EDGE DETECTION FOR MOBILE ROBOTS ON LUNAR SURFACE AND SURROUNDINGS

CSE400: THESIS

SUBMITTED BY:

Mehtab Iqbal

ID: 08241001

Department of Computer Science and Engineering (CSE)

BRAC University

Dhaka, Bangladesh

STUDENT:

MEHTAB IQBAL (08241001)

SUPERVISOR:

DR KHALILUR RAHMAN

EDGE DETECTION FOR MOBILE ROBOTS IN LUNAR SURFACE AND SURROUNDING

A THESIS BY:

Mehtab Iqbal

ID: 08241001

Bachelors in Computer Sciences (BSc) Program

The thesis is submitted to the Department of Computer Science and Engineering (CSE), BRAC University, 66 Mohakhali, Dhaka 1212, Bangladesh, in partial fulfillment of the requirements for the degree of:

BACHELOR IN COMPUTER SCIENCE (BSC)

Department of Computer Science and Engineering (CSE)

SUPERVISOR:

DR KHALILUR RAHMAN

Department of Computer Science and Engineering (CSE)

BRAC University

66 Mohakhali, Dhaka 1212

Bangladesh

April 2012

ACKNOWLEDGEMENT

I would like to thank my supervisor Dr. Khalilur Rahman for all the support and advice through out this thesis.

Additionally I would like to thank the following people for their patience and faith in me during my undergraduate program:

Professor Ziauddin Ahmed

Dr. Mumit Khan

Dr. Syed Salam

Dr. Manjur Karim

A special thanks to Dr. Peter M Cronin for his constant motivation.

Finally I would like to thank my family for all their support and Sumaiya Rahman for the encouragement, inspiration and assistance.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ALGORITHMS

1. ABSTRACT

The goal of this paper is to explore possibilities in devising a system that is able to detect obstacles in a scene or situation where color variation is limited and environment is noisy, such as that of the moon where there are many craters of different depths and rocks of various sizes and shapes.

The research is premised upon the importance of space research and the dual problem of manned missions to space finances and human survival. For space programs that involve scouting or sample collection from planetary surfaces, it is cost-effective to make use of autonomous or semi-autonomous robots. These robots need to overcome obstacles in lunar/ planetary surface without any hazard, such as falling over, being stuck and so on.

This paper will handle the methods and mechanism of the primary part of this system (obstacle detection) using the moon surface as the destination. Existing algorithms for obstacle detection dealing with edge and corner detection have been compared and modified to best determine obstacles on the lunar surface. In particular shadow masking technique along with edge detection has been used to discern shapes of ditches and mounds on the lunar surface.

2. INTRODUCTION

A visual system is a collection of devices that transform measurements of light into information about spatial and material properties of a scene [2]. Humans view the three-dimensional structure of the world with apparent ease [1], being able to discern shape and translucency of objects, count and order objects and even identify emotions in people among other aspects. Unfortunately, what humans and animals are able to perceive effortlessly, computational mechanisms are error-prone in understanding even basic images [2]. Computer vision is a discipline where research is ongoing to derive mathematical techniques that allow computers to recover the three-dimensional shape and appearance of objects in imagery [1].

Imagery depends on three broad characteristics: a) the geometry of a scene, a change in the shape of an object changes the image, b) the photometry (illumination and material properties

of an object), a skyline appears differently on a cloudy day to that of sunshine and, c) dynamics of the environment [2]. Computer vision has explored different ways of perceiving these characteristics to meet different ends. Simply put, artificial vision offers the potential of relieving human of tasks that are dangerous, monotonous or unfeasible [2] such as, Optical Character Recognition (OCR), motion capture (mocap), surveillance, face detection, visual authentication etc. In recent times, vision-guided helicopters and aircrafts can, nowadays, take off, fly and land.

The problem we are addressing in this paper is that of using computer vision to identify paths for an autonomous robot to traverse through a path on a lunar surface using shadow detection and edge detection mechanism. The robot will have a camera to capture the moon's surrounding and by processing those images, decide whether it is possible to move ahead on the rocky surface or turn to avoid the hill. Although the aim is to identify or create a system, more emphasis is placed on comparing, analyzing and combining existing methods or systems.

THE LUNAR SURFACE

The Moon is a little over one-fourth the diameter of Earth and contains less than one-eightieth of its mass [18]. While Earth's mountains are built by gradual sliding of one tectonic plate over another or by volcanic eruptions, the Moon's mountains result from the impact of asteroids. The Moon's landscape is made up of craters, lunar maria and bays, wrinkled ridges, rilles and domes. Unlike Earth, the Moon has no atmosphere or surface activity that is able to *erase* impact of asteroids, meteors or comets.

The craters are essentially impact sites, huge shallow holes dug on the lunar surface by asteroids, meteoroids or comets. On the other hand, the lunar maria represent the dark somewhat circular areas that are visible to the naked eye and are in fact, large craters covered in lava (Figure 1: Lunar Landscape) range from small ditches to giant basins spanning hundreds of kilometers[18]. The lunar rilles are mostly less than 2 km wide and are essentially long, narrow valleys and gorges that crisscross the maria. In addition, the Moon has sixteen major

mountain ranges and large number of isolated peaks. Wrinkled ridges are the largest of the Moon's tectonic features and appear around the lunar maria.

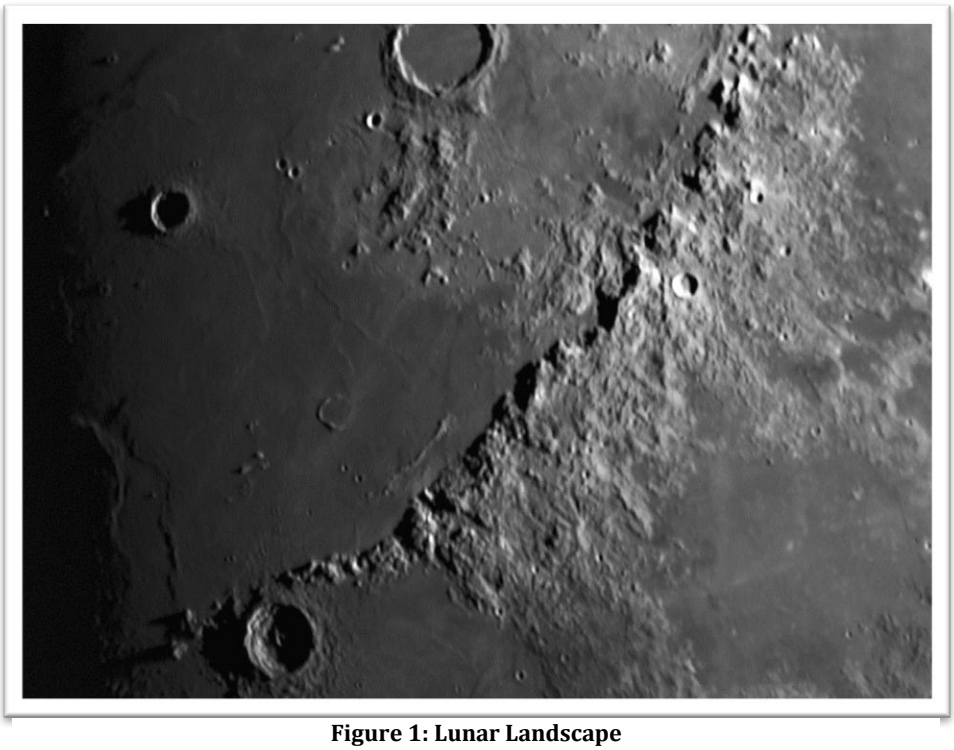

Figure 1: Lunar Landscape

Distortions to vision on the Moon are caused by the closeness of the lunar horizons and its extreme curvature [18]. If a crater is large enough, its walls may be over the horizon – that is, anyone standing in the center of a crater would not be able to see the towering walls around him or her. The shorter diameter of the Moon produces the fore-shortening effect. Features close to the North or South Pole of the Moon appear squashed in the north-south direction, explaining why certain craters and maria appears elliptical [18].

The Moon receives all its light from the Sun. While the surface of the Moon cannot create its own light, it can reflect light from the points where sunlight reaches it. At any given time, the Moon is five hundred thousand times fainter than the Sun. Some features of the Moon that need bearing for this paper [7][1]:

[1] The paper determines the features as the aspects that make it difficult to map impact craters. The features were improvised for the lunar landscape.

a) The “visibility” of impact craters in optical images depends principally on the interaction between the illumination and incidence (view) direction, surface scattering behavior and the atmospheric state.

b) Some geographical features such as small volcanic constructs or valleys have similar morphological characteristics as craters.

c) Impact craters are often concentrated into clusters resulting in overlap, and in larger craters multi-ring structures frequently occur. This means that the separation of individual craters from their background can be very difficult to generalize.

3. LITERATURE REVIEW

Several methods to detect objects automatically have been developed but the inherent limitations of imagery data and variety of objects make the task difficult [12]. Automatic identification of objects or obstacles are often required in determining if any obstacle in a given space hinders free and safe travel by an autonomous vehicle [14]. Some commonly used methods of obstacle detection are: a) edge detection, b) shadow detection, c) corner detection and d) image segmentation. Singh [14] categorized a good obstacle detection system as one containing the following features:

- able to detect obstacle in a given space within an appropriate time,
- identifying the correct obstacles, and
- identifying and ignoring ground features that may otherwise be mistaken as obstacles

Obstacle detection occurs in two steps. First, edge detection is performed, which is the fundamental of low-level image processing, and the resultant usable set of edges are then used for higher level processing required for segmentation of the objects in a scene [1].

