Robotic vehicles working in unknown environments require the ability to determine their location while learning about obstacles located around them. In this paper a method of solving the SLAM problem that makes use of compressed occupancy grids is presented. The presented approach is an extension of the FastSLAM algorithm which stores a compressed form of the occupancy grid to reduce the amount of memory required to store the set of occupancy grids maintained by the particle filter. The performance of the algorithm is presented using experimental results obtained using a small inexpensive ground vehicle equipped with LiDAR, compass, and downward facing camera that provides the vehicle with visual odometry measurements. The presented results demonstrate that although with our approach the occupancy grid maintained by each particle uses only 40% of the data needed to store the uncompressed occupancy grid, we can still achieve almost identical results to the approach where each particle filter stores the full occupancy grid.

1. Introduction

One of the most important problems in the development of an unmanned vehicle (UMV) is giving it the ability to be placed in a completely unknown environment and allowing it to determine (1) its surroundings, based on what it “sees” using available sensors, and (2) its position, based on how far it has moved and what it sees. In the field of robotics this problem is referred to as Simultaneous Localization and Mapping (SLAM) or Concurrent Mapping and Localization (CML). In this paper we develop a method of solving the SLAM problem which implements a representation of the world using a compressed occupancy grid [1].

The approach we develop is an extension of the FastSLAM algorithm [2] which is one of the first solutions to the SLAM problem that makes use of a particle filter. More specifically the FastSLAM algorithm uses a Rao-Blackwellized [3, 4] particle filter to generate a position estimate of the UMV while each particle in filter maintains its copy of the occupancy grid. Based on the expected size of the environment in which a UMV operates and the level of detail required, the size of each occupancy grid can become extremely large (matrices holding hundreds of thousands of entries). Furthermore, as discussed further in Section 3.1.2, the probability that the estimate produced by the FastSLAM algorithm is accurate increases as the number of particles in the particle filter increases. For these reasons the amount of computer resources required to perform the algorithm can grow quickly. This problem is exacerbated in small form factor and low cost UMVs as the computing resources available to vehicles of this type are much less than the resources available to larger and more expensive vehicles. The approach that we develop extends the FastSLAM algorithm [2] to use compressed occupancy grids to overcome the memory restrictions that can occur when storing full occupancy grids without making drastic changes to the original algorithm.

The remainder of this paper is organized as follows. The mathematical notations used in this paper along with some mathematical preliminaries are presented in Section 2. The updated FastSLAM algorithm that makes use of compressed occupancy grids is introduced in Section 3. In Section 4 four separate approaches for reconstructing the compressed occupancy grid are examined. Section 5 presents a set of performance improvements to some components of the
algorithm. In Section 6 experimental results are provided and concluding remarks are presented in Section 7.

2. Mathematical Preliminaries

In much of the existing literature, the SLAM problem is addressed in a probabilistic sense. In many cases we would like to estimate the probability, also referred to as the distribution, of some random variable \( x \in \mathbb{R} \) and we denote the distribution of \( x \) as \( p(x) \). In many cases we use some additional information \( y \in \mathbb{R} \) to tell us something about the random variable \( x \). In this situation \( p(x) \) is referred to as the \textit{prior probability} and it is all that we know about the probability of \( x \) without the inclusion of \( y \). In many cases this is shortened to \textit{prior}. The distribution of \( x \) with the inclusion of the data \( y \) is denoted as \( p(x \mid y) \) and referred to as the \textit{posterior probability} and in many cases is just referred to as the \textit{posterior}.

In SLAM we are attempting to estimate the pose \( \xi_k \) of a UMV and a map \( M_k \) of the environment that surrounds the UMV at some time step \( k > 0 \). In this paper we treat the world in which the UMV operates as a two-dimensional plane; thus \( \xi_k \triangleq [x_k \ y_k \ \theta_k]^T \), where \( [x_k \ y_k] \in \mathbb{R}^2 \) are the horizontal and vertical positions of the vehicle in some frame of reference and \( \theta_k \in (-\pi, \pi] \) is the heading of the UMV with respect to the positive horizontal axis in the frame of reference. From our use of occupancy grids, the estimated map is represented by a matrix \( M_k \in \mathbb{R}^{c \times r} \), where \( r \in \mathbb{N} \) are the number of rows in the grid and \( c \in \mathbb{N} \) are the number of columns. We assume that the environment that the UMV operates is static; therefore during the time in which the UMV is performing the SLAM algorithm the environment does not change. Based on this assumption, to simplify the notation, the occupancy grid is denoted with \( M \). The estimate produced by SLAM, in many cases, makes use of sensor measurements and control inputs. The set of sensor measurements \( z_{1:k} \) denotes the full set of sensor measurements for \( k > 0 \); thus \( z_{1:k} \triangleq [z_1, \ldots, z_k] \), where \( z_i \in \mathbb{R}^m, i = 1, 2, \ldots, k \), and \( m \in \mathbb{N} \) is the number of measurements in \( z_i \). In the same way the full set of control inputs are defined as \( u_{1:k} \triangleq [u_1, \ldots, u_k] \), where \( u_i \in \mathbb{R}^c, i = 1, 2, \ldots, k \), and \( c \in \mathbb{N} \) is the size of the control vector \( u_i \). With the notation defined and the required mathematical preliminaries provided, the FastSLAM algorithm that makes use of compressed occupancy grids is presented in the following section.

3. FastSLAM with Compressed Occupancy Grids

In general the goal of SLAM is to estimate the pose \( \xi_k \) of a UMV and the map, \( M \), of the unknown environment in which the UMV is operating at some time step \( k > 0 \). We present an extension of the FastSLAM algorithm that uses compressed occupancy grids to store the map. Before presenting our approach, an overview of the FastSLAM algorithm that uses standard occupancy grids (FastSLAM OG) is provided.

3.1. FastSLAM with Occupancy Grids

As opposed to many SLAM solutions that attempt to solve the online SLAM problem that is estimating the posterior that represents the pose of a UMV and the map of the environment based on a set of control inputs and sensor measurements, the FastSLAM approach attempts to estimate the full SLAM posterior. The full SLAM posterior is given as

\[
p(\xi_{1:k}, M \mid u_{1:k}, z_{1:k}),
\]

which is the distribution that represents the trajectory of UMV \( \xi_{1:k} \) and the map of the environment \( M \) based on the set of control inputs \( u_{1:k} \) and observations \( z_{1:k} \). By estimating the full SLAM posterior, \( 1 \) can be factored, using the property of conditional independence, into a pair of simpler terms which is given by

\[
p(\xi_{1:k}, M \mid u_{1:k}, z_{1:k}) = p(\xi_{1:k} \mid u_{1:k}, z_{1:k}) \times p(M \mid \xi_{1:k}, u_{1:k}, z_{1:k}),
\]

