

Feasibility Study of a Moon Sensor for Satellite Attitude Determination

R.E. Skinner



Thesis presented in partial fulfillment of the requirements for the
degree of
Master of Science in Electronic Engineering
at the
University of Stellenbosch

Study Leader: Mr. J. Treurnicht

December 2005

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously been submitted at any university, in part or in its entirety, for a degree.

Abstract

The purpose of this study was to investigate the feasibility of a moon sensor as an alternative or supplemental sensor to supply attitude information to a satellite. The visibility of the moon was evaluated with regards to that of the sun from a satellite (i.e. the feasibility of the moon sensor is compared to that of a sun sensor).

An algorithm was developed to calculate the center of the moon, regardless of phase or rotation, to offer increased accuracy of the center of the moon. The accuracy of this algorithm and the possible error margins were evaluated and discussed in both ideal and actual test cases. The sensor was implemented on embedded hardware.

The conclusion of the feasibility analysis was that the sensor would function well as a supplemental sensor (e.g. in conjunction with a sun-sensor) rather than as an alternative sensor. The accuracy of the moon center algorithm was satisfactory for attitude determination.

This sensor could thus be seriously considered for use on a future satellite.

Samevatting

Die doel van die studie is om die lewensvatbaarheid van 'n maan-sensor, as alternatiewe of aanvullende sensor, om oriëntasie data aan 'n sateliet te verskaf, te ondersoek. Die sigbaarheid van die maan en die sigbaarheid van die son is met mekaar vergelyk, vanaf 'n sateliet (m.a.w. die uitvoerbaarheid van 'n maan-sensor is vergelyk met dié van 'n son sensor).

'n Algoritme is ontwikkel om die middelpunt van die maan, ongeag van fase of rotasie, te bereken om verhoogde akuraatheid van die maan se middelpunt te bied. Die akuraatheid van die maan-middelpunt algoritme en moontlike foute is evalueer en bespreek in beide ideale sowel as werklike toets gevalle. Die sensor was op hardeware geïmplementeer.

Die gevolgtrekking van die lewensvatbaarheids analise is dat die sensor effektief sal funksioneer as 'n aanvullende sensor (d.w.s saam met 'n son sensor) eerder as 'n alternatiewe sensor. Die resultate verkry van die evaluasie van die maan-middelpunt algoritme is bevredigend en is voldoende vir oriëntasie bepaling.

Die sensor kan dus ernstig oorweeg word vir gebruik op 'n sateliet in die toekoms.

Acknowledgements

I would like to thank the following people for their help, support and contributions throughout this project:

- My study leader, Mr. J. Treurnicht for his patience, guidance, motivation and invaluable advice throughout this thesis.
- My colleagues at the ESL for their assistance.
- All my friends for their unbelievable support and motivation regardless of how tough the circumstances.
- My roommate JP Campher for the much needed comic relief, fun filled breaks and support.
- Marthélize 'Goose' Tredoux for her invaluable help, concern, support and selfless personal sacrifice in bringing this thesis to completion. Thank you so very much!
- My parents for their unconditional love, support and understanding.
- My Heavenly Father for carrying me through all of life.

Table of Contents

Chapter 1 Introduction.....	1
1.1 Background.....	1
1.2 Objectives/overview.....	1
1.3 Thesis layout.....	2
Chapter 2 Background.....	3
2.1 Introduction	3
2.2 Orbit Parameters	4
2.2.1 Satellite Orbit Parameters	4
2.2.2 Sun Orbit Parameters.....	6
2.2.3 Moon Orbit Parameters.....	8
2.3 Visual Properties of Brightest Objects in Space	9
2.4 Coordinate System.....	10
2.5 Moon Image Processing.....	11
Chapter 3 Sensor Feasibility Study	15
3.1 Introduction	15
3.2 Orbit Analysis/Feasibility of Sensor.....	15
3.2.1 Satellite Orbit Analysis	16
3.2.2 Moon Orbit Analysis.....	19
3.2.3 Feasibility Analysis.....	21
3.2.4 Additional Information on Moon Visibility Accuracy	27
3.3 Algorithms	28
3.3.1 Moon Center Algorithm	28
3.3.2 Moon Position Algorithm	38

Chapter 4	Hardware and Software Set_Up	40
4.1	Introduction	40
4.2	Hardware Overview (Implementation)	40
4.2.1	DSP Evaluation Board (Sensor Processor Board)	40
4.2.2	Sensor Camera	44
4.2.3	Physical Test Configuration	46
4.3	Software Implementation	48
4.3.1	Hardware Initialization	48
4.3.2	Basic Sensor Software Structure	49
4.3.3	Image Processing - Software Implementation	50
4.3.4	Communications Software Structure	57
Chapter 5	Simulations and Test results	59
5.1	Evaluation Procedure Overview	59
5.2	Moon Center Algorithm	60
5.2.1	Results for Varying Moon Image Size (Using Matlab)	60
5.2.2	Results for Varying Moon Image Rotation (Using Matlab)	62
5.2.3	Results for Varying Moon Position within Image (Using Matlab)	63
5.2.4	Algorithm Results for Actual Moon Images (Using Matlab)	64
5.3	Moon Center Error and Accuracy Evaluation	65
5.3.1	Algorithm Error Evaluation on a Full Moon	65
5.3.2	Algorithm Error Evaluation on Other Phases	66
5.3.3	Algorithm Error Evaluation for Images Taken by Actual Sensor	69
Chapter 6	Conclusions and Recommendations	72
6.1	Summary	72
6.1.1	Sensor Feasibility with Moon as Target (Summary)	72
6.1.2	Moon Position Algorithm	73
6.1.3	Moon Center Algorithm	74
6.2	Conclusion, Considerations and Recommendations	74

Appendix A	Matlab Implementation of Moon Center Algorithm	78
Appendix B	DSP Implementation of Moon Center Algorithm.....	88
Appendix C	Matlab Implementation of Moon and Sun Position Algorithms ...	107
Appendix D	C++ Implementation of PC Communication with DSP	111

List of Figures

Figure 2.1 Orbit Parameters for an Elliptical Orbit	5
Figure 2.2 Keplerian Orbital Elements of a Satellite in an Elliptical Orbit	6
Figure 2.3 Geocentric Inertial Coordinate System	11
Figure 2.4 Locating of Points on Moon Border	13
Figure 3.1 Satellite Orbit.....	17
Figure 3.2 Obstruction of Earth to Satellite LOS	18
Figure 3.3 Satellite-Moon Elevation and Rotation Angles.....	19
Figure 3.4 Moon Orbit and Phases.....	20
Figure 3.5 Area around New Moon (Sun and Moon would fall in Sensor FOV Simultaneously)	21
Figure 3.6 Polar Region of Satellite Orbit.....	22
Figure 3.7 Visibility of Moon during Eclipse Region of Satellite Orbit.....	23
Figure 3.8 Regions in which Moon and Sun are Visible to the Satellite.....	24
Figure 3.9 Regions of Moon Orbit where it is visible for Full Satellite Orbit.....	25
Figure 3.10 Moon Phase Relative to Degrees from New-Moon.....	26
Figure 3.11 Actual Satellite and Moon Orbits	27
Figure 3.12 Fitting of Circle to Moon Image Resulting in Center Coordinate	29
Figure 3.13 Results of Different Spacing Of selected Points on Moon border.....	31
Figure 3.14 Determining Points at x and y Extremities.....	32
Figure 3.15 Fitting of Different Size Circle through same Two Points	33
Figure 3.16 Determining Points at xy and yx Extremities.....	34
Figure 3.17 Calculation and Selection of Longest Bisecting Line	35
Figure 3.18 Calculation and Selection of Point on Actual Moon Border	36
Figure 3.19 Final Selection of Three points on Actual Moon Border.....	37

Figure 3.20 Calculation of Moon Center	38
Figure 4.1 Format of Image as Downloaded to SDRAM.....	42
Figure 4.2 Picture of the ADSP-BF533 EZKit-Lite Evaluation Board.....	43
Figure 4.3 Flow Diagram of Evaluation Board.....	43
Figure 4.4 Image Taken to Calculate Calibrate the Camera's FOV.....	44
Figure 4.5 Picture of Over Exposed Moon.....	45
Figure 4.6 Picture of Correctly Exposed Moon.....	45
Figure 4.7 Picture of Camera used for sensor.....	46
Figure 4.8 Flow Diagram of Physical Test Configuration.....	47
Figure 4.9 Picture of Entire Configuration.....	47
Figure 4.10 Flow Diagram of Sensor Software Structure.....	49
Figure 4.11 Optimization of Image Processing Code.....	51
Figure 4.12 Basic Image Processing Algorithm Flow Diagram.....	52
Figure 4.13 Evaluation of Possible Edge Pixels	53
Figure 5.1 Large Moon Images with Average Radius of 227.24 pixels	61
Figure 5.2 Small Moon Images with Average Radius of 13.08 pixels.....	61
Figure 5.3 Moon Image Rotated in 5° Increments from 0° to 45°	62
Figure 5.4 Moon Image Rotated in 45° increments from 0° to 315°	63
Figure 5.5 All Positions of Moon within Image Tested	63
Figure 5.6 Image of Perfect Circle Showing Distortion by Sensor.....	64
Figure 5.7 Actual Moon Images as Taken by Sensor.....	65
Figure 5.8 Large Moon Images (Avg. Radius 227.24 pixels).....	67
Figure 5.9 Small Moon Images (Avg. Radius 13.08 pixels).....	67
Figure 5.10 Ideal Moon Images with Exact Same Radius (Radius 38.64 pixels)	69
Figure 5.11 Perfect Circle Predicted and Calculated Centers.....	69
Figure 5.12 Actual Moon Images (Avg. Radius 15.03 pixels).....	71

List of Tables

Table 2.1 Apparent Magnitude of Brightest Heavenly Bodies	10
Table 3.1 Summary of Moon Visibility to Satellite.....	26
Table 5.1 Full Moon Center Predictions vs. Algorithm Center Calculation.....	66
Table 5.2 Distortion Results of Calculated Center of Perfect Circle.....	70

List of Abbreviations

LEO	Low Earth Orbit
LOS	Line of Sight
FOV	Field of View
GCI	Geocentric Inertial Coordinates
MOI	Moment of Inertia
AOS	Axis of Symmetry
AU	Astronomical Units
RE	Earth Radii

List of Symbols

General Orbit and Space Environment

a	Semimajor Axis of Elliptical Orbit
b	Semiminor Axis of Elliptical Orbit
r_A	Radius of Apogee
r_P	Radius of Perigee
e	Eccentricity of Orbit
i	Inclination of Orbit
Ω	Right Ascension of the Ascending Node of the Orbit
ω	Argument of Perigee
v	True Anomaly of Orbit
M	Mean Anomaly of Orbit
r	Radius from Barycenter to the Orbit
P	Orbital Period
R_E	Radius of Earth
t	Time
X_c, Y_c, Z_c	Satellite Position on Geocentric Inertial Coordinates
μ	Earth's Gravitational Constant

Sun and Moon Orbit Parameters

$\mathbf{r}_{Sun}, \mathbf{r}_{Moon}$	Vectors to Sun and Moon in Geocentric Coordinates
r_{Sun}, r_{Moon}	Radial Distances to Sun and Moon respectively
ε	Mean Obliquity of the Ecliptic
$\lambda_{Sun}, \lambda_{Moon}$	True Geometric Longitudes of Sun and Moon
$\lambda_{ecliptic}$	Ecliptic Longitude

ϕ_{ecliptic}	Ecliptic Latitude
L_{Sun}	Mean Geometric Longitude of Sun
$u_{M_{\text{Moon}}}$	Moon's Mean Argument of Latitude
D_{Sun}	Mean Elongation of Sun
φ	Parallax
T	Julian Centuries from Epoch J2000
JD	Julian Day of Observation

Moon Image Processing

x, y	Coordinate Position within Moon image
x_c, y_c	Coordinate Position of Center of Moon
r	Radius of Moon

Chapter 1

Introduction

1.1 Background

On any given satellite, sensors are required to determine the attitude. The most common of these sensors are: sun sensors, star sensors, horizon sensors and magnetometers. This study considers a moon sensor as an alternate or supplemental sensor to supply attitude information to a satellite which can be used for attitude determination and control.

