Fusion of Depth and Color Images for Dense Simultaneous Localization and Mapping

Dylan T. Conway and John L. Junkins
Department of Aerospace Engineering, College Station, Texas A&M University, TX, United States
Email: {dtconway, junkins}@tamu.edu

***Abstract*—This paper presents a system for performing dense mapping of surface geometry and texture properties. Co-registered depth and grayscale images are incrementally fused into a global map in real-time as they are collected. The resulting dense map can be rendered into a virtual depth and grayscale image from any arbitrary pose. Comparison of the rendered and observed images provides a direct means of computing the sensor pose relative to the map allowing new data to be fused into the model. This frame-to-map tracking scheme, as opposed to frame-to-frame tracking, improves system accuracy and robustness. Additionally, the use of both surface geometry and color texture better constrains the pose solution and reduces the risk of tracking failures. This paper describes an implementation of the proposed algorithm and provides experimental results.**

***Index Terms*—SLAM, data fusion, GPU, dense mapping**

I. Introduction

Localization of a sensor in an unknown scene is a central problem in navigation. Applications range from industrial robotics to spacecraft guidance. In order to plan safe trajectories and move to regions of interest, a map of the scene's surface must be generated. A reference frame must be embedded in the scene to perform this task. Then all surfaces can be described in this frame and the sensor pose relative to this frame can be estimated. Vision sensors have formed the front-end of many Simultaneous Localization and Mapping (SLAM) systems to solve this problem.

Vision sensors generate measurements that depend on both the surface geometry and the sensor pose. The goal of the SLAM problem is to solve for the geometry and pose given the vision measurements. Many applications demand real-time performance in unstructured scenes which has been a major challenge to practical systems. Many algorithms use either passive cameras or depth cameras but little work has been done on optimally fusing data for a co-registered pair of the two. The goal of this work is to accomplish that task.

The proposed method builds on earlier advances in the SLAM literature. Geometry based methods that use depth images and visual feature based methods that use color images are combined into a single algorithm that can operate in real time. First a brief review of recent developments in SLAM is given. Then details of the proposed algorithm are discussed. Finally, results on experiments tailored at spacecraft navigation applications are given.

Manuscript received February 15, 2014; revised June 19, 2014.

II. Background

The vision-aided SLAM problem has received an enormous amount of attention in the literature. SLAM methods estimate the parameters of a map structure which can grow over time. Typically images are searched for previously mapped features. The estimated map locations of those features are used with observation data to update sensor pose. Then with an updated sensor pose, new features can be initialized into the map and the location estimates of existing features can be refined.

The MonoSLAM implementation which used a single passive camera was one of the earlier successes in this field [1]. That system used an Extended Kalman Filter (EKF) where the sensor pose was augmented by the locations of discrete visual feature points in the state vector. The augmented EKF lacked robustness because incorrect feature correspondences easily corrupted the entire state vector. This was addressed in [2] and [3] by tracking features independently of one another and by using particle filter methods. These algorithms significantly improved overall robustness in real world scenes and reduced the computational requirements.

All of these systems relied on discrete feature points to populate their maps. This was done largely to fit the SLAM problem into a traditional state estimation framework. When using discrete points, a visual feature tracker or matcher is required to solve the correspondence problem. Algorithms like Lucas-Kanade optical flow or SURF have been used for this task but often fail to match features over widely varying viewpoints [4] [5]. This failure is a major hurdle to bounding error estimates and closing loops.

Dense mapping is an alternative to the discrete point approach. In these methods a three-dimensional voxel grid is embedded in the scene. Methods in this class recursively estimate the probability of each cell being occupied by a solid surface. One example is a memory efficient implementation known as OctoMap which stores the grid in a tree structure to prune away finer resolution cells covering wholly occupied or wholly unoccupied space [6]. Curless and Levoy presented an alternative dense method where a Signed Distance Function (SDF) is

doi: 10.12720/joig.2.1.64-69

discretized over the voxel grid [7]. The value of a true SDF at a given point is simply the distance from that point to the nearest surface. Positive SDF values represent free space while negative values represent solid interiors. The actual surfaces are extracted by locating the zero-crossings. The advantage of the SDF over an occupancy grid is that the resulting surface resolution is much finer than the resolution of the grid cells. Therefore more accurate maps can be made with similar memory resources.

