REAL-TIME MAPPING AND LOCALIZATION UNDER DYNAMIC LIGHTING FOR SMALL-BODY LANDINGS

Dylan Conway* and John L. Junkins†

Small-body landing missions present difficult challenges to Guidance, Navigation, and Control (GNC) systems. A typical mission profile makes a distinction between a mapping phase and a terminal navigation phase. In the mapping phase, analysts on the ground process spacecraft sensor data to generate a geometric and visual model of the body. This model is used to pick a particular landing site. Then during the terminal navigation stage, the spacecraft must autonomously drive itself towards the landing site using the map. There are two major hurdles here. The first hurdle is that the visual appearance of the map will change as the direction to the sun changes in both the body-frame and the sensor-frame. The second hurdle is that smaller landing hazards on the ground only become visible due to improved spatial resolution as the spacecraft gets closer to the surface and are therefore not a part of the map made from greater standoff range. This paper presents a method to clear both of these hurdles. An algorithm to sequentially estimate the full geometry and texture of the local terrain about the landing site is developed. With this information and an estimate of sensor-to-inertial and body-to-inertial pose available, the terrain is efficiently rendered under the actual lighting and estimated relative sensor pose conditions. The rendered images are then compared to sensor images to perform pose estimate updates. Details of the map parameterization, rendering algorithm, pose estimation method, and filtering are presented. Laboratory experiments in a simulated scene with ground truth data are used to validate the algorithm.

INTRODUCTION

The problem of estimating the state of a vehicle based on vision measurements to distinct points in an unstructured environment requires the simultaneous estimation of the locations of the observed points. This is because of the obvious coupling of the scene structure and vehicle state in the vision measurement equations. The problem is widely known as Simultaneous Localization and Mapping (SLAM) in the literature.

Proposed solutions to the SLAM problem can be classified in a number of ways. Two such classifications are batch versus filtering methods and sparse versus dense methods. The sparse methods model the environment as a set of landmarks, each with a unique and constant 3D
position in a scene-fixed reference frame. Under this assumption, the SLAM problem can be summarized as follows: extract a suitable estimate (maximum-likelihood estimate for example) from the joint distribution of landmark positions and vehicle states at the time of each image conditioned on the landmark measurements in each image.

So-called photogrammetry methods are based on batch estimation and they have a long history in aerial surveying applications\(^1\). These methods stack the vehicle states at the time of each image and the 3D position of all observed landmarks into a large parameter vector and then optimize an appropriate cost function over the parameter set. The cost function is the sum of some positive definite function of the error in the predicted and observed pixel measurements of each landmark seen in each image. It can be selected to give a Maximum Likelihood Estimate (MLE) for various measurement noise distributions. This gives excellent results but because of the large number of parameters and the connections between them, it is infeasible for real-time implementation in realistic scenes.

Beginning in the 1980s, the robotics community began investigating real-time SLAM solutions\(^2\). Traditional filtering approaches like the EKF were applied to a system state consisting of the usual vehicle states augmented with the positions of landmarks\(^3\). This provided a recursive solution for real-time implementation and reduced the complexity of the problem by marginalizing out previous vehicle poses. However it introduced certain issues. The first issue is that the size of the filter state quickly grows as new features are observed. Because the computational cost of the filter scales cubically with the size of the state, the problem can quickly become intractable. The second issue is that “bad features” that are either false matches or violate the rigid-scene assumption due to glare or shadowing effects can corrupt the entire filter state. Iterative batch methods are able to detect these outliers and explicitly remove the associated measurements from the optimization.

Modern real-time SLAM methods attempt to capture the benefits of both batch and filtering techniques\(^4,5,6\). Researchers recognized that images spaced closely in time and position contain lots of redundant information. This motivated a tradeoff between computational cost and accuracy: only consider the vehicle states at a small subset of image times. The subset of images is labeled the keyframe set. A frame can be defined as a keyframe based on the amount of motion since the last keyframe\(^5\). Another option is to define keyframes over a sliding fixed-size window\(^7\). The choice depends on the application: the first is better for a vehicle repeatedly observing a finite environment while the latter is better for continuous exploration of new regions. Both have been shown to be successful in various environments.

