Mesh Addition Based on the Depth Image (MABDI)

Lucas Chavez[1] and Ron Lumia[2], *Fellow, IEEE*

***Abstract*— Many robotic applications, especially those whose goal is to aid or assist through human robot-interaction, utilize a rich map of the world for reasoning tasks such as collision detection, path planning, or object recognition. Such map, and the method used to produce it, must take into consideration real-world constraints. Most mesh-based mapping algorithms resample a "black box" and do no provide a mechanism to close the loop and make decisions about the incoming information. MABDI leverages the global mesh by finding the difference between what we expect to see and what we are actually seeing, and using this to classify the incoming measurements as novel or not. This allows the surface reconstruction method to be run only on data that has not yet been represented in the global mesh. The result is an algorithm that becomes computationally inexpensive once the environment is known, but can also react to new objects.**

I. Introduction

Many robotic applications, especially those that involve human-robot interaction, often require a rich representation of the environment in order to perform such behavior as path planning and obstacle avoidance. In general, a rich representation, or map, is useful for providing situational awareness to an autonomous agent. A map is also important for applications such as teleoperation [1].

The methodology to build this representation is a continuously evolving subject in the field of robotics. The origins of the research into this problem date back roughly 25 years [2]. Since then the methods and the representations themselves have continued to evolve at an impressive rate. The main catalyst behind this growth is the advancement of sensing technologies over the same time period. In general, sensors have continued to generate measurements at higher rates, higher resolution, and lower cost over the years. This has provided an amazing opportunity to build richer and more useful representations of the environment.

In robotics, map building in an unknown environment is referred to as the Simultaneous Localization and Mapping (SLAM) problem [3]. This label describes the fact that a methodology which solves the SLAM problem must simultaneously locate the robot in the environment as well as map the environment. The focus of this work is the mapping aspect of the SLAM problem. Fig. 1 gives a visualization of the goal.

There are different types of data structures that can define a map. All types have both intrinsic characteristics that impact the algorithms that generate them and constraints that must be considered for real-world applications. In addition, we are concerned with rich representation types, in contrast to sparse representation types [4], because rich types have the most use in applications such as human robot interaction.

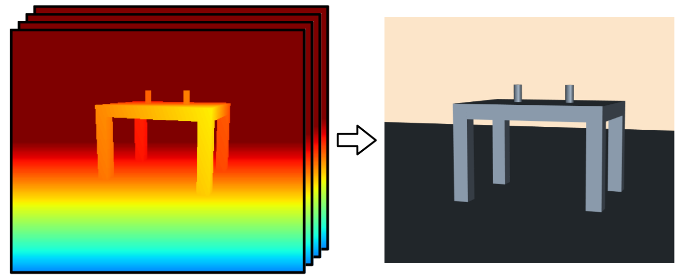

Fig. 1. Goal is to create a map from depth images

	Supported	Computationally Inexpensive	Low Memory Requirement
Point Clouds	x	x	-
Surfels	-	x	x
Implicit Functions	x	-	-
Mesh	x	x	x

TABLE I

Comparison of constraints for different map types

When considering what type of map is best for real-world applications, we must consider the constraints imposed by each type:

- Supported - Is there software, tools, research, algorithms, etc., for this type of map?
- Computationally Inexpensive - Can the algorithms run quickly on low cost computers (rather than specialized hardware)?
- Low Memory Requirement - Can the algorithms run on hardware with a standard amount of RAM?

Table I compares the constraints of common map types. We can see, in general a mesh type map satisfies real-world constraints. It has been used extensively by the gaming and graphics communities, and so benefits from an incredible amount of continued research and advances in hardware such as Graphics Processing Units (GPUs).

Currently, one of the issues with mesh mapping techniques is they are generally "black box" methods. Meaning the data comes in from the sensor, those measurements are turned into a mesh, and then that mesh is appended to a global mesh. Fig. 2 visualizes this common pipeline in black. The goal of this work is to design an algorithm to close the loop

*This work was not supported by any organization

[1]Lucas is with Mechanical Engineering Department and Electrical & Computer Engineering Department, University of New Mexico, Albuquerque, NM, USA {lucasc,lumia}@unm.edu

(as visualized in red) and allow the system to make decisions about the incoming data based on what it already knows.

