Compressed Sensing Environmental Mapping by an Autonomous Robot

M. Horning*, M. Lin*, S. Srinivasan†, S. Zou‡, M. Haberland‡, K. Yin‡, and A. Bertozzi‡
*Department of Mathematics, Harvey Mudd College, Claremont, CA 91711
†Department of Physics, Harvey Mudd College, Claremont, CA 91711
‡Department of Mathematics, University of California, Los Angeles, CA 90095

***Abstract*—This paper introduces the use of compressed sensing for autonomous robots performing environmental mapping in order to reduce data collection, storage, and transmission requirements. A prototype robot sends data collected over adaptively updated straight-line paths to a server, which reconstructs an image of the environment variable using Split-Bregman iteration. The amount of data collected is only 10% of the amount of data in the final map, yet the relative error is only 20%.**

I. Introduction

Environmental mapping is essential to making truly autonomous robots and is critical for robotic exploration and discovery underwater, outdoors, within buildings, and in space [1]. However, the necessarily limited capabilities of mobile robots and inherent communication challenges of certain environments conspire to complicate the problem of mapping, even when the location of the robot(s) is known. For instance, autonomous underwater vehicles would be immensely useful for mapping natural resources and scientific data collection in the ocean [2], but returning the information to a land-based server is difficult because radio communication is impractical underwater [3]. Acoustic communication is possible, but has greater bandwidth constraints and power requirements [4], hence there is a need to minimize data transmission. One approach would be to collect as much data as necessary and compress it prior to transmission, but this has significant drawbacks, as cost increases with the number of sensor nodes in a network and mobile robots are limited in their physical speed and data storage capability. A more promising solution is compressed sensing, a technique in which a relatively small amount of data is collected in order to reconstruct a higher-resolution signal or map. [5] considers the use of Random Access Compressed Sensing to create an energy-efficient network of static sensor nodes for oceanographic data collection, but in certain scenarios, such as exploration of unchartered territory, autonomous robots are clearly preferably to a network of fixed sensors. In this paper, we pioneer the use of mobile robots for compressed sensing environmental mapping. Section II describes the testbed hardware: a small wheeled robot tracked by an overhead camera measures the reflectivity of the testbed surface and sends the data to a central server. Section III documents the algorithm used by the server to reconstruct a complete map of the surface from the incomplete data collected by the robot. In Section IV we describe both simulated and physical simulation results, and we conclude in Section V.

II. Testbed Configurations and Communications

The experimental setup consists of four main components: a testbed surface, a mobile robot, two Overhead Imaging Source DMK 21F04 1/4 Monochrome CCD cameras, and a server hosted on a windows computer.

The testbed surface is a 1.5m x 2.0m sheet of black asphalt felt paper with white paper patterns to be mapped. The robot itself consists of 5 main components:

1) **Arduino Uno:** The Arduino Uno, powered by a 9V battery, has very limited processing power (16 MHz) and memory (32 KB Flash and 2 KB SRAM). An Adafruit WiFi Shield allows the Arduino to connect to a wireless network and thereby communicate with the server.
2) **Reflectance Sensor:** The reflectance sensor yields an analog voltage corresponding with the reflectance of the surface, which is then binarized: sensor readings above an experimentally determined threshold are read as white and those below are read as black.
3) **Chassis:** The chassis has four wheels, each directly connected to a gearmotor powered by five onboard AA rechargeable batteries.
4) **Motor controller:** The motor controller amplifies the 5V, logic-level output of the Arduino to power the motors with the AA batteries. The two motors on the left and the two motors on the right are connected in parallel to each other, enabling simple tracked-vehicle (tank) steering.
5) **Tag (Visual ID):** A unique white rectangular tag with a black-tape border and a black header strip allows the overhead cameras to visually identify the robot and determine the position of the reflectance sensor.

The two 640x480 pixel cameras send 30 frames per second to a processing computer via firewire. A python script uses OpenCV to recognize the robot identification tag and return the robot's position and orientation to the server over the serial port. Finally, the server reads the robot's location and orientation provided by the cameras, records data collected by the robot, and sends instructions to the robot. The logic and flow of data is illustrated in Figure 1.