1.2 Objectives/overview

In this thesis the following objectives were investigated:

1. The feasibility of a moon sensor was considered with regards to the visibility of the moon from a satellite in Low Earth Orbit (LEO).
2. Algorithms were implemented to be used for calculating the position of the moon relative to the sun and the satellite. The moon position algorithm would also be used by the sensor to acquire the moon for the purpose of imaging.
3. An algorithm was developed to process an image of the moon and obtain the center coordinate thereof regardless of its phase or rotation. The algorithm resulted in greater accuracy for a sensor using the moon as reference; an error analysis was then carried out. The coordinate obtained by this algorithm was passed on to the satellite.

The aim was to evaluate the sensor for consideration of implementation in a satellite system in future studies.

1.3 Thesis layout

Chapter 2: Introduction of the mathematical models describing the orbits of the satellite, the moon and the sun. The coordinate system used in this thesis and an initial consideration for a moon center processing algorithm is also described.

Chapter 3: The process of the feasibility study and its results are discussed. The development and explanation of the moon center algorithm is fully described. Factors pertaining to the moon position algorithm, including camera FOV and orbital rates of the satellite and moon, are mentioned.

Chapter 4: Discusses the hardware with which the sensor was implemented and the physical test setup is portrayed. The software structure with regards to sensor hardware initialization, communications between sensor and computer and image processing algorithm is also described.

Chapter 5: Contains simulations and test results of the image processing algorithm as well as error evaluations of these results.

Chapter 6: Discusses the results obtained in this thesis and expands on certain points not specifically covered in other chapters.

Chapter 2

Background

2.1 Introduction

This chapter describes the satellite, moon and sun's orbit details so as to determine the moon's visibility from the satellite. This is necessary since each of these factors play a direct role in the visibility of the moon from the satellite.

The orbital parameters of the satellite are obtained from equations defined in Wertz & Larson [1, page 132 – 140] and are detailed in section 2.2.1. These parameters are used to calculate the coordinate position of the satellite in the geocentric coordinate system. This coordinate position can be used, along with the positions (in the same coordinate system) of the moon (Section 2.2.2) and sun (Section 2.2.4), to evaluate the moon's visibility from the satellite.

Section 2.3 lists the visual magnitudes of some of the brightest heavenly bodies. Section 2.4 discusses the coordinate system used throughout this text, while section 2.5 looks at the background regarding the sensor image processing.

2.2 Orbit Parameters

2.2.1 Satellite Orbit Parameters

To fix the satellite position about the earth, the orbit elements which fully define the motion of the satellite, are required (Wertz & Larson [1, page 132 – 140]).

The orbital elements are as follows (Refer to Figure 2.1 and Figure 2.2 (from Wertz & Larson [1, page 136, Fig. 6-3])):

- *Semimajor axis, a* : the distance that describes the size of the ellipse of the orbit and is calculated using the radius of perigee, r_p and the radius of apogee, r_A :

$$a = \frac{r_A + r_P}{2} \quad (2.1)$$

- *Eccentricity, e* : the value describing the shape/flattening of the ellipse. Described as the ratio between the semimajor axis, a , and the semiminor axis, b , and defined as:

$$e = \left(\frac{r_A}{a} \right) - 1 \quad (2.2)$$

The orbit is elliptical for $e \neq 0$ and the orbit is circular for $e=0$.

- *Inclination, i* : the angle between the orbital plane and the equatorial plane
- *Right ascension of the ascending node, Ω* : the angle measured (as a right-handed rotation about the Z-axis) from the *vernal equinox*¹ to the *ascending node*.²
- *Argument of perigee, ω* : the angle as measured from the ascending node to the *eccentricity vector*³ in the same direction as the satellite motion.

¹ The point where the earth's orbital plane about the sun cuts the equatorial-plane going from south to north.

² The point where the satellite's orbital plane cuts the equatorial-plane from south to north.

³ The vector that points from the center of the earth to perigee, with magnitude equal to the eccentricity or the orbit.

- *True anomaly, v* : reflects where the satellite is at any specific time, t , and is shown as the angle from the eccentricity vector to the vector indicating the position of the satellite, calculated (approximately) as follows:

$$v \approx M + 2e \sin M + 1.25e^2 \sin(2M) \quad (2.3)$$

Where M (*Mean anomaly*) is: $360\left(\frac{\Delta t}{P}\right)$ with Δt the time since the satellite last passed through perigee and P the orbital period.

- *Radius, r* : the distance from the barycenter to the orbit at any time, calculated as follows:

$$r = a \frac{1 - e^2}{1 + e \cos v} \quad (2.4)$$

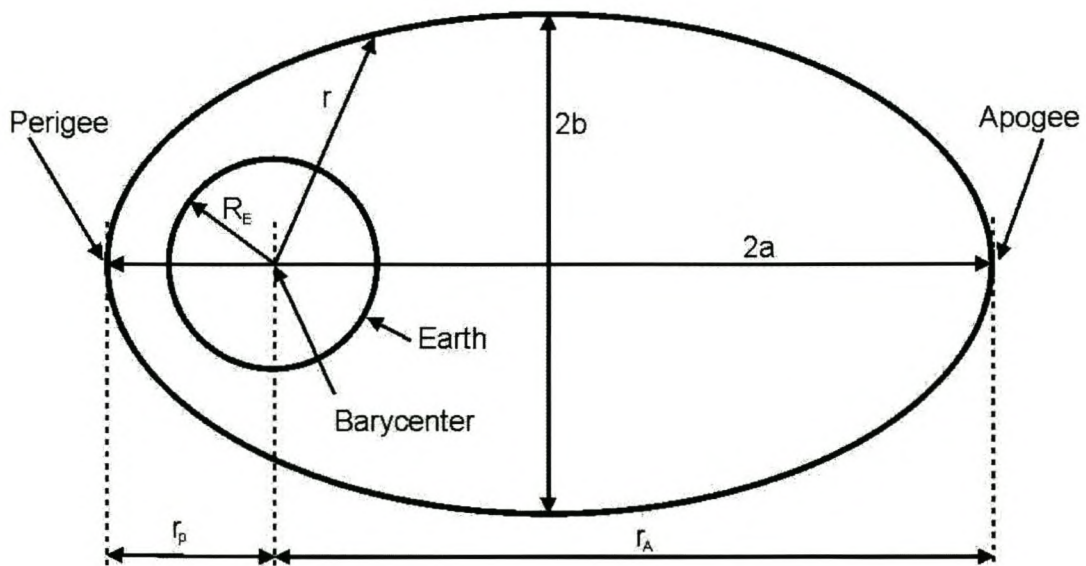


Figure 2.1 Orbit Parameters for an Elliptical Orbit

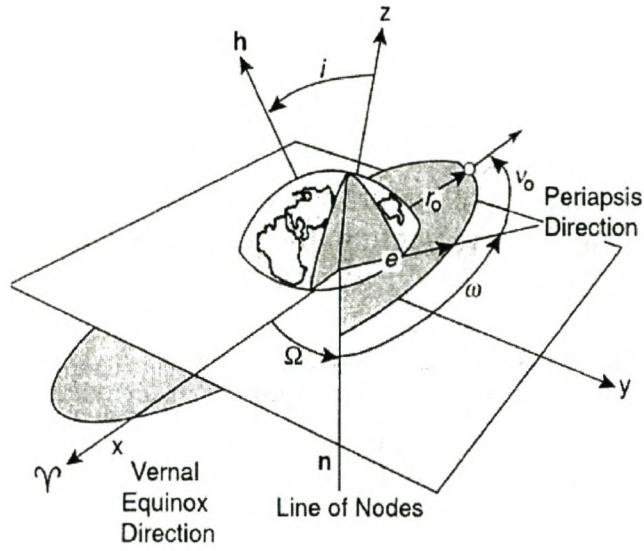


Figure 2.2 *Keplerian Orbital Elements of a Satellite in an Elliptical Orbit (GCI coordinate frame)*

The satellite's position in the Geocentric Inertial (GCI) coordinate system at specified time, t , can be calculated using the following equations (see Wertz [7, page 135]):

$$X_c = r[\cos(\omega + \nu) \cos \Omega - \sin(\omega + \nu) \sin \Omega \cos i], \quad (2.5)$$

$$Y_c = r[\cos(\omega + \nu) \sin \Omega + \sin(\omega + \nu) \cos \Omega \cos i], \quad (2.6)$$

$$Z_c = r[\sin(\omega + \nu) \sin i], \quad (2.7)$$

2.2.2 Sun Orbit Parameters

The sun's position in the GCI coordinate system can be accurately calculated to within 0.01° by using the following formula (Meeus [6, page 151]).

$$\mathbf{r}_{Sun} = r_{Sun} \begin{bmatrix} \cos(\lambda_{Sun}) \\ \cos(\varepsilon)\sin(\lambda_{Sun}) \\ \sin(\varepsilon)\sin(\lambda_{Sun}) \end{bmatrix} AU \quad (2.8)$$

The following definitions apply:

Mean obliquity of the ecliptic

$$\varepsilon = 23.4392911 - 0.0130041.T - 1.64 \times 10^{-7}.T^2 + 5.04 \times 10^{-7}.T^3$$

True geometric longitude

$$\lambda_{Sun} = L_{Sun} + C$$

Mean geometric longitude

$$L_{Sun} = 280.46645 + 36000.76983.T + 0.0003032.T^2$$

Mean anomaly

$$M_{Sun} = 357.52910 + 35999.05030.T - 0.0001559.T^2 - 0.00000048.T^3$$

Equation of center

$$C = +\left(1.914600 - 0.004817.T - 0.000014.T^2\right)\sin M \\ + \left(0.019993 - 0.000101.T\right)\sin 2M + 0.000290\sin 3M$$

Eccentricity of earth's orbit

$$e = 0.016708617 - .0000042037.T - .0000001236.T^2$$

Sun's true anomaly

$$v = M_{Sun} + C$$

Radial distance from earth to sun (Astronomical units)

$$r_{Sun} = \frac{1.000001018(1 - e^2)}{1 + e \cos v}$$

Julian centuries from epoch J2000

$$T = \frac{JD - 2451545.0}{36525}$$

JD=Julian day of observation

2.2.3 Moon Orbit Parameters

The complex motion of the moon makes an accurate calculation of its position computer intensive. The following algorithm, acquired from the Astronomical Almanac, is sufficiently accurate (to 10 arc minutes) for calculating the position of the moon in GCI coordinates. The result is in terms of earth radii (ER).