where \( p(\xi_{1:k} \mid u_{1:k}, z_{1:k}) \) is the distribution that represents the UMV trajectory and \( p(M \mid \xi_{1:k}, u_{1:k}, z_{1:k}) \) is the distribution that represents the map of the environment. The FastSLAM approach estimates the factored SLAM posterior \( 2 \) using a Rao-Blackwellized particle filter. The Rao-Blackwellized particle filter is a version of a sampling importance resampling (SIR) \( 5 \) particle filter and it is the SIR particle filter that provides the format of the FastSLAM solution. The Rao-Blackwellized particle filter estimates the distribution representing a portion of the state, which in FastSLAM is the UMV trajectory \( \xi_{1:k} \), using a particle filter, and each particle in the particle filter maintains its own estimate of the remainder of the state, which is the map \( M \). As a result, each particle in the particle filter is composed of a UMV pose and a map, which the FastSLAM OG algorithm represents using an occupancy grid; thus the \( p \)th particle at time \( k \) is defined as

\[
\xi_k^{[p]} \triangleq [\xi_k^{[p]} \ M_k^{[p]}],
\]

where \( \xi_k^{[p]} \) is the pose estimate of the \( p \)th particle at time \( k \), \( M_k^{[p]} \) is the occupancy grid of the environment maintained by the particle as it has been generated through time step \( k \), and \( p \in [1, n] \) is the index corresponding to the \( p \)th particle where \( n \) is the total number of particles used in the particle filter. The FastSLAM approach is implemented in a four-step procedure based on the use of a SIR particle filter to estimate \( 1 \) and an overview of the algorithm is presented in Algorithm 1 for time \( k \).

3.1.1. Motion Model Update. In the first step of the algorithm (Line \( 4 \) in Algorithm 1) a new particle pose is generated and added to a set of temporary particles, \( \xi_{k-1}^{[\zeta_k]} \). The new pose for the \( p \)th particle is generated by sampling from the probabilistic motion model of the UMV:

\[
\xi_k^{[p]} \sim p(\xi_k \mid \xi_{k-1}^{[p]}, u_k).
\]
The probabilistic motion model of the UMV generates a new pose based on the state transition model of the UMV, the pose of the $p$th particle at time step $k-1$, the current control input $u_k$, and the characteristics of the noise on $u_k$. The result is that the particles in $\mathcal{X}_k$ are distributed according to

$$p(\xi_{1:k} | u_{1:k}, z_{1:k-1}), \quad (5)$$

which is referred to as the proposal distribution. The actual distribution which we are attempting to estimate is referred to as the target distribution and given as

$$p(\xi_{1:k} | u_{1:k}, z_{1:k}). \quad (6)$$

The proposal distribution is transformed to the target distribution in the last stage of the algorithm, the resampling stage (Lines 10–13) in Algorithm 1. Before the resampling process can occur, $z_k$ must be incorporated into the estimate, which is achieved in the calculation of the particle’s importance weight.

3.1.2. Importance Weight Calculation. In the second step of the algorithm an importance weight for each particle is generated. The need for an importance weight for each particle comes from the use of a SIR particle filter and, as seen in the previous section, we are sampling particles from a distribution different from the one that we are attempting to estimate. According to [6], if we are unable to directly sample from the distribution that we are attempting to estimate, then the importance weight for each particle is given as

$$w_k^{[p]} = \frac{\text{target distribution}}{\text{proposal distribution}}, \quad (7)$$

and particles are drawn from $\mathcal{X}_k$ with replacement and added to $\mathcal{X}_k$ with a probability proportional to $w_k^{[p]}$. When drawing particles from $\mathcal{X}_k$ with replacement, each particle that is added to $\mathcal{X}_k$ remains in $\mathcal{X}_k$ so that a given particle that is selected has the potential of being added to $\mathcal{X}_k$ multiple times. After the resampling process $\mathcal{X}_k$ will approximate the target distribution and the quality of the approximation will improve, the number of particles increases.

Using (7) along with the previously discussed distributions (5) and (6), the importance weight of each particle simplifies to [7]

$$w_k^{[p]} = \frac{p(\xi_{1:k} | u_{1:k}, z_{1:k})}{p(\xi_{1:k} | u_{1:k}, z_{1:k-1})} = \eta p(z_k | \xi_{1:k}^{[p]}, M_{1:k}^{[p]}), \quad (8)$$

where $p(z_k | \xi_{1:k}^{[p]}, M_{1:k}^{[p]})$ is the probabilistic measurement model of the sensor being used by the UMV and $\eta > 0$ is a normalizing factor. The probabilistic measurement model is dependent on the perceptual sensor being used by the UMV and it calculates the probability that the sensor measurement would make based on the current pose estimate and map maintained by the particle. This form of importance weight incorporates the sensor observations into the estimate which were not included in the pose sampling procedure. Once the importance weight has been calculated for each particle then the occupancy grid maintained by each particle is updated using the new particle pose and current sensor measurement.

3.1.3. Occupancy Grid Update. Once the importance weight for each particle has been calculated, the third step of the algorithm updates the occupancy grid of each particle using the current sensor measurement and pose estimate. Occupancy grids decompose an unknown environment into a finite set of cells where each cell holds the probability that the location in the world enclosed by the cell is occupied by an object. From [8], the occupancy grid for a given particle at time $k$ is obtained in a probabilistic sense by calculating the posterior over maps based on the history of sensor measurements and trajectory of a particle:

$$p(M_k^{[p]} | \xi_{1:k}^{[p]}, z_{1:k}). \quad (9)$$

By decomposing the environment into cells, the posterior calculated by the occupancy grid can be restated as a product of cellular posteriors:

$$p(M_k^{[p]} | \xi_{1:k}^{[p]}, z_{1:k}) = \prod_{i,j} p(M_k^{[p]}(i,j) | \xi_{1:k}^{[p]}, z_{1:k}). \quad (10)$$

where $p(M_k^{[p]}(i,j) | \xi_{1:k}^{[p]}, z_{1:k})$ contains the probability that the location in the environment represented by the cell at $(i,j)$ is occupied based on the measurement history and particle trajectory. A solution to a problem of this form is the binary Bayes filter [8]. The binary Bayes filter stores the probability of occupancy for each cell in log odds form to prevent truncation errors that can occur during the update process. The log odds form of the variable $x \in \mathbb{R}$ is given as

$$l(x) \triangleq \log \frac{x}{1-x}, \quad (11)$$

and the value of $x$ can be recovered from the log odds form as

$$x \triangleq 1 - \frac{1}{1 + \exp(l(x))}. \quad (12)$$

Algorithm 1: FastSLAM OG algorithm.
The binary Bayes filter based occupancy grid update modifies the probability of occupancy of each cell in the grid when an observation provides information about that location. As a consequence, for locations that have been observed, the probability of occupancy quickly approaches 0 for free cells and 1 for occupied cells. While the addition of new information to the estimate makes the probability quickly approach the limits, the probability of occupancy of each cell should never equal 0 or 1 as this would assume perfect knowledge about the occupancy of the cell. Depending on the data type selected to store the probability, the precision required to store the value can quickly be exceeded resulting in the probability being rounded to either 0 or 1. Once this rounding has occurred any new information learned about the occupancy of a cell is lost. To overcome this problem the binary Bayes filter stores the probability of occupancy in log odds form which takes the potential probability values in [0, 1] and spreads them over the range (−∞, ∞). By spreading the small range of probability values over this much larger range, the data type being used to store the probability has more precision to store the probability thus preventing the truncation/rounding from occurring.

