Planetary Multi-Photogrammetry and its Visualization

Kyle D. Husmann
khusmann@calpoly.edu
California Polytechnic State University, San Luis Obispo, CA, USA

March 11, 2011

Abstract

During the summer of 2010, Dr. Taemin Kim and Kyle Husmann jointly developed a novel new algorithm to generate topographical maps of planetary surfaces using a multiple-view photogrammetry approach. To evaluate and debug the algorithm, a program called `mvpgui` was developed to visualize and examine the objective function. After verifying the correct operation of the algorithm and `mvpgui` with synthetic data, it is run with real orbital data from the Apollo 15 mission. The results confirm the validity of the multiple-view algorithm as well as demonstrate the value of objective function visualization.

1 Introduction

Topographical maps are an essential tool for scientists interested in exploring and learning more about planetary bodies like the moon or mars. These maps allow scientists do everything from identifying geological phenomena to identifying potential landing sites for probes or spacecraft.

Satellites and other spacecraft that visit planetary bodies of interest are usually equipped with a variety of sensors, some of which can be used to recover the topography of the planetary surface. LiDAR (Light Detection And Ranging) sensors give sparse (but highly accurate) measurements at periodic points called "posts". Raw images that are captured as the satellite orbits the planetary body can also be processed to create highly detailed topographical maps. By registering and aligning these two data sets, maps can be created that are both dense accurate. The methods of processing orbital images into topographical maps are the topic of this paper.

Given two images of the same scene taken from slightly different perspectives, the relative shift of objects from frame to frame (known as "disparity") is related to distance of the object: far objects appear to move less than close objects. This phenomena should be familiar, since the human brain uses this relationship to create depth perception from the differences in perspective seen by both eyes. Similarly, depth information from orbital imagery can be recovered in areas where the images overlap by matching points in the images that correspond to the same 3D location and measuring their disparity. By using the disparity along with the location and orientation of the spacecraft as well as a mathematical model for the lens of the camera, the precise location of the 3D point can be calculated. This technique of using matching points between images to calculate 3D locations is known as "stereophotogrammetry".

Before computers automated this task, points in orbital images were manually matched using mechanical stereoplotters like the one shown in figure 1. Now, computers can perform this once long and arduous process with minimal human interaction. Software suites like the Ames Stereo Pipeline [2, 5] can be fed orbital images and will simply output the topography in the form of digital elevation maps (DEMs).

While stereophotogrammetry produces good results, we are interested in going beyond the usual two views used in the stereo reconstruction process and generalizing the process to use all views available in the scene being reconstructed. By moving to a multiple-view approach, we will be able to increase the accuracy of the DEMs produced while at the same time robustly rejecting outliers in the source imagery.

During the summer of 2010, Dr. Taemin Kim[1] and I jointly developed a novel multiple-view algorithm. It works by approximating very small sections of the planetary surface as planar patches and adjusting the

[1]Intelligent Robotics Group, NASA Ames Research Center, CA, USA

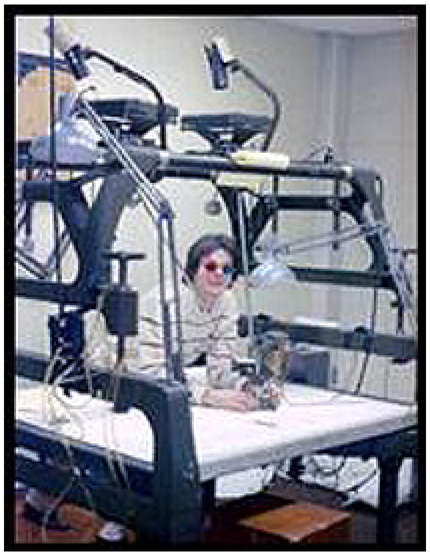


Figure 1: A Kelsh stereoplotter (source: http://en.wikipedia.org/wiki/File:Kelsh.jpg)

height and orientation of the patches to minimize the alignment error when the orbital images are projected on to them. As with any minimization process, local minima and stability are concerns. Therefore, to evaluate and debug the objective function, (the alignment error function being minimized), I have created a small program called "`mvpgui`" to help visualize its landscape and navigate its parameters.

Before going into detail about this new multiple view algorithm and presenting some results obtained with `mvpgui`, this paper will give some background on cartography to give context for the multiple-view algorithm.

2 Background

Before discussing the multiple-view algorithm in this paper, some basic concepts in cartography must be introduced.

2.1 Map Projections

Topographical maps are typically stored and distributed as DEMs (Digital Elevation Maps). DEMs are two-dimensional, single-channel digital images of the planetary surface. Rather than each pixel representing an illumination value, as with typical images, in a DEM each pixel represents a height at that location or "post".