There are various modifications to the methods described above to deal with different sensor sets. When using an IMU with a monocular camera, the SLAM algorithms require, at the time of each image, a set of pixel measurements and a data structure indicating which landmark each measurement is associated with. This requires a front-end vision module to solve the correspondence problem: to extract features from an image and match them to landmarks in a database. This has been one of the great challenges to practical and robust SLAM systems and is especially challenging for many space applications.

There are an enormous number of proposed solutions for solving the correspondence problem in the literature\(^8,9,10\). Nearly all of them work in the following way. First an image is processed to detect local maxima in a suitably designed feature detection function. This function must ensure that the detected maxima have large spatial gradients along two distinct directions to avoid the so-called aperture problem. Next, local descriptions of the pixel values around the
maxima are extracted from the image. This can simply be a patch of pixel values or some compressed descriptor designed to be insensitive to image motion or illumination changes. Despite all attempts to make local descriptors robust to such changes, it is impossible for a local descriptor from to characterize how a region in a 3D scene will respond to changes in lighting and view. As a result, the correspondence solution will inevitably contain false matches.

The typical approach to deal with false matches in SLAM is to have some kind of outlier detection scheme that preprocesses a measurement set before handing it to the filter. RANSAC and its variants are the most commonly used\textsuperscript{11,12}. While this has been successful in many settings, it may not be suitable for space applications. Consider the image of Itokawa in Figure 1\textsuperscript{*}. The red circles have been added to indicate certain features that will all move in a similar manner due to shadows this is clearly inconsistent with the rigid-scene assumption. Similar issues can be seen with non-Lambertian surfaces. These problems can cause enormous errors in state estimates.

The problem of explicitly dealing with moving shadows and non-Lambertian surfaces has been largely ignored in the literature. This is likely due to the degree of difficulty involved and the fact that many SLAM applications do not have to work in such challenging settings. In this paper, we attempt to take a step towards solving this problem. In contrast to the sparse SLAM approaches, we have developed a dense SLAM algorithm. The map characterizes the full structure and texture of the scene. Like other dense SLAM approaches, we use a co-registered monocular camera and depth camera\textsuperscript{13}. Unlike other dense SLAM approaches, our map enables real-time rendering to enable a direct comparison of mapped and observed visual and depth data. This paper will describe the proposed system.

The rest of the paper is structured as follows. First an overview of a navigation system is given. Then a filter is derived that uses IMU data and “pose measurements” from the independent vision module. Next, a dense mapping system detailed in a separate paper is outlined. Modifications to the system to deal with shadows are discussed\textsuperscript{14}. Then an experimental setup is described and results against ground-truth data are provided. Finally, some concluding remarks on the system and future work are made.

\textbf{METHOD}

\textsuperscript{*} Hayabusa Project Science Data Archive: https://darts.isas.jaxa.jp/pub/planet/darts/hayabusa/amica/20050930/ST_2420708174_v.jpg
The relative navigation system consists of two main components: a vision module and a filter module. The vision module performs mapping and computes corrections to the sensor-to-terrain pose estimate at the time of each image. The filter module propagates the navigation state based on IMU data and performs an update each time the vision module returns a measured pose correction.

In the usual EKF implementation, it is assumed that the update step can use a measurement related to the current state. This is not the case in our system because there is some delay between the time of an image observation and the availability of a pose measurement. Therefore, a reduced order stochastic cloning approach is used. This type of approach can be used anytime the current state has to be updated based on measurements that depend on a previous state.

In this application, stochastic cloning is applied as follows. The filter receives a flag at the time an image is captured. The filter augments the time-evolving state and covariance with the position and attitude at the image time. Then the filter resumes state propagation with IMU data. Because the augmented portion of the state refers to pose parameters at a particular time, it and its covariance are constant. However, the covariance between the augmented and original state parameters does change. The measurement, which only depends on the augmented parameters, can be used to update the full state once it becomes available because the full covariance is computed.

Reference Frames

There are several reference frames of interest for this algorithm. These are shown in Figure 2.