The occupancy grid update algorithm based on the binary Bayes filter is provided in Algorithm 2.

In Algorithm 2, \( p(M(i, j) | \xi_k, z_k(l)) \) is the inverse sensor model of the sensor being used to generate the map and it returns the probability that the cell at \( (i, j) \) is occupied based on the \( l \)th sensor measurement and the pose estimate of the particle. The value of \( l_0 \) in Algorithm 2 is referred to as the prior of occupancy and is defined as

\[
l_0 = \log \frac{p(M(i, j))}{1 - p(M(i, j))}
\]

and gives the probability that a cell is occupied before any sensor measurements are integrated into the map. Typically \( p(M(i, j)) \) in (13) is selected to be 0.5.

### 3.1.4. Resampling

After the occupancy grid for each particle in \( \mathcal{X}_k \) has been updated, the final step of the algorithm is the resampling process. During the resampling process particles are drawn with replacement from \( \mathcal{X}_k \) with a probability proportional to \( w^{[l]} \). This resampling step takes the set of temporary particles that are distributed according to (5) and ensures that there is a high probability that \( \mathcal{X}_k \) is distributed according to (6).

One of the problems with this approach is the large amount of memory required to run the algorithm. The estimate generated by the algorithm improves as the number of particles increases; thus we like to have as many particles as possible. However, each particle must keep a copy of the occupancy grid and each occupancy grid can be quite large. It is easy to see that while we would like to run the particle filter with as many particles as possible we can be limited by the fact that each particle must maintain a full copy of the occupancy grid. To address this problem we propose an updated version of the algorithm that decreases the amount of memory required for each particle by storing a compressed version of the occupancy grid as opposed to the full uncompressed occupancy grid.

### 3.2. FastSLAM with Compressed Occupancy Grids

Before discussing the updated SLAM algorithm, we will first address the compressed form of the occupancy grid. To compress \( M_k^{[p]} \) we first reshape the occupancy grid so that \( r \times c \) matrix becomes a column vector containing \( n = rc \) elements and we refer to the vector form of the occupancy grid as \( m_k^{[p]} \). The reshaping process is performed in such a way that the cell in the original occupancy grid located at \((i, j)\) can be accessed in the vector form at \( l \), where \( l = ic + j \). In order to compress the occupancy grid without major degradation, we assume that the occupancy grid can be represented in an alternate basis in which it is sparse. The relationship between the two different representations is given as

\[
m_k^{[p]} = \Phi c_k^{[p]},
\]

where \( \Phi \) is a transformation matrix that defines the relationship and \( c_k^{[p]} \) is the set of coefficients that represent the occupancy grid in the alternate basis. In order for the occupancy grid to be compressible, \( c_k^{[p]} \) must be sparse [9]; that is \( \#\supp(c_k^{[p]}) < n \), where \( \#\supp(\cdot) \) is the support operator and \# denotes the cardinality of the set. This means
that, in order for us to be able to compress the occupancy grid, we must find a basis in which the coefficient vector is composed of a large number of zeros. If this condition is met then the occupancy grid for a given particle can be uncompressed using

\[ y_k^{[p]} = Hc_k^{[p]}, \]  

where \( H \) is a matrix that performs the compression by selecting a subset of coefficient data that is smaller than the amount of data in the complete signal.

The FastSLAM COG algorithm is a modification of Algorithm 1. Since we are now maintaining the compressed form of the occupancy grid, the steps of the algorithm that require the occupancy grid, the occupancy grid is stored in vector form, and the occupancy grid update. Therefore, the only change that must be addressed are the steps of the algorithm that require the occupancy grid, the importance weight calculation, and occupancy grid update.

3.2.1. Importance Weight Calculation. As discussed in Section 3.1.2, the importance weight for each particle is the normalized probabilistic measurement model for each particle. The probabilistic measurement model is a measure of how well the current set of sensor measurements matches up with what we expect based on the current pose and map of the particle. In calculating the probabilistic measurement model, the algorithm must have access to the raw occupancy grid values so that an expected sensor measurement can be generated. Based on this requirement, the compressed occupancy grid coefficients are first uncompressed. For now a general reconstruction operator denoted as \( H \) is used; however the details of several reconstruction methods are explored in detail in Section 4. Once the compressed coefficients have been reconstructed, the occupancy grid can be extracted using (14). The importance weight calculation can be found in Algorithm 3 on Line (6).

3.2.2. Compressed Occupancy Grid Update. As previously seen in Algorithm 2, the standard approach to updating the occupancy grid is to examine every cell in the occupancy grid and if a given cell falls within the perceptual range of the sensor attached to the UMV then the corresponding cell of the occupancy grid is updated according to

\[ M_k^{[p]} (i, j) = M_{k-1}^{[p]} (i, j) + \log \frac{p \left( M(i, j) \mid x_k^{[p]}, z_k(l) \right)}{1 - p \left( M(i, j) \mid x_k^{[p]}, z_k(l) \right)} - l_0. \]

Using Algorithm 2 as our basis, we rewrite the occupancy grid update in vector form. If we assume that the full occupancy grid is stored in vector form, then the occupancy grid update can be performed using

\[ m_k^{[p]} = m_{k-1}^{[p]} + \alpha_k^{[p]} - \beta, \]

where \( \alpha_k^{[p]} \) is constructed using Algorithm 4 and \( \beta \) is vector with every element containing \( l_0 \).

Algorithm 4 first initializes \( \alpha_k^{[p]} \) using \( l_0 \). This ensures that cells that do not fall within the perceptual range of the sensor remain unchanged by the algorithm. For cells that fall into the perceptual range of the sensor, their \( \alpha \) value is set as the log odds form of the inverse sensor model and it is this value that either increases or decreases the likelihood that the cell is occupied. Equation (18) updates the occupancy grid in vector form; however we need to modify it so that it updates the occupancy grid which is stored using an alternative basis.

Algorithm 3: FastSLAM algorithm with compressed occupancy grids.
Incorporating (14), the vector form of the occupancy grid update in the alternate basis becomes

\[ c_k^{[p]} = c_{k-1}^{[p]} + \Phi^\top \alpha_k^{[p]} - \Phi^\top \beta, \]  

(19)

where \( \Phi^\top \) is used to convert \( \alpha_k^{[p]} \) and \( \beta \) to the alternate basis. In (19), \( \Phi^\top \) is used to change basis as opposed to \( \Phi^{-1} \) because we assume that our choice of basis is orthonormal so \( \Phi^{-1} = \Phi^\top \); however if the alternative basis that is selected is not orthonormal then \( \Phi^{-1} \) should replace \( \Phi^\top \) in (19). We can now simplify (19) by grouping the two components that are multiplied by \( \Phi^\top \) to reduce the number of matrix-vector multiplications; this yields

\[ c_k^{[p]} = c_{k-1}^{[p]} + \Phi^\top [\alpha_k^{[p]} - \beta]. \]  

(20)

Equation (20) updates the occupancy grid in the alternate basis; hence the final step is to incorporate compression into the update. The occupancy grid maintained by each particle contains the compressed set of basis of coefficients; thus (20) can be updated using (15) to yield

\[ c_k^{[p]} = H_r y_k^{[p]} + \Phi^\top [\alpha_k^{[p]} - \beta]. \]  

(21)

Finally, the basis coefficients used to represent the occupancy grid must be recompressed in order to be stored by the particle. Including the final compression the compressed occupancy grid update can be performed according to

\[ y_k^{[p]} = H_r y_{k-1}^{[p]} + \Phi^\top [\alpha_k^{[p]} - \beta], \]  

(22)

which is seen in Algorithm 3 Line (8).