In order to store a curved, three-dimensional surface of a planetary body as a two-dimensional plane, one must develop some sort of relationship between points on the surface and points on the plane. In cartography, this is known as a "map projection": a function that maps locations on a planet's surface to locations on the planar map.

It is a well known fact that it is impossible to project the curved surface of a spheroid onto a plane without some kind of distortion. Depending on the map projection being used, the perceived distance, direction, area or the shape of features on the map may be distorted or preserved [1]. For example, the Mercator projection (figure 2) preserves area but heavily distorts shapes, while the sinusoidal projection (figure 3) preserves small shapes but distorts areas.

2.1.1 The Equirectangular Projection

The equirectangular projection (also known as plate carrée) is a common representation used in digital cartography. Although it preserves neither distance, direction, area nor the shape of features in the map, it

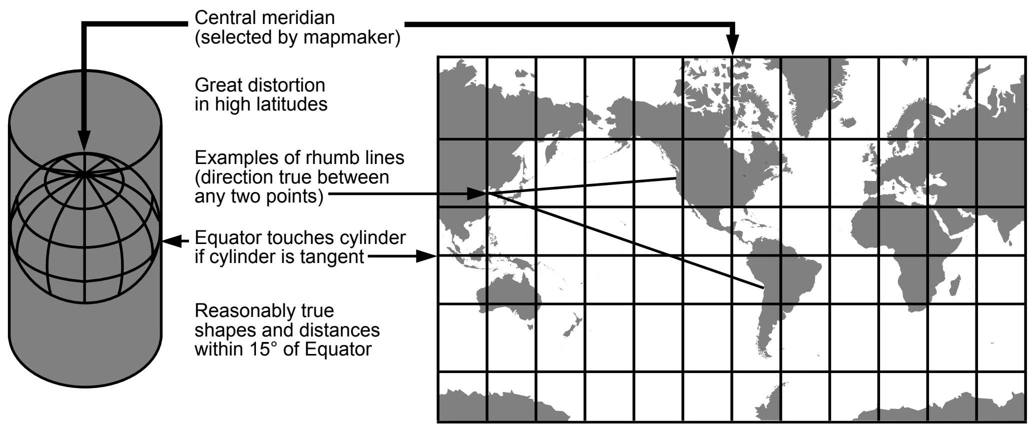


Figure 2: The Mercator projection accurately depicts small shapes but distorts areas. (Source: `http://egsc.usgs.gov/isb/pubs/MapProjections/graphics/sinousidal.gif`)

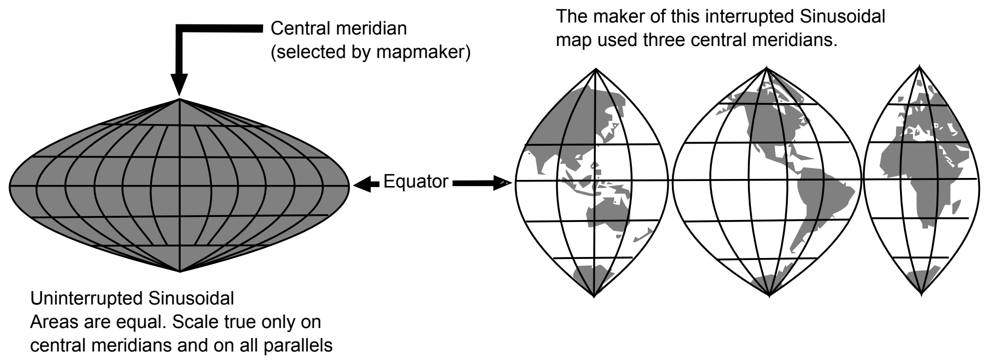

Central meridian
(selected by mapmaker)
The maker of this interrupted Sinusoidal
map used three central meridians.
Equator
Uninterrupted Sinusoidal
Areas are equal. Scale true only on
central meridians and on all parallels

Figure 3: The sinusoidal projection accurately depicts relative sizes, but distorts shapes and directions. (Source: `http://egsc.usgs.gov/isb/pubs/MapProjections/graphics/mercator.gif`)

has a computational advantage: The longitude and latitude $\boldsymbol{\theta} = (\theta, \varphi)$ of a location[2] on the map is related to the coordinates $\mathbf{z} = (c, r)$ of the pixel representing that location by a simple affine relationship:

$$\begin{bmatrix} \boldsymbol{\theta} \\ 1 \end{bmatrix} = \mathbf{S} \begin{bmatrix} \mathbf{z} \\ 1 \end{bmatrix} \tag{1}$$

where

$$\mathbf{S} = \begin{bmatrix} s_c & 0 & t_c \\ 0 & -s_r & t_r \\ 0 & 0 & 1 \end{bmatrix} \tag{2}$$

s_u and s_v are the c and r scale of the map in degrees per pixel, and t_c and t_r are the longitude and latitude of the upper-left corner of the map. Note that equation (1) is a homogeneous equation, as described in [6].