Newcombe *et al.* presented a state-of-the-art implementation of dense surface mapping in [8] which relied on a Truncated Signed Distance Function (TSDF). Their method was demonstrated with a Microsoft Kinect sensor. Each voxel of their map contained two values: the TSDF and the TSDF weight. This representation is convenient for two reasons. First, given a depth image from a known sensor pose, the observed data can be fused into the map in a recursive fashion. Second, given an arbitrary sensor pose and known map, a depth image can be *rendered* from that pose by ray casting into the map. Iterative Closest Point (ICP) is applied to a depth image rendered from an a priori pose estimate and the observed image to find a pose correction. Using this accurate pose update, the observed data can be fused into the map. Each stage of the algorithm is parallelized onto GPU hardware and can operate at the sensor's 30 Hz frame rate. Their method directly inspired this paper.

In [8], an ICP algorithm is used to align the rendered and observed depth images [9] [10] [11]. This method works well when the pose estimate is close to the truth and the scene's geometry is sufficient constrain the pose. However, this may not always be the case. This paper proposes the use of co-registered grayscale images to incorporate surface texture to better constrain the pose solution.

III. Method

The proposed method has a similar structure to that in [8]. The major difference is that the surface texture is estimated and used, along with the geometry, to constrain the pose solution. Each voxel in the map has an additional two values: an albedo and an albedo weight. This allows both a depth image and a grayscale image to be rendered from an arbitrary pose under the assumption of a diffuse lighting model by ray casting into the scene. On the next frame, the previous pose serves as an a priori estimate of the current pose. A depth image and grayscale image are rendered from this pose. The host computer receives the latest sensor data and passes a copy of it to a GPU. Then two separate host threads are launched that run independently of one another.

The first thread operates entirely on the host and computes a pose update based on visual feature matching between the rendered and observed grayscale images. The second thread manages GPU kernel launches to get a geometry based pose update. This thread first passes sensor data through a preprocessing stage. Then an ICP algorithm aligns the rendered and observed depth image to get a pose update. Once both updates have been found, the main thread of execution resumes. The two estimates are merged in a probabilistic sense and the new pose estimate is passed to the GPU. The GPU then performs a map update and a surface prediction to be used on the next raw data set. The algorithm is outlined in Fig. 1.

By fusing both data sources the algorithm can overcome the failure modes of low texture variation and low geometry variation when either is present alone. Additionally, the visual-based pose update is better suited to perform large updates unattainable by ICP as shown below.

A. Notation

A notation similar to that in [8] is used for easy comparison of the algorithms. Two reference frames of interest are defined. A global reference frame $\mathbf{g}^+$is fixed in the scene. A second frame is fixed to the sensor. The sensor frame at time *k* is located at position $[\mathbf{t}_{g,k}]_g$ relative to the global frame. Note that the subscript outside of the brackets indicates the frame in which the vector is coordinatized in. The rotation between the two frames is represented by the orthogonal matrix $\mathbf{R}_{g,k}$. This matrix maps vector coordinates from one frame to the other: $[\mathbf{r}]_g = \mathbf{R}_{g,k}[\mathbf{r}]_k$ and $[\mathbf{r}]_k = \mathbf{R}_{g,k}^T[\mathbf{r}]_g$.

The Microsoft Kinect sensor provides a measured depth map $\tilde{d}(\mathbf{u})$ over a two-dimensional pixel domain $U = [0,479]x[0,639]$.

A depth measurement at a given pixel $\mathbf{u} = [u,v]^T$ can be converted into a measured three-dimensional point as follows. First define the homogenized pixel $\dot{\mathbf{u}} = [\mathbf{u}^T, 1]^T$and the camera calibration matrix K.

Then the measured three-dimensional point at a given pixel $\mathbf{u} = [u,v]^T$ in sensor frame coordinates is $[\tilde{\mathbf{V}}_\mathbf{k}(\mathbf{u})]_k = \tilde{d}(\mathbf{u})\left(K^{-1}[\dot{\mathbf{u}}]\right)$. Note that the range to the point is $\tilde{d}(\mathbf{u})/\lambda$ where $\lambda = |\left(K^{-1}[\dot{\mathbf{u}}]\right)|^{-1}$.