1) IMU frame $m^+$
2) Vision Sensor frame $s^+$
3) Inertial frame $n^+$
4) Terrain frame $t^+$

Preliminaries

The notation $[r_{a/b}]_c$ is a position vector from the origin of frame $b^+$ to the origin of frame $a^+$ in the coordinates of frame $c^+$. Rotation matrices that transform the coordinates of a vector from frame $a^+$ to frame $b^+$ are represented by $R_{b/a}$ so that $[r]_b = R_{b/a} [r]_a$. The transpose of this matrix performs the reverse transformation: $[r]_a = R_{b/a}^T [r]_b$.

The estimate and measurement of a true variable $x$ will be written as $\hat{x}$ and $x$ respectively. Small errors in a rotation matrix estimate will be approximated by a small angle rotation matrix.
\[ R_{alb} = (I_{3x3} - [\delta \Theta_{alb} \times]) R_{alb} \] where \([x \times]\) is the skew-symmetric matrix formed from the three elements of \(x\). All other errors are expressed as \(x = x + \delta x\).

**Filter**

The filter state consists of the inertial-to-IMU position, velocity, and attitude, and the biases in the accelerometer and gyroscope.

\[
x^T = \left[ \begin{array}{c} r_{m/n} \ 
\dot{r}_{m/n} \\
\beta_a \\
\beta_g \\
q_{m/n} \ 
\dot{q}_{m/n} \end{array} \right]
\]

(1)

**Measurements**

The IMU output is related to the angular rotation and acceleration of the IMU frame relative to the inertial frame as follows.

\[
\omega_m = \left[ \begin{array}{c} \omega_{m/n} \\
\beta_g + w_g \\
\end{array} \right]
\]

(2)

\[
\tilde{a}_m = R_{m/n} \frac{\partial}{\partial t} \left( \left[ \begin{array}{c} \omega_{m/n} \\
\beta_a + w_a \\
\end{array} \right] \right)
\]

(3)

Note that in the above equations \(\beta_g\) and \(\beta_a\) are the bias terms of the gyroscope and accelerometer respectively. These biases are modeled as Gaussian random walk processes with covariance matrices \(\sigma_{wg}^2 I_{3x3}\) and \(\sigma_{wa}^2 I_{3x3}\). The \(w_g\) and \(w_a\) terms are zero mean Gaussian noise variables with covariance matrices \(\sigma_{wg}^2 I_{3x3}\) and \(\sigma_{wa}^2 I_{3x3}\) respectively.

The vision module initializes a terrain fixed reference frame when it is first turned on. This frame has a known position and attitude relative to the inertial frame at initialization time (since it is arbitrary). We assume that the inertial-to-terrain frame transformation is a well-known function of time based on a priori information (rotational period of the body for example).

Once initialized, the vision module uses the current state estimate from the filter to get a prior estimate of the sensor-to-terrain pose:

\[
R_{tls} = R_{tls}^T R_{wls} R_{slm}^T
\]

(4)

\[
\hat{r}_{tls} = R_{tls} \left( \hat{r}_{m/n} + R_{wls} R_{slm}^T r_{tls/n} - r_{tls/n} \right)
\]

(5)

Then the vision module returns a measured correction to these quantities

\[
y^T = \left[ \Delta r_{tls}^T \ 
\Delta \Theta_{tls}^T \right]
\]

(6)

which can be used with the prior estimate to construct a 'measured pose':

\[
R_{tls} = \left( I_{3x3} - \left[ (\Delta \Theta_{tls} \times) \right] \right) R_{tls}
\]

(7)

\[
\left[ \tilde{r}_{tls} \right]_t = \left[ r_{tls} \right]_t + \Delta \tilde{r}_{tls}
\]

(8)
The measured correction $y$ has some error $\delta y^T \equiv [\delta r_{ts}^T \ \delta \theta_{ts}^T]$ with covariance $P_y$. The true terrain-to-sensor pose is related to the measured pose and prior estimate of pose using this error:

$$R_{ts} = (I_{3x3} - [(\delta \theta_{ts}) \times]) R_{ts} = (I_{3x3} - [(\delta \theta_{ts}) \times])(I_{3x3} - [(\Delta \theta_{ts}) \times]) R_{ts}$$

