Summer Research: SPEARS 3D Microscope *

Pranav Konda

1 Abstract

Implementation and benchmarking of 3D reconstruction algorithms was done in C++ in order to drive a solid-state microscope capable of returning 3D topographic maps of planetary surfaces at a granular level. Using a support vector machine, a data quality evaluation program was created, using image sharpness metrics, in order to tag images that should not be analyzed in the microscope pipeline. Progress is being made to analyze the sharpness of regions within images to disregard their reconstruction output. Furthermore, image segmentation is being explored, in order to identify interesting objects in soil regolith and on planetary surfaces.

2 Motivation

Microscopy is a valuable tool for gathering data across planetary bodies, whether that be on the Earth or another extraterrestrial body. Due to its high resolution, microscopy is particularly suited to analyze the fine details of the surfaces, which often yields very insightful data on planetary surface composition. Microscopes in space are not uncommon, such as the Mars Hand Lens Imager aboard the Curiosity rover. [3]

However, such devices do not have the resolution required to image individual soil granules and regolith. Furthermore, the rovers they are placed on are often too large to reach certain environments. This project proposes a high-resolution miniature solid state 3D microscope, with capabilities to reconstruct three dimensional topographic maps of planetary surfaces from image data. This device is targeted for next generation versatile small robotic platforms, such as a CubeRover, which can reach places previously unreachable. It is also lightweight, weighing 150 grams. [3]

Due to the dangers of these areas, a robust device is needed to withstand the harsh natural conditions of these environments. This microscope has no moving parts, making it solid state, and has one optical pathway. A transparent liquid crystal display functions as a programmable aperture, while a constellation of LEDs allows for the incident illumination to be controlled at various angles. [3]

Using a combination of light field imaging (multiview stereo reconstruction) and photometric stereo (shake from shading algorithms), the device can provide both reflectance data and topographic data of the environment. [3]

The device uses a Rock Pi S embedded computer, which measures around 1.6in on all sides. This specific device is remarkable in its low power usage, but this comes at a cost of performance. This requires all algorithms to be optimized for this embedded system.

3 Reconstruction Algorithms

3.1 Methods and Frameworks

To accomplish the aforementioned photometric stereo and light field imaging, various reconstruction algorithms were implemented to return a 3D reconstruction of the surface from an image stack. However, these were implemented using MATLAB, which is not ideal for the embedded hardware the device targets. Rewriting the prototype code in a more performant framework and language was necessary, while attempting to preserve as much functionality as possible.

C++ was chosen as the primary language for development due to its high performance and extensive library support. To emulate the image processing and computer vision functions of MATLAB, OpenCV was chosen both for its ease of use and high performance. MATLAB is notable in its powerful linear algebra functions, and to emulate this Eigen was used, due to it being very lightweight (as it is completely static, only template code in its headers) and for its ease of use with OpenCV.

3.2 Algorithms

Two primary algorithms were used for reconstruction, a multiview stereo algorithm and a photometric stereo (shake from shading) algorithm. The photometric stereo algorithm functions better on a smooth monotone surface, while the multi-view stereo algorithm functions better on rough and heterogeneous surfaces. Employing both of these algorithms, and combining their output into a final stack ensures a generally more accurate reproduction.

3.2.1 Multiview Stereo

This algorithm is based on processing different images taken of the same scene at different angles to create a height map. A multiview stereo correlation is computed to reconstruct the height of the sample, using multiple image pairs. These pairs have a shift which can be calculated according to the parallax shift p, where

$$p = h\frac{a}{f_0},$$

where h is the object height, a is the aperture distance off axis, and f_0 is the objective lens focal length. [3] Once this shift is

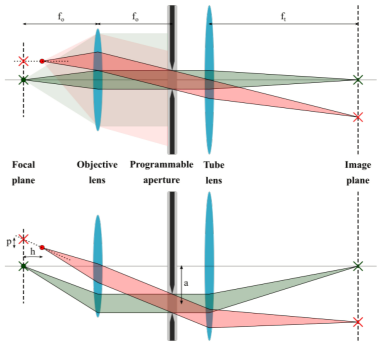

Figure 1: Calculation of the parallax shift.