"An edge detector can be defined as a mathematical operator that responds to the spatial change and discontinuities in a gray-level (luminance) of a pixel set in an image[1]. An edge is indicated by abrupt changes within an image that shows characteristic features and thus can be categorized as a set of pixels whose surrounding intensity follows a continuous variation [11]. Edge-dividing areas in the lunar landscape can have very similar properties such as when considering craters; but also have different aspects to it, for instance, when rilles run along a crater, thus having too many rapid variations. Although edge detection may vary based on output requirement, they share the need for precise edge information [1] to determine the closed area between objects that edges bind [18].

Different Edge detectors work better under different conditions, and thus it can be surmised that there is no one algorithm which can perform best under all circumstances [11].Boolean function based edge detectors produce thick edges in its output and thus, are problematic in

images with finer details [11].On the other hand, Marr-Hildreth cannot distinguish between weak and strong edges since it has only one threshold operator [11]. The Canny edge detector does not perform well for heavily textured backgrounds as it draws the textures as edges [12].

The quality of edges discerned by any algorithm is dependent on the quality of image, surface properties, [4], lighting conditions, objects with similar intensities, edge density in the scene and noise [11]. There are algorithms to overcome specific limitations by adjusting certain values and approximating thresholds. A very common property taken into consideration during edge detection is the intensity variation within an image. Figure 2: Derivatives of Intensity at an Edgeshows the relationship between the intensity variation and existence of an edge [12].

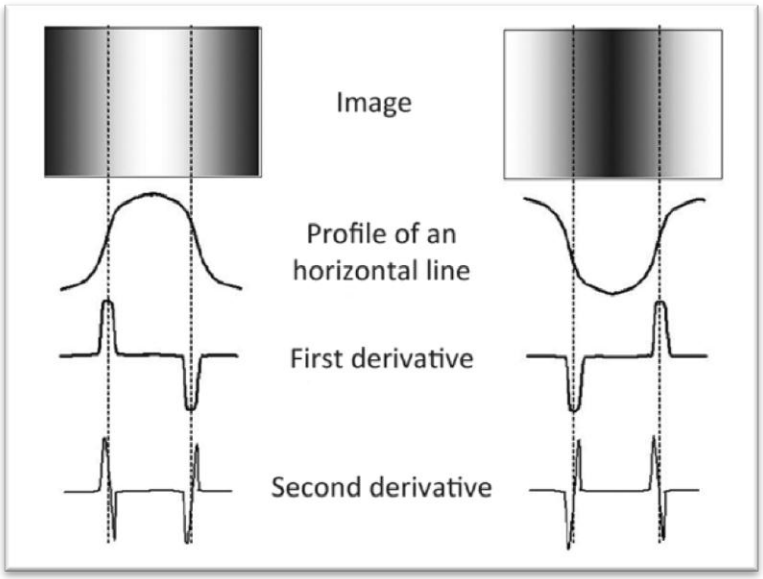

Figure 2: Derivatives of Intensity at an Edge

Edge detection methods are mainly as follows:

a) Use of Gradient Operator: The Sobel, Prewitt and Roberts [13] methods uses derivatives on an intensity map to calculate the maximum change in the gradient at an edge [12]. Figure 3 show the common gradient operators used by these methods.

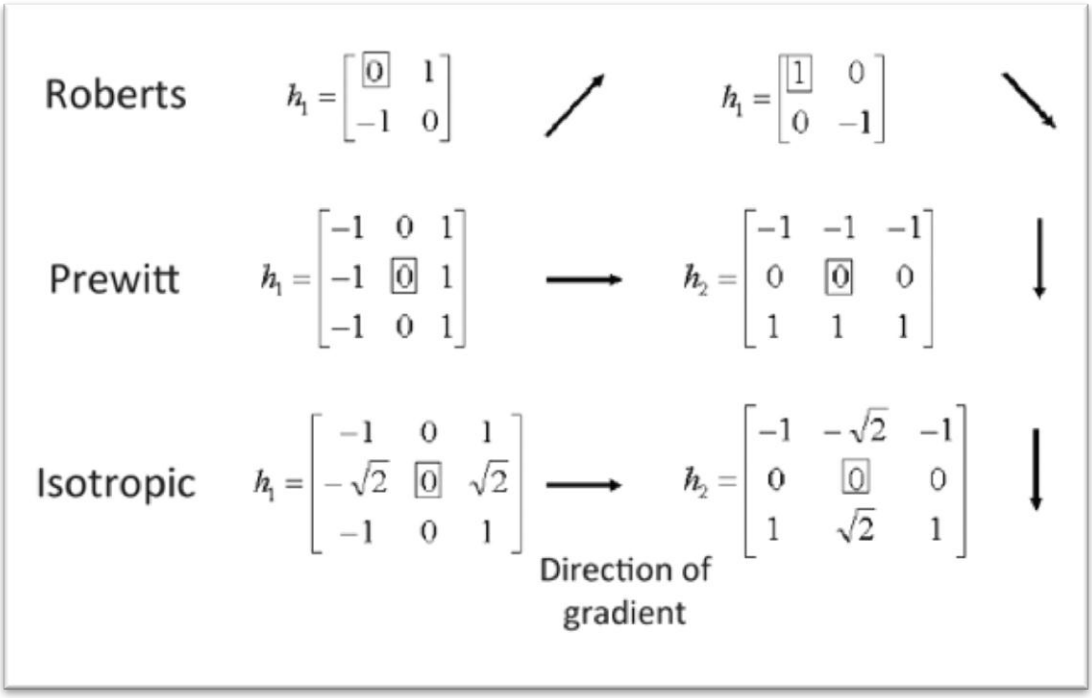

Figure 3: Common gradient operators of edge detection methods

All the above methods are both susceptible to noise and in case of homogeneous intensity distribution, are inaccurate.

b) Use of Optimum Operator: The Marr-Hildreth Edge Detector uses a Gaussian smoothing followed by the application of a rotation invariant Laplacian, $r^2 f = \frac{\partial^2 y}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$, to evaluate the gradient change and then denote changes that is within a specific threshold as an edge [11]. This provides more accurate results than the previous mentioned algorithms and thus was a widely acclaimed edge detection method.

Canny [5] modelled edge detection as a signal processing problem [18] theorized that it should satisfy a)signal to noise ratio (SNR) criterion, b)location accuracy criterion and c)single-edge response criterion. He improved on the Marr-Hildreth edge detector by making modifications to the final processing of the image by introducing a low and high threshold instead of a single threshold.

Nadernejad, Sharifzadeh, &Hassanpour [11] found some good distinction between the abilities of various edge detections by comparing the Canny edge detector, Marr-Hildreth edge detector, Local Threshold and Boolean Function Based edge detector and Color edge detector

using Canny operator. Canny edge detector outperformed all the rest on a variety of sample images. Boolean function based edge detection is comparable in terms of its results but Canny edge's characteristic single pixel outline makes it a more desirable choice. One interesting fact to note, the Color Canny edge detector seemed to be a good candidate because of its access to more information (three color channel instead of a single intensity map), however combining the three channels post processing proved to be a challenge. The vector angle/Euclidean distance perform poorly under inconsistent illumination and the Multi-Flash edge detector failed on outdoor images [11].

Furthermore, Canny edge is susceptible to picking up noise and unwanted features on a rough terrain. Agaian, Almuntashri, & Papagiannakis [1] came up with a modified Canny edge detector which uses a smoothing and gradient kernel matrices to detect edges from images of Asphalt Concrete. Their version seemed to work well on images which did not have noise artifacts and produced remarkable results.. However, their kernel matrices were of fixed sizes and proved to be a problem in case of edges that was much larger than the kernel dimension and thus requires manual tweaking.

Other than noisy backgrounds and textured terrain, edge detectors also face an issue in case of shadows. In case of a lot of distinct shadows in an image, edge detectors such as Canny tend to draw their edges. This is however undesirable, since it cuts down on movable space in case of an autonomous vehicles. Recent researches have delved deep into addressing this problem and worked on another augmented image processing phenomena for detecting shadows [8].

Shadow detection is usually done to remove the shadows from the images before further image processing techniques such as edge detection is applied on them [3]. There are more usages of shadows such calculating the homography of an image in order to do depth mapping and other processing based on multi-image systems [12].

Guo, Dai and Hoiem [6] have used a method of estimating fractional shadow coefficient using color matting. Their "pairwise" method gave better result than using simple appearance-based model. Also the application of soft matting allowed them to restore the image without

shadow with sufficient accuracy. Their only problem was that the method relied on segmentation which in case of certain illumination tend to group soft shadow regions with non-shadow regions causing a problem in the presence of orientation discontinuities such as a building or a wall [6].

Another way of looking at shadows can be through modeling energy [8]. Images with colorful backgrounds can be accounted for by increasing the energy intensity at the shadow points by the mean energy level on the image. This method allowed for the removal of large percentage of foreground shaded colors without losing pertinent image data.

Finally a simpler yet faster method of shadow removal using statistical analysis of intensity distribution within an image was introduced in 2011 by Blajovici, Kiss, Bonus, & Varga [3]. Even though, their method is similar to that of modelling energy function, they used three separate channels of color to model a high light region and a low light (shadow) region. Once the gray median is calculated from the high and low point of illumination a difference can be obtained. The difference between the high and the low intensity of lighting is then added to the shadow region thus removing the low points of illumination in the image [3].

Taking into consideration the results of Canny edge and the improved Canny detector applied on the Asphalt Concrete images [1], it promises to be a good algorithm to try and detect obstacles on the lunar surface. Additionally since the lunar surface is highly textured due to craters with low and ambient illumination, we would like to try methods of shadow removal before using the Canny edge detector. Since Sobel is a gradient based operator [18] it tends to pick up texture noise without much accuracy and thus forming weak edges. We would like to try using that relation to try and reduce the texture noise from our final output image.

4. METHODOLOGY

In this project we segregated lunar obstacles to be of two different types – holes and rocks. In order to detect the aforementioned obstacles we have theorized that holes or craters are lowest intensity points in the image intensity map while the rocks or mounds contain the highest intensity points at the peak.