Each voxel of the map is located by a position vector $[\mathbf{r}]_g$. If the sensor pose is known, this voxel location can be projected on to the pixel array. The sensor-to-cell vector in the sensor frame is $[\mathbf{V_r}]_k = \mathbf{R}_{g,k}^T([\mathbf{t_{g,k}}]_g - [\mathbf{r}]_g)$. The projection function $\pi([X,Y,Z]) = [X/Z, Y/Z]$ is applied to this vector to get a pixel $\mathbf{u} = \pi([\mathbf{V_r}]_\mathbf{k})$. Then the measured cell-to-surface vector can be determined from the measured point at $\mathbf{u}$.

$$\delta = [\tilde{\mathbf{V}}_\mathbf{k}(\mathbf{u})]_k - [\mathbf{V_r}]_k \tag{1}$$

The $\ell_2$norm of δ is scaled by $cos(\theta)$, where θ is the angle between the ray and surface normal, $[\hat{\mathbf{n}}]_k$, to give a measurement of the cell's SDF. This geometry is seen in Fig. 2.

B. Data Processing

The proposed method employs a similar processing stage as in [8]. A bilateral filtered version of the depth image is created to remove noise while preserving depth discontinuities. This is then used to compute a surface normal map. In this implementation, no map hierarchy is required. This is likely due to the more robust pose estimation enabled by the fusion of visual data.

The value of $cos(\theta) = [\hat{\mathbf{n}}]_k^T\left(K^{-1}[\dot{\mathbf{u}}]\right)\lambda$ is required during the map update. This can be found by taking the

numerical gradient of the SDF about the particular cell during the map update. However, any added steps to the map update are extremely costly because of the number of cells that must be processed. Instead, the value of $cos(\theta)$ is most efficiently and accurately obtained during the normal map computation for two reasons. First, the normal vectors will be stored in on-chip registers at that time which avoids global memory accesses. Secondly, the $cos(\theta)$ computation becomes independent of the pose estimate errors: both the normal and ray vector are computed directly in the current sensor frame. Therefore the proposed method computes $cos(\theta)$ during this stage and writes it to an additional array of surface memory which is read during the map update function.

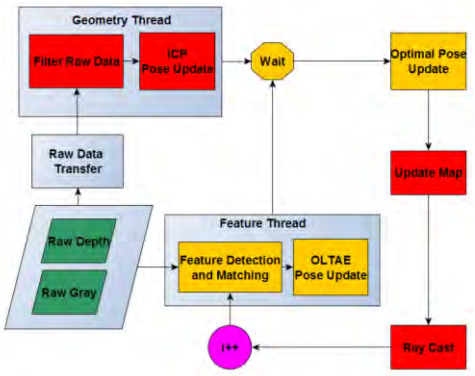

Figure 1. System block diagram. GPU based operations in red. Host based operations in yellow.

C. Map Update

The N_xxN_yxN_z voxel grid is stored in GPU surface memory. With the most recent sensor pose update available, N_xxN_y kernels are launched. Each kernel has a unique (x, y) grid location and marches in step with the other kernels along the z direction. At each cell, the estimated δ vector is computed using (1). If the norm of δ is greater than the maximum noise of the sensor, μ, then the cell must be occluded by another surface and is therefore not updated.

If the norm of δ is less than the negative of μ, the cell is in free space. In this case the measured SDF is simply truncated to μ and is given a weight of *1.0*. Otherwise, the cell is within μ of the observed surface.

In this case the SDF is computed by scaling the norm of δ by $cos(\theta)$ as shown in Fig. 2. Note that the pixel ray is assumed to pass through the cell and hit a surface that is planar in the immediate vicinity of the cell. To account for poor measurements on surfaces viewed at low angles, a weight of $cos(\theta)$ is given to the measurement.

The map estimate of the SDF is then updated using a weighted average between the original and measured value. The original SDF is weighted by that cell's weight which is then incremented by the weight of the measurement (*1.0* or $cos(\theta)$). The cell weight is updated simply be adding the weight of the current measurement to it.