### 4. Compressed Signal Reconstruction Methods

As previously described we would like to store a compressed version of the occupancy grid as opposed to the complete occupancy grid to decrease the memory requirements of the algorithm. In order to make use of the compressed occupancy grid, a method for reconstructing the full occupancy grid from the compressed version must be selected. In this section several reconstruction methods are examined with the goal of selecting an approach to be used in FastSLAM COG. In FastSLAM COG we are attempting to compress the vector form of an occupancy grid; however in this section we will examine methods for reconstructing a generic discrete signal \( x \in \mathbb{R}^n \) that has been compressed.

We represent \( x \) using some alternate basis in which it is sparse and the relationship between \( x \) and its sparse representation is given by

\[ x = \Phi c, \]  

(23)

where \( c \in \mathbb{R}^n \) is a vector of coefficients that represent \( x \) in the alternate basis and \( \Phi \in \mathbb{R}^{m \times n} \) is a matrix of basis functions that determines the relationship between \( x \) and \( c \). The discrete signal used in this section for testing is displayed in Figure 1(a) and the set of coefficients used to represent
the signal in the discrete cosine basis [10] can be seen in Figure 1(b). The test signal is generated by evaluating
\[ f(t) = \sin(t) - \cos(2t) - 3 \sin(0.25t), \]
at 128 evenly spaced points in [0, 10].

If \( c \) is sparse then \( x \) can be compressed; this sparseness can be seen by examining Figure 1(b) and noticing that a large number of the coefficients are zero. The compression is performed by selecting a subset of the coefficient information using
\[ y = Hc, \]
where \( H \in \mathbb{R}^{m \times n} \) is the compression matrix and \( m \in \mathbb{N} \) is the size of the compressed signal with \( m < n \).

4.1. \( l_2 \) Reconstruction \((l_2)\). One of the most simple methods for reconstructing a compressed signal is to reconstruct the signal while minimizing the error between the original and reconstructed signals as measured by \( l_2 \) norm, where \( l_2 \) norm of a signal \( x \in \mathbb{R}^n \) is defined as
\[ \|x\|_2 = \left( \sum_{i=1}^{n} x(i)^2 \right)^{\frac{1}{2}}. \]

In examining (25), if the size of the compressed signal is the same as the original then the reconstruction can be performed according to
\[ c = H^{-1}y. \]
In reality the system is underdetermined due to \( m < n \). A common solution to problems of this form uses the Moore-Penrose pseudoinverse of \( H \).

The Moore-Penrose pseudoinverse \([11,12]\) is used to find the solution that minimizes \( l_2 \) error for systems of the form
\[ Ax = b, \]
where \( x \in \mathbb{R}^n \), \( b \in \mathbb{R}^m \), and \( A \in \mathbb{R}^{m \times n} \). The solution to the system found using the pseudoinverse is defined as
\[ z = A^\dagger b, \]
where \( A^\dagger \) is the pseudoinverse of \( A \) and it minimizes \( l_2 \) error so that
\[ \|Ax - b\|_2 \geq \|Az - b\|_2, \]
for all \( x \in \mathbb{R}^n \) if \( x \) is defined as
\[ x = A^\dagger b + (I - A^\dagger A)w, \]
for any vector \( w \in \mathbb{R}^n \).

Using the pseudoinverse of \( H \) a set of reconstructed coefficients is generated according to
\[ \hat{c} = H^\dagger y, \]
and the reconstructed value of the signal is generated from
\[ \hat{x} = \Phi \hat{c}. \]

The results of \( l_2 \) reconstruction using four separate amounts of signal information, 25%, 50%, 75%, and 95%, are presented in Figure 2. The form selected for \( H \) needed to obtain such level of compression is an orthonormal Gaussian random matrix as it possesses the RIP property whose importance will be discussed in the following section.

4.2. Compressed Sensing Reconstruction \((CS)\). The second reconstruction algorithm selected is based on an approach referred to in the literature as compressed sensing [13]. Compressed sensing allows for sparse signals to be reconstructed using fewer measurements that would be required according to the Nyquist/Shannon sampling principle. In compressed sensing the full set of sparse coefficients can be reconstructed from the compressed set by solving
\[ \min \|\bar{c}\|_0 \]
subject to \[ \|y - H\bar{c}\|_2 \leq \epsilon, \]
for some sufficiently small \( \epsilon \). In (34) \( \|\cdot\|_0 \triangleq \#\text{supp}(\bar{c}) \) is a count of the number of nonzero elements in the vector \( \bar{c} \). Solving (34) is NP-hard and the solution cannot be found efficiently. A key finding of compressed sensing is that if the measurement matrix, \( H \), is constructed in such a way that it obeys the Restricted Isometry Property (RIP), which is defined as
\[ (1 - \delta) \|c\|_2^2 \leq \|Hc\|_2^2 \leq (1 + \delta) \|c\|_2^2, \]
form some small \( \delta \geq 0 \), then \( \|\cdot\|_0 \) in (34) can be replaced by \( \|\cdot\|_1 \) resulting in
\[ \min \|\bar{c}\|_1 \]
subject to \[ \|y - H\bar{c}\|_2 \leq \epsilon. \]
As opposed to being NP-hard to solve, (36) is a constrained convex optimization problem and can be solved efficiently using one of several available software packages with Lasso [14], NESTA [15], and l1 magic [16] being some examples. As in \( l_2 \) case, once the complete set of coefficients have been estimated using the compressed set, an estimate of the original signal is generated according to (33).

In order to examine the performance of the compressed sensing reconstruction approach, the test signal is reconstructed using the same four amounts of data, 25%, 50%, 75%, and 95%, with the same orthonormal Gaussian random matrix being used to perform the compression. Equation (36) is solved using the NESTA software library and the results of the reconstruction are shown in Figure 3.