Using several edge detection methods have lead us to the conclusion that Canny Edge detector gives the optimal result. However, due to the sensitivity and accuracy of Canny edge it tends to perform undesirably on lunar surfaces. Lunar surface is anything but homogenous or smooth and Canny edge tends to outline every detail on the terrain. Furthermore it is possible to reduce the texture noise by tweaking the upper and lower threshold of the Canny operator but that poses an inconvenience to the autonomy of a machine.

Another challenge faced by Canny algorithm is the presence of shadows on the scene. With so many dunes and rilles, the entire surface is pock marked with shadows. We decided to either remove the shadow altogether or use a technique to try ignoring the shadows. However, the latter risks avoiding the ditches and craters when outlining the edges.

By using an Additive Shadow Removal model [3] we thought it might help us in determining a shadow coefficient and also to figure out the difference between the high and low points in the image. This seems to be a useful technique in order to eliminate ground level shadows and retain those inside the craters.

Once we manage to estimate the mean gray level in an image we can easily apply a shadow mask and run the Canny Operator on the resultant. Finally we used the noisy edge obtained from the Sobel algorithm to perform an intersection to remove all weak edges that appear to be disjointed from any strong edges in our Canny result.

The entire process flow is outlined in Figure 4 below and Algorithm 1 details the steps used to achieve our results.

OUR PROCESS FLOW

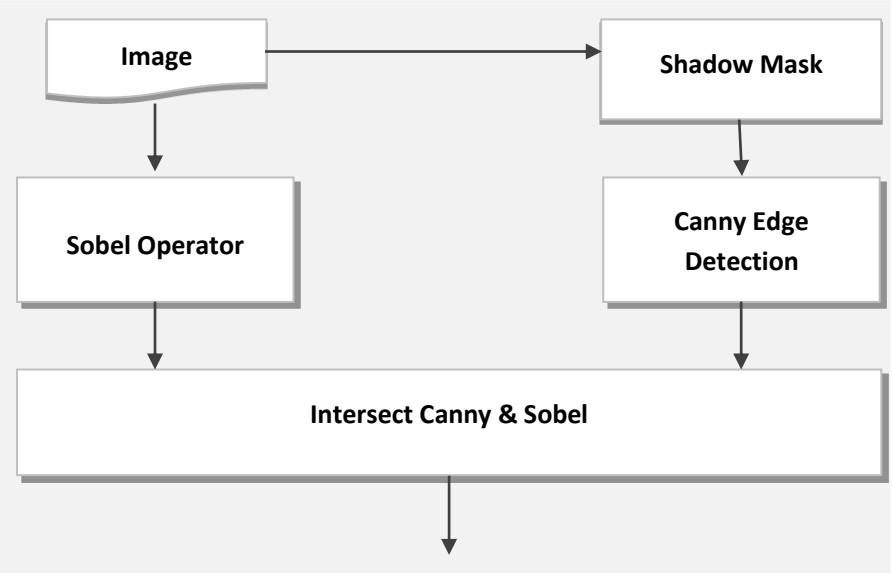

Figure 4: Our Process Flow

1. *compute Sobel edges from image I and store in J*
2. *extract shadow mask from I using additiveShadowRemoval(I)and store in K*
3. *extract Canny edge from the K and store in E*
4. *compare edges in J and E*
5. *if weak edge in J*
   then remove edge from E
6. *return E*

Algorithm 1: Shadow Mask Canny Edge Detector

Canny Edge algorithm and an Additive Shadow detection algorithm were used extensively for our algorithm. The aforementioned algorithms are outlined below.

ADDITIVE SHADOW DETECTION AND REMOVAL

Blajovici, Kiss, Bonus, & Varga [3], described the use of direct and ambient light present in an image to detect shadows. Shadows are part of the image where the direct light is occluded to some degree. According to them the shadow model can be defined as,

$$l_i = (t_i \cos\theta_i L_d + L_e) R_i$$

where, l_i represents each pixel in RGB image, L_d is the direct light and L_e is the environment light. R_i represents the surface reflectance, θ_i is the angle of direct light against the normal to the incident plane; t_i denotes the attenuation factor;

$$if t_i = 1 \; then \; l_i \; is \; a \; spotlight \; region$$

$$if t_i = 0 \; then l_i \text{is in the shadow region}$$

The shadow coefficient is given by:

$$k_i = (t_i \cos\theta_i)$$

The ratio between direct and environment light is defined as:

$$r = \frac{L_d}{L_e}$$

Thus based on the above criteria modeling, relighting the shadow region is done through the following equation:

$$l_i^{shdaow\ free} = \frac{r+1}{k_i r + 1} l_i$$

1. *Convert the image to grayscale to compute a mask from the global gray threshold.*
2. *S = shadow matrix determined by the gray mask*
3. *L = light matrix determined by 1 – gray mask*
4. *Total Shadow = sum of* S_i
5. *Total Light = sum of* L_i
6. *Compute Mean Shadow coefficient for each (red, green and blue) channel*
7. *Computer Mean Light coefficient for each (red, green and blue) channel*
8. *Calculate the intensity difference for each (red, green and blue) channel by subtracting Mean Shadow coefficient from the Mean Light coefficient*
9. *Add the intensity difference to the image* l_i *with a convolved smoothing mask*

Algorithm 2: Additive Shadow Removal

CANNY EDGE DETECTOR

Canny uses an optimum operator for its edge detection technique [18] and utilizes a Gray Scale image as an intensity matrix.

$$f(x,y) = \begin{bmatrix} f(1,1) & F(1,2) & \dots & f(1,n) \\ f(2,1) & f(2,2) & \dots & f(2,n) \\ \dots & \cdots & \dots & \dots \\ f(n,1) & f(n,2) & \dots & f(n,n) \end{bmatrix}$$

The elements in the matrix correspond to each pixel and $f(x,y)$ denote brightness value within a range of 0(black) to 255(white) [Zhou].

John Canny [5] devised a signal processing problem in order to solve the edge detection problem [11] by setting up three criteria corresponding to the characteristics he defined [18]:

a) Signal to noise ratio: to avoid true negative and false positive edge detection.

$$SNR = \left| \frac{\int_{-w}^{w} E(-x) f(x) dx}{\sqrt[\delta]{\int_{-w}^{w} f^2(x) dx}} \right|$$

where, $f(x)$ is the filter impulse response of the edge$[-w, +w]$, $E(x)$is the edge function and $\delta$is the root mean square of Gaussian noise.

b) Location accuracy: marked edges should be within an expected threshold of the actual edge.

$$Loc = \left| \frac{\int_{-w}^{w} E'(-x) f'(x) dx}{\sqrt[\delta]{\int_{-w}^{w} (f'(x))^2 dx}} \right|$$

The mathematical expression shows the proximity between a drawn edge and the actual edge, thus the higher the value of Loc, the better the result is.

c) Single-edge response: the edge output should be a line no more than a single pixel thick. This was modeled by Canny in the following equation:

$$x_{max}(f) = 2x_{zc}(f) = 2\pi\left[\frac{\int_{-\infty}^{\infty}(f'(x))^2dx}{\int_{-\infty}^{\infty}f''(x)dx}\right]^{\frac{1}{2}}$$

This criterion ensures there is one response to the single edge.

A Gaussian filter is used on the gray-scale image to smooth the image [11][17] and remove noise. The first order Gaussian function is defined as [Agaian]:

$$G(x,y,\sigma) = \frac{1}{2\pi\sigma^2}e^{\left(\frac{-x^2+y^2}{2\sigma^2}\right)}$$

Gradient vector is given by:

$$\nabla G = \left[\partial G/\partial x \quad \partial G/\partial y\right]$$

$$\frac{\partial G}{\partial x} = kx\exp\left(-\frac{x^2}{2\sigma^2}\right)\exp\left(-\frac{y^2}{2\sigma^2}\right) = h_1(x)h_2(y)$$

$$\frac{\partial G}{\partial y} = ky\,exp\left(-\frac{x^2}{2\sigma^2}\right)exp\left(-\frac{y^2}{2\sigma^2}\right) = h_1(y)h_2(x)$$

The Gaussian kernel is denoted by$\sigma$and is used for the smoothing filter. The kernel σcan be tweaked manually for different situation to increase or decrease the level of blurring required.

Canny edge algorithm calculates the gradient magnitude and direction once the points are smoothed. The first order partial derivatives of the two directions of point $(x,y)$are

$$P_x(i,j) = \frac{[I(i,j+1) - I\,(i,j) + I(i+1,j+1) - I(i+1,j)]}{2}$$

$$P_y(i,j) = \frac{[I(i,j) - I\,(i+1,j) + I(i,j+1) - I(i+1,j+1)]}{2}$$

The gradient magnitude and direction of the point (i,j) are:

$$M(i,j) = \sqrt{P_x^2(i,j) + P_y^2(j,j)}$$

$$\theta(i,j) = \arctan\frac{P_x(x,y)}{P_y(x,y)}$$

Where, $M(i,j)$ is the strength of the edge and $\theta(i,j)$ is the direction of the gradient at the point.

The final step before applying the Canny operator is the Non-maxima Suppression on the gradient magnitude $M(i,j)$. All points along the gradient matrix are compared along a 2x2 neighborhood around the center (i,j) along the direction of $\theta(i,j)$. If (i,j) is the maximum point in the neighborhood, it is marked as an edge, otherwise it is suppressed.