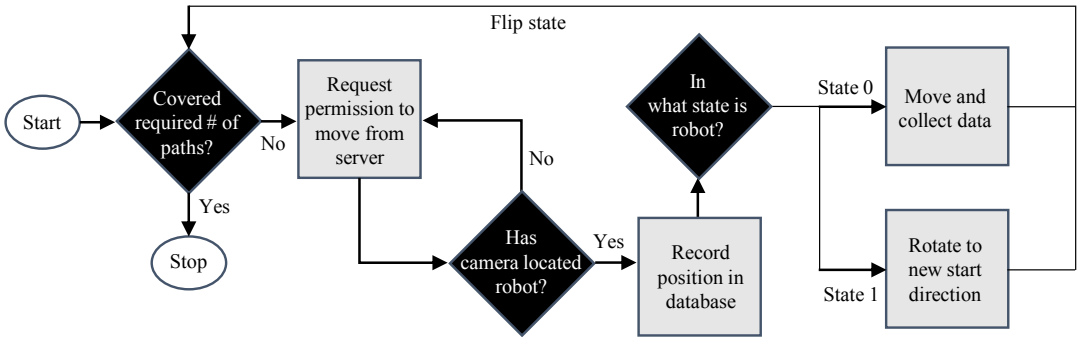


Fig. 1: The robot operates as a finite state machine. During state 1, the robot attempts to rotate to a start orientation previously commanded by the server, then requests for the server to record its actual position from the video cameras. The robot switches to state 0 upon confirmation, integrates sensor data along a straight path for a duration commanded by the server, and requests for the server to record the actual final position.

III. Algorithm

A. Models and Assumptions

Without loss of generality, we assume the area to be explored is a rectangle denoted by Ω, and the interested environment variable $u(x, y)$ is a piece-wise constant function defined on Ω. See Figure 2 for an illustration. Assume the robot travels through n different paths, which are denoted by $C_1, \ldots, C_n$. Along each path, the robot sensor reads the value of u at each point, and an integration of u is performed on-board by adding up the readings. Therefore, the robot collects the integral of u along C_k, which is denoted by b_k. The robot sends b_k to the server once it completes C_k. Formally, b_k is expressed as

$$b_k = \int_{C_k} u(x, y)\, \mathrm{d}\Gamma,\ k = 1, \ldots, n. \tag{1}$$

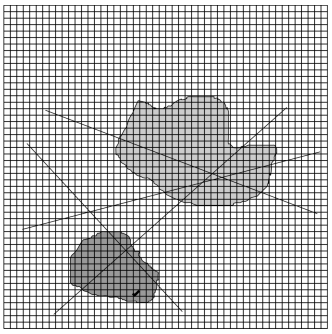


Fig. 2: An illustration of the paths of robots. The shaded regions denote the area of interest (unknown), where the values of environment variable u is significantly different from the surroundings. The size of the unit pixel (each small rectangle) is determined by the accuracy of the positioning system.

For computational purpose, the whole domain is discretized into rectangular pixels (See Figure 2). Each path C_k defines a weight $a_{k,ij}$ for each pixel (i, j). If C_k does not intersect with pixel (i, j), then $a_{k,ij} = 0$, otherwise $a_{k,ij}$ is defined to be the length of the part of C_k that falls within pixel (i, j). It is also assumed that the value of u is a constant within each pixel (i, j), which is denoted by u_{ij}. After discretization, Eq (1) becomes

$$b_k = \sum_i \sum_j a_{k,ij} u_{ij}. \tag{2}$$

Let $b = (b_1, \ldots, b_n)^t$, $u = (u_{ij})$, then b and u have a linear relation, which is written as

$$Au = b, \tag{3}$$

where the linear operator A is specified in Eq (2). Eq (3) is the model equation, which poses an inverse problem. In this equation, A is determined from the paths that the robot travelled in - it travels in straight line paths, hence we only need the start and end locations to determine its trajectory. Each path contributes one row to A. b is obtained from the experiment, and is the vector of the integral of sensor values as the robot travelled along each of its paths. u is the variable that we want to solve. Of course, if the paths form a complete raster scan over the whole domain, then theoretically Eq (3) has a unique solution. For economical reasons, it is desirable to use much fewer paths and still being able to reconstruct a solution for u. The problem is well suited for compressed sensing based image reconstruction techniques, which provide a way to solve these underdetermined systems.