$$\mathbf{r}_{Moon} = r_{Moon} \begin{bmatrix} \cos(\phi_{ecliptic}) \cos(\lambda_{ecliptic}) \\ \cos(\varepsilon) \cos(\phi_{ecliptic}) \sin(\lambda_{ecliptic}) - \sin(\varepsilon) \sin(\phi_{ecliptic}) \\ \sin(\varepsilon) \cos(\phi_{ecliptic}) \sin(\lambda_{ecliptic}) + \cos(\varepsilon) \sin(\phi_{ecliptic}) \end{bmatrix} ER \quad (2.9)$$

The following definitions apply:

Ecliptic longitude

$$\begin{aligned} \lambda_{ecliptic} = & \lambda_{Moon} + 6.29^\circ \cdot \sin(M_{Moon}) - 1.27^\circ \cdot \sin(M_{Moon} - 2D_{Sun}) \\ & + 0.066^\circ \cdot \sin(2D_{Sun}) + 0.21^\circ \cdot \sin(2M_{Moon}) - 0.019^\circ \cdot \sin(M_{Sun}) - 0.11^\circ \cdot \sin(2u_{M_{Moon}}) \end{aligned}$$

Moon's longitude

$$\lambda_{Moon} = 218^\circ.3165 + 481267^\circ.8813.T$$

Ecliptic latitude

$$\phi_{\text{ecliptic}} = 5.13^\circ \cdot \sin(u_{M_{\text{Moon}}}) + 0.28^\circ \cdot \sin(M_{\text{Moon}} + u_{M_{\text{Moon}}}) - 0.28^\circ \cdot \sin(u_{M_{\text{Moon}}} - M_{\text{Moon}}) - 0.17^\circ \cdot \sin(u_{M_{\text{Moon}}} - 2D_{\text{Sun}})$$

Moon's mean anomaly ($k=360^\circ$)

$$M_{\text{Moon}} = 134^\circ.9729814 + (1325k + 198^\circ.867398)T + 0.0086972^\circ \cdot T^2 + 1.778^\circ \times 10^{-5} \cdot T^3$$

Moon's mean argument of latitude

$$u_{M_{\text{Moon}}} = 93^\circ.2719103 + (1342k + 82^\circ.0175381)T - 0.0036825^\circ \cdot T^2 + 3.06^\circ \times 10^{-6} \cdot T^3$$

Mean elongation of the Sun

$$D_{\text{Sun}} = 297^\circ.8503631 + (1236k + 307^\circ.111480)T - 0.00191417^\circ \cdot T^2 + 5.28^\circ \times 10^{-6} \cdot T^3$$

Radial distance from the earth to the moon (in earth radii units)

$$r_{\text{Moon}} = \frac{1}{\sin(\varphi)}$$

Parallax

$$\varphi = 0.9508^\circ + 0.0518^\circ \cos(M_{\text{Moon}}) + 0.0095^\circ \cos(M_{\text{Moon}} - 2D_{\text{Sun}}) + 0.0078^\circ \cos(2D_{\text{Sun}}) + 0.0028^\circ \cos(2M_{\text{Moon}})$$

2.3 Visual Properties of Brightest Objects in Space

The visual magnitudes of the brightest objects in space that could influence the performance of a moon sensor are listed in the table below (Values obtained from Wertz [7, page 813] and Kaler [9] :

<u>Object</u>	<u>Apparent Magnitude</u>
Sun	-26.74
Moon	-12.73
Venus	-4.22
Sirius	-1.46

Table 2.1 *Apparent Magnitude of Brightest Heavenly Bodies*

2.4 Coordinate System

The Geocentric Inertial (GCI) coordinate system (Wertz & Larson [1, page 96 – 97, 135]) is used throughout this text (See Figure 2.3, as from Wertz & Larson [1, page 97, Fig. 5-1, D]). This system's X -axis points from the center of the earth to the first point of Aries (i.e. the position of the sun at the vernal equinox). This direction is the intersection of the earth's equatorial plane and the ecliptic plane. The Z -axis is parallel to the rotation axis of the earth and Y -axis completes the right-handed orthogonal set.

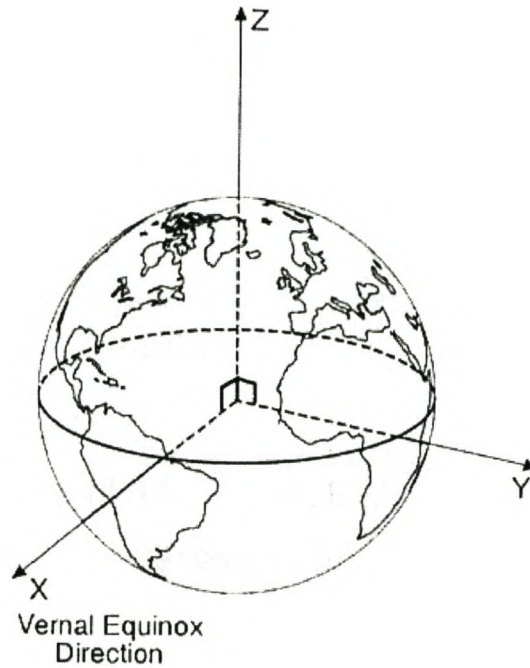


Figure 2.3 *Geocentric Inertial Coordinate System*

This coordinate system changes slowly in time, owing to the effects of astronomical precession and the nutation of the earth's rotation axis. The coordinate system can be seen as sufficiently inertial for this application and J2000 coordinates are used.

2.5 Moon Image Processing

A paper regarding moon image processing for attitude estimation in a satellite application (Bellezza, Mortari, Perfetti [2]), was studied as a starting point. Data from that paper, relevant to the project, is summarized below:

The sensor is based on the fact that the same face of the moon is always facing towards the earth. It is thus possible to compare reference images to acquired images and so deduce attitude information of the satellite

The main objectives were to acquire the center of the moon regardless of phase (and therefore also the satellite-moon vector) and to calculate the phase. The acquired image is then compared to a reference image of the same phase and rotated until the two images overlap. From this the rotation of the satellite about the satellite-moon axis is deduced (phase angle). Only the image processing for the moon center will be discussed further since it is the only information relevant to this thesis.

- The edge of the moon image is extracted by applying a 1-dimensional convolution of the image with a Derivative of a Gaussian impulse response $h(x)$. This convolution is performed separately in the horizontal and vertical directions.

$$h(x) = g(x) * \delta(x) \quad (2.10)$$

$$g(x) = \exp\left(\frac{-x^2}{2\sigma^2}\right) \quad (2.11)$$

$$\delta(x) = \frac{dg(x)}{dx} = -\frac{x}{\sigma^2} \exp\left(\frac{-x^2}{2\sigma^2}\right) \quad (2.12)$$

An edge linking procedure is applied to the acquired pixels, which results in a moon border.

- In the case of full moon, the center of the moon was acquired by performing a least-squares estimation with given radius (radius is assumed to be known).
- In the case when the moon is not full, the image is separated into two parts: the actual moon edge (circular part) and the part deformed by shadow (terminator). The moon edge is filled with white and the image is treated as a uniform plane mass distribution: white pixels have mass one and black pixels have mass zero. The main axes of inertia are calculated by applying a tensor of inertia. As it would be, the axis of symmetry (AOS) has the largest moment of inertia (MOI); this fact along with the calculation of the center of gravity is used to find the AOS (See Figure 2.4(a)).
- Two lines parallel to the AOS, at a distance of one third the radius of the moon, are constructed on either side of the AOS. These three lines intersect both the

circular part and terminator of the moon image resulting in three points on each of the two parts of the moon edge (See Figure 2.4(b)).

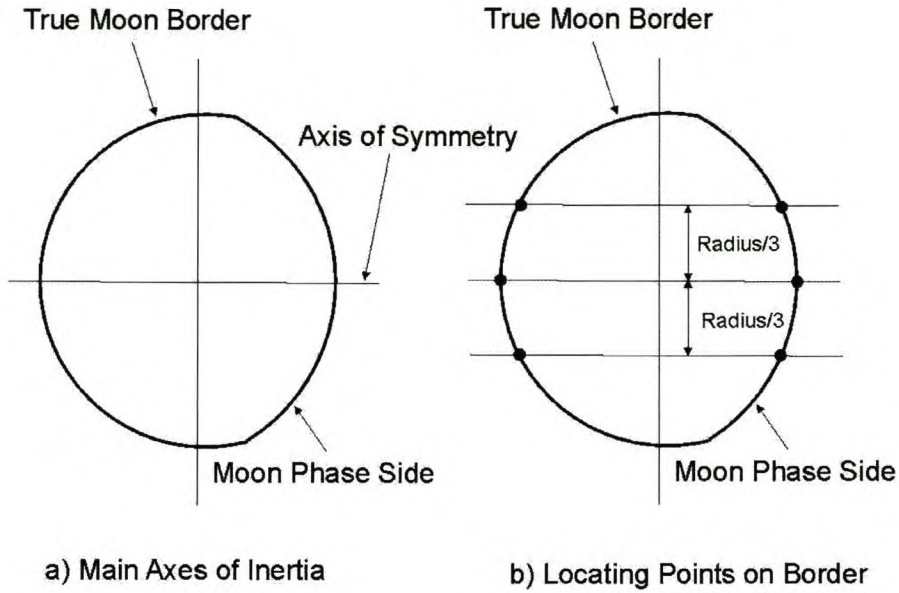


Figure 2.4 *Locating of Points on Moon Border*

- Each of these two groups of three points belongs to a separate circle. The radius of each of these circles is now calculated and compared to the expected radius. The value closest to this is selected and the three points related to it are assumed as lying on the correct edge of the moon and the center is calculated. The equations used are as follows:

Equation of a circle:

$$x^2 + y^2 + ax + by + c = 0 \quad (2.13)$$

The coordinates of the 3 points on each curve $(x_1, y_1); (x_2, y_2); (x_3, y_3)$ are set into the equation for a circle resulting in 3 equations which are then solved for a, b, c :

$$x_1^2 + y_1^2 + ax_1 + by_1 + c = 0 \quad (2.14)$$

$$x_2^2 + y_2^2 + ax_2 + by_2 + c = 0 \quad (2.15)$$

$$x_3^2 + y_3^2 + ax_3 + by_3 + c = 0 \quad (2.16)$$

The resultant values for a , b , c are now used in the following equations to calculate the center of each of the 2 circles and their radii:

$$x_c = \frac{-a}{2} \quad (2.17)$$

$$y_c = \frac{-b}{2} \quad (2.18)$$

$$r = \sqrt{x_c^2 + y_c^2 - c} \quad (2.19)$$

- All pixels within the newly found circle (terminator pixels) are now deleted leaving only the circular part. A least squares estimation is now applied to the remaining pixels resulting in a more accurate moon center estimate.