The albedo and albedo weight are only updated for cells within μ of the surface because the albedo is undefined for cells in free space. The grayscale image value at the projected pixel location is used directly as an albedo measurement. It is possible to estimate a scale factor for each image to remove the effects of exposure compensation performed by the camera. However this was found to be unnecessary through experimentation. It may be necessary when large variations in lighting occur throughout the scene.

The map albedo estimate is updated with the same weighted running average scheme used for the SDF update. This is computationally efficient and the $cos(\theta)$ weight reduces the effects of poor measurements on glancing surfaces in a similar manner. Essentially this part of the algorithm 'paints' the surfaces of the scene.

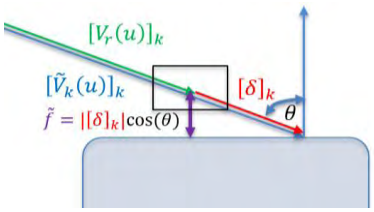

Figure 2. Signed distance function geometry.

D. Rendering

A simple ray casting algorithm as described in [8] is implemented here with a few modifications. The first modification is that a grayscale image is rendered in addition to the depth image. This is made possible by the albedo estimates stored in the map. Second, the sensor is allowed to leave the grid map and when it does, the free space between the sensor and grid is skipped over in one step. To do so, a check is first applied to the estimated ray starting location $[\mathbf{t}_{g,k}]_g$. If any of three components are beyond the fixed map, the estimated pose (denoted by a *hat*) is used to compute the unit ray vector through a given pixel in global frame coordinates:

$$[\hat{\mathbf{s}}(\mathbf{u})]_g = \left|K^{-1}[\dot{\mathbf{u}}]\right|^{-1}\left(\hat{\mathbf{R}}_{g,k}\left(K^{-1}[\dot{\mathbf{u}}]\right)\right) \quad (2)$$

If the x component of $[\mathbf{t}_{g,k}]_g$ is beyond the $+x$ bound of the grid $L_x/2$, the x component of $[\hat{\mathbf{s}}(\mathbf{u})]_g$ must be negative in order for the ray to intersect the map. If so, the distance skipped to reach the $+x$ bound of the grid is

$$d = \frac{\left([\mathbf{t}_{g,k}]_g^T\hat{\mathbf{i}}_\mathbf{x} - L_x/2\right)}{[\hat{\mathbf{s}}(\mathbf{u})]_g^T\hat{\mathbf{i}}_\mathbf{x}} \quad (3)$$

where $\hat{\mathbf{i}}_\mathbf{x}$ is the unit vector along the global x direction. The sign in front of the $L_x/2$ term is flipped if the sensor is beyond the $-x$ bound and the x component of the ray is greater than zero. Once found, a step of d along the ray is taken. This is then repeated for the y and z directions. The skipping of free space outside the map bounds saves a large amount of computation and allows the sensor to exit the map without any issue.

One kernel is launched per pixel of the output image pair. The ray either starts in the map because the sensor is in the map or it reaches the map edge in one step as described above. Steps of μ are then taken along the ray. On each step, the map SDF is polled at the nearest cell. If the SDF is truncated (i.e. equal to ±1), the steps of μ

continue. Note that this step size should be a few times larger than the cell resolution. A zero-crossing in the map will not be "stepped-over" because all cells within μ of the surface have a non-truncated SDF.

Once a truncated cell is hit, the step size is reduced to the cell resolution. The smaller steps continue until a zero-crossing is passed. If the ray returns to a truncated SDF region without hitting a zero-crossing, the step size is scaled back up to μ.

There are two stopping conditions for the ray stepping. First, if the ray exits the map, null values are written to the ray's corresponding pixel of the rendered images. Second, if a zero-crossing is reached a trilinear interpolation of the SDF is performed to find the zero-crossing to sub-cell-resolution accuracy. The depth to this cell is computed and written to the rendered depth image. Also, the SDF weight and the albedo value are written to a "weight-image" and grayscale rendered images respectively. The weight-image is another unique aspect of this algorithm and will be justified in the next section.