B. Solving the Inverse Problem

Based on the observation that the image u to be reconstructed is piecewise constant, its directional gradients, $\nabla_x u$ and $\nabla_y u$, are sparse over the domain. Also we assume that the interesting feature of the image is relatively small compared to the whole domain to be explored. Therefore, we postulate that the solution u minimize the energy functional

$$E_0(u) = \alpha |u|_0 + \beta \left(|\nabla_x u|_0 + |\nabla_y u|_0 \right), \tag{4}$$

subject to the data fitting constraint (3). Here $|\cdot|_0$ denotes the ℓ^0 norm. α and β are parameters balancing the weights of the sparsity of the gradients and the feature. We note that

minimizing ℓ^0 norm is generally considered intractible, we replace it with ℓ^p $(p > 0)$ norm, which still enforces sparsity and much easier to solve:

$$E_p(u) = \alpha|u|_p + \beta\left(|\nabla_x u|_p + |\nabla_y u|_p\right). \tag{5}$$

Prior work in the field of compressed sensing and sparse image reconstruction normally chooses $p = 1$ [6], and some theoretical work has demonstrated its efficiency [7]. One popular algorithm for solving ℓ^1 minimization problems is based on Bregman iteration, which is carefully described in [8]. And we follow this method to solve our problem.

Although it is a nonconvex optimization problem in the case $0 < p < 1$, some work such as [9], [10] has shown that it can also be solved using the Bregman-type method, and that the solution converges to the global minimum of the associated ℓ^0 formulation.

C. Path Planning Algorithm

The server program generates the next path for the robot every time the robot is in State 1 (see Figure 1). Without any prior information about the distribution of features in the domain to be explored, it is natural to choose random paths that are uniformly distributed over the whole domain. Although we would demonstrate later that it should work, it is desirable to adopt an adaptive path planning strategy based on the data already collected by the robot as it moves around.

Initially, the robot collects data along some constant number of uniformly distributed random paths. Then all the data that have already been collected are processed off-board on the server, and a coarse resolution image is reconstructed. Based on this image, the server program would segment interesting areas that contain more features than elsewhere. And it generate a new set of paths, still random but with higher probability passing through those interesting areas. As the robot travels more paths and collects more data, the resolution of the reconstructed image would improve, and the new paths based on that would be more likely directed into areas rich in features.

The segmentation of interesting areas consists of two steps. First a simple thresholding is applied to zero out small pixel values. Then it finds connected components containing more than n pixels and put them into the set of interesting areas. Every new path passes through the current location and the next location. The next location is chosen randomly from all viable pixels, with probability p_1 of being one inside interesting areas, and p_0 for non-interesting areas. It is required that $p_0 < p_1$ and the probabilities add up to 1. The pseudocode for adaptive path planning is in Algorithm 1.

With the adaptive path planning, the workflow of our algorithm is in Algorithm 2.

Algorithm 1 Adaptive path selection

function ADAPTIVEPATHS(u, Ω, p_0, p_1, max_num, n)
  for $i = 1$ to max_num **do**
    $u_1 \leftarrow$ THRESHOLDING(u, threshold)
    $\Omega_1 \leftarrow$ CONNECTEDCOMPONENTS(Ω, u_1, n)
    $P_i \leftarrow$ NEWPOINT($\Omega, \Omega_1, p_0, p_1$)
    $C_i \leftarrow$ NEWPATH(Ω, P_{i-1}, P_i)
    $C_{\text{new}} \leftarrow C_{\text{new}} \cup \{C_i\}$
  end for
  return new paths C_{new}
end function

Algorithm 2 Envirionment Mapping Algorithm

procedure ENVIRIONMENTMAPPING
  initialize u, A, b with zeros
  while $\|u_{\text{old}} - u_{\text{new}}\| <$ tol **do**
    $C_{\text{new}} \leftarrow$ ADAPTIVEPATHS($u_{\text{new}}, \Omega, p_0, p_1$, num, n)
    $b_{\text{new}} \leftarrow$ COLLECTNEWDATA(C_{new}) ▷ on-board
    $A, b \leftarrow$ UPDATESYSTEM($\Omega, A, C_{\text{new}}, b, b_{\text{new}}$)
    $u_{\text{old}} \leftarrow u_{\text{new}}$
    $u_{\text{new}} \leftarrow$ SPLITBREGMAN(A, b, α, β)
  end while
  $u \leftarrow u_{\text{new}}$
  return u
end procedure

