Development of a Robust Indoor 3D SLAM Algorithm

Timothy Murphy
Ohio University
Honors Tutorial College
35 Park Place
Athens, OH 45701
tm507211@ohio.edu

Dr. David Chelberg
Ohio University
School of Electrical Engineering and Computer Science
Russ College of Engineering and Technology
Ohio University
chelberg@ohio.edu

ABSTRACT
Mobile robots need to continuously navigate their environment. Doing so necessitates using sensor data to both map that environment and locate their position. A 3D Simultaneous Localization and Mapping (SLAM) algorithm for use with indoor robots has been developed that addresses these problems. It is based upon using a Microsoft Kinect sensor, extracting features and keypoints, matching the resulting features and keypoints, and finally using the matched keypoints to compute transformations that produce a consistent global map consisting of aligned point clouds. The algorithm aligns RGB-D point clouds in an efficient manner; however encounters problems for visually sparse environments (small or no color differentiation).

The current results of the algorithm are presented with comparisons to other frequently used techniques and include possibilities for extensions to the algorithm.

This paper includes specifics about the implementation using the Point Cloud Library (PCL), OpenCV, and the actual experimental robot, Turtlebot 2.0.

1. INTRODUCTION

Mobile robots, by definition, traverse the real world. In doing so, the robot needs to know where it is, where it is going and what objects are in its surroundings. Given an unknown location, the robot has to both create a map of its environment and locate itself within that map. A frequent method used to achieve this utilizes a Simultaneous Localization and Mapping (SLAM) algorithm. The idea behind SLAM was first presented in [2]. This paper addresses the problem of how to create a 3D map of the robot's surrounding environment while at the same time determining the location of the robot itself within this map. This is done through the use of an original SLAM algorithm. This paper will have three focuses: specifying how the algorithm should behave, describing the current state of the algorithm with comparison to other mapping and alignment techniques, and detailing potential further improvements to the algorithm.

Many SLAM algorithms are used with two-dimensional (2D) sensors such as a video camera producing color data, or three-dimensional (3D) sensors such as a laser depth scanner. This new algorithm takes color (RGB) in addition to depth (D) information as its input. Specifically, it receives sequential RGBD point clouds and creates a 3D representation of the surrounding environment and determines the origin of the sensor (the location of the robot) for each point cloud. The algorithm presented in this paper is implemented on the Turtlebot 2.0 robot using the ROS, PCL, OpenCV libraries, and is written in C++. For more information about the robot or libraries used visit the links in [10], [11] and [12].

2. ALGORITHM BEHAVIOR

The algorithm described within this paper creates a robust 3D SLAM algorithm for use

in indoor environments. The algorithm is created for use with ground-based robots that can produce RGBD point clouds. These point clouds contain the 3D information of a specific scene. The algorithm makes the assumption that the robot's sensor produces RGBD point clouds; however, the algorithm can still be used if sensory input from a robot can be transformed to create a RGBD point cloud. For this reason, the SLAM algorithm was implemented on a Turtlebot 2.0 with a Microsoft Kinect sensor that directly produces RGBD point clouds without the need for extra transformational computation.

The algorithm takes a series of sequential RGBD point clouds and successively aligns the point clouds into one consistent frame. The algorithmic output is a single RGBXYZ (color and 3D position) point cloud that contains each input point cloud aligned into one consistent frame.

The end goal for the algorithm is to both align successive point clouds and also to perform loop closure – loop closure is when the robot recognizes that it has already visited its current location – and to be able to do both in real time. Loop closure detection is important to reduce accumulated errors in the computed map.