4.3. Compressed Sensing Embedded Kalman Filtering \((CSEKF)\). In the previous section a reconstruction approach was introduced that allows for the reconstruction of compressed signals using less data than expected based on
the Nyquist/Shannon sampling principle. As seen in Figure 3, this method does a better job at reconstructing a sparse signal than the approach that reconstructs the signal by minimizing $l_2$ norm. The authors of [17] introduce an approach that performs the compressed sensing reconstructing using an augmented Kalman filter (KF) [18] which is easy to implement, compared to the compressed sensing approach. The compressed sensing approach must solve a complex convex minimization problem, where problems of this type are typically being solved using an external software package. In order to use the KF it is assumed that the full set of coefficients evolve from time step to time step according to

$$c_k = A_k c_{k-1} + \epsilon_k,$$

(37)

where $A_k \in \mathbb{R}^{m \times n}$ is the state transition matrix of the system which describes how the signal coefficients change between time steps, and $\epsilon_k \in \mathbb{R}^n$ is a zero mean Gaussian vector with covariance $R_k \geq 0$. The compressed form of the coefficients is generated according to

$$y_k = H_k c_k + \delta_k,$$

(38)

where $H_k \in \mathbb{R}^{m \times n}$ is the same compression matrix discussed in (25) and $\delta_k \in \mathbb{R}^m$ is a zero mean Gaussian vector with covariance $Q_k > 0$.

The CSEKF algorithm first performs a standard KF estimation that is carried out in a two parts, a prediction phase and a correction phase. The KF estimates $p(c_k \mid y_k)$ as a Gaussian distribution that can be represented using a mean
vector \( \tilde{c}_k \) and covariance matrix \( \Sigma_k \). The KF generates the estimate using a standard set of five equations [18]:

\[
\tilde{c}_k = A_k \tilde{c}_{k-1}, \tag{39}
\]

\[
\Sigma_k = A_k \Sigma_{k-1} A_k^T + R_k, \tag{40}
\]

\[
K_k = \Sigma_k H_k^T (H_k \Sigma_k H_k^T + Q_k)^{-1}, \tag{41}
\]

\[
\tilde{c}_k = \tilde{c}_k + K_k (y_k - H_k \tilde{c}_k), \tag{42}
\]

\[
\Sigma_k = (I - K_k H_k) \Sigma_k, \tag{43}
\]

where \( I \) in (43) is an \( m \) dimensional identity matrix. The estimate produced by the KF minimizes the error, using \( l_2 \) norm, between the estimate and the actual set of coefficients

\[
\min \| \mathbf{c}_k - \tilde{\mathbf{c}}_k \|_2^2; \tag{44}
\]

however, this is not what we are trying to solve according to the compressed sensing approach. The CSEKF presented by the authors replaces the classic compressed sensing convex optimization problem (36) with the dual problem:

\[
\min \| \mathbf{c}_k - \tilde{\mathbf{c}}_k \|_2^2 \quad \text{subject to} \quad \| \mathbf{c}_k \|_1 \leq \epsilon'. \tag{45}
\]

They solve this constrained optimization problem by iteratively applying the pseudoobservation method presented.
in [19] to the estimate generated by the KF. According to this approach a pseudoobservation is generated using the constraint being applied to the estimate which is defined as

\[ \tilde{z}_k = \| \tilde{c}_k \|_1 - \epsilon', \]  

(46)

where the constraining value \( \epsilon' \) is now treated as a Gaussian noise on the pseudomeasurement with a covariance of \( \mathbf{P} \). Using the definition of \( \| \cdot \|_1 \) the pseudomeasurement can be written as

\[ \tilde{z}_k = \mathbf{H}_k \tilde{c}_k - \epsilon', \]  

(47)

where the observation matrix of the pseudoobservation is defined as

\[ \mathbf{H} = [ \text{sign}(\tilde{c}_k(1)) \cdots \text{sign}(\tilde{c}_k(n))] , \]

(48)

and

\[ \text{sign}(x) = \begin{cases} 1 & x \geq 0 \\ -1 & \text{otherwise}. \end{cases} \]

The constraint is then iteratively applied \( \tau \) times to the estimate produced by the KF using the standard KF correction equations (41)–(43). First, a new Kalman matrix is generated using the observation matrix of the pseudomeasurement and the covariance of the constraint:

\[ \mathbf{K} = \Sigma_k \mathbf{H}_k^T (\mathbf{H}_k \Sigma_k \mathbf{H}_k^T + \mathbf{P})^{-1} . \]  

(49)

Next, the mean of the estimate is updated using the standard mean update equation with the assumption that the measured value is 0. Using this assumption the mean update equation can be simplified to

\[ \tilde{c}_k = (\mathbf{I} - \mathbf{K} \mathbf{H}) \tilde{c}_k . \]  

(50)

Finally, the covariance of the estimate is constrained using the standard covariance measurement update equation

\[ \Sigma_k = (\mathbf{I} - \mathbf{K} \mathbf{H}_k) \Sigma_k . \]  

(51)

The full algorithm that combines all of these separate components to reconstruct the compressed signal using the CSEKF is shown in Algorithm 5.

The reconstructed test function for the same four test percentages as in the previous two algorithms is shown in Figure 4. The algorithm is implemented as shown in Algorithm 5 with the tuning parameters being chosen as \( \mathbf{Q}_k = 0.01^2, \mathbf{R}_k = 0, \mathbf{P} = 100^2 \), and \( \tau = 1000 \). Since the input vector is static the initial estimate of the coefficient vector was zero, \( \tilde{c}_{k-1} = 0^T \), the state transition matrix is chosen as \( \mathbf{A}_k = \mathbf{I}^T \), and the initial covariance matrix of the estimate is chosen to be \( \Sigma_{k-1} = \mathbf{I}^T \).

4.4. Kalman Filter Constrained with Quasinorm (CSEKF-p). In the previous section an approach was presented that solves a problem similar to the standard compressed sensing problem (36) using a KF that is constrained by iteratively applying a pseudoobservation to the estimate generated by the KF. The standard approach to solving the compressed sensing problem replaces \( \| \cdot \|_1 \) in (34) with \( \| \cdot \|_p \). A new approach has been presented by the authors of [20] that replaces the zero norm with \( p \)-norm \( \| \cdot \|_p \) which for \( 0 < p < 1 \) is defined as

\[ \| \mathbf{x} \|_p = \left( \sum_{i=1}^{n} |x(i)|^p \right)^{1/p} . \]  

(52)

This approach has been shown to yield better accuracy than using \( \| \cdot \|_1 \) in some cases. Using this approach, we would like to constrain the estimate produced by the KF by the quasinorm \( \| \cdot \|_p \) as opposed to \( \| \cdot \|_1 \).