Ultimately the Dual Threshold that characterizes the Canny operator termed as the "hysteresis" is carried out [18][11]. The algorithm is outlined as follows:

Set T_h;

Set T_l;

if ($M(i,j) > T_h$)

    Set (i,j) *as edge*

if ($M(i,j) < T_l$)

    Set (i,j) *as non-edge*

if ($M(i,j) < T_h \& M(i,j) > T_l$)

    if (($i+1, j+1$) *is edge* | ($i+1, j-1$) *is edge* | ($i-1, j-1$) *is edge* | ($i-1, j+1$) *is edge* | ($i+1, j$) *is edge* | ($i-1, j$) *is edge* | ($i, j+1$) *is edge* | ($i, j-1$) *is edge*)

        Set (i,j) *as edge*

    else

        set (i,j) *as non edge*

Algorithm 3: Canny Edge Operator

The complete Canny edge detector algorithm is outlined in **Error! Reference source not found.**.

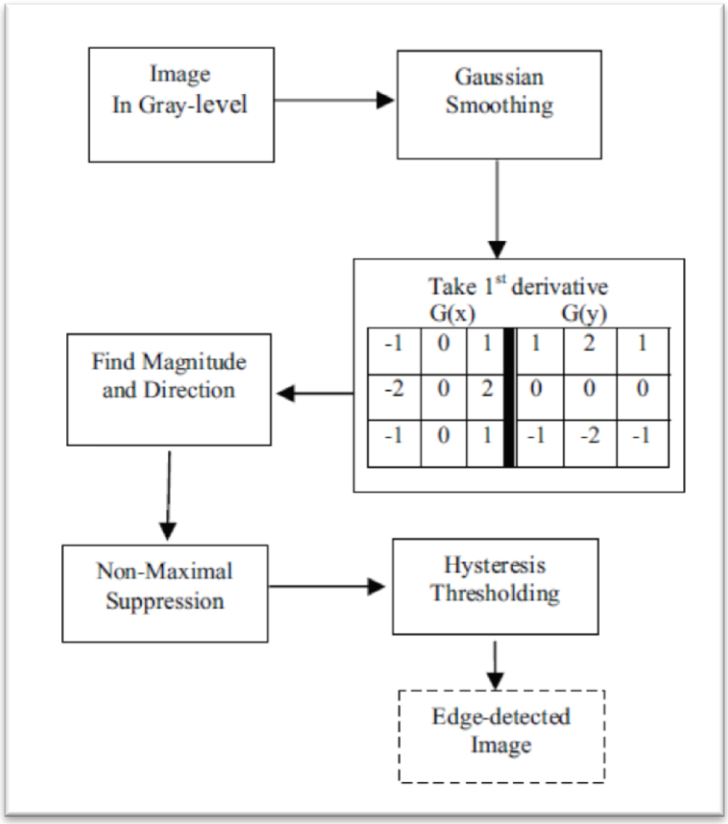

Figure 5: Canny Edge Process Flow

5. IMPLEMENTATION

Originally we toyed with the OpenCV library, which has a collection of very powerful image processing functionalities. However MATLAB also provided a good set of tools known as the "Image Processing Tools" which sufficed for our purpose. Furthermore, MATLAB has some built in image loading capabilities allowing for easy and fast loading and manipulation of images.

MATLAB's "imread()" function reads images of many common formats and stores them in a 2x2 pixel matrix for each color channel. This functionality enabled us to get right to the image processing work without having to develop pre-processing methods for our images.

We used fairly modest hardware to perform our processing and even the worst case brute force methods turned out to take no longer than a few minutes.

Language: MATLAB & MATLAB Image Processing Tools

Hardware:

Processor: Inter Core i3 M350 @ 2.27GHz x 2 (64-bit Processor)

RAM: 8.00 GB at 1333MHz

THE DEVELOPMENT ENVIRONMENT

The development environment holds important bearing to the implementation of a system and may even alter the required output. In choosing the environment for this paper, we considered C and MATLAB. These tools can handle similar problems and MATLAB can even be interfaced with C. However, due to the complex nature of images required and the inability of the programming language to efficiently handle complex matrix manipulations – MATLAB was used. In addition, MATLAB also sports some inbuilt functions that make easier and less time consuming to implement certain features.

ADVANTAGES OF MATLAB

i. As an interpreted language, MATLAB allows the programmer greater flexibility in terms of coding and updating data while the program is in execution.

ii. It is a functional-based language, making it easier to develop and document programs in.

iii. The system operates on matrices rather than scalar quantities, implying many mathematical operations and being especially useful for image processing.

iv. It does not require variable type and size determination before usage.

v. With built in features, it allows programmer to perform calculations and receive visual results.

vi. MATLAB allows use of undefined values, such as division by zero. For image processing where features such as gradient of vertical lines tend to be undefined, this is a very important feature.

DISADVANTAGES OF MATLAB

i. C allows codes that are more efficient than MATLAB in many areas, especially compared to use of some embedded features of MATLAB.

ii. Unlike C, MATLAB can be slow with slight inefficiencies in programming.

iii. With undeclared variables, chances of programmers making errors increases.

DIFFERENCES BETWEEN C AND MATLAB

Some reasons considered in selecting MATLAB over C for the project are given below:

i. Although C presents greater time-efficiency, MATLAB is preferred during development.

ii. Without the need for declaring variables in advance, MATLAB allows for fast coding.

iii. MATLAB does not have restrictions on data structures unlike C, and can perform certain array calculations using pre-defined commands.

iv. MATLAB has in-built functions and libraries for reference that is more reliable than the many available libraries of C.

SHADOW MASKED CANNY PROCESS

Figure 6 to Figure 10 shows the full process flow output at every step. Figure 6 shows an image of a lunar surface. Once the image is converted to gray scale to generate the intensity

map, the average global gray value of the image was calculated to compute out the shadow mask.

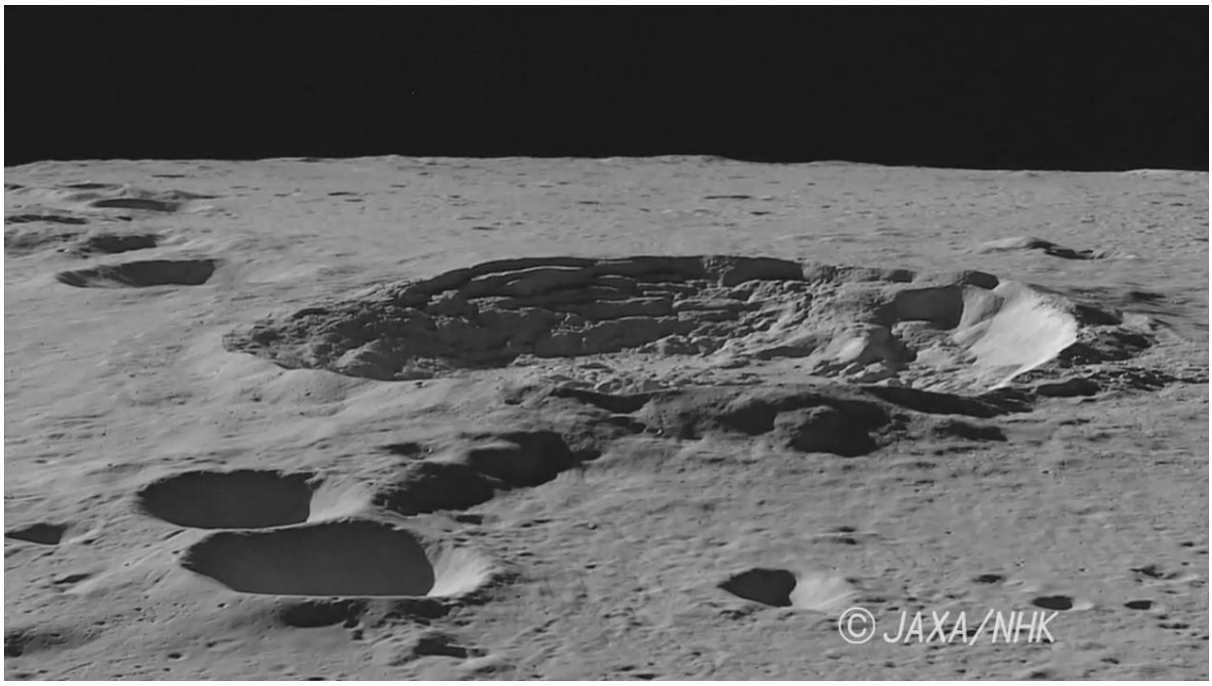

Figure 6

Once the shadow mask was calculate are able to easily computer the Canny edge of the binary image. This helps reduce noise in the environment by averaging the subtle texture variation out of the image. Once the threshold is applied to the gray scale image to form a binary map, the middle gray values are lost to either black or white as shown in Figure 7.

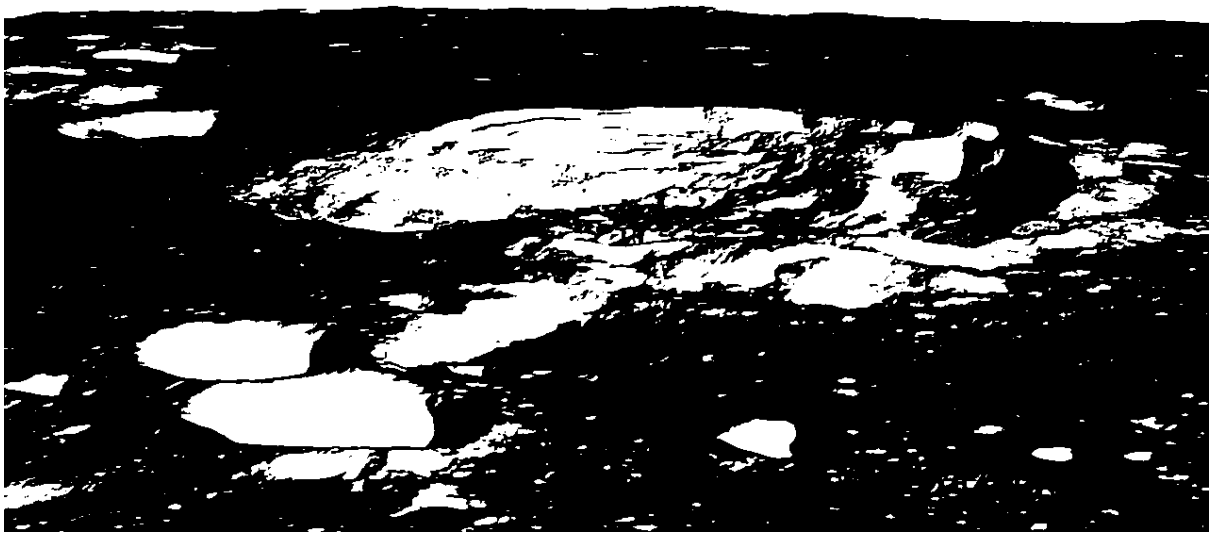

Figure 7: Shadow Mask

The resulting Canny edge is much cleaner due to the lost texture. However, some small granularities remain on the image as we can see in Figure 8. This is then compared with the Sobel edges of the original image (Figure 9).