3. CURRENT STATE OF ALGORITHM

3.1 Overview

The algorithm is split between several steps, setting up the initial variables, converting between different 3D and 2D images, extracting features and keypoints, matching keypoints, and finally estimating the transformation between images. The following pseudocode gives a high level description of the algorithm. The algorithm continues until the user exits. The algorithm performs initial set up and starts the loop by accessing the next point cloud from the sensor and then processes the point cloud.

```
3D_SLAM{
  Clouds = {};
  Keypoints = {};
  Descriptors = {};
  Transform = {1, 0, 0, 0,
               0, 1, 0, 0,
               0, 0, 1, 0,
               0, 0, 0, 1};
  while(!user.quit()){
    cloud = get_next_cloud();
    Clouds.append(cloud);
    img = get_2d_image(cloud);
    feature, keypoint = get_2D_keypoints(img);
    Keypoints.append(keypoint);
    Descriptors.append(feature);
    if (length(Clouds) > 1){
      2D_corr = get_2D_correspondences(
                        Keypoints[-1], Descriptors[-1],
                        Keypoints[-2], Descriptors[-2]);
      3D_corr = get_3D_correspondences(
                        2D_corr, Clouds[-1], Clouds[-2]);
      RANSAC(3D_corr);
      Transform = get_transform(3D_corr);
      Clouds[-1] *= Transform;
    }
  }
 }
}
```

3.2 Initial Setup and Conversions

As shown above, several variables for holding the point clouds, keypoints, description, and transform are set to an initial state. The transform is set to the 4x4 identity matrix and the rest are set to be empty lists. When the program receives the point cloud input, it is in RGBD format, which then needs to be converted into RGBXYZ (color and 3D position) format for easier use. The Point Cloud Library (PCL) provides an easy conversion between point cloud types. The first step, after receiving the point cloud, is to convert the 3D point cloud into a 2D image. This is relatively simple as the sensor produces a point cloud such that each point matches exactly to a pixel of the 2D image. That is the point cloud is stored in the same order as the corresponding 2D image is. This 2D image is stored as an OpenCV Mat image for compatible use with OpenCV functionalities. OpenCV is a computer vision library (focusing on 2D computer vision) that

implements several useful computer vision algorithms and visualization tools in C++ and Python. These features include changing the format of images, extracting and matching features, and providing easy to use methods for showing images and their calculated features on the screen.

3.3 Feature and Keypoint Extraction

Having obtained a 2D image of the environment, the next step is to extract important features such as Speeded Up Robust Features (SURF) and Normal Aligned Radial Features (NARF) using a local feature detector. SURF features are calculated locally within the image using a Hessian Blob Detector. NARF features take a range image, normalize the range image, and calculate feature descriptors by determining how much the range has changed within a set distance from the point of interest. Extraction of these keypoints is important as it allows for a much smaller set of points to be used when calculating the transform between two point clouds. In theory, each point within the point clouds could be used as a keypoint for calculating the transform between them; however, this would be very computationally expensive and therefore would not be feasible in an application meant to run in real time. Another advantage to keypoints is that a keypoint is representative of the points surrounding it – that is the keypoint is calculated using not only the point it represents but also the surrounding points, allowing keypoints to be more stable than any one point. Two different methods for calculating keypoints that can both be associated with 3D coordinates were tested. See section 5 for comparison of times needed to calculate NARF and SURF keypoints on experimental robot.

The first method used the feature detector SURF. SURF keypoints were calculated using OpenCV's SURF descriptor class with a Hessian threshold of 400. This threshold was chosen after testing several other thresholding values. Values greater than 400 produced too few keypoints while values less than 400 produced less distinct keypoints that created poor matches. The Hessian value refers to the measure of the point using the Hessian Corner Detector. The Hessian Corner Detector is described in [8] as being an affine-invariant feature detector that uses the second partial derivatives of the image after being smoothed with a Gaussian Kernel. Affine-invariant features are stable across affine-transforms which include scale, rotation, translation and any other transform that preserves points, straight lines, and planes. SURF extracted keypoints from the 2D image that were then associated with 3D space (Figure 1). The circles on the OpenCV image represent each keypoint where the circle size illustrates the distinctiveness of the keypoint, the line represents its 2D orientation and the centroid is the actual location of the keypoint. Each green point in the point cloud represents the XYZ position of the keypoint.