IV. EXPERIMENT

A. Simulated Results

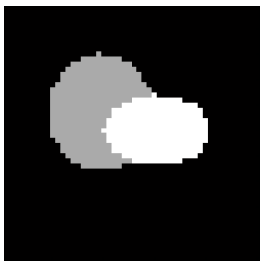
Before actually using the robot to collect data, we run simulated reconstructions. Figure 3a is a 50x50 test image for simulations. We note that he test image has multiple gray scales. In our experiment the robot can only distinguish black and white colors since it is equipped with a binary sensor. With more advanced sensor our method should work as well.

Figure 3b is a reconstruction of a 50x50 image from 100 random paths, with data to unknowns ratio of 1:25. The shape and the intensity of the region of interest are successfully identified. Figure 3c illustrates the paths for the robot, with each one generated by connecting current position and randomly selected target position while ensuring the robot stays within the testbed. In the figure, brighter pixels represent points that are visited more times. Figure 3d shows the difference between the original image and the reconstructed one. It shows that most of the error in the reconstruction is around the edges of the region of interest. The relative error of the reconstruction is 0.19715.

B. Comparison with Other Approaches

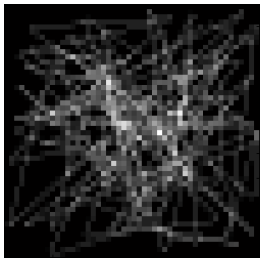
In order to judge the effectiveness of our approach, we examine other methods of collecting data and reconstructing an image. An intuitive one is to sample individual pixels and try to reconstruct the image from these samples. One reconstruction approach is to use the same Split Bregman iteration that we are using for reconstruction from path integrals by assuming that each path is degenerated to a sample point. Alternatively, we can take the value found at each sampled pixel and assign the same value to nearby pixels, a method that we refer to as pixel expansion.

Figures 4 and 5 present reconstructions of the previously shown 50x50 test image, Figure 3a, using these approaches,



(a) 50x50 test image.

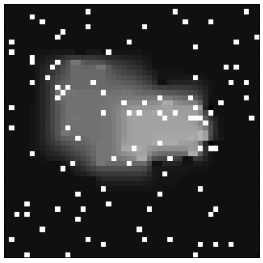
(b) Simulated reconstruction.



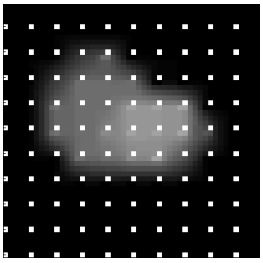
(c) 100 random paths used for simulated reconstruction.

(d) Difference between the original image and the reconstruction.

Fig. 3: Simulated reconstruction using random paths.



(a) Reconstruction using points sampled randomly (Random Point Reconstruction).



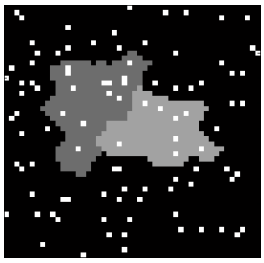
(b) Reconstruction using points sampled on a grid (Grid Point Reconstruction).

Fig. 4: Values are found at sampled points, and Split Bregman iteration is used to reconstruct the image.

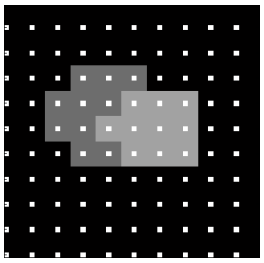
both with randomly sampled points and points sampled on a grid. In all cases, 100 pixels are sampled, and the sampled points are colored white.

In Figure 4, Split Bregman iteration was used with the collected data for reconstruction. By comparing Figures 4a and 4b, it seems that the reconstruction is more accurate when the points are sampled on a grid. This is reasonable because by sampling on a grid, the spread of sampled points is very even and well distributed. When the points are sampled randomly, certain areas will be less thoroughly investigated than others, hindering accurate reconstruction.

Figure 5 illustrates pixel expansion. This process creates



(a) Expansion of points sampled randomly (Random Point Expansion).



(b) Expansion of points sampled on a grid (Grid Point Expansion).

Fig. 5: Values are found at sampled points, and those values fill the area surrounding the sampled points.