Just as in the previous section the initial estimate is produced using the standard KF equations. The pseudoobservation used to apply the constraint to the KF estimate using \( \| \cdot \|_p \) is defined as

\[ \tilde{z}_k = \| \tilde{c}_k \|_p - \epsilon', \]  

(53)

where \( \epsilon' \) is again treated as noise on the pseudomeasurement with a covariance of \( \mathbf{P} \). Unlike in the previous case where the constraint was linear and could be applied using the pseudoobservation matrix, the constraint applied by \( p \)-norm is nonlinear and can be rewritten as

\[ \tilde{z}_k = \mathbf{h}(\tilde{c}_k) - \epsilon' , \]  

(54)

where

\[ \mathbf{h}(\tilde{c}_k) = \| \tilde{c}_k \|_p . \]  

(55)

Since the constraint using \( p \)-norm is nonlinear the constraint is applied to the KF estimate by iteratively performing a measurement update using (54) and the Extended Kalman Filter (EKF) [21] update equations. Just as in the linear case the first step in applying the constraint is to construct the Kalman matrix according to

\[ \mathbf{K} = \Sigma_k \mathbf{H}_k^T (\mathbf{H}_k \Sigma_k \mathbf{H}_k^T + \mathbf{P})^{-1} , \]  

(56)
where $H$ is now the Jacobian of $h(\cdot)$ and is defined as

$$H(i) = \begin{cases} \left( \sum_{j=1}^{n} |\tilde{c}_k(i)|^p \right)^{1/(p-1)} |\tilde{c}_k(i)|^{p-1}, & \tilde{c}_k(i) \geq 0, \ \ (57) \\ -\left( \sum_{j=1}^{n} |\tilde{c}_k(i)|^p \right)^{1/(p-1)} |\tilde{c}_k(i)|^{p-1}, & \tilde{c}_k(i) < 0, \end{cases}$$

where $\tilde{c}_k = \bar{c}_k$. The constraint is applied to the mean of the estimate using the standard EKF mean measurement update equation, again with the assumption that the measured value is zero, which when simplified is written as

$$\bar{c}_k = \bar{c}_k - K \|\bar{c}_k\|_p. \ \ (58)$$

The final step in applying the constraint is to update the covariance of the estimate using the standard EKF covariance update equation

$$\Sigma_k = (I - K \bar{H}_k) \Sigma_k. \ \ (59)$$

The reconstruction results for the compressed sensing Kalman filter using a quasinorm to constrain the estimate are presented in Figure 5 for the same four data sizes used for the three previous algorithms. A parameter that greatly affects the performance of the algorithm is the value chosen for $p$. To determine the best value of $p$ the algorithm was...
5. Performance Improvements

Our overall goal is to augment the FastSLAM algorithm to use compressed occupancy grids. In order for us to accomplish this task the reconstruction approach selected must run thousands or hundreds of thousands of times during the operation of a UMV. By using a Rao-Blackwellized particle filter, we can increase the probability that our SLAM posterior matches the true SLAM posterior by increasing the number of particles maintained by the particle filter. As seen in Algorithm 3 the reconstruction process must be performed by each particle at each iteration; thus the execution time of the reconstruction process is important in the overall performance of the algorithm.

In Section 4 four different algorithms were presented that can be used to reconstruct an occupancy grid from the compressed form. From the timing results (Figure 8), the two approaches that make use of the Kalman filter (CSEKF and CSEKF-p) take much longer time to run than \( l_2 \) and CS. The longer run times of the Kalman filter based approaches are due to the large number of matrix-matrix multiplications. Based on their performance the CSEKF and CSEKF-p algorithms will not be used in FastSLAM COG. We will now present a method that can be used to improve the accuracy of the remaining two algorithms and a method that improves the run time of \( l_2 \) approach.

5.1. Updated Compression Step. As seen in Section 3.2 the compressed occupancy grid is generated by selecting a subset
of the information stored in the coefficients. This compression is performed according to (15) which makes use of $H$ to perform the compression. As stated in Section 4.2 we would like for this compression matrix to obey the RIP property.

Several common matrices that are used in compressed sensing applications are the discrete Fourier transform [23],
the discrete cosine transform [10], the Hadamard transform [24], and wavelet transform [25]. Each of these matrices is constructed in such a way that each of the elements stored in the compressed vector stores a small amount of information about each of the coefficients used to represent the original signal and this information can be recovered by knowing H. These types of matrices are useful when there is no knowledge about the sparse signal that is being compressed; however, we are constructing the sparse signal thus we have full knowledge of the signal before the compression is performed. Using this knowledge we present an alternative form of H that allows for better reconstruction using a smaller amount of information. The form of H selected comes from the definition of the square error defined according to $l_2$ norm. For some discrete signal $x \in \mathbb{R}^n$ the square of the error between the signal and the approximation of the signal $\tilde{x}$ generated using $m < n$ elements is defined as

$$\|x - \tilde{x}\|_2^2 = \langle x - \tilde{x} | x - \tilde{x} \rangle,$$

where $\langle \cdot | \cdot \rangle$ is the standard inner product. The discrete signal $x \in \mathbb{R}^2$ can be represented in some alternate basis so that a single element can be expressed as

$$x(i) = \sum_{j=1}^{n} \Phi(i, j) c(j).$$

Recalling that $\bar{y} \in \mathbb{R}^m$ denotes the compressed set of coefficients (25), a single element of the compressed signal can be expressed in the alternate basis as

$$\tilde{x}(i) = \sum_{j=1}^{m} \bar{\Phi}(i, j) \bar{c}(j),$$

where

$$\bar{\Phi} = \Phi H^\top,$$

$$\bar{c} = y.$$  

While this is a valid approach for compressing a signal, we would like to simplify the compression process for performance reasons. Instead of generating a new set of basis function it is easier to keep the basis functions the same at all times and just select a subset of the coefficients to represent the signal. In order to decide which coefficients should be selected that minimize the error based on $l_2$ norm, we define several sets. The set $\zeta = \{x | x \in \mathbb{N}, 1 \leq x \leq n\}$ is the full set of indices of the discrete vector being compressed. The set $\lambda \in \mathbb{N}_m$ is the set of indices that are used to represent the compressed signal, $\lambda \subset \zeta$. Finally, the set $y \in \mathbb{N}_m$ is the set of indices of coefficients not used in the compressed signal, $y = \{x | x \in \zeta$ and $x \notin \lambda\}$. Using these sets and the fact that a constant basis is used to represent the uncompressed and compressed signal an element in the reconstructed signal can be written as

$$\tilde{x}(i) = \sum_{j=1}^{m} \Phi(i, \lambda(j)) c(\lambda(j)).$$

The square of the error in $l_2$ norm for a single element becomes

$$\|x(i) - \tilde{x}(i)\|_2^2 = \langle x(i) - \tilde{x}(i) | x(i) - \tilde{x}(i) \rangle = \sum_{k=1}^{n-m} \Phi(i, y(k)) c(y(k)) \sum_{j=1}^{n-m} \Phi(i, y(j)) c(y(j))*c(y(k)) = \sum_{j=1}^{n-m} \Phi(i, y(j)) c(y(j))^2 \sum_{k=1}^{n-m} \Phi(i, y(k)) c(y(k))^2$$

and if we assume that the set of basis function is orthonormal, which is common, then $l_2$ error for the entire compressed signal becomes

$$\|x - \tilde{x}\|_2^2 = \sum_{i=1}^{n-m} \sum_{j=1}^{n-m} c(y(j))^2.$$  

This tells us that the best way to minimize the error in the reconstructed signal is to select the coefficients that have the largest magnitudes. Using this knowledge a new compression algorithm that does not require matrix-vector multiplications and selects only $m$ coefficients with the largest magnitudes was developed and shown in Algorithm 7.