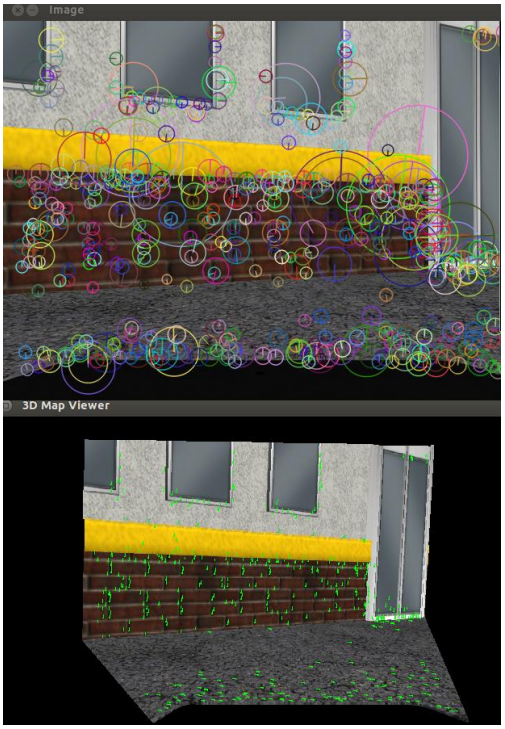

Figure 1. OpenCV Image with Surf Keypoints (Top), Surf Keypoints Associated to 3D Point Cloud (Bottom)

The second tested method utilized the NARF method. Figure 2 shows a 2D image, range image (depth image with violet being the closest depth and red the farthest away), and down-sampled point cloud with the results of the NARF method shown in green. The point cloud is down-sampled in order to reduce the amount of space needed to store the point cloud, as well as reducing the amount of time need to perform the transformation into a consistent frame. Down-sampling keeps the general shape of the point cloud while reducing the total number of points needed to represent the point cloud.

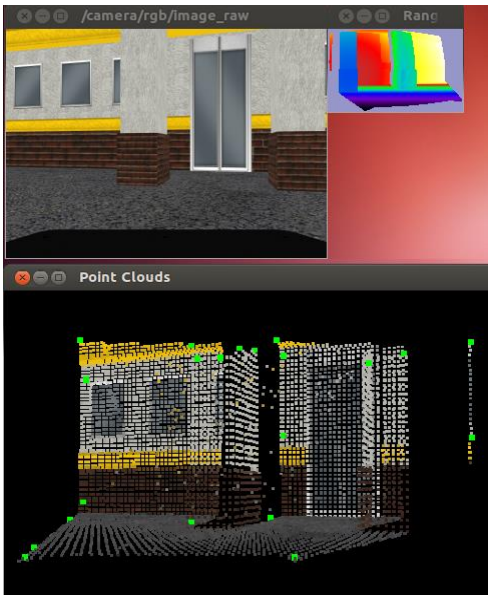

Figure 2. 2D Image (Top Left), Range Image (Top Right), Down Sampled with Narf Features (Bottom)

From the two examples it can be seen that the SURF keypoint extractor produces many more keypoints than the NARF keypoint extractor. In practice the SURF keypoints were more stable than the NARF keypoints. This can also be seen in the Figure 2 as many of the NARF keypoints were found at the edge of the point cloud. The keypoints found at the edge are not repeatable from different positions and therefore are not suitable for use with keypoint matching between images. When the robot changes pose, the edges of the point cloud will too. The SURF keypoints also have the added advantage of incorporating the use of color information in the SLAM algorithm instead of only using depth information. The current state of the algorithm uses SURF features for keypoint matching and does not calculate NARF keypoints as the number and quality of NARF keypoints was unsatisfactory for aligning point clouds.

3.4 Keypoint Matching

The SURF keypoints are matched using OpenCV's Fast Library for Approximating Nearest Neighbors (FLANN) algorithm. The next step involved finding the minimum distance between two matching points and filtering the correspondences by only accepting the points that are less than twice the minimum distance or some other small threshold (0.2). This initial filtering of the keypoint matches tries to determine good matches. The transformations between one image to the next only include rotation, translation, or scale which are linear transformations. This implies that the distance between any two corresponding keypoints should be fairly consistent. Any distance greater than the threshold is likely to be a bad match and is rejected. These specific thresholds were chosen by experimentally testing several thresholds. Thresholding based only on the minimum distance produced very few keypoints when the minimum distance was less than 0.1. With a threshold of more than twice the minimum distance, poorly matched keypoints became more prevalent. Figure 3 shows two consecutive images where SURF keypoint correspondences are shown by connecting lines from one image to the other.

Figure 3. SURF keypoint correspondences.