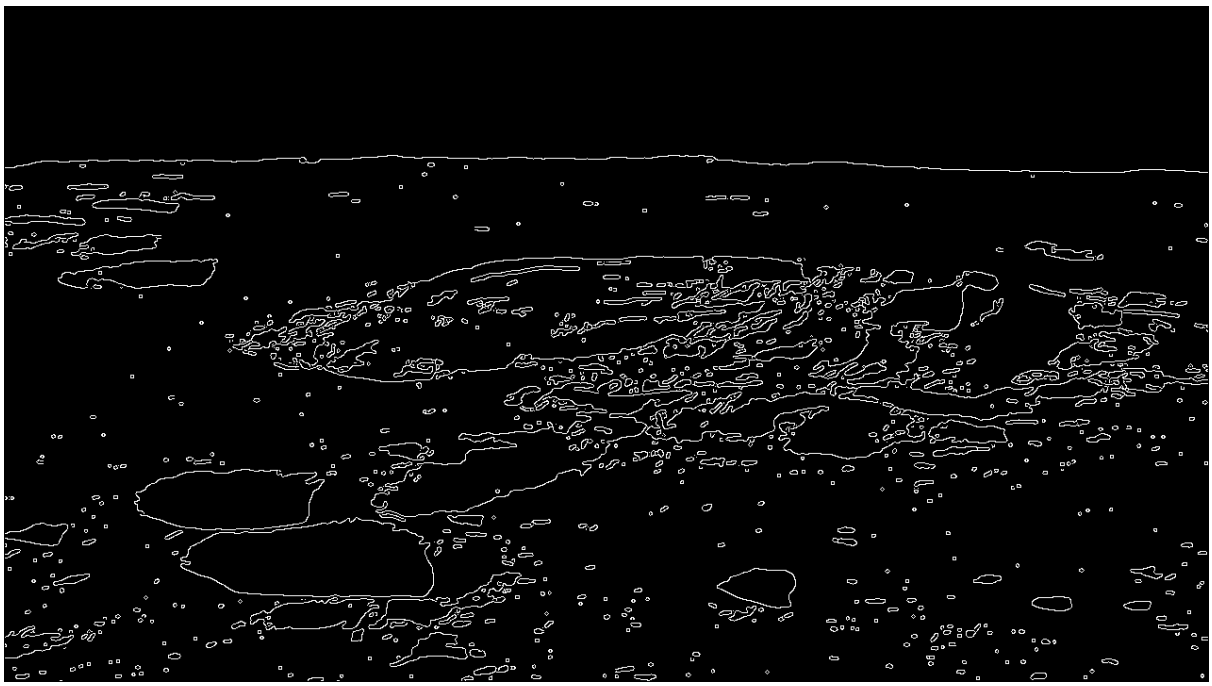

Figure 8: Canny Edge on Shadow Mask

During the comparison we check for weak edges. Every weak edge in the Sobel output is removed from the output shown in Figure 8.

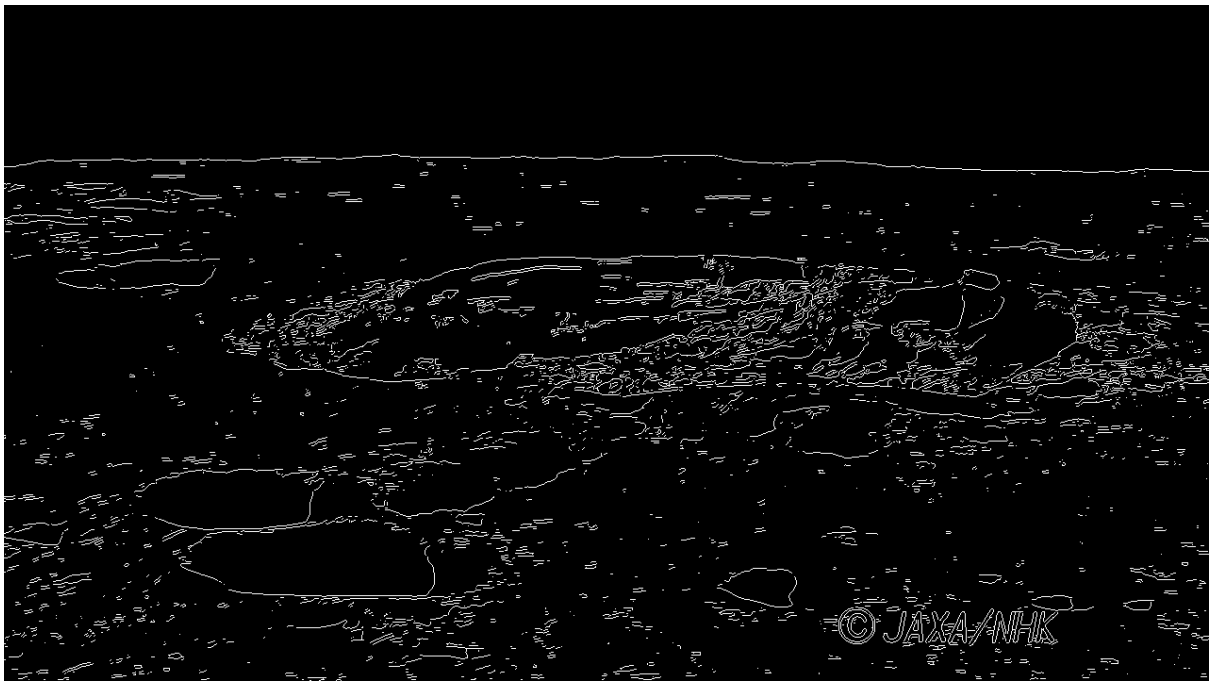

Figure 9: Sobel Edges

Figure 10 shows the final output of our process. It can be seen that the noise is considerably less and the edges are very well defined for every single crater on the image presented in Figure 6.

Figure 10: Final Output

6. ANALYSIS

We compared our results with three other edge detection algorithms. Canny edge algorithm and Marr-Hildreth Lagrange of Gaussian [11] are optimum edge detection algorithm [18] that was used in our comparison. Additionally Sobel edge detection was also compared since it is a gradient based edge detector, a different set of results were expected.

All algorithms were either implements in MATLAB or used directly from the MATLAB "Image Processing Tools" library. The results were compared based on their overall accuracy and noise ignoring capabilities.

In Figure 11 our algorithm clearly out performs all the other edge detectors, both in terms of noise reduction and accuracy. However, in Figure 13 and Figure 14 it suffers when there is a point with a spot light effect. Even though there isn't a single edge detector which performed well in that situation, Canny edge managed to give the most detailed result, even if unusable.

Interestingly enough in Figure 15 we can see that Sobel's output seems to be more informative than the others. However, it isn't very useful in terms of any concrete data.

Overall we can conclude that the Shadow Masked Canny Algorithm as implemented in this paper gave the best result.

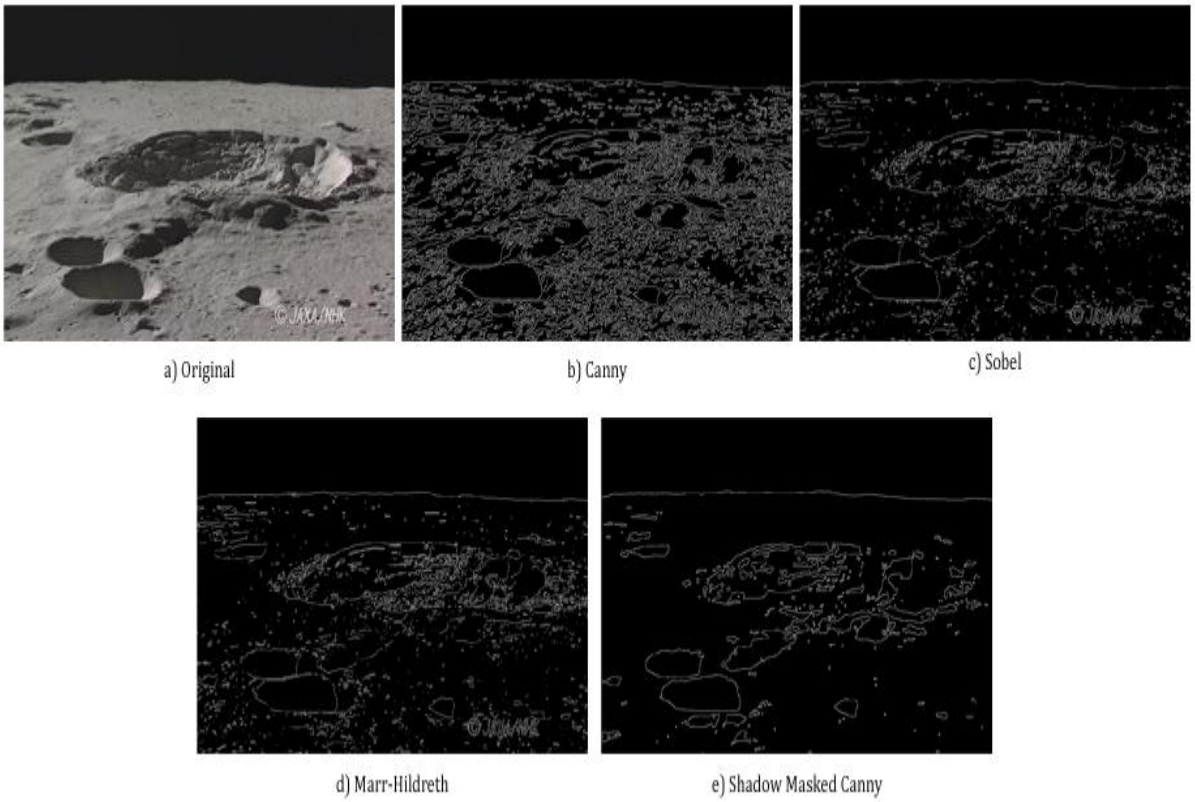
a) Original
b) Canny
c) Sobel
d) Marr-Hildreth
e) Shadow Masked Canny