(9)

$$\begin{bmatrix} r_{ts} \\ \theta_{ts} \end{bmatrix} = \begin{bmatrix} \hat{r}_{ts} \\ \hat{\theta}_{ts} \end{bmatrix} + \Delta r_{ts} + \Delta \theta_{ts}$$

(10)

To be clear, the $\Delta r_{ts}$ and $\Delta \theta_{ts}$ are the actual measured corrections while the $\delta r_{ts}$ and $\delta \theta_{ts}$ are the errors in those measured corrections. Note that these errors will depend on the terrain observed at the time of the image. The vision module computes the terrain dependent covariance of these errors $P_y$ and passes it to the filter module along with the actual pose measurements.

Filter Propagation

The propagation of the state estimates takes on a simple form. First define the estimated inertial acceleration and angular rate as:

$$\begin{bmatrix} \hat{a}^k \\ \omega^k \end{bmatrix} = R_{mn}^T \begin{bmatrix} a^k \\ \omega^k \end{bmatrix} + [g]^k$$

(11)

$$\omega_{m/n} = \omega_m - \beta_g^k$$

(12)

Note that $[g]^k$ is an estimate of local gravity in the inertial frame. Then the state propagation is:

$$\begin{bmatrix} \hat{r}^{k+1} \\ \nu^{k+1} \\ \beta^{k+1} \\ q^{k+1} \end{bmatrix} = \Omega \begin{bmatrix} \omega_{m/n}^{k+1} \\ \nu_{m/n}^{k+1} \end{bmatrix}$$

(16)

$$\beta^{k+1} = \beta^k$$

(15)

The rotational kinematics in Equation (16) uses the definitions below.

$$\psi(\omega) = \frac{\sin(\frac{1}{2} \Vert \omega \Vert \Delta t)}{\Vert \omega \Vert}$$

(18)

$$\Omega(\omega) = \begin{bmatrix} \cos(\frac{1}{2} \Vert \omega \Vert \Delta t) & -\psi(\omega)^T \\ \psi(\omega) & \cos(\frac{1}{2} \Vert \omega \Vert \Delta t) I_{3x3} - [\psi(\omega) \times] \end{bmatrix}$$

(19)

The state covariance propagates as
\[ P_{s}^{k+1} = \Phi_{k} P_{s}^{k} \Phi_{k}^{T} + Y_{k} Q Y_{k}^{T} \]  
(20)

where the following quantities are used.

\[
\Phi_{k} = \begin{bmatrix}
I_{3 \times 3} & \Delta t I_{3 \times 3} & -\frac{1}{2} \Delta t R_{m/n}^{T} & -\frac{1}{2} \Delta t B & 0_{3 \times 3} \\
0_{3 \times 3} & I_{3 \times 3} & -\Delta t R_{m/n} & \Delta B & 0_{3 \times 3} \\
0_{3 \times 3} & 0_{3 \times 3} & I_{3 \times 3} & -\Delta t I_{3 \times 3} \\
0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & I_{3 \times 3}
\end{bmatrix}
\]  
(21)

\[
Y_{k} = \begin{bmatrix}
-\frac{1}{2} \Delta t R_{m/n}^{T} & 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \\
-\Delta t R_{m/n} & 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \\
0_{3 \times 3} & \sqrt{\Delta t} I_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \\
0_{3 \times 3} & 0_{3 \times 3} & \Delta t I_{3 \times 3} & \sqrt{\Delta t} I_{3 \times 3}
\end{bmatrix}
\]  
(22)

\[
Q_{k} = \begin{bmatrix}
\sigma_{w}^{2} I_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \\
0_{3 \times 3} & \sigma_{w}^{2} I_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} \\
0_{3 \times 3} & 0_{3 \times 3} & \sigma_{w}^{2} I_{3 \times 3} & 0_{3 \times 3} \\
0_{3 \times 3} & 0_{3 \times 3} & 0_{3 \times 3} & \sigma_{w}^{2} I_{3 \times 3}
\end{bmatrix}
\]  
(23)