Next the 2D correspondences are converted into 3D coordinate space. This process is the inverse of the one used to convert the point cloud into a 2D image. The 3D correspondences are run through a Random Sample Consensus Algorithm (RANSAC) implemented in the PCL Library. The RANSAC algorithm finds a set of inliers and outliers based on the distance between correspondences. The filtered correspondences are then passed on to the transformation estimation. These points are being filtered again through the RANSAC algorithm in order to find a consistent set of corresponding keypoints. This will find a set of inliers and outliers and try to maximize the number of inliers. Inliers are corresponding keypoints that have a small distance between them (in a 3D Space). The process iteratively chooses a random set of initial corresponding keypoints and tests how many corresponding keypoints follow the same transform (these corresponding keypoints are the set of inliers). All outliers, after the final iteration, are discarded, in order to produce the set of corresponding keypoints that produces a more consistent transform. The largest set of inlying corresponding keypoints is more likely to produce the correct transform between the two point clouds. If all correspondences were used for transformation estimation, including the outlying correspondences, the likelihood of errors in the transformation would increase due to the inconsistent correspondences.

3.5 Transformation Estimation

Estimating the transform between the two point clouds is necessary in order to align the two point clouds together. This is done by computing the transform from the previous point cloud to the current point cloud – doing this transforms each point cloud into the same frame as the first point cloud. This is because the previous cloud was already aligned into the same frame as the cloud before it. The transform from one point cloud to the previous can be calculated, after the 3D correspondences are calculated. The algorithm uses PCL's implementation of Levenberg Marquardt (LM) based estimation. The PCL implementation iteratively computes the least-square of cost function (distance between corresponding points) and will continues until a minima is reached. This means this implementation works well for point clouds that are already relatively closely aligned. The LM based estimation returns a 4 by 4 transformation matrix that can then be applied to the newest point cloud to align it into the previous point clouds frame.

3.6 Summary of Algorithm Behavior

At this point, the algorithm takes a sequence of RGB-D point clouds. It computes the 2D color image from this incoming point cloud as well as the corresponding RGBXYZ point cloud. The algorithm takes the 2D image and computes SURF features and keypoints using OpenCV. The keypoints are matched to the keypoints calculated for the previous point cloud using OpenCV's FLANN algorithm. The matches are filtered down in 2D space then used to create the list of 3D correspondences. The 3D correspondences are filtered down again using a RANSAC algorithm using PCL. The resulting correspondences are then used to calculate a transform using PCL's Levenberg Marquardt Transformation Estimation algorithm to align the newest point cloud to the one before it. Figure 4 shows a flow chart (left to right) that shows how the algorithm is organized. The lines show that the block on the left's output is an input to the one on the right of the line.

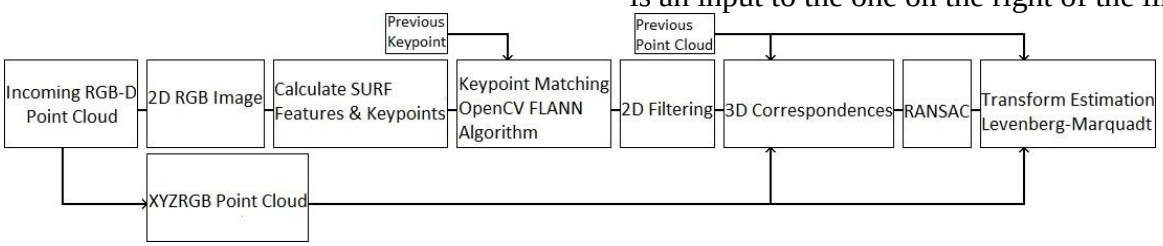

Figure 4. Flowchart of Algorithm organization. The process goes from left to right.

3.7 Comparison of SLAM Techniques

SLAM is a common method of aligning sensor readings together into a common frame. The authors of [5] and [6] describe SLAM algorithms that implement Appearance based SLAM algorithms. That is each implement a SLAM algorithm that uses the RGB information of their sensor's input. The author of [9] describes a SLAM algorithm that uses Depth sensors in order to build its map and determine the location of the robot. All three of these algorithms describe various SLAM algorithms for use with mobile robots; however, these do not discuss using both color and depth information for calculating the transform between each frame. The algorithm described in [6] does create a RGB-D map as its final product; however, the transformations are calculated using only color information. The algorithm described within this paper uses both color and depth information to calculate the transformation between frames. This is done by first calculating the color features in 2D space then associating these features to their 3D location within the point clouds. These keypoints that are a combination of color features and 3D position are then used to calculate the transform between frames.