Ultimately, only the concept of finding three points on the actual moon border (circular part) was employed and also the equations required in calculating the moon center and radius from these three points. The result is that moon phase and rotation are not specified or calculated and rotation information of the satellite about the satellite-moon axis is lost.

Chapter 3

Sensor Feasibility Study

3.1 Introduction

This chapter discusses the process and decisions regarding the feasibility of a moon sensor for attitude determination on a satellite. The orbits of the satellite and moon, and the development of moon center and position algorithms are expanded on.

3.2 Orbit Analysis/Feasibility of Sensor

The satellite and moon orbit the earth in different planes and with different orbit velocities. The moon is therefore not visible to the satellite's sensor at all times, due to obstruction by the earth. The sensor is calibrated (by adjusting the aperture until the moon image is correctly exposed) to detect an object's brightness equivalent to that of the moon. In situations when both the moon and sun fall into the sensor's field of view (FOV), it is impossible to obtain an image of the moon due to the sun saturating the sensor. The sun is approximately 401791 times brighter than the full moon. Another factor to take into consideration would be the influence of planets and stars on the sensor, this equates to the 'noise level' as seen by the sensor. The apparent magnitudes of the brightest planet, Venus, and the brightest star, Sirius, are approximately 2535 and 32211 times, respectively, less bright than the moon. (Refer to Table 2.1 for brightness levels of each respective object) As a result the Signal (moon image brightness) to Noise

(Planet/Star brightness) ratio is very large and planets or stars will have no effect on the sensor as their brightness is far below the sensitivity level of the sensor.

Consequently it is necessary to calculate the percentage of the moon's visibility. This evaluation is done for all positions of the satellite in its orbit and also for all positions of the moon throughout its orbit. In the following sections all orbit details/assumptions are presented and discussed pertaining to the orbits of both the satellite and moon, and the position of the sun.

3.2.1 Satellite Orbit Analysis

Before any algorithms can be developed for calculating the moon's center, an evaluation of the moon's visibility to the sensor must be performed. This requires that an orbit for the particular satellite be decided upon.

The satellite used as example in this study is a micro-satellite. The orbit selected for this satellite is a sun-synchronous, low earth orbit (LEO).

Satellite orbit details (Figure 3.1):

- LEO (~550km above earth's surface)
- Sun-synchronous (orbital plane at $\sim 30^\circ$ to sun)
- Satellite orbital plane approximately orthogonal to the moon's orbital plane.

From the orbit radius, $a = 6378km + 550km = 6928km$ (earth's radius + Orbit height) the orbital period can be calculated using Equation (3.1) (Wertz & Larson [1, pg 137]):

$$P = 2\pi \left(\frac{a^3}{\mu} \right)^{\frac{1}{2}} = 95.65 \text{ min} \quad (3.1)$$

Where μ is the earth's gravitational constant ($= 398600.5 \text{ km}^3 \text{ s}^{-2}$)

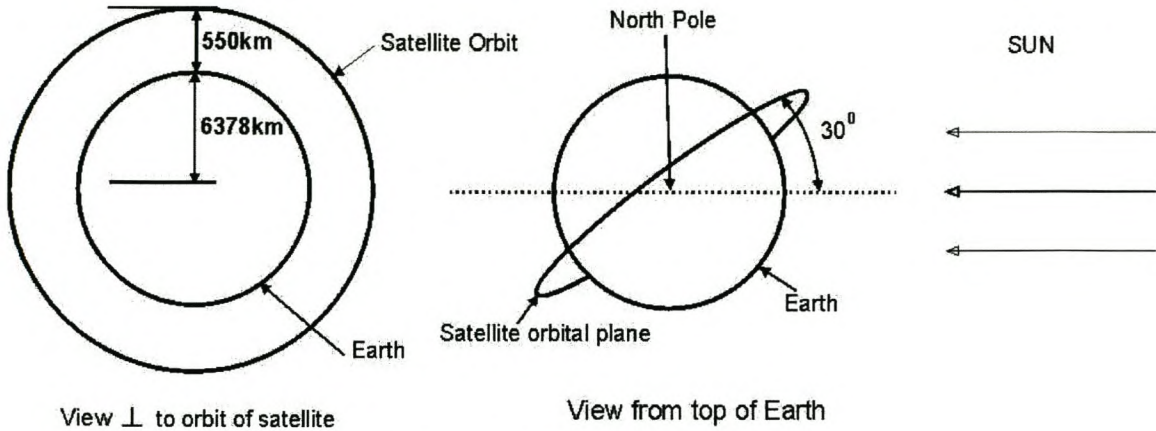


Figure 3.1 *Satellite Orbit*

As the satellite orbits the earth, the only object that obstructs its line of sight (LOS) will be the earth (there will be time periods when the earth will be between the satellite and the moon, making the sensor ineffective). The orbital details of the satellite were used to calculate the obstruction of the earth to the satellite's LOS. The resultant obstruction is calculated using basic geometry as illustrated in Figure 3.2, and Equation (3.2).

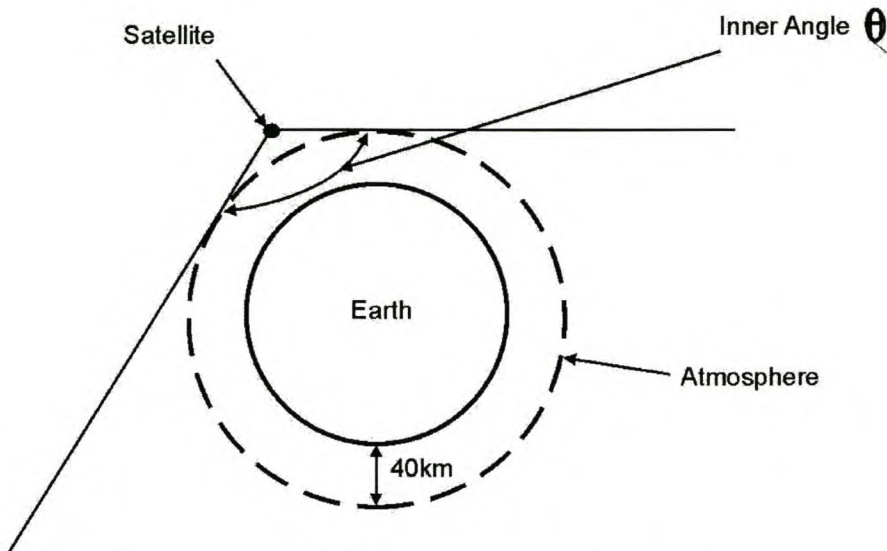


Figure 3.2 *Obstruction of Earth to Satellite LOS*

The earth's maximum radius is 6378km and the height of the atmosphere is assumed as approximately 40km (~6418km total radius):

$$\theta = 2 \sin^{-1} \left(\frac{6418}{6928} \right) \approx 136^{\circ} \quad (3.2)$$

The resultant obstruction is a cone with approximate inner angle of $\theta = 136^{\circ}$

With the satellite's orbit sufficiently defined, a study of the moon's orbital details is necessary for further evaluation of the moon sensor's feasibility.

It is assumed that the satellite keeps the same side (solar panel) facing the sun during the non-eclipse period of its orbit and, to maximize energy, flips around quickly during the eclipse period so as to have the solar panel correctly positioned as the satellite exits the eclipse period. Throughout the majority of the orbit the elevation angle (angle between the moon orbital plane and the satellite) will vary between approximately $+1^{\circ}$ and -1° (refer to Figure 3.3a). As the moon orbits the earth the rotation angle (angle between the satellite-orbital plane and the moon) varies between 0° and 360° (refer to Figure 3.3b). If a

sensor with a FOV of $<15^\circ$ is employed, the sensor could remain inertially fixed with regards to elevation. It would, however, have to be inertially maneuverable in rotation, or a number of sensors would have to be situated around the satellite to accommodate for change in rotation angle.

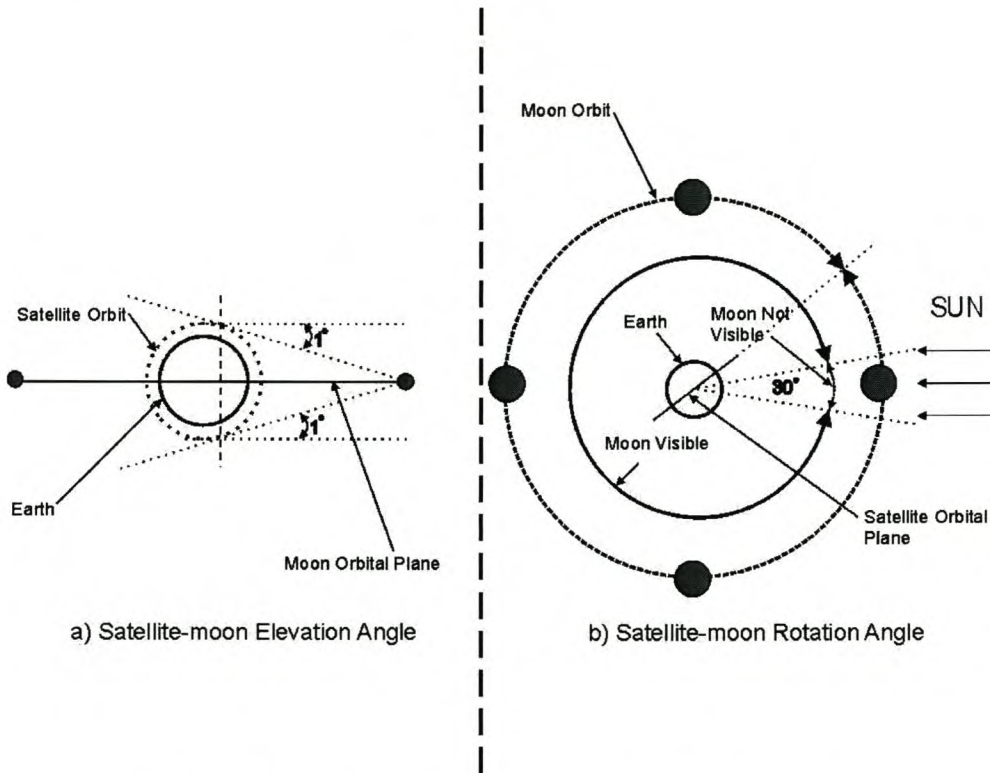


Figure 3.3 *Satellite-Moon Elevation and Rotation Angles*

3.2.2 Moon Orbit Analysis

The moon orbits the earth in a complex fashion. However this can (for the sake of this feasibility study) be simplified to a circular orbit with average distance from the earth center at approximately 384 467 km, and an orbital period of 29.53 days. This orbital period is known as the synodic period (This orbital period is used throughout the visibility evaluation) and represents the time from one new-moon to the next (See Figure 3.4).