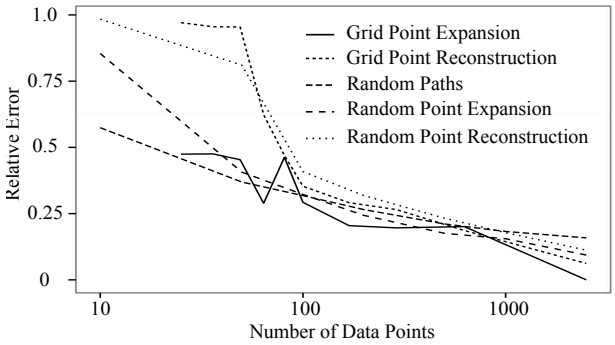
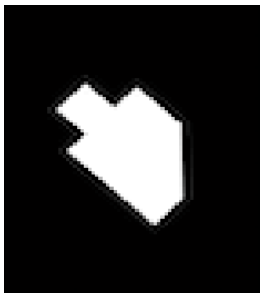


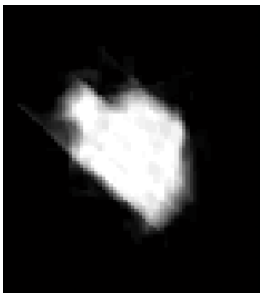
Fig. 6: Plot of error for the various methods of reconstruction with respect to the number of data samples used.

expanded pixels of a single color around each sampled pixel. When the pixels are sampled on a grid, the even distribution causes the expanded pixels to simply be squares, essentially creating a low resolution version of the image, as in Figure 5b. Figure 5a is a result of pixel expansion with randomly sampled pixels. The unpredictable spread of sampled pixels causes the expanded pixels to vary widely in shape and size, and the reconstruction does a poor job of determining the shape of the region of interest.

To quantitatively compare these methods, Figure 6 plots the errors of reconstruction, with respect to the number of data points collected. The reconstruction from randomly generated path integrals achieve lower error than the other methods for a small number of data points. As the number of data points grows, all methods converge to similar error; this is expected because enough data would turn the problem from under-determined to fully-determined. Worth noting is that while pixel expansion from samples on a grid appears to do better than random path integrals for some of the low data situations, the effectiveness of grid point expansion is highly variable, and as such, its overall usefulness is limited. The accuracy of reconstruction is very dependent on how well the grid aligns with the given image, and choosing the best grid size for a given environment would require prior knowledge of the environment. Another flaw with pixel expansion is the dependence on accurate sampling; pixel expansion

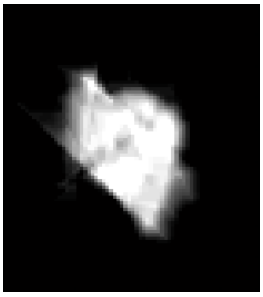


(a) Image of the testbed, sized at 70x90.



(b) Simulated reconstruction from the 178 paths that the robot travelled.

(c) Reconstruction from the data collected by the robot.



(d) Reconstruction from the data collected by the robot using tuned parameters.

Fig. 7: Reconstruction using the data collected by the robot.

will be very heavily affected by noise. If a pixel value is read inaccurately, the entire expanded pixel associated with that sample will be inaccurately reconstructed. Using path integrals mitigates the impact of inaccurate reading of individual pixels.

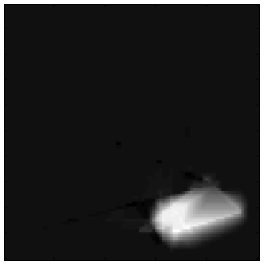
C. Experimental Results

We proceeded to carry out experimental testing with the actual robot. Figure 7a shows a testbed that we used. The testbed is modeled as a 70x90 image.

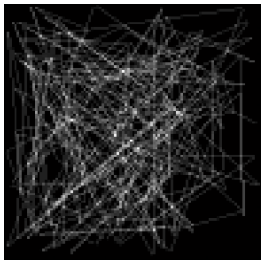
As a means of comparison, we first carried out the reconstruction as a simulation, shown in Figure 7b. This simulated reconstruction uses the same set of 178 paths that the robot travelled along. Thus, this reconstruction is an ideal reconstruction.