4. FUTURE IMPROVEMENTS

4.1 Features

The next goal is to make the algorithm more robust i.e. the algorithm is made more likely to find the correct transform from one frame to the next. The algorithm would benefit from computing more feature types. Currently only SURF features are being computed as NARF features produced poor results in most situations. Adding additional feature types will likely produce more reliable transformation estimates, because there are more methods of calculating the transform between frames. [1] and [7] describe the algorithms for calculating SURF and SIFT features as well as what these features represent, respectively. Each of the features can be used to calculate the transform. If multiple feature sets produce a similar transform, there becomes a greater certainty that that transform is correct. Another issue could be that one type of feature could produce the wrong transform in certain situations. If this is noticed, the algorithm could exclude it from the results and rely only on the other feature transforms in those situations.

Several popular features for use in feature matching are Scale Invariant Feature Transform (SIFT), Binary Robust Invariant Scalable Keypoints (BRISK), and Features from Accelerated Segment Test (FAST). Fox et al [3] describes a similar mapping algorithm with the use of SIFT features. The authors of [6] discuss an online mapping algorithm for large-scale areas by using SURF and SIFT keypoints.

The added features and keypoints can be used to calculate a transformation estimate for each incoming point cloud and then would allow the program to choose between the transformations returned. If multiple keypoints produce a similar transform then it is more likely for that transform to be correct. If each of the keypoints produces several different transforms, then this also lets the algorithm know more work needs to be done to calculate an acceptable transform.

4.2 Transformation Estimation

The transformation estimation portion of the algorithm can be improved by implementing the LM transform estimation algorithm so that a method for testing convergence can be implemented. The newly implemented LM transform estimation algorithm would be very similar; however, it would offer a method of testing the alignment of the point clouds. [4] offers an implementation of Levenberg-Marquardt algorithm and a method of calculating the covariance matrix that can be used to determine how well the point clouds are aligned. If two point clouds are aligned,

this means they have been transformed into the same frame of reference. If each of the point clouds are fully aligned, the resulting map would be a reconstruction of the environment that has been seen by the robot. This means that the better the alignment between frames, the more accurate the map is.

Another common method for calculating rigid transformation estimation using corresponding points is Singular Value Decomposition (SVD) that is also implemented in PCL.

Being able to use both of these estimation techniques will increase the robustness of the algorithm. Being able to compare the output of each technique allows the determination that the transform is more likely to be correct if both produce similar results.

4.3 Loop Closure

Loop closure occurs when the robot comes back to a place it has already been to (the robot has performed a loop). This can also be defined as two non-consecutive point clouds that share a number of matching keypoints that are greater than a given threshold. Loop closures can reduce the error of mapping systems. Each loop closure adds another constraint to the map. The map then distributes the error along the loop evenly (visually this appears as if the map is snapping into place such that the two ends of the loop match up). Figure 5 shows a depiction of loop closure where each is numbered in sequential order and each line shows a known connection between sensor readings. Each numbered circle's position represents the X,Y position of the robot when it took the sensor reading.

5. Experimental Robot

The robot used for testing this algorithm is the Turtlebot 2.0. A picture of the robot can be seen in figure 6. The robot consists of a Kobuki base, a Microsoft 360 Kinect sensor, a mounting structure and an ROS enable laptop. The laptop used for testing was the ASUS X200CA with an Intel Celeron 1007U processor and 4 GB RAM. Other sensors include the Kobuki's wheel encoders, gyroscope, bump, cliff, and wheel drop sensors. Feature extraction of NARF and SURF keypoints were performed using the Turtlebot's Laptop. Each feature type was computed on 30 point clouds that had been previously captured. SURF keypoints were calculated on average in 133 ms while the NARF Features took on average 14.405 seconds for the same set of point clouds.