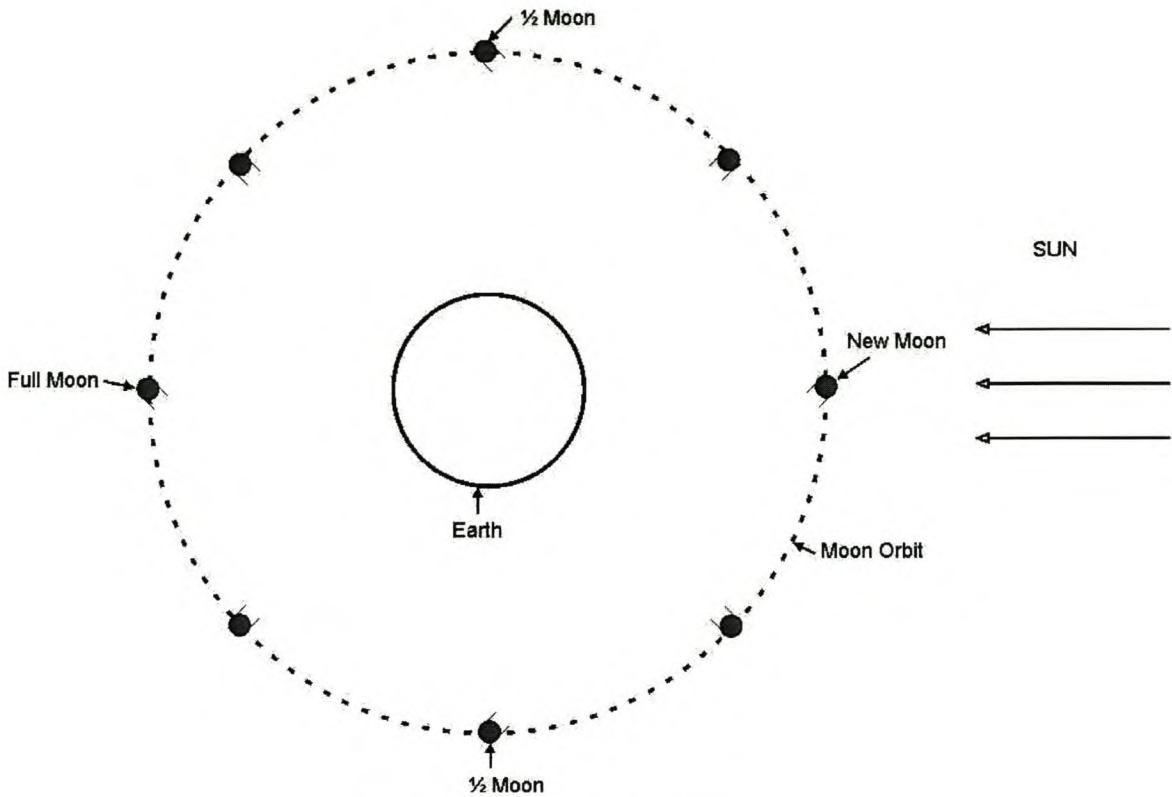


Figure 3.4 *Moon Orbit and Phases*

The case where the moon and sun both fall within the sensor's FOV occurs during new-moon, or very close to new-moon. The camera used in this project is assumed to have a FOV of less than 15° , if the moon is centered in this FOV then there has to be an angle of at least 7.5° between the sun and moon for the moon to be visible. The assumption is made that the moon will not be visible if within 15° of the sun (see Figure 3.5) thus not requiring that the moon be centered in the FOV. This will not have a far reaching effect since the moon center algorithm is unable to calculate the center of a new-moon (the moon would not have been of use to the sensor during this part of its orbit in any case). It follows that the moon will not be visible for at least 30° of its orbit. This translates to approximately 2.5 days out of every 29.53 days (synodic period) that the moon will not be visible. However, for the remaining 27.03 days the moon will be visible to the sensor for some part during every orbit of the satellite.

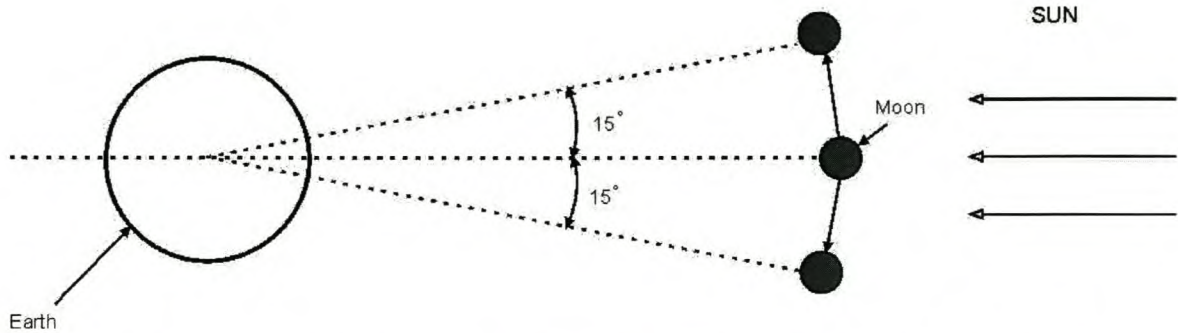


Figure 3.5 *Area around New Moon (Sun and Moon would fall in Sensor FOV Simultaneously)*

The satellite completes a number of orbits during the time it takes for the moon to complete a single orbit. It is assumed that the moon remains stationary for any one full satellite orbit (In reality the moon moves, relative to the sun-earth direction at approximately 0.8098° in a single satellite orbit).

3.2.3 Feasibility Analysis

Having considered the satellite and moon orbit details, and discussed the obstructions to the sensor's view of the moon, the visibility of the moon from the satellite for a few specific scenarios will be looked at:

1. When the satellite passes over the top or bottom of the earth, (positions nearest north and south poles) the moon will always be visible (unless the moon is in the 30° space around new-moon as discussed in Section 3.2.2). (See Figure 3.6). This allows a few degrees of the satellite's orbit that the moon will virtually always be available for the sensor to make use of.

From basic geometry, as can be seen in Figure 3.6, the approximate percentage of the satellite's orbit that the moon is always visible can be calculated. Note that during this period the sun will also always be visible, thus offering two points of reference for the satellite to work with (if a sun sensor is also employed on the satellite).

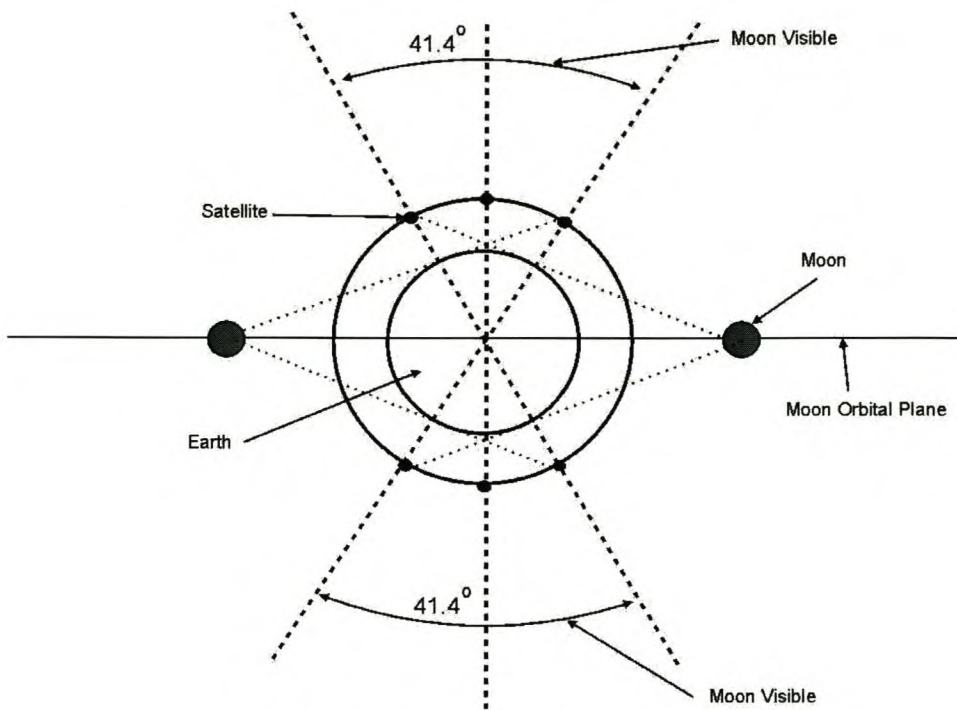


Figure 3.6 *Polar Region of Satellite Orbit*

When the moon and sun do not both fall within the sensor FOV, the moon is visible for about 11.5% of the satellite's orbit above the north-pole and also below the south-pole. This translates to two 11 minute periods in the satellite orbit in which the moon will be visible.

2. When the satellite enters the eclipse part of its orbit it would be convenient to see the moon, since the sun is not visible and the sun sensors are of no use. It is here that the moon offers the opportunity of more accurate attitude sensing for the satellite attitude and orientation systems.

There are a number of days in which the moon is not visible during the eclipse period of the satellite orbit (approx. < 11.16 days) but a number of days where it is visible every satellite orbit (approx. > 18.37 days). (See Figure 3.7)

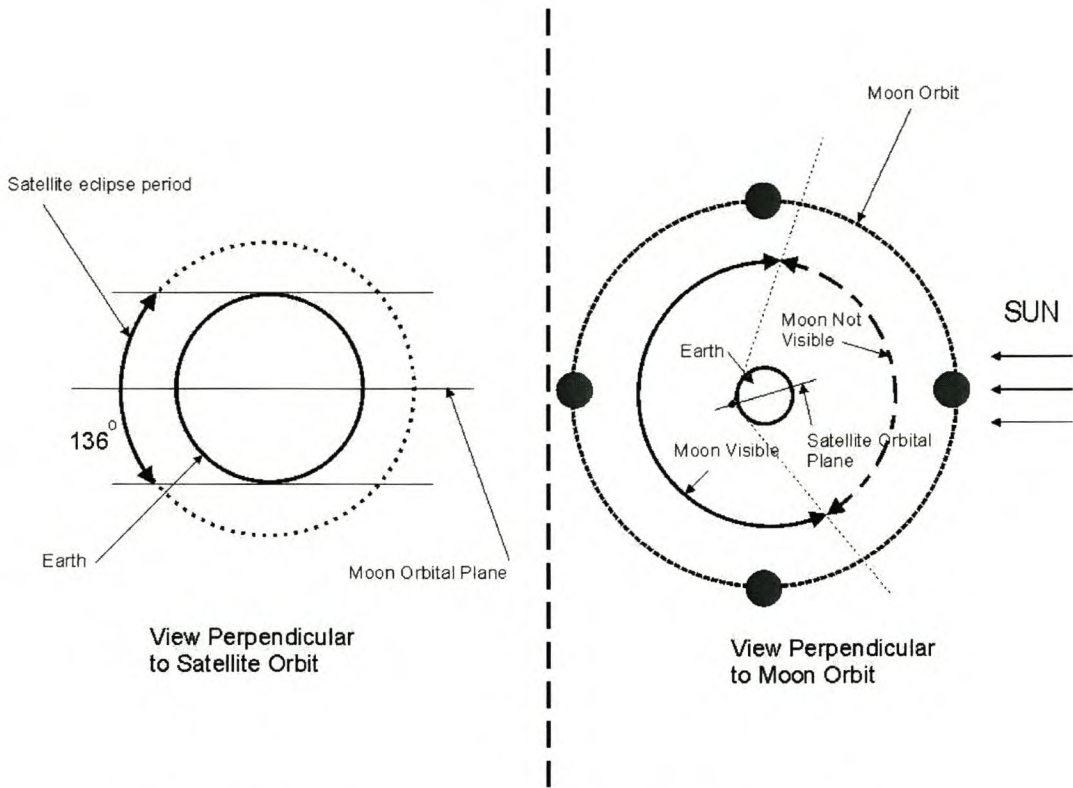


Figure 3.7 *Visibility of Moon during Eclipse Region of Satellite Orbit*

As shown in the figure above, the eclipse period of the satellite orbit is approximately 136° (or 36.13 minutes) in length. The moon will be visible for this time during every satellite orbit whilst the moon is not behind the earth (the moon is visible for approximately 62% of its orbit)

3. When the moon and sun are both visible to the satellite but do not both fall in the sensor's FOV, the moon offers a second point of reference (See Figure 3.8). The moon and sun sensors, when able to simultaneously capture the moon and sun respectively, can ensure attitude information that would allow complete control of attitude and orientation about all 3 axes of the satellite.