This projection has the highest amount of distortion when used near the poles of the planetary surface. In situations where maps of the poles are needed, a different projection should be used.

2.1.2 Moving From Map To Cartesian Coordinates

It is a simple procedure to calculate Cartesian xyz locations on a planetary surface given $D(\mathbf{z})$, a DEM, and its map projection (stored in a data structure called a "georeference"). For a pixel on the DEM $\mathbf{z} = (c, r)$, the longitude and latitude $\boldsymbol{\theta} = (\theta, \varphi)$ at that point are calculated using equation (1). Then, let

$$\hat{\mathbf{e}} = \begin{bmatrix} \cos\theta\cos\varphi \\ \sin\theta\cos\varphi \\ \sin\varphi \end{bmatrix} \tag{3}$$

be the direction vector in Cartesian coordinates pointing from the origin longitude and latitude in question. Now, using the radius of the planetary body's surface at this point, $D(\mathbf{z})$, the Cartesian coordinate $\mathbf{x} = (x, y, z)$ can be calculated:

$$\mathbf{x} = D(\mathbf{z})\,\hat{\mathbf{e}} \tag{4}$$

When calculating the Cartesian coordinates of points in a DEM around a small patch of pixels centered around a given pixel $\mathbf{z}'$, the above trigonometric functions can be approximated by their first order Taylor approximations. First, let

$$\frac{\partial \mathbf{e}}{\partial \mathbf{x}} = \begin{bmatrix} -\sin\theta'\cos\varphi' & \cos\theta'\sin\varphi' \\ \cos\theta'\cos\varphi' & -\sin\theta'\sin\varphi' \\ 0 & \cos\varphi' \end{bmatrix} \tag{5}$$

where $\boldsymbol{\theta}' = (\theta', \varphi')$ is the longitude and latitude for $\mathbf{z}'$ found using equation (1). Then, using the Taylor approximation of equation (3) centered at θ' (equivalently $\hat{\mathbf{e}}'$), we have an affine mapping between $\hat{\mathbf{e}}$ and $\boldsymbol{\theta}$:

$$\begin{bmatrix} \hat{\mathbf{e}} \\ 1 \end{bmatrix} = \mathbf{L} \begin{bmatrix} \theta \\ 1 \end{bmatrix} \tag{6}$$

where

$$\mathbf{L} = \begin{bmatrix} \frac{\partial \mathbf{e}}{\partial \mathbf{x}} & \hat{\mathbf{e}}' - \frac{\partial \mathbf{e}}{\partial \mathbf{x}}\theta' \\ \mathbf{0} & 1 \end{bmatrix} \tag{7}$$

Using this result, can now easily map a pixel $\mathbf{z}$ to its Cartesian location $\mathbf{x}$:

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = D(\mathbf{z}) \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{LS} \begin{bmatrix} \mathbf{z} \\ 1 \end{bmatrix} \tag{8}$$

This relationship very is important performance-wise when using small patches of the DEM to create orthoimages (defined in the next section). This performance gain is essential for the multiple-view algorithm described later in this paper because the algorithm generates many orthoimage patches at each iteration.

[2]Typically in cartography, coordinates are represented as (latitude, longitude) rather than (longitude, latitude). However, we choose the latter because it mirrors the representation of pixels in an image: (c, r).

2.2 Orthoimages

After a DEM is generated, the source imagery can be projected onto the DEM to give it texture. This resulting image is called an orthoimage, because it appears as if the camera that took the image was orthographic to the planetary body's surface.

To do this, a mapping between the orbital image coordinates $\mathbf{u} = (u, v)$ and world coordinates $\mathbf{x} = (x, y, z)$ is needed. For a pinhole camera, this mapping is given by its fundamental matrix (see [6]):

$$\begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} = \mathbf{F} \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{9}$$

Then, using the approximation from the last section, a orbital image $O(\mathbf{u})$ can be mapped onto a small patch centered at $\mathbf{z}'$ on the DEM given by $D(\mathbf{z})$. The resulting orthoimage patch $I(\mathbf{z})$, is given by:

$$I(\mathbf{z}) = O\left(\mathbf{F}D(\mathbf{z}) \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{LS} \begin{bmatrix} \mathbf{z} \\ 1 \end{bmatrix}\right) \tag{10}$$

3 Multiple-View Photogrammetry

The multiple-view photogrammetry method jointly developed by Dr. Taemin Kim and I is described in this section.