Figure 11

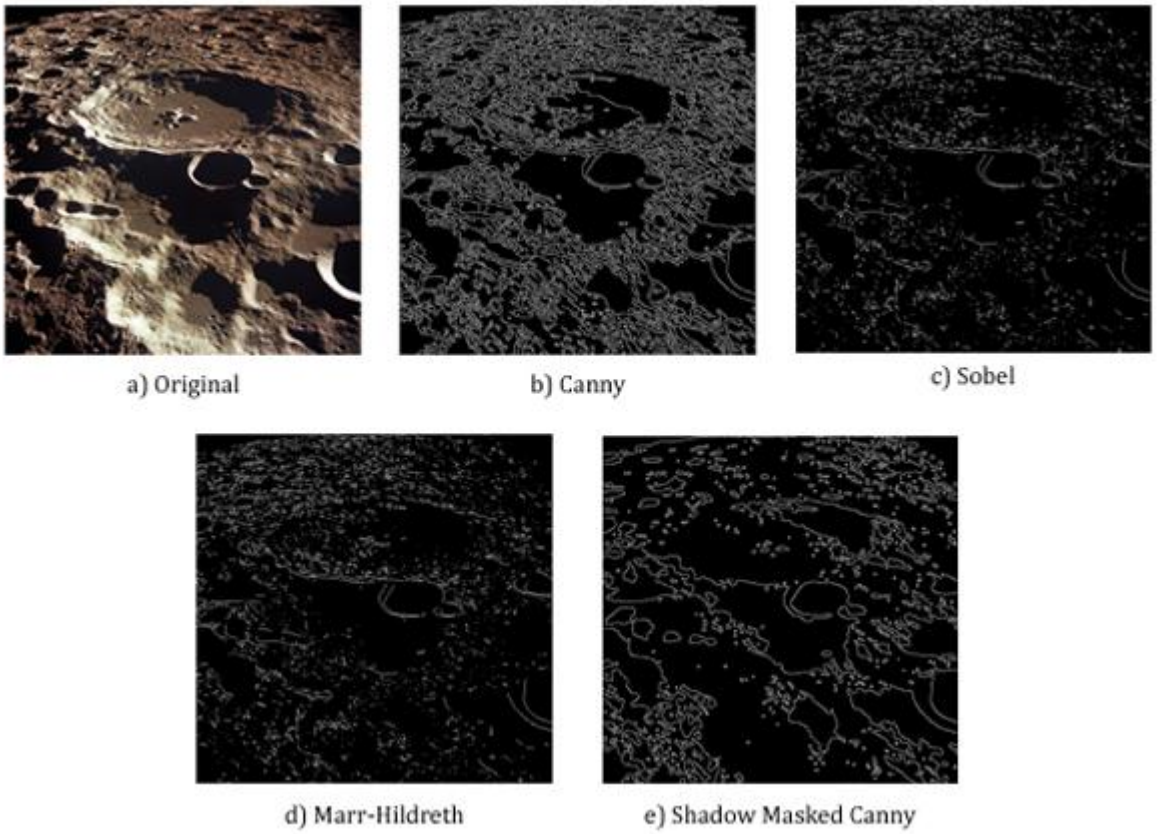

Figure 12

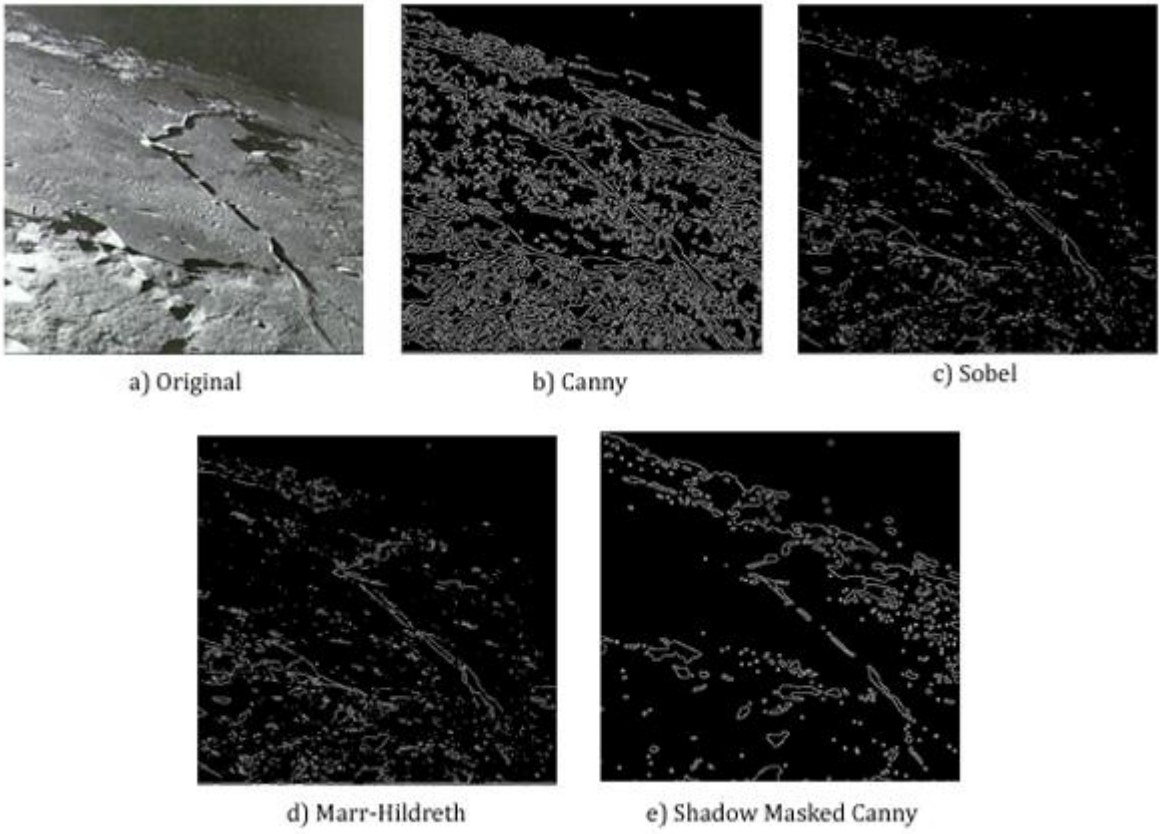

Figure 13

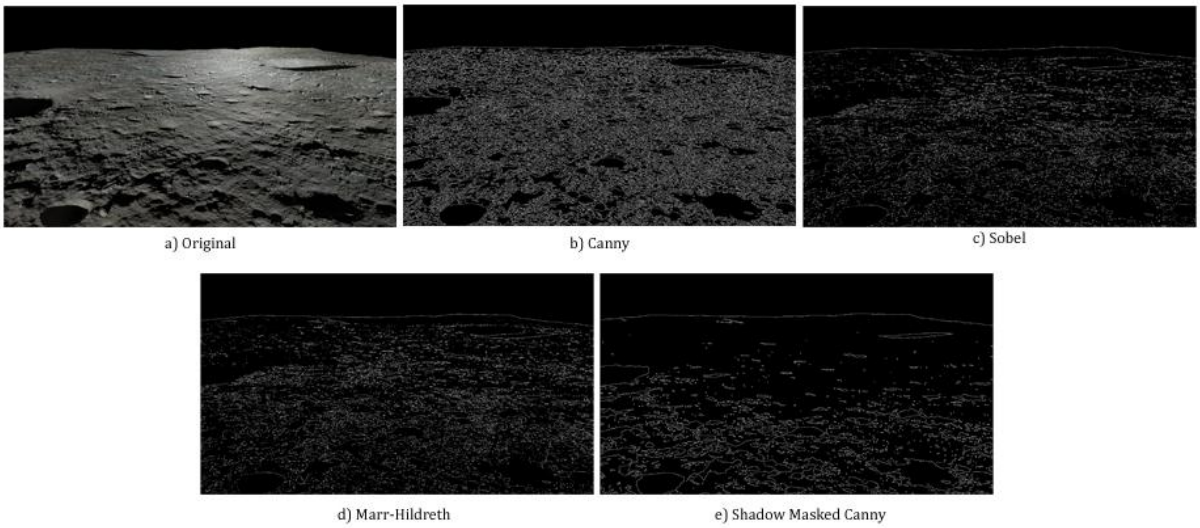

Figure 14

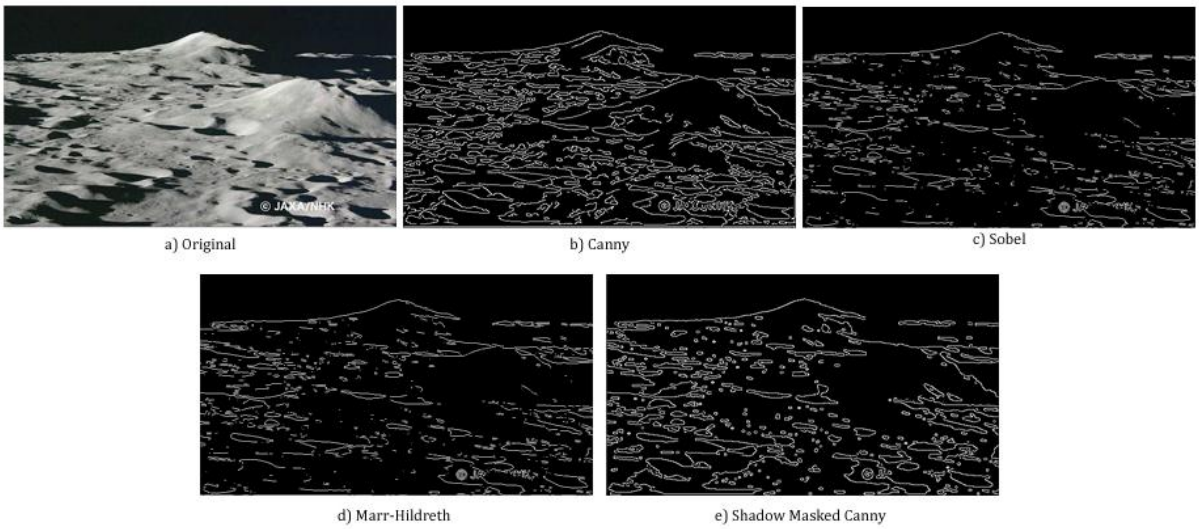

Figure 15

7. DISCUSSION

Object detection on lunar surface using traditional edge detection methodologies posed two major challenges, the lack of global illumination and the high textured terrain. Additionally there were two general features that we decided was required for our mobile vehicle to detect in order for safe travel – ditches and mounds.

It is noticed that while Canny edge algorithm results in highly detailed edges, it is not desirable in a defined textured scene such as that of lunar landscape. Our method of using Sobel operator to detect the high concentration edges made by the texture and then subtracting it from the Canny result on the shadow optimized images provided the best result.

Due to the sensitivity of the canny edge algorithm, hard shadows were being detected as shapes too. This isn't always desirable unless it is a ditch. Our decision to preprocess the image by removing all shadows or reducing the clarity of shadows other than those caused by strong edges was fruitful. It allowed our system to notice more accurate edges on the lunar surface.

Finally the use of erosion or dilation based on the average image intensity helped us remove the texture noise that was otherwise being drawn as shape edges. However, this method of using four major steps could use some optimization and that leave room for further research.

One major failure of our algorithm is in a situation where there is direct light incident at the center of the slope of a small mound – small compared to the illuminated area. This causes the shadow detector to ignore it and our edge detector is not able to find the base since the illuminated area is rather large, making the gradient across it, appear flat. This leaves scope for further research on the topic and may require a more elaborate set of input information.

WORKS CITED

[1] **Agaian, Sos, Almuntashri, Ali and Papagiannakis, T A,** *Improved Canny Edge Detection Application for Ashpalt Concret,***.** San Antonio : IEEE, IEEE International Conference on Systems, Man, and Cybernetics. pp. 3683-3687, 2009.

[2] **Ahmed, M B and Choi, T S,** *Local Threshold and Boolean Function Based Edge Detection*, IEEE Transaction on Consumer Electronics, pp. 74-79, 1991.

[3] **Blajovici, Corina; Kiss, Peter Jozsef; Bonus, Zoltan; Varga, Laszlo** *Shadow detection and removal from a single image.* Szeged, Hungary : SSIP , 19th Summer School on Image Processing, 2011.

[4] **Bue, Brian and Stepinski, Tomasz F** *Machine Detection of Martial Impact Craters From Digital Topography Data,* IEEE Transactions on Geoscience and Remote Sensing, pp. 265-274, 2007.

[5] **Canny J, Member**, IEEE, A Computational Approach to Edge Detection, *IEEE Trans. Pattern Analysis and Machine Intelligence*, 8(1):679-697, 1986.

[6] **Guo, Ruiqi, Dai, Qieyun and Hoiem, Derek** *Single-Image Shadow Detection and Removal using Paired Region,.* IEEE Computer Vision and Pattern Recognition (CVPR), Colorado Springs : IEEE, 2011.

[7] **Kim, Jung Rack; Muller, Jan-Peter; Gasselt, Stephen van; Morley, Jeremy G; Neukum, Gerhard** *Automated Crater Detection, A New Tool for Mars Cartography and Chronolo,* Photogrammetric Engineering & Remote Sensing, pp. 1205-1217, 2005.

[8] **Kumar, Sanjeev and Kaur, Anureet** *Shadow Detection and Removal in Color Images Using MATLAB*, International Journal of Engineering Science and Technology, pp. 4482-4486, 201

[9] **Ma, Yi.** *An Invitation to 3-D Vision: From Images to Geometric Models.* s.l. : Springer, 2004.