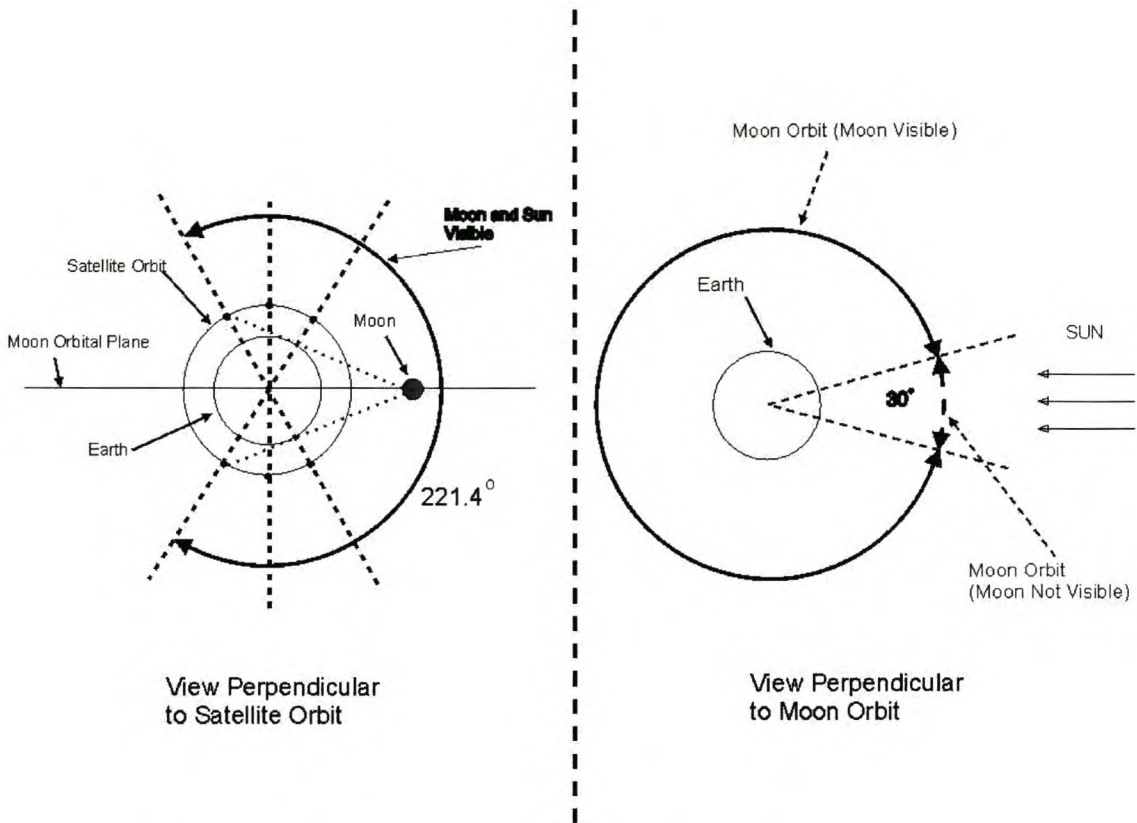


Figure 3.8 *Regions in which Moon and Sun are Visible to the Satellite*

This situation occurs during every satellite orbit when the satellite is not in the eclipse part of its orbit and the moon and sun, as mentioned previously, do not both fall within the moon sensor's FOV. The results, as can be seen in the figure above, are a combination of the periods over the poles as discussed in the first scenario of this section and the period between these polar regions on the sunlit side of the earth.

4. There are parts of the moon's orbit in which it will always be visible from the satellite for the entire orbit of the satellite (See Figure 3.9). This is true at two sections of the moon's orbit when the moon is in a position that is approximately orthogonal to the satellite orbital plane. These situations offer a few days in which the moon is a more available reference than the sun, it can be used by the satellite at any time a point of reference is required.

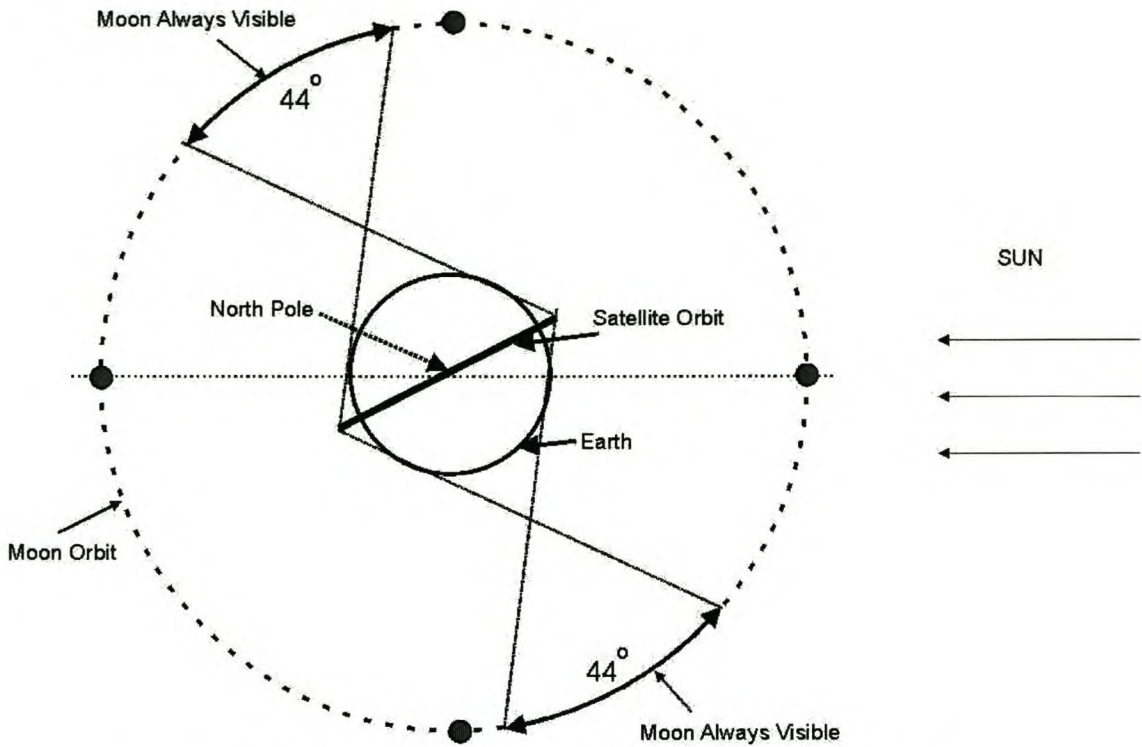


Figure 3.9 *Regions of Moon Orbit where it is Visible for Full Satellite Orbit*

As can be seen in the figure above, the two periods of time that the moon is always visible from the satellite are about 44° of the moon’s orbit each, translating to two, 3.6 day long, periods that the moon is always visible throughout the satellite’s orbit.

In conclusion to the feasibility study of the moon sensor, the approximate percentages that the moon is visible from the satellite is compiled and listed in Table 3.1 below. The calculations begin at 0°, where the moon lies directly between the earth and sun (new-moon). The approximate phase of the moon which is visible from the satellite is evaluated at each selected point of the moon in its orbit. It is assumed that the moon’s phase changes in a sinusoidal manner. The percentage of the moon that is visible at a specific angle (θ from new moon) is calculated using the following formula:

$$\phi = 50 * (1 - \cos(\theta)) \tag{3.3}$$

This formula results in a value between 0% and 100% as the angle changes between 0° and 360° (Figure 3.10).

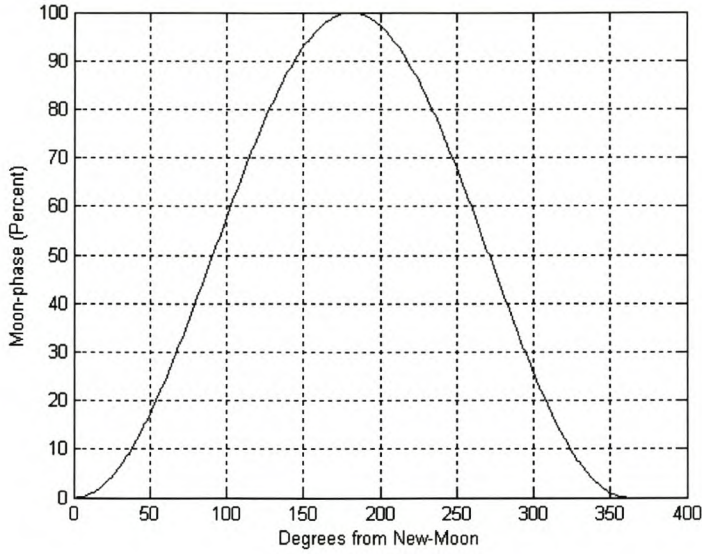


Figure 3.10 Moon Phase Relative to Degrees from New-Moon

<u>Position of Moon in orbit</u> (Degrees from new-moon)	<u>Moon Phase Visible</u>	<u>Percentage of the Satellite's Orbit that the Moon is Visible</u>
0° – 15°	0%	0%
15° – 98°	1.7% - 57%	$61.7\% < x \leq 100\%$
98° – 142°	57% - 89.4%	100%
142° – 180°	89.4% - 100%	$100\% \geq x \geq 61.7\%$
180° – 278°	100% - 43%	$61.7\% \leq x \leq 100\%$
278° – 322°	43% - 10.6%	100%
322° – 345°	10.6% - 1.7%	$100\% \geq x > 61.7\%$
345° – 360°	0%	0%

Table 3.1 Summary of Moon Visibility to Satellite

The sun is visible to the satellite for approximately 61.7% of each satellite orbit. Using the visibility of the sun as a point of reference, it follows from Table 3.1 that the moon's visibility is equal to or greater than that of the sun for approximately 91.6% of the moon orbit. The only time that the moon is less visible than the sun is when the moon and sun fall within 15° of each other (approximately 8.4% of the moon's orbit). This shows that the moon is approximately as consistent a point of reference as the sun. This moon sensor was therefore decided to be a viable project to further research.