finished, the **Normalized Cross Correlation** (NCC) between the images in the pair is calculated for a small window centered around each corresponding pixel in the image. For windows with pixel values of a_i and b_i, the NCC is calculated as

$$NCC(a,b) = \frac{\sum_i (a_i - \overline{a})(b_i - \overline{b})}{\sqrt{\sum_i (a_i - \overline{a})^2 \sum_i (b_i - \overline{b})^2}}, \quad (1)$$

*NASA Ames Research Center, Mountain View, CA

where a bar denotes the mean. For each possible height, the candidate height with the greatest mean NCC is chosen as the height for that pixel. [3] No exclusive library functions were used

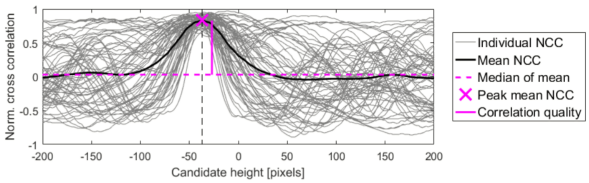

Figure 2: Calculation of the candidate height from greatest mean NCC.

when rewriting this code, meaning there are no performance discrepancies in the prototype implementation and the C++ implementation down to floating point accuracy.

3.3 Performance and Benchmarking

The Shake from Shading (photometric stereo) has not finished being ported, so no comments can be made on accuracy or time execution. Most of the general computation time is spent on that algorithm and the multiview stereo algorithm. Proper full suite benchmarks have not been conducted yet, but from basic observation of compile times and execution times the speed of execution has improved, as expected. No excessive memory usage has been detected yet, which could be a potential problem on the embedded hardware if detected later. We are flexible in what computation could be sent back to Earth via downlinking, so a specific benchmark does not have that much importance yet for the embedded hardware, but this should be determined soon.

In the future, proper benchmarks of the multiview stereo algorithm should be conducted on both benchtop and embedded hardware to get a detailed sense of execution times. Progress should be made on rewriting the photometric stereo algorithm as well, enabling the combining process to be implemented and a full stack benchmark to be conducted.

4 Data Quality and Segmentation

4.1 Motivation

Image and data processing is very relevant for measuring data quality, and attempting to classify what the reconstruction output actually is. Data quality is important as execution time should not be wasted on faulty data, as this produces output that will not be useful. Material segmentation of the reconstruction output, and by extension classification enables us to see the boundaries between objects, and get a classification of what each object is. This is of particular use to planetary scientists as it helps them distinguish between critical objects (for instance, reflective or crystalline material) to analyze and regular regolith and surface soil.

4.2 Methods

As these were prototypes of a more robust future system, Python was chosen due to its extensive library support and ease of implementation. Once again, OpenCV was used, along with Numpy for linear algebra and Scikit-Learn for machine learning models. The OpenCV implementation in Python, however, is simply a wrapper for the C ABI underneath, so it is performant enough to be used in production.

4.3 Blur Detection

One primary way to control data quality is blur detection: checking whether the input images from the microscope are blurry and out of focus. Such an image would waste computation power if it were to be constructed, as the lack of fine resolution would cause strange and useless outputs from the reconstruction algorithms.

4.3.1 Metrics

Two metrics were used, the **Laplacian Variance** and the **Mean Gradient Magnitude**.

- **Laplacian Variance**: We can convolve the image with a discrete Laplacian operator, and find the variance [2]:

$$\phi = \mathrm{Var}\left(\Delta[x, y]\right),$$

where

$$\Delta = I \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}.$$

I denotes the image.