3.1 Overview

Say the terrain of the planetary body is represented as the DEM $D(\mathbf{z})$. Projecting all the available orbital imagery on a small patch of this DEM yields orthoimages $\tilde{I}_1(\mathbf{z}), \tilde{I}_2(\mathbf{z}), \ldots, \tilde{I}_n(\mathbf{z})$. If the DEM exactly represents the planetary surface, the orthoimages should line up exactly (neglecting occlusions, shadows and other artifacts). Error in the DEM will manifest as misalignment in the orthoimage patches.

If the DEM patches are small, we can approximate the curved planetary surface that the patch represents with a plane tangent to the surface at the center point of the patch (see figure [FIG]). Our algorithm attempts to find this plane for each DEM post by adjusting the plane's position and orientation until the alignment error is minimized. The plane's height at the DEM post in question becomes the value for that DEM post.

3.2 Efficiently Generating Orthoimages

For a given DEM post $\mathbf{z}'$, we parameterize the plane passing through it in terms of its height h at the DEM post and its normal in spherical coordinates (n_θ, n_φ). With the surface defined with these parameters, it is easy to create the orthoimages $\tilde{I}_1(\mathbf{z}), \tilde{I}_2(\mathbf{z}), \ldots, \tilde{I}_n(\mathbf{z})$ from their the orbital imagery by creating a planar DEM patch and then projecting onto it. However, with a little manipulation it is possible to find a homography between the orbital images and their orthoimages, which is derived here.

First, to form an equation for the plane, the normal must be converted into Cartesian coordinates:

$$\hat{\mathbf{n}} = \begin{bmatrix} \cos n_\theta \cos n_\varphi \\ \sin n_\theta \cos n_\varphi \\ \sin n_\varphi \end{bmatrix} \tag{11}$$

The equation for a plane using these parameters is:

$$\hat{\mathbf{n}}^T (\mathbf{x} - \mathbf{x}') = 0 \tag{12}$$

where $\mathbf{x}'$ is the Cartesian xyz location of the surface at the DEM post:

$$\mathbf{x}' = h\hat{\mathbf{e}}' \tag{13}$$

($\hat{\mathbf{e}}'$is found using equation (3)). Solving the plane equation for $\mathbf{x}$ gives:

$$\mathbf{x} = \frac{\mathbf{n}^T \mathbf{x}'}{\mathbf{n}^T \mathbf{e}} \hat{\mathbf{e}} \tag{14}$$

or, in matrix form:

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} \hat{\mathbf{e}} \\ 1 \end{bmatrix} \tag{15}$$

where

$$\mathbf{P} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \frac{1}{\mathbf{n}^T \mathbf{x}'} \mathbf{n}^T & 0 \end{bmatrix} \tag{16}$$

Using equations (2), (7), (9) and (16), the homography $\mathbf{H_k}$ between the orbital image $O_k(\mathbf{u})$ and its corresponding orthoimage $\tilde{I}_k(\mathbf{z})$ is found:

$$\begin{bmatrix} \mathbf{u} \\ 1 \end{bmatrix} = \mathbf{H_k} \begin{bmatrix} \mathbf{z} \\ 1 \end{bmatrix} \tag{17}$$

where

$$\mathbf{H_k} = \mathbf{F_k PLS} \tag{18}$$

3.3 The Objective Function

After $\tilde{I}_1(\mathbf{z}), \tilde{I}_2(\mathbf{z}), \ldots, \tilde{I}_n(\mathbf{z})$ have all been found using their homographies, they must be normalized to remove any illumination changes before any alignment measure can be developed. To this end, they are normalized by their mean and standard deviation:

$$I_k(\mathbf{z}) = \frac{\tilde{I}_k(\mathbf{z}) - \overline{\tilde{I}_k(\mathbf{z})}}{\operatorname{std}\left(\tilde{I}_k(\mathbf{z})\right)} \tag{19}$$

Once the normalized orthoimage patches $I_1(\mathbf{z}), I_2(\mathbf{z}), \ldots, I_n(\mathbf{z})$ are found, the objective function can be defined as the measure of alignment error as a function of plane position and orientation:

$$f(h, n_\theta, n_\varphi) = \sum_{k=1}^{n} \int g_c(\mathbf{z}) \left(A(\mathbf{z}) - I_k(\mathbf{z})\right)^2 \tag{20}$$

where

$$A(\mathbf{z}) = \frac{1}{n} \sum_{k=1}^{n} I_k(\mathbf{z}) \tag{21}$$

is the estimated normalized albedo for the patch, and $g_c(\mathbf{z})$ is a Gaussian correlation window centered around the DEM point being solved for.

4 Evaluating The Objective Function Using `mvpgui`

To evaluate and explore the parameters of the multiple-view objective function, I developed the program `mvpgui`.