3.2.4 Additional Information on Moon Visibility Accuracy

The assumptions made with regards to the moon's orbit in the previous section were over simplified. For more accurate results the following orbit details should be taken into account. The moon's orbit plane, in reality, has an inclination of 5.1454° relative to the ecliptic plane, and the ecliptic plane is 23.44° relative to the equatorial plane. The moon orbital plane precesses and thus its inclination relative to the equatorial plane varies between $23.45^\circ + 5.15^\circ = 28.60^\circ$ and $23.45^\circ - 5.15^\circ = 18.30^\circ$. As result of this, the worst case angle between the moon and satellite orbital planes is approximately 53.81° (this is illustrated in figure 3.11).

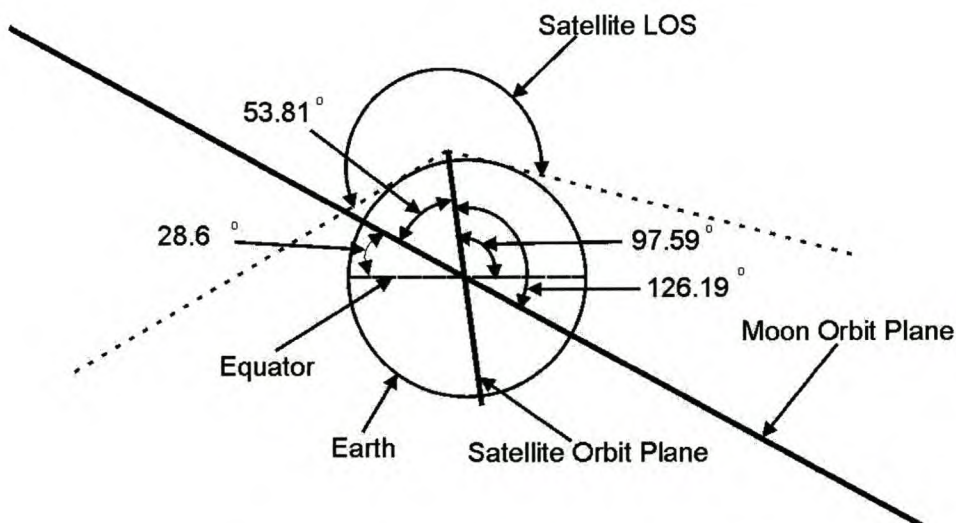


Figure 3.11 *Actual Satellite and Moon Orbits*

As a result of the actual orbit planes, the moon will not always be visible during both of the two satellite polar periods as previously stated (figure 3.6) and also not for 100% of the satellite orbit during the 2 periods as indicated in figure 3.9. The latter case will only be true during times when the moon is nearest to the normal of the satellite orbital plane, which is not true for every orbit of the moon.

In conclusion, the moon will be less visible than stated in Table 3.1, but still visible for a large part of every satellite orbit (unless both moon and sun fall simultaneously within sensor FOV). The visibility can be calculated more accurately by creating a simulation of the satellite, moon and sun orbits. When the moon is visible it is for at least 61.7% or more of the satellite orbit. This is equal to and above the visibility of the sun, thus it is still a viable option to further research on the use of this sensor.

3.3 Algorithms

3.3.1 Moon Center Algorithm

The full-moon is approximately $\frac{1}{2}^\circ$ in diameter when viewed from the earth. Any point selected as reference on the moon would be accurate within $< \frac{1}{2}^\circ$. A higher accuracy and thus improved performance can be achieved if the same point on the lunar body could be selected consistently. To start this process it was necessary to examine the actual physical planet and its shape.

The moon's equatorial radius (1738.1km) differs from the polar radius (1736.0km) by approximately 1.9km which is small enough a difference to be ignored. For the purposes of this study the moon can be seen as round.

To achieve a more accurate reference point for the sensor, a means of consistently calculating the center of the actual lunar body - regardless of phase or rotation - was required. This requirement resulted in the development of an algorithm that calculates the center of the lunar body from any given moon image. The algorithm was initially

developed to calculate the center of the full-moon, after which it was expanded and refined to calculate the center of the lunar body regardless of phase and rotation.

The full-moon appears as a near-white disc on a black background. By taking a threshold of the image (threshold being when all intensities above a certain level become white and all intensities below that level are made black), the moon can be isolated within the image. If a circle is fitted to this moon image to fall precisely on the outline of the moon, the center of this circle would lie exactly on the center of the moon (Figure 3.12).

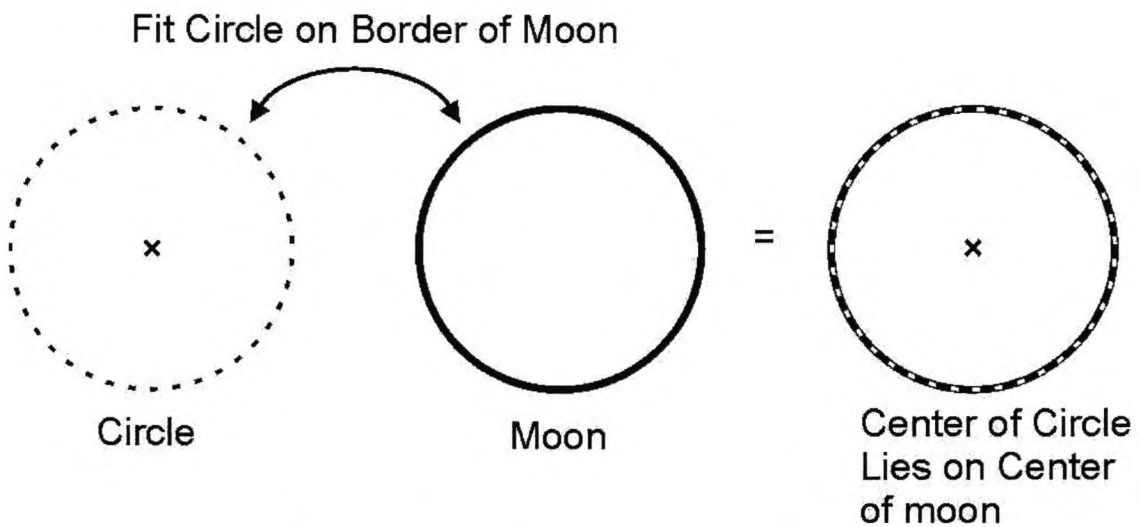


Figure 3.12 *Fitting of Circle to Moon Image Resulting in Center Coordinate*

The formulation of the algorithm is approached by breaking it into two parts. Firstly, fitting a circle on the outline of the moon image, and secondly, calculating the center of that circle. The process of calculating the center of a given circle will be examined first.

Determining the center of a circle requires three separate points on its circumference. For any three points only one circle exists that passes through all the points. The center coordinate for this circle is calculated as follows:

The equation of a circle:

$$x^2 + y^2 + ax + by + c = 0 \quad (3.4)$$

The center coordinates and radius result from Equations (2.17) – (2.19).

If (x_1, y_1) , (x_2, y_2) , (x_3, y_3) are three points on the circle circumference, and they are substituted into the equation for a circle, Equations (2.14) – (2.16) are generated.

These equations when written into a matrix form:

$$\begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ x_3^2 + y_3^2 \end{bmatrix} + \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \quad (3.5)$$

This matrix equation is solved for the unknowns a , b and c as follows:

$$\begin{bmatrix} a \\ b \\ c \end{bmatrix} = - \begin{bmatrix} x_1^2 + y_1^2 \\ x_2^2 + y_2^2 \\ x_3^2 + y_3^2 \end{bmatrix} \begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix}^{-1} \quad (3.6)$$

And the inverse matrix above is calculated using the following formula:

$$A^{-1} = \frac{Adj(A)}{Det(A)} \quad (3.7)$$

And expanded to:

$$A^{-1} = \frac{\begin{bmatrix} y_2 - y_3 & -(y_1 - y_3) & y_1 - y_2 \\ -(x_2 - x_3) & x_1 - x_3 & -(x_1 - x_2) \\ x_2 y_3 - y_2 x_3 & x_1 y_3 - y_1 x_3 & x_1 y_2 - y_1 x_2 \end{bmatrix}}{x_1(y_2 - y_3) - y_1(x_2 - x_3) + (x_2 y_3 - y_2 x_3)} \quad (3.8)$$

The values for a , b and c , are now set into Equations (2.17), (2.18), (2.19) resulting in the required center coordinate and radius for the circle under investigation.

The solution to the requirement of fitting a circle to the moon image outline is satisfied by locating three separate points on the moon's circumference. Since the full-moon is viewed as a perfect circle, theoretically, any three separate points on its circumference can be used. These points fulfill the first and second requirements of the algorithm, and the center can be calculated accordingly.

If the selection of three points on the moon border is done at random, the complexity of the processing algorithm becomes very high and does not guarantee sufficient spacing of the points. Also, the three points cannot be adjacent to each other; the reason shown in Figure 3.13 for low resolutions and/or small images.

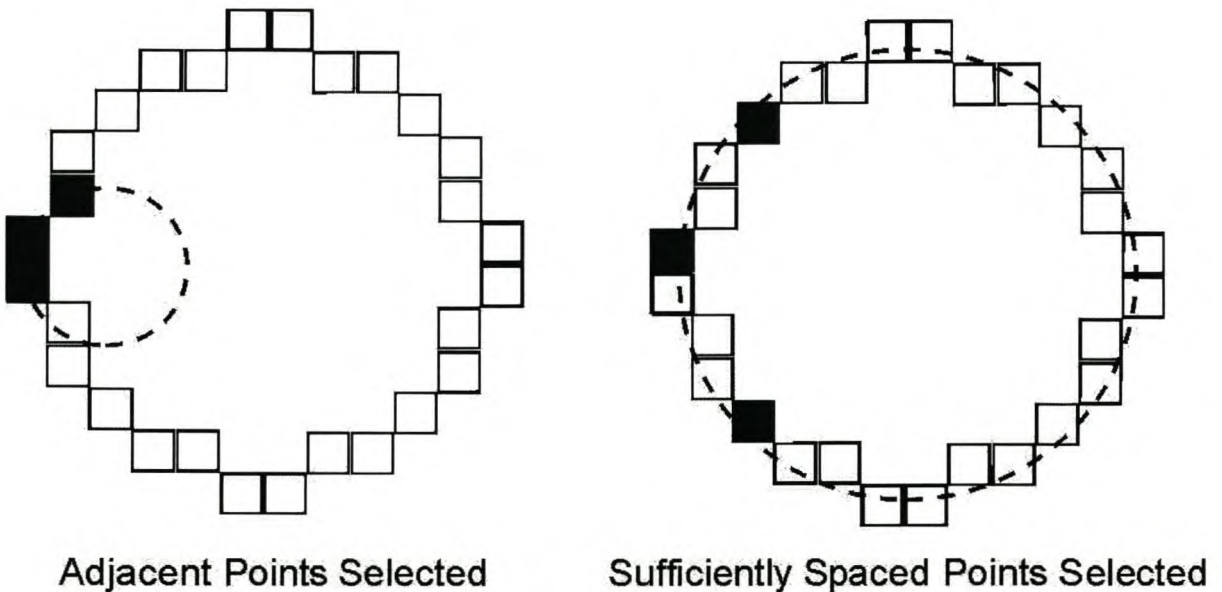


Figure 3.13 Results of Different Spacing Of selected Points on Moon border

In a digital image a circle - when viewed at pixel level - consists of a number of short straight lines, each only a few pixels long. To achieve a sensible fitting, the 3 required pixels must be selected as far apart as possible. The selection of the points must be achieved by a process that can be repeated with consistently accurate results. It was

decided to select uniformly spaced points along the moon border (the selection of which would not be too computationally complex).