Figure 6. Turtlebot 2.0 (picture from www.turtlebot.com)

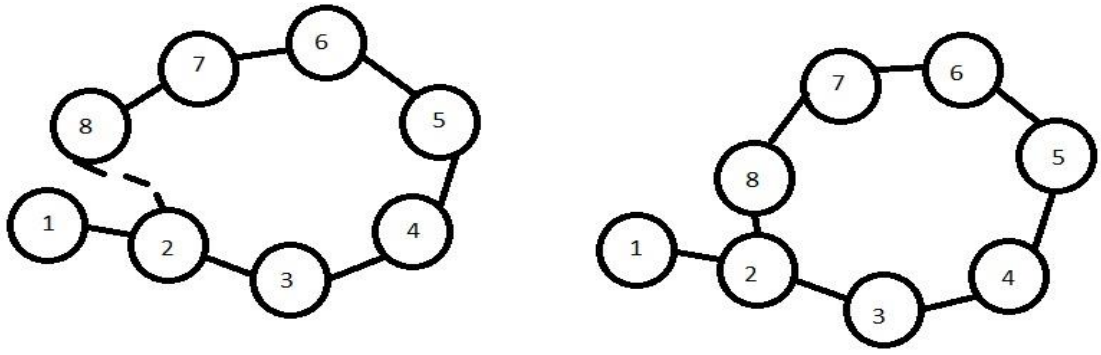

Figure 5. Depiction of Loop Closure: Loop Detected between 8 and 2 (Left) Loop Closed (right)

6. CONCLUSION

This paper described the current status of a new RGB-D slam algorithm, how keypoints and features are calculated, how the keypoints are associated into 3D space, how the correspondences are filtered to reduce the number of bad correspondences and how the remaining correspondences are used to calculate the transform between the current and previous point cloud. The algorithm described above uses both color and 3D position information to determine the transform between point clouds, which differs from previous SLAM techniques. Also discussed in this paper is future work to be performed to create a more robust algorithm by incorporating more feature types as well as changing the implementation of the transformation estimation algorithm to allow for testing for alignment of the point clouds.

7. REFERENCES

[1] Bay, H., Ess, A., Gool, L., Tuytelaars, T. SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding (CVIU).* (2008). Vol 110, No. 3, pgs 346-359.

[2] Cheeseman, P., Self, M., Smith, R. Estimating Uncertain Spatial Relationships in Robotics. *Proceedings of the Second Annual Conference of Uncertainty in Artificial Intelligence.* (1986). University of Pennsylvania, Philadelphia, PA, USA: Elsevier. pgs 435-461.

[3] Fox, D., Henry, P., Herbst, E. Krainin, M., Ren, X. RGB-D Mapping: Using Depth Cameras for Dense 3D Modeling of Indoor Environments. Web. Viewed 21 April 2014. http://www.cs.washington.edu/robotics/postscripts/3d-mapping-iser-10-final.pdf. 2013.

[4] Flannery, B., Press, W., Teukolsky, S., Vetterling, W. Numerical Recipes 3rd Edition: The Art of Scientific Computing. *Columbia Press*. (September 2007).

[5] Hung, D., Sun, C., Wang, Y. Improving Data Association in Robot SLAM with Monocular Vision. *Journal of Information Science and Engineering 27*. (March 2011). pgs 1823-1837.

[6] Labbe, M., Michaud, F. Appearance-Based Loop Closure Detection for Online Large-Scale and Long-Term Operation. *IEEE Transactions on Robotics*. (June 2013). pgs 734-745

[7] Lowe, D. G. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*. (2004). Vol 60, No. 2, pgs 91-110.

[8] Mikolajczyk, k., Schmid, C. An affine invariant interest point detector. *Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada.* 2002.

[9] Mirowski, P. Depth Camera SLAM on a Low-cost WiFi Mapping Robot. Web. Viewed 21 September 2014. http://www.academia.edu/2527447/Depth_camera_SLAM_on_a_low-cost_WiFi_mapping_robot

[10] Description of Turtlebot 2.0 can be found at http://www.turtlebot.com/

[11] Description of Microsoft Kinect and how it can be used can be found at http://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx

[12] For specifics on OpenCV, PCL, and ROS visit http://www.opencv.org, http://pointclouds.org, and http://www.ros.org respectively.