4.1 Implementation Details

`mvpgui` was written in `octave`, an open source `MATLAB` clone [3]. Since `octave` does not natively support loading 32-bit images, georeferences, or their camera models, the interpreter was extended using oct-files: bridges between `octave` and custom C++ code [4]. In the oct-files, loading the images, georeferences and camera models was accomplished using the NASA Ames Vision Workbench, an open-source computer vision library .

4.2 Basic Commands And Usage

mvpgui is invoked by passing it a georeferenced DEM (to use for initial post values) followed by orbital images and their camera models:

```
./mvpgui dem.tif  orbit1.tif  orbit1.pinhole
                  orbit2.tif  orbit2.pinhole
                 [orbit3.tif  orbit3.pinhole...]
```

After loading the images, it gives a simple prompt. Entering "help" will give you the list of commands:

```
Loading images... Done!
mvp> help

Usage: help [command]

Available commands:
   help
   poi
   hwin
   cp
   cpi
   rplot
   llplot
   replot
   save
   exit

mvp>
```

When the program starts, it selects the current "point of interest" (see "`help poi`") on the DEM to be the center of the DEM. The "current plane" (see "`help cp`") is initialized to be tangent to the surface of the DEM at that point. Window size defaults to 10, and can be adjusted with "`hwin`".

Graphs can be generated with the "`rplot`" and "`llplot`" commands, and can be saved to file using the "`save`" command.

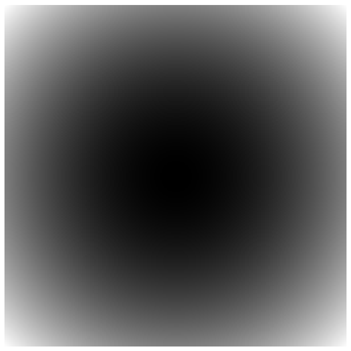
4.3 Results With Synthetic Data

Synthetic data was created to examine the performance of the multiple-view algorithm under ideal circumstances (no noise) and to verify the correct operation of `mvpgui`. The synthetic scene is simply DEM representing a planar surface with a uniform noise texture, as seen in figure 4. Notice how the planar geometry appears curved due to the equirectangular projection. Four "orbital images" were "taken" of the scene and are shown in figure 5. The camera models and locations as well as the location of the surface were chosen to be the same as the real orbital data used in the next section.

Figure 6 of the objective function was created using the "`rplot`" command. It shows the objective function as a function of DEM post radius while keeping the orientation constant. It has a well defined minimum at exactly the height of the post.

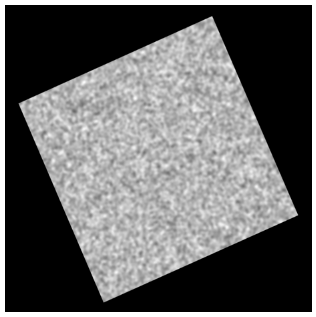
Figure 7 of the objective function was created using the "`llplot`" command. It shows the objective function as a function of plane orientation. It also has a well defined minimum at the expected orientation.

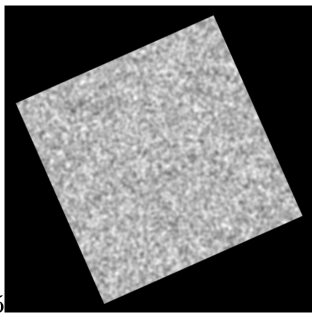
Both plots of the objective function show that it behaves quadratically as expected, and doesn't suffer from local minima. This good behavior can be attributed to the synthetic data being free of noise, as well as