- **Mean Gradient Magnitude**: This is also known as the Tenengrad method. We can calculate the mean gradient magnitude, where the gradient is computed by the norm of the Sobel filters [2]:

$$\phi = \frac{1}{n} \sum_{x,y} \sqrt{\nabla_x[x, y]^2 + \nabla_y[x, y]^2}$$

where

$$\nabla_x = I \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

and

$$\nabla_y = I \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}.$$

4.3.2 Training and Model Selection

A public dataset [4] was used, containing 10,000 scenes with 2 images each, one in focus and one out of focus. A Python script was used to calculate the values of the metrics for each image. The data was then plotted below:

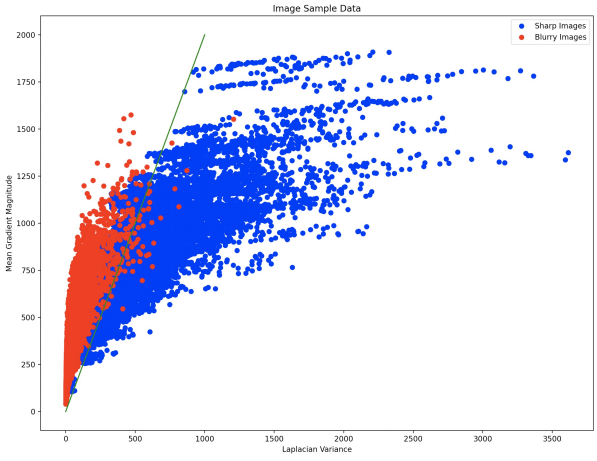

Figure 3: Plot of the data, where blue points are sharp images, and red points are blurred images.

A support vector machine was chosen as a way to classify the images into one of two classes: sharp or blurred. This was implemented in Python using the `scikit-learn.SVC()` object. A radial basis function (RBF) was used as the kernel. A γ value of 10^{-1} was used along with a C value of 10^{0}. To handle the data, `numpy` was used while `opencv` was used to actually compute the Laplacian Variance and Mean Gradient Magnitude.

4.3.3 Testing

Another public dataset [1], consisting of 1050 images, was used for unit testing. From these, 350 sharp and 350 blurred images were chosen. The images were preprocessed by the same script that the training data was processed by, stored in a file, and tested using the Support Vector Machine, which was dumped as an object and easy to access. A short `C++` script was created to test the expected values against the observed values, and report the total accuracy along with a side-by-side comparison.

4.3.4 Biasing

Initially, the model did not report results in acceptable error ranges: many images were being tagged as sharp when they were actually blurry, a Type II error, which was undesirable. To solve this, a biasing technique was used. By scaling the training data values by a factor of 5, the distinction between the space of blurry images and the space of sharp images was widened. This allows the Support Vector Machine to be more accurate in its classification. By using this technique, the model became more accurate and had less false negatives.

4.3.5 Results

A total of 23 differences were measured, yielding a total accuracy of $\frac{677}{700} \implies \approx \boxed{96.7\%}$ across the test data. Here we take positive to be an image that is identified as blurred:

		Predicted	
		Positive	Negative
Actual	Positive	337	13
	Negative	10	340

Table 1: Confusion matrix for the test data.

4.3.6 Future Progress

Proper benchmarking must be done on the embedded hardware in order to determine whether further optimization is necessary. Work is being done to shift the algorithm from an entire image classifier to one which would determine blurred *regions* of images, the currently proposed method for this is by a sliding window algorithm where such regions could be identified by metric scores.

Exploration is also being made in the realm of image segmentation, in order to identify interesting objects within the data. One proposed method is **Mask R-CNN**, which requires a custom dataset to implement. This process is time consuming as boundaries must be manually traced.

5 References

References

[1] Aleksey Alekseev. *Blur Dataset*. Online. Can be found here: https://github.com/Kwentar/blur$_d$*ataset*. 2019.

[2] Charles Hermann et al. "Learning to Autofocus". In: (2020). DOI: `https://arxiv.org/abs/2004.12260v3`.

[3] Gustav M. Petterson et al. "Miniature 3D Microscope and Reflectometer for Space Exploration". In: *IEEE International Conference on Computational Photography* (2019). DOI: `https://doi.org/10.1109/ICCPHOT.2019.8747331`.

[4] Jaesung Rim et al. "Real-World Blur Dataset for Learning and Benchmarking Deblurring Algorithms". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2020.