[10] **Mead, Aram.** *Development of a Vision Enhancement System for Use on the Lunar Surface.* s.l. : ProQuest, 2008.

[11] **Nadernejad, Ehsan, Sharifzadeh, Sara and Hassanpour, Hamid,** *Edge Detection Techniques: Evaluations and Comparison,* Applied Mathematical Sciences, pp. 1507-1520, 2008.

[12] **Rahman, Syedur.** Obstacle Detection for Mobile Robots Using Computer Vision (Thesis). s.l. : University of York, March 2005.

[13] **Roberts L**, Machine perception of three-dimensional solids, *Optical and electrooptical information processing*, Massachusetts, MIT Press, 1965.

[14] **Singh, S. and Keller, P** *Obstacle Detection for High Speed Autonomous Navigation.***.** Proceedings of International Conference on Robotics and Automation, s.l. : IEEE, 1992.

[15] **Szeliski, Richard.** *Computer Vision: Algorithms and Applications.* s.l. : Springer, 2010.

[16] **Wlasuk, Peter.** *Observing the Moon.* s.l. : Springer, 2000.

[17] **Yuan, Y B** *A fast algorithm for determining the gaussian filtered mean line in surface metrology,* Precision Engineering, pp. 62-69, 2000.

[18] **Zhou, Ping; Ye, Wenjun; Xia, Yaojie; Wang, Qi,** *An Improved Canny Algorithm for Edge Detection***,** Journal of Computational Information Systems, pp. 1516-1523, 2011.

APPENDIX

LITERATURE REVIEW TABLE

Author (Date)	Concept	Camera	Subject model	Key Finding	Limitation
Rahman(2005)	Planar homography, Epipolar geometry, Edge detection	Stereoscopic	Indoor setting, boxes and wooden planks.	Suitable for environments with little noise. Sufficient camera movement and minimum background texture detected.	Based on threshold, either ground is detected or object of decent heights are not detected at all. Excessive texture on object and not on ground may be detected. Epipolar lines may be inaccurate due to matrix limitations. Color similarity between object and ground leads to segmentation failure.
Nadernejad, Sharifzadeh, & Hassanpour(2008)	Canny Edge Detection, Marr-Hildreth Edge Detector and Local Threshold, Boolean Function Based Edge Detection, Color Edge Detection using Canny Operator	Single Image	Indoor and Outdoor images	Canny Edge detector has the best results. It out performs the Boolean function based edge detector because of the single pixel edges it produces. Canny color edge detector has good prospects since it has more information access.	The vector angle/Euclidian distance detector misses fine grained details. The color canny edge faces a challenge in proper combining of the three channels.
Blajovici, Kiss, Bonus, Varga (2011)	Shadow removal using statistical analysis of intensities related to illumination.	Single image	Simple images	Overall very smooth output	Only highly textured images are not smooth.
Kumar, Kaur (2010)	Energy function (shadow removal)	Single color image	Images with different colored backgrounds	Removal of large percentage of shaded colors without losing pertinent image data.	
Guo, Dai, Hoiem (2011)	Matting to estimate fractional shadow coefficient.	Single color image	Various outdoor images	Pairwise method performs better than the simple appearance-based model. Application of soft matting results in better lighting conditions on the result.	Detection relies on segmentation which may group soft shadows with non-shadow regions. This does not account for shading due to orientation discontinuities such as building walls.
Agaian, Almuntashri, &Papagiannakis(2009)	Altered Canny edge detector with a modified Gaussian smoothing kernel and gradient kernel	Single grayscale image	Asphalt Concrete Images	The concept of fusing Sobel with the improved Canny algorithm provides better output for noisy terrain than the original Canny algorithm	The kernel matrices are of fixed sizes. This causes an issue in case of large disparity between the size of the subject and the kernel dimension.