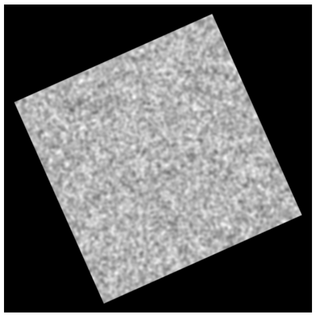

(a) Synthetic DEM

(b) Synthetic Orthoimage

Figure 4: Synthetic planar DEM and orthoimage


5


(a) Image 1 (b) Image 2


(c) Image 3

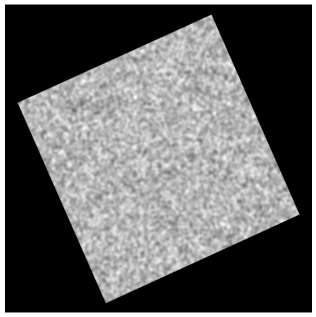

(d) Image 4

Figure 5: “Orbital images” that were “taken” of the synthetic scene

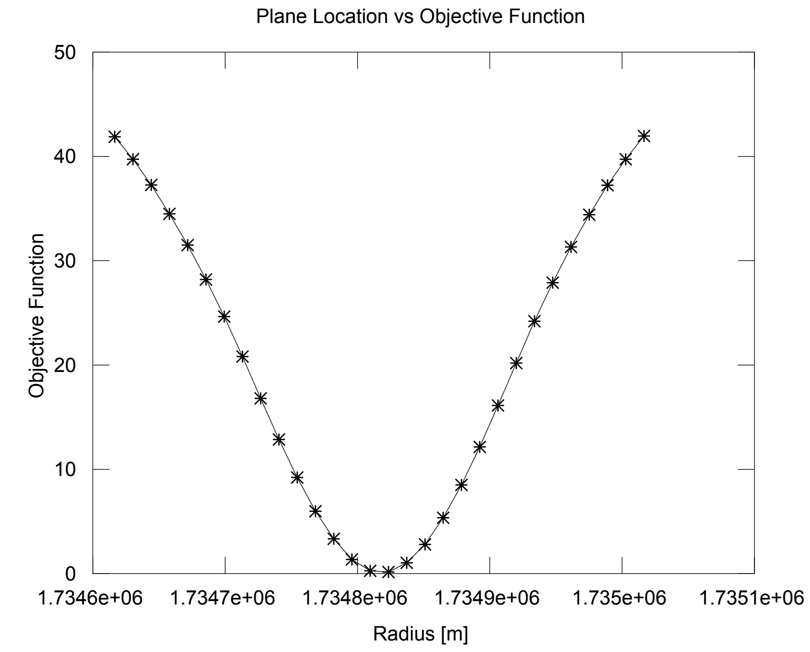


Figure 6: The objective function landscape as a function of radius with orientation held constant as created by the "`rplot`" command with a synthetic scene.

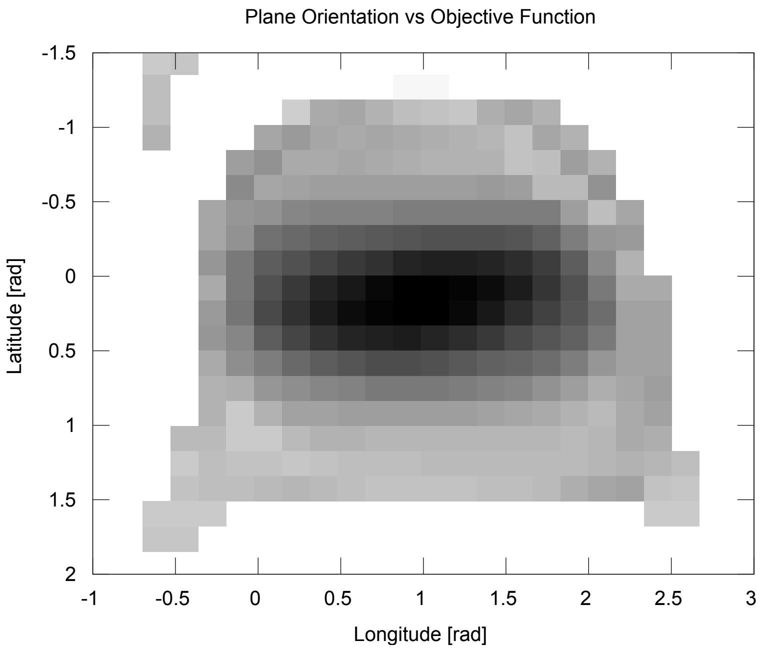
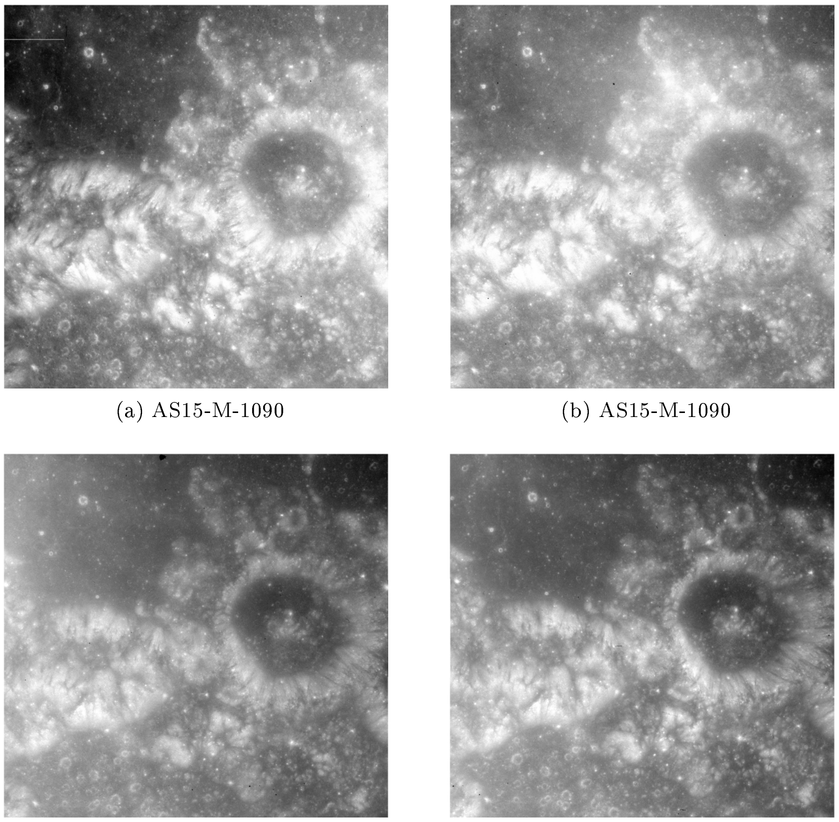