E. Pose Update

Two solutions to the pose are computed. One is based on geometric surface alignment and the other is based on visual feature matching. The fusion of these two estimates is a key component of this paper. Both methods use data rendered from the *a priori* pose estimate and data from the latest sensor image pair. The two resulting estimates for the pose between the rendered and observed images are first fused and then applied to the *a priori* pose estimate.

The ICP algorithm is used for the geometry based pose update. On each iteration the current pose estimate is used to solve the correspondence problem by mapping pixels in one image to pixels in the other. Each correspondence contributes one equation to a linear system. Each measurement is weighted by the corresponding "weight-image" mentioned above. The effect of this is to give greater weight to rendered depths from cells that have more certain SDF values. It reduces the influence of cells that have only been seen a few times or that have only been seen from poor angles.

The solution to the system gives a correction to the pose estimate. Details of the algorithm can be found in [8][9][10]. In this implementation, the components of the linear system are partially summed up in parallel on the GPU. The result is passed to the host which finishes the summation and solves the 6x6 linear system.

One host thread is instantiated to handle the launching of GPU kernels for ICP and the other minor computations needed for ICP. A second host thread independently performs the visual feature based update.

The visual feature based update runs entirely on the CPU which would otherwise be nearly idle while the GPU ICP operations are performed. This thread has full access to the host's copy of the rendered and observed grayscale image. Distinct features in both grayscale images are identified with the SURF detector and then described with the FREAK descriptor in OpenCV [12] [13]. Then a preliminary set of matches between the two sets are found. Since the depth at the features' pixel locations can be polled from both the predicted and observed depth image, gross outliers based on the *a priori* pose estimate can be quickly eliminated. Additionally the drawback of poor matching performance due to large variations in viewing angle is largely avoided because the predicted image is rendered from the *a priori* pose estimate which is close to the actual pose. In fact, the feature matching was found to be far more robust to pose errors than the ICP. The result of the feature matching process is a set of corresponding vector pairs: the coordinates of a set of 3D points coordinatized in each of the two reference frames. This set is sent to the Optimal Linear Translation and Attitude Estimator (OLTAE) algorithm [14].

OLTAE provides a direct means to compute the relative translation and attitude between two reference frames given the set of vector pairs. Since the OLTAE solution will be merged with ICP, the same pose parameters estimated in ICP will be used here. Given the point with predicted position in the previous frame $[\hat{\mathbf{V}}_{\mathbf{k-1}}]_{k-1}$ and the corresponding measured point $[\tilde{\mathbf{V}}_{\mathbf{k}}]_k$ in the current frame, a pose constraint can be generated:

$$\begin{aligned}[\hat{\mathbf{V}}_g]_g &= [\hat{\mathbf{R}}_{g,k-1}][\hat{\mathbf{V}}_{\mathbf{k-1}}]_{k-1} + [\hat{\mathbf{t}}_{\mathbf{g,k-1}}]_g \\ &= [\delta\mathbf{R}][\hat{\mathbf{R}}_{g,k-1}][\tilde{\mathbf{V}}_{\mathbf{k}}]_k + [\hat{\mathbf{t}}_{\mathbf{g,k-1}}]_g + [\delta\mathbf{t}]\end{aligned} \quad (4)$$

The top line of (4) is known. In the bottom line, the current pose is parameterized as an unknown rotation and translation from the previous pose. This parameterization is required for ICP and is adopted for OLTAE out of convenience. The unknown rotation is represented by the three Classical Rodriquez Parameters $[q]$ which are used to form the skew-symmetric cross product matrix $[Q]$. Then $[\delta\mathbf{R}] = ([I]+[Q])^{-1}([I]-[Q])$ where $[I]$ is the 3x3 identity matrix. Substituting $[\delta\mathbf{R}]$ into (4) gives (5). Rearranging (5) with the definitions provided in (7) and (8) leads to (6) where $[\delta\breve{\mathbf{t}}] \equiv ([I]+[Q])[\delta\mathbf{t}]$.