Note that the \( B \) matrix used in (21) is

\[
B = -R_{m/n}^{T} \left( \bar{a}_{m} - \bar{B}_{s}^{T} \right) \times
\]  
(24)

**State Augmentation**

In order to deal with the measurement delay, the filter state is augmented using the stochastic cloning approach. For each image capture, the vision module sends a flag to the filter module indicating an image observation at the current time. The state then becomes

\[
\begin{bmatrix}
\tilde{x}_{a}^{k} \\
\tilde{r}_{m/n}^{k} \\
\tilde{v}_{m/n}^{k} \\
\tilde{B}_{s}^{k} \\
\tilde{q}_{m/n}^{k} \\
\tilde{r}_{g}^{k} \\
\tilde{j}_{m/n}^{k} \\
\tilde{j}_{g}^{k}
\end{bmatrix}^{T} = \begin{bmatrix}
\tilde{r}_{m/n}^{T} \\
\tilde{v}_{m/n}^{T} \\
\tilde{B}_{s}^{T} \\
\tilde{q}_{m/n}^{T} \\
\tilde{r}_{g}^{T} \\
\tilde{j}_{m/n}^{T} \\
\tilde{j}_{g}^{T}
\end{bmatrix}
\]  
(25)

Note the superscripts on the variables to indicate the time dependence. At the time of image capture and state augmentation \( k = i \). Then once state propagation continues, \( k > i \) as the time index \( k \) continues to be incremented while the \( i \) is fixed.

This augmentation requires a modification to the covariance.

\[
P_{s}^{k} \leftarrow \begin{bmatrix}
I_{15 \times 15} \\
I_{3 \times 3} & 0_{3 \times 12} \\
0_{3 \times 9} & I_{3 \times 3}
\end{bmatrix} P_{s}^{k} \begin{bmatrix}
I_{15 \times 15} \\
I_{3 \times 3} & 0_{3 \times 12} \\
0_{3 \times 9} & I_{3 \times 3}
\end{bmatrix}^{T}
\]  
(26)
Note that Equation (26) expands the covariance from a 15x15 matrix to a 21x21 matrix. The new parts of the matrix are filled in with the appropriate parts from the old matrix. Then in the propagation equations, $\Phi_k$ and $Y_k$ defined in Equation (21) and (22) are replaced by

$$
\Phi_k \leftarrow \begin{bmatrix}
\Phi_k & 0_{15x6} \\
0_{6x15} & I_{6x6}
\end{bmatrix}
$$

$$
Y_k \leftarrow \begin{bmatrix}
Y_k \\
0_{6x12}
\end{bmatrix}
$$

**Measurement Update**

The measured correction of terrain-to-sensor pose is $y^T = [\Delta \tilde{r}_{sl} \ \Delta \theta_{tls}]$ which has an error covariance of $P_y$. The first step in the measurement update at time $k$ is to relate the measured corrections to errors in the inertial-to-IMU pose at the time $i < k$ of the image. To do so, replace the true quantities in Equations (9) and (10) with the estimated quantities and error terms to obtain:

$$
R_{lm} R_{ws}^T \begin{bmatrix}
I_{3x3} + \left( \left( \delta \theta_{mln} \right) \times \right) \\
0_{3x3} + \left( \left( \delta \theta_{tls} \right) \times \right)
\end{bmatrix} R_{slm}^T = \begin{bmatrix}
I_{3x3} - \left( \left( \delta \theta_{tlm} \right) \times \right) \\
I_{3x3} - \left( \left( \delta \theta_{tls} \right) \times \right)
\end{bmatrix}
$$

$$
R_{lm} \left( \hat{r}_{ml} \right) + \delta r_{ml} = R_{lm} \left( \hat{r}_{ml} \right) + R_{ws}^T \begin{bmatrix}
I_{3x3} + \left( \left( \delta \theta_{mln} \right) \times \right) \\
0_{3x3} + \left( \left( \delta \theta_{tls} \right) \times \right)
\end{bmatrix} R_{slm}^T \left[ r_{slm} \right]_n - \left[ r_{slm} \right]_n
$$