Figure 7: The objective function landscape as a function of orientation with radius held constant as created by the "`llplot`" command with a synthetic scene. The darker the data point, the lower the cost.



(a) AS15-M-1090 (b) AS15-M-1090

(c) AS15-M-1092 (d) AS15-M-1093

Figure 8: Orbital imagery from the Apollo 15 mission

the surface being planar. The results here verify that the objective function and mvpgui are indeed working correctly.

4.4 Results With Real Data

Orbital data from the Apollo 15 mission was also used to examine the performance of the multiple-view algorithm under real conditions. Crops of frames AS15-M-1090 through AS15-M-1093 were used, shown in figure 8.

Figure 9 shows the objective function as a function of DEM post radius, while figure 10 varies the plane orientation. Both minima are well defined, so it is safe to assume that they reflect the position and orientation of the actual lunar surface at that post.

The previous results were obtained using a Gaussian window with a standard deviation of 8 pixels. It is useful to see how the objective function begins to become noisy when a smaller window size is used. Figure 11 shows the objective function as a function of DEM post radius using a Gaussian window with a standard deviation of 4. Notice how the objective function begins to develop a local minimum. Clearly, window size will be an important parameter for this algorithm.

5 Conclusion

Through the use of the `mvpgui` program, it is apparent the cost function for the new multiple-view photogrammetry algorithm satisfies the needs of the algorithm: it has well-defined minima, is stable, and (if a sufficiently sized window is chosen), does not suffer from local minima.

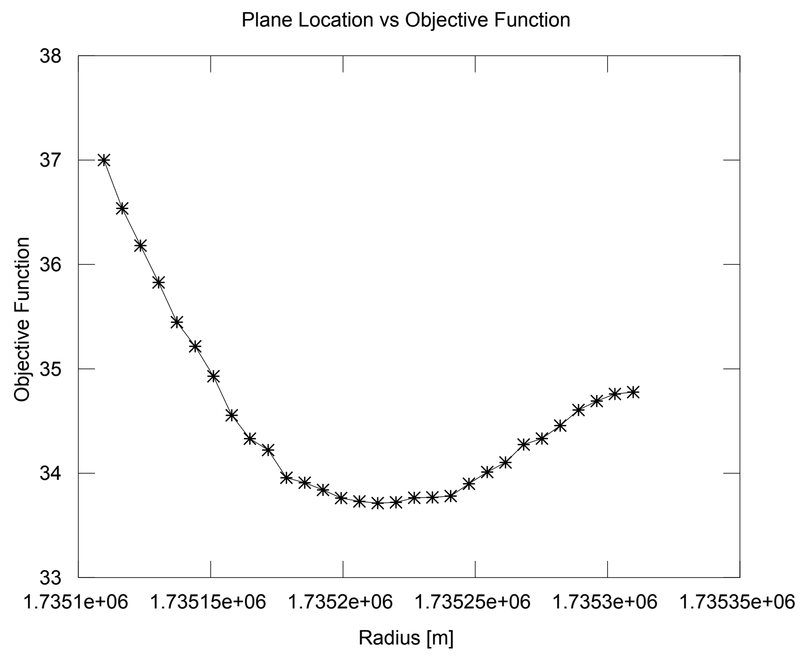


Figure 9: The objective function landscape as a function of radius with orientation held constant as created by the "`rplot`" command with orbital imagery from Apollo 15.