$$\begin{aligned}[\hat{\mathbf{V}}_g]_g = ([I]+[Q])^{-1}([I]-[Q])[\hat{\mathbf{R}}_{g,k-1}][\tilde{\mathbf{V}}_{\mathbf{k}}]_k \\ + [\hat{\mathbf{t}}_{\mathbf{g,k-1}}]_g + [\delta\mathbf{t}]\end{aligned} \quad (5)$$

$$[a] = -[Q][b] + [\delta\breve{\mathbf{t}}] \quad (6)$$

The other definitions used in (6) are:

$$[a] \equiv [\hat{\mathbf{V}}_g]_g - [\hat{\mathbf{t}}_{\mathbf{g,k-1}}]_g - [\hat{\mathbf{R}}_{g,k-1}][\tilde{\mathbf{V}}_{\mathbf{k}}]_k \quad (7)$$

$$[b] \equiv [\hat{\mathbf{V}}_g]_g - [\hat{\mathbf{t}}_{\mathbf{g,k-1}}]_g + [\hat{\mathbf{R}}_{g,k-1}][\tilde{\mathbf{V}}_{\mathbf{k}}]_k \quad (8)$$

Then the cross product identity $-[Q][b] = [B][q]$ can be used where $[B]$ is the skew-symmetric matrix constructed from $[b]$. This results in the least squares system:

$$\begin{aligned}[a] &= [B][q] + [\delta\breve{\mathbf{t}}] \\ &= [\ B \quad I\ ][\delta\breve{p}] \\ &= [H][\delta\breve{p}]\end{aligned} \quad (9)$$

The parameter vector $[\delta\breve{p}]$ contains the CRP and the modified translation update. Its least squares solution is the standard pseudo-inverse: $[\delta\breve{p}] = ([H]^T[H])^{-1}[H]^T[a]$. Each vector pair has an independent contribution to both the $[H]^T[H]$ and $[H]^T[a]$ term which are summed up and

then used to solve for $[\delta\breve{p}]$. Then the actual translation update $[\delta\mathbf{t}]$ can be computed to form the unmodified update parameters $[\delta p]$.

In practice, some false feature matches may seep through the initial filter. RANSAC is applied to defend against these false matches from corrupting the solution. Using an OLTAE hypothesis sample size of 2 for RANSAC, a sufficient inlier set is typically found on the first iteration. The inlier set is batch processed with OLTAE to refine the pose update.

To merge the geometric and texture based pose updates in a statistically near-optimal fashion, the error covariance matrices of both pose updates are needed. A simple and accurate approximation is to scale the $([H]^T[H])^{-1}$ matrix by a rough guess of the measurement error variances. This is done separately for OLTAE and ICP using predefined values for their associated measurement error variances. These can be tuned for particular scenes but experimentation suggests quality pose estimates occur whenever the variances are on the same order of magnitude. The merged solution is shown in (10) through (13) where the subscript *O* is for OLTAE and *I* is for ICP.

$$[P_O] = \sigma_O^2([H_O]^T[H_O])^{-1} \tag{10}$$

$$[P_I] = \sigma_I^2([H_I]^T[H_I])^{-1} \tag{11}$$

$$[W] = [P_O]\left([P_O] + [P_I]\right)^{-1} \tag{12}$$

$$[\delta p] = [\delta p_O] + W\left([\delta p_I] - [\delta p_O]\right) \tag{13}$$

Note that even when the variances used in (10) and (11) are the same, the resulting pose estimate in (13) is more than a simple average. If one of the solutions has week constraints about a particular pose parameter, that specific parameter will contribute less to the solution. This can occur for example when performing ICP on a planar surface or for OLTAE if all features lie near a straight line edge.

IV. Experiments

The proposed method was implemented on a laptop equipped with Intel i7 processors and a Nvidia GeForce GTX 780 graphics card. A Microsoft Kinect provided co-registered 640x480 pixel depth and color images. The current implementation processes this data and updates the pose at approximately 15 Hz for 5.6 million map cells. The results of two tests are shown below.

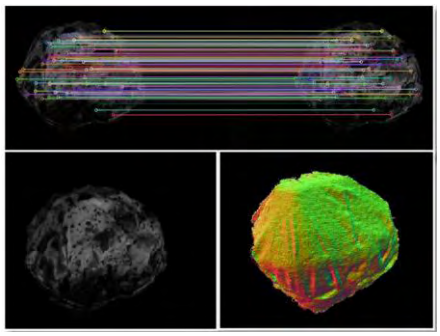

Figure 3. Feature matching (top) between rendered (top-right) and observed (top-left) images. The rendered grayscale image (bottom-left) and normal map (bottom-right).

