (2005). Visually navigating the RMS Titanic with SLAM information filters. In *Proc. of Robotics Science and Systems*.

Fairfield, N., Kantor, G. A., and Wettergreen, D. (2005). Three dimensional evidence grids for SLAM in complex underwater environments. In *Proc. of the 14th Intl. Symposium of Unmanned Untethered Submersible Technology*.

Fairfield, N., Kantor, G. A., and Wettergreen, D. (2006). Towards particle filter slam with three dimensional evidence grids in a flooded subterranean environment. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*.

Fox, D. (2003). Adapting the sample size in particle filters through KLD-sampling. *I. J. Robotic Res.*, 22(12):985–1004.

Gary, M. O. (2002). Understanding Zacatón: Exploration and initial interpretation of the world's deepest known phreatic sinkhole and related karst features, southern Tamaulipas, Mexico. *Karst Frontiers, Karst Waters Institute Special Publication*, 7:141–145.

Hähnel, D., Burgard, W., and Thrun, S. (2003). Learning compact 3d models of indoor and outdoor environments with a mobile robot. *Robotics and Autonomous Systems*, 44(1):15–27.

Havran, V. (1999). A summary of octree ray traversal algorithms. *Ray Tracing News*, 12(2).

Healey, A. J., An, P. E., and Marco, D. B. (1998). On line compensation of heading sensor bias for low cost AUV's. In *Proc. of the IEEE Workshop on Autonomous Underwater Vehicles*, pages 35–42.

Larsen, M. B. (2000). High performance doppler-inertial navigation experimental results. In *Proc. of IEEE/MTS OCEANS*, pages 1449–1456.

Leonard, J. J., Bennett, A. A., Smith, C. M., and Feder, H. J. S. (1998). Autonomous underwater vehicle navigation. Technical report, MIT Marine Robotics Laboratory.

Mahon, I. and Williams, S. B. (2003). Three-dimensional robotic mapping. In *Proc. of the Australasian Conf. on Robotics and Automation*.

Martin, M. C. and Moravec, H. (1996). Robot evidence grids. Technical Report CMU-RI-TR-96-06, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.

Montemerlo, M., Thrun, S., Koller, D., and Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the AAAI National Conference on Artificial Intelligence*, pages 593–598.

Murphy, K. (1999). Bayesian map learning in dynamic environments. In *Neural Information Processing Systems*, pages 1015–1021.

Newman, P. M., Leonard, J. J., and Rikoski, R. J. (2003). Towards constant-time slam on an autonomous underwater vehicle using synthetic aperture sonar. In *Eleventh Intl. Symposium of Robotics Research*, pages 409–420.

Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. *Autonomous robot vehicles*, pages 167–193.

Stone, W. C., am Ende, B. A., Wefer, F. L., and Jones, N. A. (2000). Automated 3d mapping of submarine tunnels. In *Robotics 2000*, pages 148–157.

Thrun, S., Haehnel, D., Ferguson, D., Montemerlo, M., Triebel, R., Burgard, W., Baker, C., Omohundro, Z., Thayer, S., and Whittaker, W. L. (2003). A system for volumetric robotic mapping of abandoned mines. In *Proc. of the 2003 IEEE Intl. Conf. on Robotics and Automation*.

Urick, R. J. (1983). *Principles of Underwater Sound*. McGraw-Hill, New York, NY, 3rd edition.

Weingarten, J. and Siegwart, R. (2005). EKF-based 3D SLAM for structured environment reconstruction. In *Proc. of Intelligent Robots and Systems*, pages 3834–3839.

Whitcomb, L., Yoerger, D. R., and Singh, H. (1999). Combined doppler/LBL based navigation of underwater vehicles. In *Proc. of the 11th Intl. Symposium on Unmanned Untethered Submersible Technology*.

Williams, S. (2003). A terrain aided tracking algorithm for marine systems. In *Intl. Conf. on Field And Service Robotics*, pages 55–60.

Williams, S. B. and Mahon, I. (2004). Simultaneous localisation and mapping on the great barrier reef. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, volume 2, pages 1771–1776.

Williams, S. B., Newman, P., Dissanayake, G., and Durrant-Whyte, H. (2000). Autonomous underwater simultaneous localisation and map building. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 22–28.