To examine how the use of the new compression algorithms affects the performance of the reconstruction algorithms, the occupancy grid shown in Figure 9(a) is reconstructed from the compressed form. The compression matrix selected for the baseline results is different from the orthonormal Gaussian random matrix used in Section 4; in its place a normalized Hadamard matrix was used. Each of the two algorithms was then run using each of the two compression methods for 20 separate compression percentages in $0, 100\%$. The performance results of each of the two algorithms are provided in Figure 10. As seen from the results for each of the two reconstruction algorithms the occupancy grid is reconstructed with a smaller MSE using the
Algorithm 7: Updated compression algorithm that represents the compressed occupancy grids with a set of coefficients and corresponding indices.

```
(1) procedure FastCompression(c, m)
(2)   x = sort(abs(c), "descend")
(3)   for i = 1, m do
(4)       i(i) = x(i)
(5)       y(i) = c(x(i))
(6)   end for
(7) end procedure
```

5.2. Updated $l_2$ Reconstruction. In Section 4.1 a method for reconstructing a compressed signal was introduced that minimizes the error according to $l_2$ norm. The method makes uses of the Moore-Penrose pseudoinverse to reconstruct the set of coefficients from their compressed form according to

$$
\hat{c} = H^\dagger y.
$$

This method of reconstructing the signal is made up of two steps where the performance of each step is based on the size of the signal ($n$). The first step in the process is the calculation of the pseudoinverse of the compression matrix $H$. Many common software implementations of this calculation, for example, Matlab's `pinv()` function [26], makes use of the Singular Value Decomposition (SVD) of the matrix. The process of calculating the SVD of a matrix is worse from a performance perspective than that of matrix-matrix multiplication which is $O(mn^2)$ [27] for $m \times n$ matrix.

The second step of the algorithm is a matrix-vector multiplication operation which in a naive implementation has a performance of $O(n^2)$. We would like to improve the performance of the reconstruction algorithm to get rid of these computationally intensive steps. By making use of the compression matrix developed in the previous section that just selects $m$ coefficients with the largest magnitudes, the performance of the reconstruction process can be improved. Each of the rows in the compression matrix $H$ is orthogonal; that is, for two rows in the selection matrices $H(i,:)$ and $H(j,:)$ the discrete inner product is zero; $\langle H(i,:) \mid H(j,:) \rangle = 0$. A second property is that each row of $H$ is normalized; $\sum_{j=1}^{m} H(i,j) = 1$; thus $H$ is orthonormal. The pseudoinverse compression matrix that keeps $m$ most significant coefficients as opposed to the compression step using the normalized Hadamard matrix using the same amount of data.

Figure 9: An example occupancy grid for an unknown environment (a) and the set of coefficients used to represent the occupancy grid when represented using the Haar wavelet basis (b).

Figure 10: Performance results of each of the reconstruction methods using a normalized Hadamard matrix and a selection matrix to keep $m$ most significant coefficients.
of a orthonormal matrix can be easily computed without the computationally intensive SVD process according to

$$H^\dagger = H^\star,$$  \hspace{1cm} (68)

where $H^\star$ is the conjugate transpose of $H$ and is found by taking the transpose of $H$ followed by the complex conjugate of each element in $H$. Since the compression matrix is composed of elements containing just 0 or 1 the pseudoinverse is just the transpose $H^\dagger = H^\top$.

Finally, in (67) we can replace the pseudoinverse of $H$ with the transpose; thus the reconstruction process becomes

$$\hat{c} = H^\top y.$$  \hspace{1cm} (69)

We can improve the performance even more if we assume that the compression is performed as discussed above, that is, where $m$ coefficients with the largest magnitudes are selected. In that case then $H$ is full of zeros except for elements that correspond to the coefficients with largest magnitudes. Using this information we simplify $l_2$ reconstruction algorithm to just generate a vector of zeros and then copy the stored coefficients into their proper locations in the reconstructed vector. This replaces the matrix-vector multiplication with $m$ value copies. The updated form of $l_2$ reconstruction algorithm that does not require the potentially expensive calculation of the matrix pseudoinverse or matrix-vector multiplication is presented in Algorithm 8.

To examine the performance benefit of the new $l_2$ reconstruction algorithm a compressed form of the occupancy grid seen in Figure II is reconstructed using the original algorithm explained in Section 4.1 and the new algorithm described in Algorithm 8. The algorithm is run for $x$ varying compressed data sizes in [0%, 100%] and the timing results are seen in Figure II. As seen from the results the run time for the new $l_2$ reconstruction remains small as the size of the compressed signal grows while the original $l_2$ reconstruction approach slows down as the size of the compressed signal grows.

6. Experimental Results

We would now like to examine how using compressed occupancy grids affects the FastSLAM OG algorithm. Based on the timing results presented in Section 4 only two of the uncompression methods, $l_2$ (Section 4.1) and CS (Section 4.2), were selected to use in the FastSLAM COG algorithm. Both of the algorithms are compared using data captured by a small unmanned ground vehicle (UGV). The UGV is equipped with Mecanum wheels [28] which allow the vehicle to move in any direction without the constraints that are typical with many ground vehicles. The use of Mecanum wheels makes it difficult to use standard wheel encoders for localization purposes; to overcome this a downward facing camera is attached to the UGV which provides visual odometry data for localization using the approach described in [29]. Other sensors attached to the UGV include a digital compass to provide the global heading of the vehicle and a Hokuyo UTM-30LX LiDAR sensor, with reported accuracy of 0.1 to 10 m ± 30 mm and 10 to 30 m ± 50 mm, to provide range and bearing measurements to objects in the environment. In order to provide a “ground truth” trajectory for our test, a Hagisonic StarGazer indoor localization system was used. This sensor uses an upward facing camera and static landmarks placed on the ceiling to triangulate a robotic vehicle in an environment. This sensor has a precision between 1 mm and 10 mm based on how many landmarks the sensor can see at a given time. This accuracy was much higher than the visual odometry algorithm that provided the odometry input to the SLAM algorithm; thus this trajectory was used to compare the effect of using compressed occupancy grids in our FastSLAM algorithm as opposed to the full occupancy grid.

For comparison purposes, the standard FastSLAM OG algorithm described in Algorithm 1 was used to generate a baseline vehicle path and occupancy grid map. Once the baseline was generated the FastSLAM COG algorithm described in Algorithm 3 was run using the same data used by the standard approach. For each of the two reconstruction algorithms the compression method used is the optimized approach described in Section 5.1 and $l_2$ approach is implemented as described in Section 5.2. For the CS reconstruction algorithm the convex optimization problem is solved using the NESTA library.

The UGV was remotely driven around a small enclosed 10 ft × 12 ft environment while all of the sensor data was logged to memory on a small attached computer. The data was then postprocessed to produce a baseline vehicle path and an occupancy grid representing the testing environment. The baseline vehicle path, along with the true vehicle path, can be
Figure 12: The baseline path generated by the FastSLAM algorithm with occupancy grids algorithm along with the true path of the vehicle (a) and the occupancy grid generated by the algorithm that represents the environment in which the vehicle is operating (b).