In the first test the sensor was mounted on a tripod and pointed at an asteroid model rotating on a turntable at $4^o/s$. A sample feature matching set between the predicted and observed grayscale image is shown in Fig. 3 prior to RANSAC outlier rejection. Over three minutes of data was recorded allowing the model to complete more than two rotations. The resulting trajectory is shown in Fig. 4. As expected, the trajectory is confined to a circle. Note that the short arc length between the green and red dots is traversed three times and the rest is traversed twice. This accurate loop overlap is a direct benefit of the frame-to-model tracking.

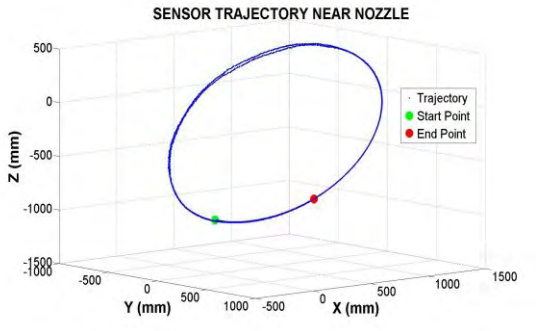

Figure 4. Trajectory about a rotating asteroid for just over two rotations. The loops accurately overlap each other.

Next a rocket nozzle model was imaged. The nozzle is radially symmetric and has little variation in geometry along most of its axis of symmetry. An image of the nozzle and a rendered normal map are shown in Fig. 5. The sensor was moved down the nozzle axis and back up to its original starting location. This trajectory is seen in Fig. 6. As expected, the starting and ending locations are co-located.

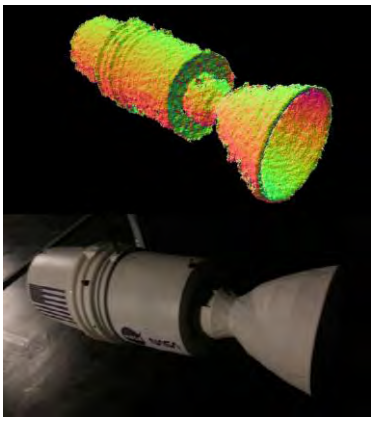

Figure 5. Observed grayscale image and rendered normal map of a rocket nozzle model.

The geometry solution alone is not sufficient to constrain the pose in this case. When the visual based update was turned off, the sensor pose estimate rolls about the axis of symmetry. Mathematically, this appears as a rank deficiency in the ICP linear system.

When the visual based update is turned on, the system seamlessly picks up visual cues from logos on the nozzle to constrain the pose. The probabilistic fusion of the two estimates will inherently weight the visual based update much higher along the direction that ICP is uncertain.

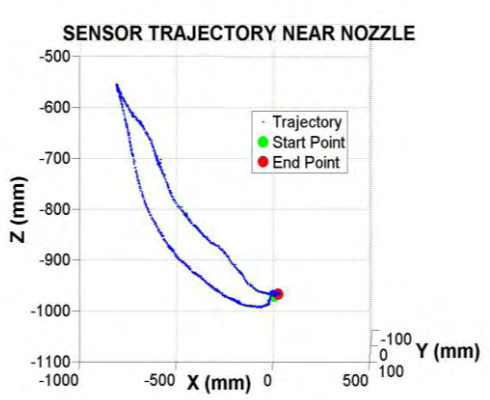

Figure 6. Trajectory along a rocket nozzle model. The sensor was manually returned to its original starting location.

V. Conclusion

An algorithm that fuses depth and color sensor data to solve the SLAM problem in fixed volume environments has been presented. It is a natural extension of the great work done in [8] which only used depth data. That work relied on geometry alone to constrain the pose solution. In this paper, the visual properties of the surface are estimated and used to further constrain the pose. The advantage of this was demonstrated experimentally for a case where surface geometry alone could not fully constrain the pose solution. This is critical in many applications.