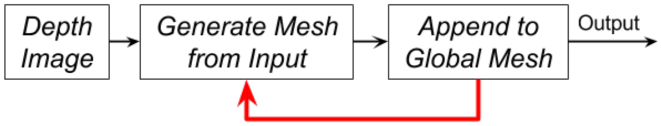

Fig. 2. Common “black box” pipeline in black. The contribution of MABDI in red.

II. Related Works

Works related to MABDI are generally based on RGB-D sensors. This type of sensor has become very popular since the release of the Kinect from Microsoft, which was the first mass produced RGB-D sensor of its kind. RGB-D sensors are inexpensive and produce noisy 640x480 depth images at 30fps. The RGB-D sensor has excited the robotics community because this has been the first time that depth data has been so readily accessible from such an inexpensive sensor. Therefore, methodologies that use RGB-D data must be able to quickly deal with very high rates of information.

One very impressive work came from Henry et al. in 2012 [5]. In this work they designed a system which used a RGB-D sensor to build a map made of surfels (Surfels are circular disks which have a particular position and orientation and also a radial size based on confidence.). In order to generate and maintain the surfel map they used the work of Weise et al. [6]. The map consists of a large number of surfels. The surfel map can be updated given new registered depth images from the sensor. Decisions are made how to handle each measurement in the depth image based on the difference between an expectation generated using the current map and the actual readings from the sensor. Rendering a surfel map requires special methods [7] and is difficult to use in applications such as obstacle avoidance.

One of the next major advances was published by Whelan et al. in 2012 [8] and more recently in 2013 [9]. The system they developed was named Kintinuous and was able to produce a high quality mesh representation of the environment. Their hybrid system utilized the KinectFusion method [10] of Newcombe et al. to create a volumetric representation of the portion of the environment in front of the sensor. As the sensor moves, portions of the environment that leave the volume in front of the sensor are ray cast and turned into a mesh. They obtain very impressive results but also mention a limitation of their system for future work. The limitation is that the mesh can not be updated once created, which is an issue when revisiting parts of the environment. One of the most impressive current works which has an adaptable mesh came from Cashier et al. in 2012 [11]. In this work, they were able to generate and update a mesh with new measurements from a ToF sensor. They used the difference between the existing model and the actual measurements to decide whether to adapt the mesh or add new elements. The mesh topology was not adaptive to the environment and their experiments only showed results of mapping a single flat wall with no robot movement. The system needs to be tested for object addition and removal.

Research and development of new mapping algorithms trend towards leveraging the information in the global map to make decisions about the incoming data. One can see parallels with how we as humans see the world. MABDI proposes do this in a computationally feasible way by simply using differencing and thresholding imaging methods.

III. Approach

The algorithmic structure of MABDI can be seen in Fig. 3. The diagram is very similar to Fig. 2 with the exception of the Classification component, shown in blue. This Classification component is MABDI’s contribution to the state-of-art in mesh based mapping algorithms, and is what gives MABDI the ability to make decisions about the incoming data.

The Classification component consists of two elements:

1) *Generate Expected Depth Image E* - Here we take the global mesh M, render it using computer graphics, and use the depth buffer of the render window to create a depth image E of what we expect to see from our sensor. This method requires the current pose P of the actual sensor (simulated for our experiments).
2) *Classify Depth Image D* - Here we classify the actual depth image D (simulated for our experiments) by first taking the absolute difference between E and D and thresholding. If the differences are small, those points are thrown away and if the differences are large, those points are kept as D_n. The idea behind this is, if the difference is large, the measurements are coming from a part of the environment that has not been seen before i.e., novel. The implication of this assumption is that this version of MABDI can not handle object removal. It is worth noting that MABDI can be extended to handle object removal by using the sign of the difference between E and D instead of the absolute value.

From a software perspective, the major difficulty of implementing the MABDI algorithm was found to be creating both the simulated depth image D and the expected depth image E. In addition, managing the complexity of the data pipeline needed to run the algorithm and the simulation of the sensor proved to be quite overwhelming. Thankfully, Kitware, who is a leading edge developer of open-source software, created the Visualization Toolkit (VTK) [12], [13]. At the time of this writing the VTK Github repository has over 60,000 commits and is contributed to by supporters such as Sandia National Labs [14].