The actual reconstruction from the robot collected data is shown in Figure 7c. Clearly, this reconstruction is more noisy, giving black spots in the middle of the region and creating white spots where it should be black. This is due to the fact that the data collection by the robot is not as perfect as the data in simulation. To make the model more robust against the noise, we retry the reconstruction with a lowered data fidelity parameter. This yields Figure 7d. It does a reasonably good job of showing the region of interest.

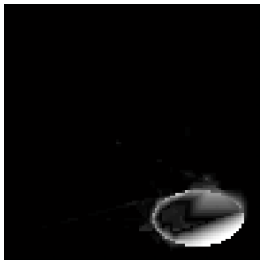
(a) 100 x 100 pixel test image with features in bottom right.



(b) Simulated reconstruction from the 200 paths

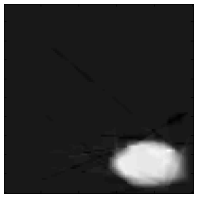


(c) The 200 randomly generated paths.

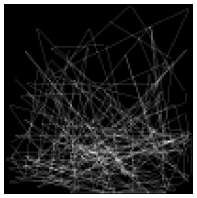


(d) Difference between the original image and the reconstruction

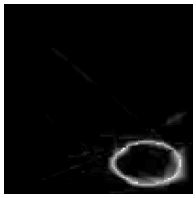
Fig. 8: Simulated reconstruction using uniformly distributed random paths.



(a) Simulated reconstruction from 200 adaptive paths.



(b) The 200 adaptive paths.



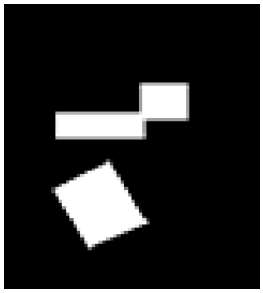
(c) The difference between the reconstruction and original image.

Fig. 9: Reconstruction using the adaptive paths.

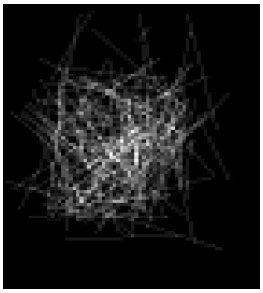
D. Adaptive Pathing

We first run simulations on a test image as in Figure 8a. As a comparison, we run the reconstruction using uniformly chosen random paths as well as adaptive paths as described in Sec III-C. Each method uses the same Split-Bregman algorithm with the same number of paths.

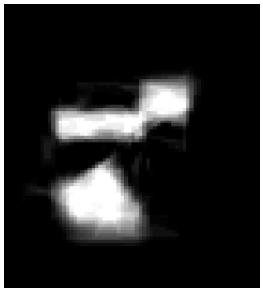
Figure 8c shows the 200 uniformly random paths for the simulation. The reconstruction obtained with these paths is shown in Figure 9a. This reconstruction is able to recover successfully generally the region of interest and the size of the area, but is not able to ascertain some of the more detailed information about the shape and boundary of the area accurately. Figure 8d shows the difference between the



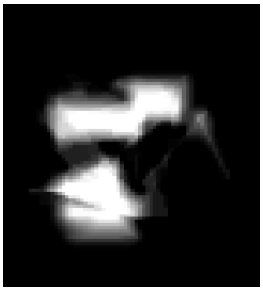
(a) Image of the testbed, sized at 70x90



(b) The 224 adaptive paths traveled by the robot



(c) Simulated reconstruction for the adaptive paths traveled by the robot



(d) Image reconstruction using robot collected data with tuned parameters

Fig. 10: Experimental results of adaptive pathing using robot collected data

the reconstructed image and the ground truth. The error is primarly along the edges but more prominent along the bottom edge. The relative error of the reconstruction is 0.5023.

We compare these results against those displayed in Figure 9, which are generated by the adaptive pathing algorithm on the same test image in 8a. Figure 9b shows the 200 adaptive paths taken by the simulation. There are visibly clustered near the bottom of the image, where it identified early on had the area of interest. Figure 9a then shows the results of the Split Bregman reconstruction with the data collected from the adaptive paths. Because the paths tend to cluster in the region of interest, the reconstruction is able to more accurately capture the curve along the borders and thus the shape of the area itself. This is illustrated in Figure 9c, which shows the difference between this reconstruction and the original test image. The relative error of the reconstruction is .19614, which is nearly half as that for uniformly random paths.