There are certainly some drawbacks to the proposed method. The first is that the computation and memory requirements scale linearly with the environment volume. Compared to other dense volume representations like tree structures, the fixed voxel grid used here reduces the computation time per voxel and enables easy parallelization at the expense of more voxels being required. This is desirable for smaller volumes but becomes unmanageable as the number of voxels grows. We are currently investigating this tradeoff to extend the method to larger environments.

One immediate area of work in this direction is in aerial terrain mapping applications. Unlike complicated indoor scenes, the terrain elevation and albedo can be represented as a single-valued function over a 2D domain. A method tailored to this application can proceed in an analogous way to the method presented in this paper with two main differences. First, only a 2D map must be stored which greatly reduces the computational burden and memory requirements. Second, the ray casting can be sped up significantly by using the intersection of a ray and the ground plane as a starting estimate for an iterative routine.

One motivation of this work is to use it as the vision front-end of a sequential filter for vehicle navigation. The pose updates can be fused with estimates propagated in dynamic models in a traditional state estimation framework. This would improve the system accuracy and robustness as compared to no state propagation by reducing error drift. Work in this area is currently in development.

Acknowledgment

This work was supported by a NASA Space Technology Research Fellowship.

References

[1] A. Davison, I. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Trans. Pattern Analysis Machine Intelligence*, vol. 29, no. 6, pp. 1052–1067, June 2007.

[2] M. Pupilli and A. Calway, "Real-time camera tracking using a particle filter," in *Proc. British Machine Vision Conf.*, Oxford, U.K., September 2005, pp. 519-528.

[3] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "Fast-SLAM: A factored solution to the simultaneous localization and mapping problem," in *Proc. AAAI Nat. Conf. Artificial Intelligence*, July 2002, pp. 593–598.

[4] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proc. Seventh International Joint Conference on Artificial Intelligence*, Vancouver, Canada, August 1981, pp. 674–679.

[5] H. Bay, A. Ess, T. Tuytelaars, and L. V. Gool, "SURF: Speeded up robust features," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[6] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An efficient probabilistic 3D mapping framework based on octrees," *Journal of Autonomous Robots*, vol. 34, no. 3, pp. 189-206, 2013.

[7] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," in *Proc. 23rd Annual Conference on Computer Graphics and Interactive Techniques*, New Orleans, LA, August 1996, pp. 303-312.

[8] R. Newcombe, A. 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 *Proc. IEEE Int. Symp. Mixed Augmented Reality*, 2011, pp. 127–136.

[9] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239-256, February 1992.

[10] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Proc. 3rd International Conf. on 3-D Digital Imaging and Modeling*, 2001, pp. 145-152.

[11] G. Blais and M. D. Levine, "Registering multiview range data to create 3D computer objects," *IEEEE Trans on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 820-824, Aug 1995.

[12] A. Alahi, R. Ortiz, and P. Vandergheynst, "FREAK: Fast retina keypoint," in *Proc. IEEE Conference on Computer Vision and Pattern Recognition*, New York, 2012, pp. 510-517.

[13] Open CV library. [Online]. Available: http://code.opencv.org.

[14] M. Majji, B. Flewelling, B. Macomber, J. L. Junkins, A. B. Katake, and H. Bang, "Registration of LiDAR point clouds using image features," in *Proc. ASPRS 2010 Annual Conf.*, San Diego, CA, April 2010.

Dylan Conway is a graduate student in Aerospace Engineering at Texas A&M University. He obtained his B.S. degree in Mechanical Engineering in 2012 at the University at Buffalo, Buffalo, NY. He is currently a NASA Space Technology Research Fellow and has a research focus on vision-aided navigation.

John L. Junkins is a Distinguished Professor of Aerospace Engineering at Texas A&M University where he holds the Royce E. Wisenbaker Endowed Chair. He obtained his B.S. in Aerospace Engineering in 1965 from Auburn University, Auburn, AL. He then obtained an M.S. and PhD in Aerospace Engineering from UCLA in Los Angeles, CA. Dr. John L. Junkins is a member of the National Academy of Engineering, the International Academy of Astronautics, and an Honorary Fellow of the American Institute of Aeronautics and Astronautics. His research interests include spacecraft navigation, guidance, dynamics, and control.