MABDI is implemented with Python and uses VTK. The code is freely available on my Github account [15]. At the time of this writing, it consists of over 1,400 lines. The code that implements the MABDI algorithm itself is around 750 lines.

VTK is aptly designed for the implementation of MABDI for many reasons. Perhaps the most important is the concept

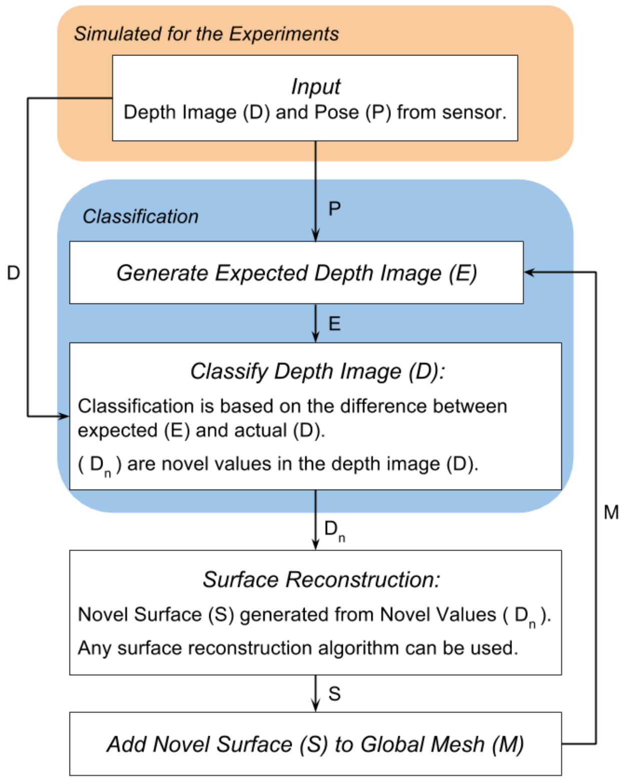

Fig. 3. MABDI system diagram

of a vtkAlgorithm (often called a Filter). This allows a programmer to create a custom and modular processing pipeline by defining classes that inherit vtkAlgorithm and then defining the connections between these classes. For example, you could have a pipeline that reads an image from a source (component 1), performs edge detection (component 2), and then renders the image (component 3). Using this concept, the individual elements of MABDI can be succinctly defined in individual classes. With that in mind, we can see in Fig. 4 the layout used in my implementation of MABDI:

- *Source* - Classes with the prefix Source define the environment that is used for the simulation and provide a mesh in the form of a vtkPolyData.
- *FilterDepthImage* - Render the incoming vtkPolyData in a window and output the depth buffer from the window as a vtkImageData. The output additionally has pose information of the sensor.
- *FilterClassifier* - Implements the true innovation of MABDI, takes the difference between the two incoming depth images (vtkImageData) and outputs a new depth image where the data that is not novel is marked to be thrown away.
- *FilterDepthImageToSurface* - Performs surface reconstruction on the novel points. In this simple implementation the topology of the mesh is defined in the image coordinates and can be thought of as a checkerboard pattern with two triangles in every square. The data is then projected to real-world coordinates. The topology and the real-world coordinates are combined to define a surface and output as a vtkPolyData.
- *FilterWorldMesh* - Here we simply append the incoming novel surface to a growing global mesh that is also output as a vtkPolyData.