Now we can rearrange, cancel terms using Equations (4) and (5), and keep only first order error terms to obtain:

$$
\left( \left( \delta \theta_{tls} \right) \times \right) = -R_{slm} \left( \left( \delta \theta_{mln} \right) \times \right) R_{slm}^T - \left( \left( \delta \theta_{tls} \right) \times \right)
$$

$$
\Delta r_{tls} = R_{lm} \left( \delta r_{mln} + R_{ws}^T \left( \left( \delta \theta_{mln} \right) \times \right) R_{slm}^T \left[ r_{slm} \right]_n \right) - \delta r_{tls}
$$

Note that Equation (31) has a 3x3 matrix on both sides. The first term on the right hand side can be interpreted as the transformation of a matrix by $R_{slm}$. Therefore the analogous 3x1 vector form of the equation is

$$
\Delta \theta_{tls} = -R_{slm} \left( \delta \theta_{mln} \right) - \left( \delta \theta_{tls} \right)
$$

The next step is to get the sensitivity matrix that relates the measurement errors to the state errors. This is done by taking the partial derivative of $\delta y$ shown in Equations (32) and (33) with respect to errors in the augmented state $\delta x_a$. The result is

$$
H = \begin{bmatrix}
0_{3x15} & R_{lm} \left( \delta r_{mln} + R_{ws}^T \left( \left( \delta \theta_{mln} \right) \times \right) R_{slm}^T \left[ r_{slm} \right]_n \right)
\end{bmatrix}
$$

$$
- R_{slm}
$$

$$
\left( \begin{array}{c}
0_{3x15} \\
0_{3x3}
\end{array} \right)
$$

$$
\left( \begin{array}{c}
0_{3x3} \\
- R_{slm}
\end{array} \right)
$$

$$
\left( \begin{array}{c}
0_{3x3} \\
- R_{slm}
\end{array} \right)
$$
Note that the measurement depends only on the augmented part of the state which leads to the 6x15 zero matrix on the left-hand side of $H$.

With all the above definitions, the usual EKF update equation can be used:

$$K = P_x^k H^T \left( H P_x^k H^T + P_y \right)^{-1}$$  \hspace{1cm} (35)

$$P_x^{k'} \leftarrow \left( I_{21x21} - KH \right) P_x^k$$  \hspace{1cm} (36)

$$\delta x_o = Ky$$  \hspace{1cm} (37)

Then the correction term $\delta x_o$ can be applied to the state estimate.

**Vision Module**

The vision module is covered in detail in a second paper$^{14}$. It is an extension of the work demonstrated by Microsoft in KinectFusion$^{13}$. A brief outline of the system is given here.

The vision module estimates a Signed Distance Function (SDF) and albedo model of the terrain. The SDF is a scalar function of 3D space defined as the distance from a particular point in space to the nearest surface. Solid space is assigned negative values while free space is assigned positive values. A simple 2D cross section is shown in Figure 3 for illustrative purposes. The algorithm maintains an estimate of the SDF and albedo over a fixed 3D grid of voxels (representing some piece of terrain).

**Figure 3:** A 2D cross-section of a simple SDF example. The black box is a solid surface.

**Figure 4:** Feature matching between rendered image (left) and observed image (right).
If an accurate estimate of the scene’s SDF and albedo grid is available, the scene can be rendered from arbitrary viewpoints: a synthetic depth and grayscale image is created. The rendering is essentially an intersection problem between a ray through each pixel of the virtual sensor and the terrain surface. The vision module uses an *a priori* estimate of the sensor pose to render the scene. The rendered images are then compared to the sensor images to determine the difference in pose between the two images. Visual features are first extracted from the grayscale images. Then OLTAE and RANSAC are used to determine the difference in pose. An example of the feature matching is shown in Figure 4. Iterative Closest Point (ICP) alignment with a point-to-plane error metric is used to determine the pose difference between the depth images. The computed pose differences are passed on to the filter so that a state update can occur.