CODES

CANNY EDGE IMPLEMENTED IN MATLAB

```
function out = cannyEdge(originalImg)
originalImg = rgb2gray(originalImg);
[h, w] = size(originalImg);
originalImg = im2double(originalImg);

derivativeX=zeros(h,w);
derivativeY=zeros(h,w);

sigma = 0.8;
maxHysteresisThresh = 1.5;
minHysteresisThresh = 0.05;

sizeOfKernel = 6*sigma+1;

adjust = ceil(sizeOfKernel/2);
yGaussian = zeros(ceil(sizeOfKernel), ceil(sizeOfKernel));
xGaussian = zeros(ceil(sizeOfKernel), ceil(sizeOfKernel));

for i = 1:sizeOfKernel
        for j = 1:sizeOfKernel
                yGaussian(i, j) = -((i - ((sizeOfKernel-1)/2) - 1) / (2 * pi * sigma^3)) * exp(-((i - ((sizeOfKernel
- 1) / 2) - 1)^2 + (j - ((sizeOfKernel - 1) / 2) - 1)^2) / (2 * sigma^2));
        end
end
```

```
for i = 1:sizeOfKernel
        for j = 1:sizeOfKernel
                xGaussian(i, j) = -((j - ((sizeOfKernel - 1) / 2) - 1) / (2 * pi * sigma^3)) * exp(-((i - ((sizeOfKernel - 1) / 2) - 1)^2 + (j - ((sizeOfKernel - 1) / 2) - 1)^2) / (2 * sigma^2));
        end
end

gradient = zeros(h, w);
nonMax = zeros(h, w);
postHysteresis = zeros(h, w);

for i = 1 + ceil(sizeOfKernel / 2):h - ceil(sizeOfKernel / 2)
        for j = 1 + ceil(sizeOfKernel / 2):w - ceil(sizeOfKernel / 2)
                referenceRow = i - ceil(sizeOfKernel / 2);
                referenceColumn = j - ceil(sizeOfKernel / 2);
                for yyy = 1:sizeOfKernel
                        for yyyColumn = 1:sizeOfKernel
                                derivativeX(i, j) = derivativeX(i, j) + originalImg(referenceRow + yyy - 1, referenceColumn + yyyColumn - 1) * xGaussian(yyy, yyyColumn);
                        end
                end
                end
end

for i = 1 + ceil(sizeOfKernel / 2):h - ceil(sizeOfKernel / 2)
        for j = 1 + ceil(sizeOfKernel / 2):w - ceil(sizeOfKernel / 2)
                referenceRow = i - ceil(sizeOfKernel / 2);
                referenceColumn = j - ceil(sizeOfKernel / 2);
                        for yyy = 1:sizeOfKernel
```

```
                    for yyyColumn = 1:sizeOfKernel
                        derivativeY(i, j) = derivativeY(i, j) + originalImg(referenceRow + yyy - 1, referenceColumn + yyyColumn - 1) * yGaussian(yyy, yyyColumn);
                    end
                end
            end
        end

for i = 1 + ceil(sizeOfKernel / 2):h - ceil(sizeOfKernel / 2)
    for j = 1 + ceil(sizeOfKernel / 2):w - ceil(sizeOfKernel / 2)
        gradient(i, j) = sqrt(derivativeX(i, j)^2 + derivativeY(i, j)^2);
    end
end

nonMax = gradient;

for i = 1 + ceil(sizeOfKernel / 2):h - ceil(sizeOfKernel / 2)
    for j = 1 + ceil(sizeOfKernel / 2):w - ceil(sizeOfKernel / 2)
        if(derivativeX(i, j) == 0) tangent = 5;
        else tangent = (derivativeY(i, j) / derivativeX(i, j));
        end
        if(-0.4142 < tangent & tangent <= 0.4142)
            if(gradient(i,j) < gradient(i, j + 1) | gradient(i, j) < gradient(i, j - 1))
                nonMax(i, j)=0;
            end
        end
        if(0.4142 < tangent & tangent <= 2.4142)
            if(gradient(i, j) < gradient(i - 1, j + 1) | gradient(i, j) < gradient(i + 1, j - 1))
```

```
                    nonMax(i, j)=0;
                end
            end
            if(abs(tangent) > 2.4142)
                if(gradient(i, j) < gradient(i - 1, j) | gradient(i, j) < gradient(i + 1, j))
                    nonMax(i, j) = 0;
                end
            end
            if(-2.4142 < tangent & tangent <= -0.4142)
                if(gradient(i, j) < gradient(i - 1, j - 1) | gradient(i, j) < gradient(i + 1,j + 1))
                    nonMax(i, j)=0;
                end
            end
        end
end

postHysteresis = nonMax;

for i = 1 + ceil(sizeOfKernel / 2):h - ceil(sizeOfKernel / 2)
        for j = 1 + ceil(sizeOfKernel / 2):w - ceil(sizeOfKernel / 2)
            if(postHysteresis(i, j) >= maxHysteresisThresh) postHysteresis(i, j) = 1;
            end
            if(postHysteresis(i, j) < maxHysteresisThresh & postHysteresis(i, j) >= minHysteresisThresh)
postHysteresis(i, j) = 1;
            end
            if(postHysteresis(i, j) < minHysteresisThresh) postHysteresis(i, j) = 0;
            end
        end
```

```
end



edgeFlag = 1;
while(edgeFlag == 1)
	edgeFlag = 0;
	for i = 1 + ceil(sizeOfKernel / 2):h - ceil(sizeOfKernel / 2)
		for j = 1 + ceil(sizeOfKernel / 2):w - ceil(sizeOfKernel / 2)
			if(postHysteresis(i, j) > 0)
				if(postHysteresis(i, j) == 2)
					if(postHysteresis(i - 1, j - 1) == 1 | postHysteresis(i - 1, j) == 1 | postHysteresis(i - 1, j + 1) == 1 | postHysteresis(i, j - 1) == 1 | postHysteresis(i, j + 1) == 1 | postHysteresis(i + 1, j - 1) == 1 | postHysteresis(i + 1, j) == 1 | postHysteresis(i + 1, j + 1) == 1)
						postHysteresis(i, j) = 1;
						edgeFlag == 1;
					end
				end
			end
		end
	end
end

for i = 1 + ceil(sizeOfKernel / 2):h - ceil(sizeOfKernel / 2)
	for j = 1 + ceil(sizeOfKernel / 2):w - ceil(sizeOfKernel / 2)
		if(postHysteresis(i, j) == 2)
			postHysteresis(i, j) == 0;
		end
	end
end
```

```
out = postHysteresis;

imwrite(originalImg, 'originalImage.bmp');
imwrite(derivativeX,'derivativeX.bmp');
imwrite(derivativeY,'derivativeY.bmp');
imwrite(gradient,'gradient.bmp');
imwrite(nonMax,'nonMax.bmp');
imwrite(postHysteresis,'postHysteresis.bmp');
```

ADDITIVE SHADOW REMOVAL CODE IN MATLAB

```
function out = additiveShadowRemoval(img)

imgSize = size(img);

gray = rgb2gray(img);
mask = 1 - double(im2bw(gray, graythresh(gray)));

strel = [0 1 1 1 0; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 0 1 1 1 0];

shadow = imerode(mask, strel);
light = imerode(1-mask, strel);

shadowSum = 0;
lightSum = 0;

smoothMask = conv2(mask, strel/21, 'same');
```

```
shadowMeanRed = sum(sum(img(:,:,1).* shadow)) / sum(sum(shadow));
shadowMeanGreen = sum(sum(img(:,:,2).* shadow)) / sum(sum(shadow));
shadowMeanBlue = sum(sum(img(:,:,3).* shadow)) / sum(sum(shadow));

lightMeanGreen = sum(sum(img(:,:,2).*light)) / sum(sum(light));
lightMeanRed = sum(sum(img(:,:,1).*light)) / sum(sum(light));
lightMeanBlue = sum(sum(img(:,:,3).*light)) / sum(sum(light));



out = img;

intensityDiffRed = lightMeanRed - shadowMeanRed;
intensityDiffGreen = lightMeanGreen - shadowMeanGreen;
intensityDiffBlue = lightMeanBlue - shadowMeanBlue;

                out(:, :, 1) = img(:, :, 1) + smoothMask * intensityDiffRed;
                out(:, :, 2) = img(:, :, 2) + smoothMask * intensityDiffGreen;
                out(:, :, 3) = img(:, :, 3) + smoothMask * intensityDiffBlue;

imwrite(out, 'shadowRemoved.bmp');
```

SCRIPT USED FOR THESIS OUTPUT AND COMPARISON

```
function out = thesisOut(img)

imgSize = size(img);
```

```
gray = rgb2gray(img);
imwrite(gray, 'gray.bmp');

tradCan = edge(gray, 'canny');
imwrite(tradCan, 'Canny.bmp');

sob = edge(gray, 'sobel');
imwrite(sob, 'sobel.bmp');

marrHildreth = edge(gray, 'log');
imwrite(sob, 'marrHildreth.bmp');

mask = 1 - double(im2bw(gray, graythresh(gray)));
imwrite(mask, 'greyMask.bmp');

maskCan = edge(mask, 'canny');
imwrite(maskCan, 'shadowMaskCanny.bmp');

strel = [0 1 1 1 0; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 0 1 1 1 0];

shadow = imerode(mask, strel);
imwrite(shadow, 'shadowMap.bmp');

shadowCan = edge(shadow, 'canny');
imwrite(shadowCan, 'shadowCanny.bmp');
```

```
light = imerode(1-mask, strel);
imwrite(light, 'lightmap.bmp');

lightCan = edge(light, 'canny');
imwrite(lightCan, 'lightCanny.bmp');

shadowSum = 0;
lightSum = 0;

smoothMask = conv2(mask, strel/21, 'same');

shadowMeanRed = sum(sum(img(:,:,1).* shadow)) / sum(sum(shadow));
shadowMeanGreen = sum(sum(img(:,:,2).* shadow)) / sum(sum(shadow));
shadowMeanBlue = sum(sum(img(:,:,3).* shadow)) / sum(sum(shadow));

lightMeanGreen = sum(sum(img(:,:,2).*light)) / sum(sum(light));
lightMeanRed = sum(sum(img(:,:,1).*light)) / sum(sum(light));
lightMeanBlue = sum(sum(img(:,:,3).*light)) / sum(sum(light));


out = img;

intensityDiffRed = lightMeanRed - shadowMeanRed;
intensityDiffGreen = lightMeanGreen - shadowMeanGreen;
intensityDiffBlue = lightMeanBlue - shadowMeanBlue;

                out(:, :, 1) = img(:, :, 1) + smoothMask * intensityDiffRed;
                out(:, :, 2) = img(:, :, 2) + smoothMask * intensityDiffGreen;
```

```
                out(:, :, 3) = img(:, :, 3) + smoothMask * intensityDiffBlue;

imwrite(out, 'shadowRemoved.bmp');

cannyEdge(out);
```