We implement adaptive path planning for the robot and the results are shown in Figure 10. Figure 10a is an image of the original testbed, sized to 70x90, which contains multiple areas of interest that are away from the center of the test bed and have sharply defined edges. Figure 9b shows the 224 straight line paths the robot vehicle traveled on the testbed and it can be observed that it did not travel often to black areas or corners where there were no areas of interest. We then generate Figure 10c, which is an ideal reconstruction assuming no error in data collection. If we run the same reconstruction with robot data, the resulting reconstruction is significantly affected by the noise, as shown in Figure 10d. It is close to the reconstruction with ideal data.

V. Conclusion

We have demonstrated a simple yet useful prototype autonomous robot for mapping environment variables. It can be applied to existing robot platforms that are equipped with wireless communication and tracking systems. The required computing power of the robot is only onboard addition and a storage big enough to hold one data point. The data collection strategy based on compressed sensing enables minimal data communication. It is further improved by adopting adaptive path planning using already collected data. The image reconstruction algorithm minimizes the data fitting error plus the ℓ^1 norm of the gradient of the image, which captures piecewise constant features. Split-Bregman algorithm is efficient for solving the minimization problem. Future work includes improving data processing methods that incorporates statistical methods such as cross validation and outlier pursuit, so that the reconstruction is more robust against the noise. Another work that can be done is to use sensors that can distinguish multiple levels of intensities, or even trying to use multiple sensors on the same robot. We also note that our method is scalable to using multiple robots, as long as the path planning can make robots avoid collisions.

Acknowledgment

This work was supported by NSF grants CMMI-1435709, DMS-1045536, and DMS-1118971 and UC Lab Fees Research grant 12-LR-236660. The authors would also like to thank Rick Chartrand and collaborators at Los Alamos National Laboratory for their technical recommendations.

References

[1] S. Thrun *et al.*, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, pp. 1–35, 2002.

[2] I. F. Akyildiz, D. Pompili, and T. Melodia, "Underwater acoustic sensor networks: research challenges," *Ad hoc networks*, vol. 3, no. 3, pp. 257–279, 2005.

[3] ——, "Challenges for efficient communication in underwater acoustic sensor networks," *ACM Sigbed Review*, vol. 1, no. 2, pp. 3–8, 2004.

[4] J. Heidemann, W. Ye, J. Wills, A. Syed, and Y. Li, "Research challenges and applications for underwater sensor networking," in *Wireless Communications and Networking Conference, 2006. WCNC 2006. IEEE*, vol. 1. IEEE, 2006, pp. 228–235.

[5] F. Fazel, M. Fazel, and M. Stojanovic, "Random access compressed sensing for energy-efficient underwater sensor networks," *Selected Areas in Communications, IEEE Journal on*, vol. 29, no. 8, pp. 1660–1670, 2011.

[6] E. J. Candès and M. B. Wakin, "An introduction to compressive sampling," *Signal Processing Magazine, IEEE*, vol. 25, no. 2, pp. 21–30, 2008.

[7] E. J. Candès, J. Romberg, and T. Tao, "Robust uncertainty principles: Exact signal reconstruction from highly incomplete frequency information," *Information Theory, IEEE Transactions on*, vol. 52, no. 2, pp. 489–509, 2006.

[8] T. Goldstein and S. Osher, "The split bregman method for l1-regularized problems," *SIAM Journal on Imaging Sciences*, vol. 2, no. 2, pp. 323–343, 2009.

[9] R. Chartrand, "Nonconvex compressive sensing and reconstruction of gradient-sparse images: random vs. tomographic fourier sampling," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*. IEEE, 2008, pp. 2624–2627.

[10] ——, "Fast algorithms for nonconvex compressive sensing: Mri reconstruction from very few data," in *Biomedical Imaging: From Nano to Macro, 2009. ISBI'09. IEEE International Symposium on*. IEEE, 2009, pp. 262–265.

[11] M. Gonzalez, X. Huang, D. S. H. Martinez, C. H. Hsieh, Y. R. Huang, B. Irvine, M. B. Short, and A. L. Bertozzi, "A third generation micro-vehicle testbed for cooperative control and sensing strategies." in *ICINCO (2)*, 2011, pp. 14–20.