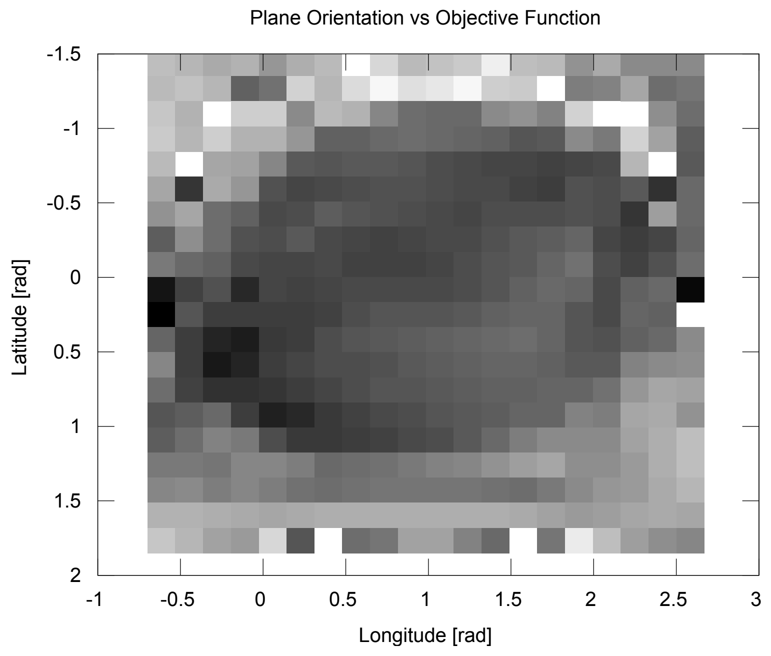


Figure 10: The objective function landscape as a function of orientation with radius held constant, created by the "`llplot`" command with orbital imagery from Apollo 15. The darker the data point, the lower the cost.

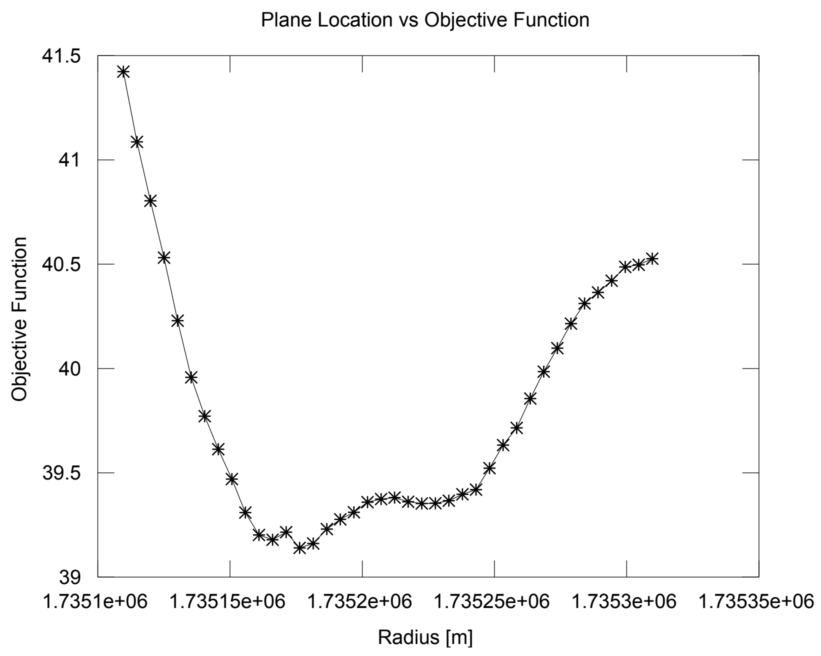


Figure 11: The objective function landscape as created by the "`rplot`" command with orbital imagery from Apollo 15, with a Gaussian window of standard deviation 4. Notice how the objective function begins to become noisy when a smaller window size is used.

As the multiple-view photogrammetry algorithm is continued to be developed, mvpgui should be maintained in parallel as a debugging and visualization tool for the algorithm. It's visualization capabilities will be invaluable as the complexity of the algorithm increases.

The next step in the development of the multiple-view photogrammetry algorithm will be to implement an optimization algorithm to find the minimum of the objective function automatically. `mvpgui` will be invaluable for debugging because it will allow us to visualize the objective function at every iteration.

References

[1] USGS Eastern Region PSC 4. Map projections, 2006. Available from: `http://egsc.usgs.gov/isb/pubs/MapProjections/projections.pdf`.

[2] Michael J. Broxton and Larry J. Edwards. The ames stereo pipeline: Automated 3D surface reconstruction from orbital imagery. In *Lunary and Planetary Institute Science Technical Report*, volume 29, page 2419, 2008.

[3] John W. Eaton. Gnu octave, 2011. Available from: `http://www.gnu.org/software/octave/`.

[4] John W. Eaton. *Octave Manual*, 2011. Available from: `http://www.gnu.org/software/octave/doc/interpreter`.

[5] The Intelligent Robotics Group. The nasa ames stereo pipeline, 2011. Available from: `http://ti.arc.nasa.gov/tech/asr/intelligent-robotics/ngt/stereo`.

[6] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2008.