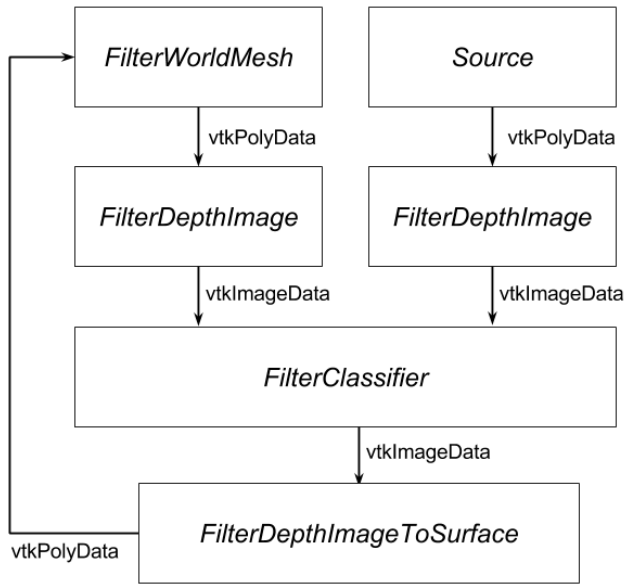

Fig. 4. MABDI software diagram

IV. Experimental Setup

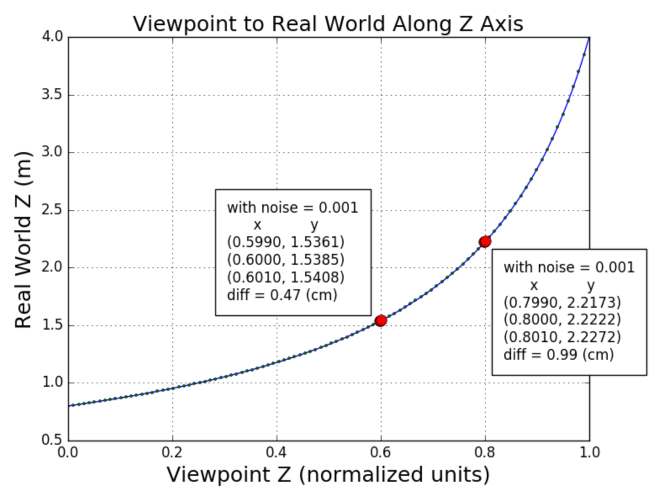

Fig. 5. Viewpoint coordinates to real world coordinates analysis. Viewpoint coordinates are obtained when a mesh is rendered into a render window, and can be transformed to real-world coordinates using the transformation matrix of the camera. Noise is added in simulation to the viewpoint coordinates. This graph shows the effect of that noise in real-world coordinates.

References

[1] M. W. Kadous, R. K.-M. Sheh, and C. Sammut, “Effective user interface design for rescue robotics,” in *Proceeding of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction - HRI ’06*. New York, New York, USA: ACM Press, mar 2006, p. 250.

[2] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3D surface construction algorithm," *Computer*, vol. 21, no. 4, pp. 163–169, 1987.
[3] S. Thrun, "Robotic mapping: A survey," *Exploring artificial intelligence in the new millennium*, no. February, 2002.
[4] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 3, pp. 229–241, jun 2001.
[5] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments," *The International Journal of Robotics Research*, vol. 31, no. 5, pp. 647–663, feb 2012.
[6] T. Weise, T. Wismer, B. Leibe, and L. Van Gool, "In-hand scanning with online loop closure," *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pp. 1630–1637, sep 2009.
[7] H. Pfister, M. Zwicker, J. van Baar, and M. Gross, "Surfels : Surface Elements as Rendering Primitives," in *SIGGRAPH 2000*. New York, New York, USA: ACM Press, jul 2000, pp. 335–342.
[8] T. Whelan, M. Kaess, and M. Fallon, "Kintinuous: Spatially Extended KinectFusion," 2012.
[9] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. B. McDonald, "Robust Tracking for Real-Time Dense RGB-D Mapping with Kintinuous," in *ICRA 2013*, no. MIT-CSAIL-TR-2012-031. Computer Science and Artificial Intelligence Laboratory, MIT, sep 2012.
[10] R. A. Newcombe, A. J. Davison, S. Izadi, P. Kohli, O. Hilliges, J. Shotton, D. Molyneaux, S. Hodges, D. Kim, and A. Fitzgibbon, "KinectFusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. IEEE, oct 2011, pp. 127–136.
[11] L.-K. Cahier, T. Ogata, and H. Okuno, "Incremental probabilistic geometry estimation for robot scene understanding," in *ICRA 2012*. Ieee, may 2012, pp. 3625–3630.
[12] W. J. Schroeder, B. Lorensen, and K. Martin, *The visualization toolkit*. Kitware, 2004.
[13] Kitware. (2016) Vtk the visualization toolkit. [Online]. Available: http://www.vtk.org/
[14] ——. (2016) Vtk about. [Online]. Available: http://www.vtk.org/overview/
[15] Github. (2016) Github mabdi repository. [Online]. Available: https://github.com/lucasplus/MABDI