Figure 13: The path generated by the FastSLAM COG algorithm along with $l_2$ reconstruction method using 40% of the original data (a) and the reconstructed occupancy grid generated by the algorithm (b).

seen in Figure 12(a) and the occupancy grid with a 0.1 m cell size is shown in Figure 12(b). The environment used to generate the baseline shown in Figures 12(a) and 12(b) is a simple rectangular indoor environment. Our choice of this simple environment, as opposed to a more complex environment, was made for two reasons. First, in order to properly compare how the use of compressed occupancy grids affected the localization and mapping an accurate baseline was required for comparison purposes. Based on the sensors that we had at our disposal, an indoor testing was more ideally suited for this task. Secondly, the goal of this research was not to examine how well an occupancy grid based SLAM algorithm
performs in complex environment that was beyond our scope and addressed by others; rather our goal was to examine how the performance of the approach was affected by compressing the occupancy grid used to represent the environment, thus testing in complex environment was not attempted for this initial research.

The vehicle path estimate generated by FastSLAM COG algorithm using $l_2$ reconstruction can be seen in Figure 13(a) and the reconstructed occupancy grid is shown in Figure 13(b). The generated path estimate using compressed sensing reconstruction approach is shown in Figure 14(a) and the reconstructed occupancy grid is shown in Figure 14(b). For each of the algorithms 40% of the amount of data used to represent the full occupancy grid is used. Figure 15 shows the mean square error (MSE) in the path estimate and reconstructed occupancy grid between the uncompressed form of the algorithm and the compressed form of the algorithm. As Figure 15 shows using more than 40% of...
the data does not decrease the MSE any more than the error that exists when using 40% of the data.

The plots provided in Figure 16 illustrate how a decrease in the amount of stored data decreases the quality of the reconstructed occupancy grid. This set of plots shows a series of occupancy grids that were generated as the final map of by the FastSLAM COG algorithm using $l_2$ reconstruction approach for decreasing amounts of data. As can be seen from the first several grids, up to 40% of the data show no decrease in quality; however as the amount of data used continues to decrease the amount of error in the final occupancy grid increases until the grid becomes unusable. A similar set of plots are provided in Figure 17 for the FastSLAM COG algorithm using the compressed sensing reconstruction approach and similar results can be seen.

By examining Figure 15 it can be seen that there is a constant error that appears in both the path estimate and the occupancy grid reconstruction from using the compressed sensing approach. This error comes from the parameters that are chosen for the NESTA library. Of most significance
is the value of $\varepsilon$ from (36) which is the amount of error that is allowed between the coefficients that represent the compressed form of the occupancy grid and the value of those coefficients in the reconstructed occupancy grid. For the presented results the value of $\varepsilon$ was chosen to be $10^{-10}$ which is quite small; however it does allow for small errors to occur in the reconstructed occupancy grid. These errors can be reduced or removed completely, from the estimate by choosing a smaller value of $\varepsilon$ and adjusting the parameters used by the NESTA library. However, as previously discussed, the accuracy of the grid reconstruction is not the only property of importance to us when deciding which approach should be used for storing and reconstructing the compressed occupancy grid. The other significant property of each algorithm that must be investigated is the time it takes to execute the algorithm. A comparison of the average time it takes to run a single iteration of the FastSLAM COG algorithm using $l_2$ approach and the compressed sensing approach for varying amounts of data is shown in Figure 18. As seen in these results the average run time of the FastSLAM
The average time to complete a single iteration of the FastSLAM COG algorithm using $l_2$ reconstruction approach and the compressed sensing reconstruction approach for varying amounts of data stored.

COG algorithm using the compressed sensing approach is significantly longer at each compression level than the algorithm using $l_2$ reconstruction approach.

6.1. How Much Compression Can Be Achieved?

There are two key components that affect how much an occupancy grid can be compressed; first is the complexity of the environment in which the UMV is operating. If the environment is simple, such as a rectangular room, then a relatively small amount of data is needed to represent it with respect to a more complex environment. The second component that affects how much an occupancy grid can be compressed is the cell size used. If a large cell size is selected then there are fewer cells to represent the environment compared to when a smaller cell size is selected. It follows that when a larger cell size is selected a larger percentage of the cells must be used to store the information about the environment, even for simple environments so the grid cannot be compressed as much. To illustrate these two components, three separate environments of increasing complexity where generated and a compressed occupancy grid was generated for each environment using four separate cell sizes (1.0 m, 0.5 m, 0.2 m, and 0.1 m) and varying amounts of compression. The first environment is a simple rectangular room and an occupancy grid for this environment generated using the full set of data with the smallest cell size is shown in Figure 19. The MSE as function of data stored for this environment is shown in Figure 20 for each of the four cell sizes. As expected the environment with the largest cell size, when fewer coefficients are needed to store all of the environment information, has the largest error when a small amount of data is used to represent the environment. It can also be seen that, for each of the four cell sizes, as more information is stored, the error in the reconstructed form of the occupancy grid decreases.

A second environment with more complexity is shown in Figure 21 which is a rectangular room with complex corners. The MSE between the true occupancy grid and the reconstructed grid as a function of the amount of data stored is shown in Figure 22 for each of the cell sizes. The overall behavior of the error is the same as in the simple case with the largest cell size having the largest error when small amounts of data are used. However, because of the added complexity in this environment, the errors at the lower data percentages are larger than the simple case which is what we expected because more information is required to store the more complex environment.

Finally, a complex environment of a rectangular room with complex corners along with rectangular objects located throughout the rectangular room is shown in Figure 23. As with the previous two environments the MSE between the true occupancy grid and the reconstructed occupancy as a function of data is shown in Figure 24 for each cell size. As expected the use of larger cell sizes causes more errors in the reconstructed form of the grid when small amounts of data are used. Also, the additional complexity causes the errors at lower data percent to be larger than in the more simple environments. It can be seen from the presented results that there appears to be a “magic” compression level of 40% at which, no matter the complexity, the compressed occupancy can be reconstructed with significant accuracy.
7. Conclusions

In this paper an approach for solving the SLAM problem was presented that makes use of compressed occupancy grids. The approach is based on an extension to the FastSLAM algorithm that represents an unknown environment by using occupancy grids. A modified form of the SLAM algorithm, FastSLAM COG, was presented in Section 3 which makes use of a generic compression and reconstruction method to solve the SLAM problem. Four specific reconstruction methods were examined in Section 4 in order to see how well each of the approaches could reconstruct a compressed signal along with an analysis of the time needed for each algorithm to complete. Based on the reconstruction time, two of the four algorithms were used to replace the generic reconstruction method in the FastSLAM COG algorithm and their performance was examined in Section 6 using experimental data. From the presented experimental results it was concluded that the FastSLAM COG algorithm using $l_2$ reconstruction approach could solve the SLAM problem with almost no errors compared to the FastSLAM OG algorithm while storing only 40% of the data required to store the complete occupancy grid. The experimental results presented in this paper were achieved by exploring how compressing the occupancy grid affects the performance of the algorithm in simple environments. In order to further examine our approach, a next step would be to examine how well this approach performs in more complex and unstructured environments. Also, further investigation into the effect of using more complex compression and reconstruction approaches as well as the possibility of dynamically resizing the cell size could lead to higher data compression rates without inducing errors into the SLAM solution.
Competing Interests
The authors declare that there are no competing interests regarding the publication of this paper.

Acknowledgments
This material is based upon work supported by (while serving at) the National Science Foundation of the United States.

References