The vision module then uses the updated state and observed data to perform a map update. With the position and attitude of the camera known, each pixel’s depth and direction are used to trace rays from the camera to the observed surface through the map. The SDF of each voxel along all rays is updated. The albedo values at voxels near the surface are also updated.

One extension to the rendering model currently in development is a shadow generator. The first pass of rendering intersects rays from the sensor with the solid terrain surface. A second pass then traces rays from these terrain intersection towards the sun. If an intersection with the terrain is detected, then a shadow is present. An example image rendered with and without shadows is seen in Figure 5. This type of rendering can improve both the number of quality of visual features used in pose estimation.

**EXPERIMENTS**

The system is implemented in C++ and CUDA (for parallel computations on a GPU) and run on a consumer gaming laptop equipped with Intel i7 processors and an Nvidia GTX 780M graphics card. The Microsoft Kinect RGB-D sensor and a VectorNav VN100 IMU are used for measurement updates and state propagation respectively. A 6-DOF motion emulation robot is used to move the camera and IMU around a 6.5 m² mock terrain target.

The Vicon motion capture system is used to record the position and attitude of the sensor during run-time to obtain ground truth data. This data was compared to the system output to obtain the plots below. The results of a sample run are shown in Figure 6 for an EDL-like trajectory. The top plots show the position and attitude estimates during a run. The sensor under goes a coning

![Figure 5: An example of terrain rendered without shadows (left) and with shadows (right) for a light source in the 'up' direction.](image_url)
motion as it is translated towards the terrain. The estimation errors are shown in the bottom plots. The position error is kept to within ±2 cm and attitude error is kept to within ±1 degree. Note that the high frequency oscillations in the error plot are mainly due to noise in the Vicon system. A longer sequence of constant motion is shown in Figure 7.

These results are for the system running without shadow generation. In nearly all space applications, an accurate inertial attitude estimate and at least a coarse inertial position estimate will be available. This would allow the direction to the sun to be computed for rendering. To test this in the lab, we would need to know the position of the light source as it moves. Unfortunately this is difficult to achieve because the light saturates the Vicon sensors near the Vicon beacons on the light. Nevertheless, the tested system demonstrated very low error drift even on longer runs. This is due to the frame-to-map tracking. Images are compared directly to the map which avoids the daisy-chaining of error normally associated with frame-to-frame tracking.

CONCLUSION

A navigation system for dense mapping was presented. The system was implemented and achieved real-time performance at 10-15 frames per second for a map containing over 5 million voxels. Testing with real sensor data in the laboratory verified the system accuracy to within 1 degree in attitude and a few centimeters in position. The tested portion did not explicitly render the shadows. This is because the direction to the light source could not be determined using the Vicon system. In practice, a spacecraft with a star tracker for inertial attitude and even a very coarse position estimate would be able to determine the sun direction for shadow rendering.

Nevertheless, the system enables real-time rendering to handle occlusion and shadows as seen in some of the sample images above. This is because the full scene structure and texture is captured in a dense map unlike the sparse SLAM methods. This prevents filter degradation that sparse local descriptors can cause by incorrect matches or improper handling of occlusion, especially in a space environment.

![Figure 6](image.jpg)

**Figure 6:** Results of a sample run during an EDL-like trajectory. Position estimates (top left), attitude estimates (top right), position error (bottom left), and attitude error (bottom right).
The system comes with one major drawback: large computational cost. Although a real-time implementation was achieved, it was done so with a high end laptop and graphics card: not currently an option for spacecraft. Nevertheless, the paper demonstrates the potential for large improvements in robustness and accuracy if such a processor was available. There has been some effort in recent years to develop parallel processors for space applications and it is the hope of the authors that this paper will provide additional motivation for such efforts\textsuperscript{20}.

Future work by the authors will investigate alternative strategies for dealing with difficult viewing conditions. This will likely include tailoring local descriptors to specific space applications like entry, descent, and landing. The appearance of shadows is typically characterized by a very sharp boundary between light and very dark regions. Image processing algorithms may be designed to detect and ignore such feature types.

ACKNOWLEDGMENTS

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

REFERENCES


Figure 7: Navigation errors for constant motion over a 300 second sequence