The first points selected were at the extremities of the moon outline parallel to the x and y axes. This selection results in four uniformly spaced points on the moon circumference (refer to Figure 3.14). Any three of these points could be used to define the circle that lies on the moon's outline. This method will however only work in the case of a full-moon.

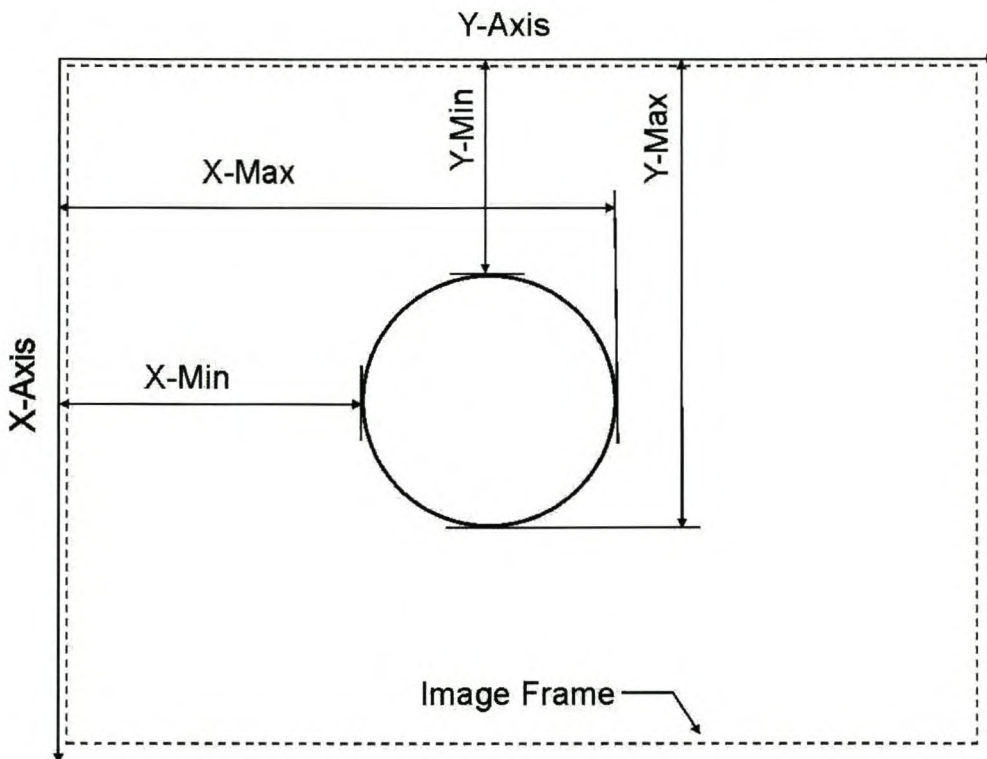


Figure 3.14 *Determining Points at x and y Extremities*

When the moon is not full, one side reflects the true border of the moon whilst the opposite side portrays the terminator (i.e. the side where shadow falls across the moon). The moon's rotation is not taken into account by the sensor, as the moon images taken could be rotated from 0° through 360° due to the rotation of the satellite as it orbits the earth. This requires that the algorithm be rotation independent with regards to the moon image.

In a worst case, as the moon's phase changes, two of the previously selected four points will no longer lie on the actual border, but on the side of the moon that is in shadow, leaving only 2 points on the true border (refer to Figure 3.15). This makes the calculation of the moon center impossible as an infinite number of different sized circles can pass through the same 2 points.

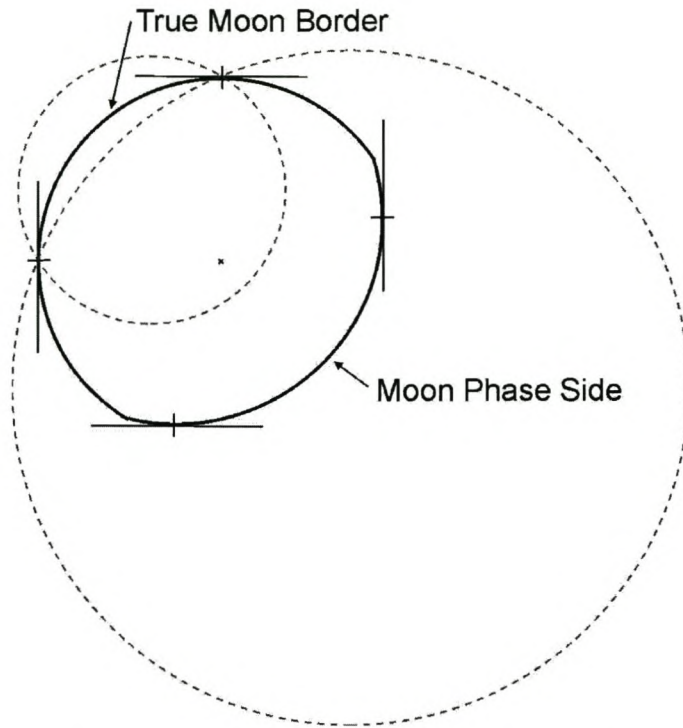


Figure 3.15 *Fitting of Different Size Circle through same Two Points*

To now find three points on the actual moon border, it is necessary to select more points along the outline of the moon image. The number of points acquired along the moon border can be doubled by rotating the reference axes by 45° and locating points at minimum and maximum perpendicular distances from the xy and yx axes (refer to Figure 3.16).

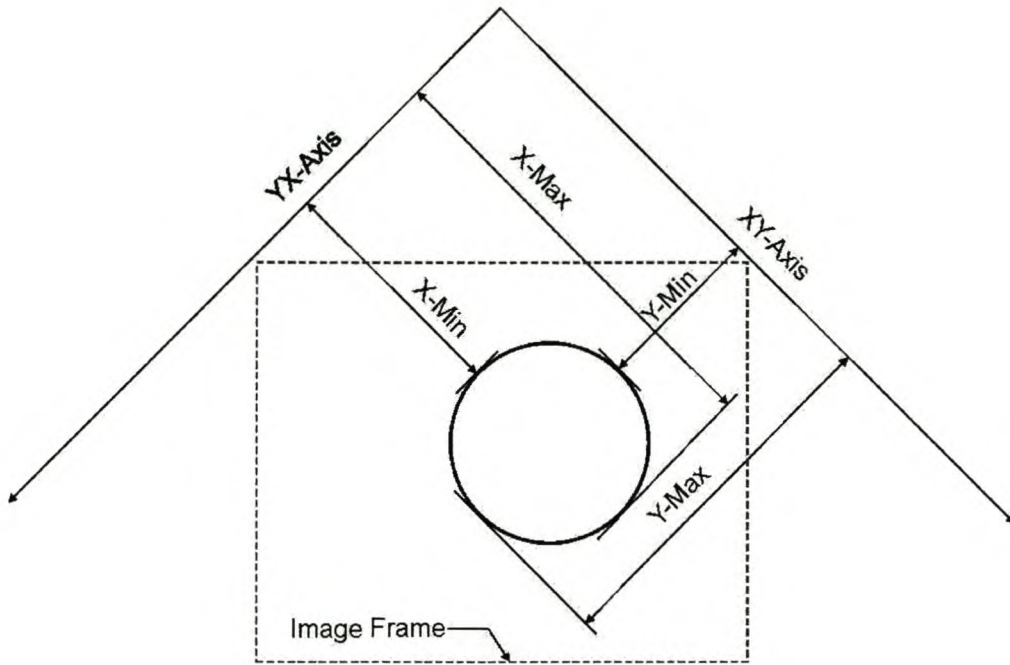


Figure 3.16 *Determining Points at xy and yx Extremities*

From extensive testing it was found that the 8 points obtained sufficed in meeting the desired requirement for all phases of the moon, offering at least 3 of the 8 points on the actual moon border. The final algorithm is implemented using this information.

Calculation of Moon Center

The next step is to select three out of the eight points which lie on the actual moon border. The algorithm is designed so that the selection process of the required three points remains consistent regardless of the moon's phase or rotation. A moon phase between 75% and 100% rotated by approximately 45° is used to illustrate this process. The steps to calculate the moon center are discussed below:

1. For each axis (x , y , $x-y$ and $y-x$) a point is selected at both the maximum and minimum distance perpendicular to each particular axis. The distance between each pair of points per axis is calculated, resulting in four separate lengths (called: x -bisecting, y -bisecting, xy -bisecting and yx -bisecting lines). During full-moon these pairs of points are equal distances apart and it does not matter

which pair is selected for the next step. When the moon is not full, the lengths will begin to differ. The longest bisecting line will be closest to dividing the moon across its phase (a dividing line separating the actual border from the shadow border, Figure 3.17). This bisecting line is selected for use in the next step.

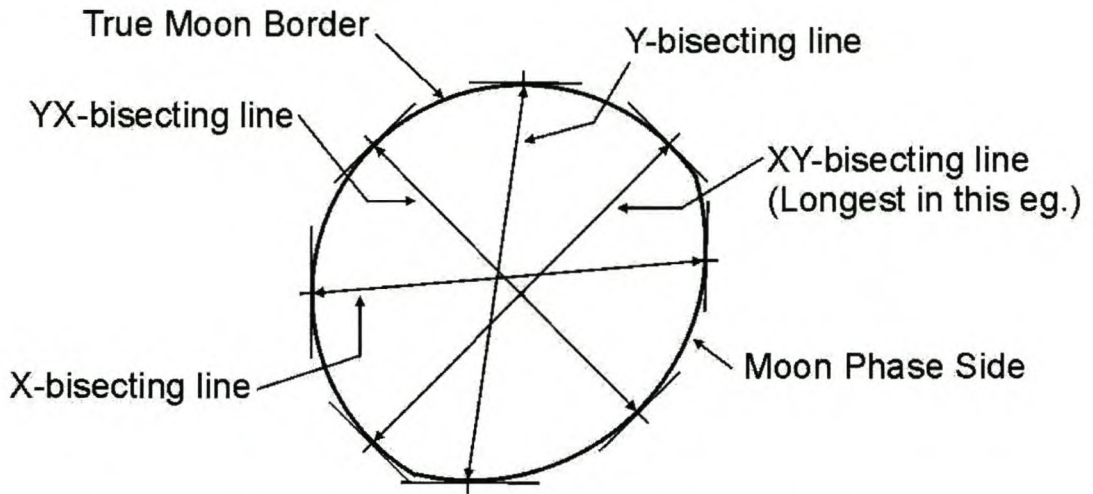


Figure 3.17 *Calculation and Selection of Longest Bisecting Line*

2. Using the bisecting line subtended by the two points from step 1, it is necessary to calculate to which side of this line the actual moon border lies (for full-moon this step is just a formality as any three points on the border can be selected). Using basic geometry, the perpendicular distance from the bisecting line to the points on either side, furthest from this line, can be calculated. As can be seen in Figure 3.18, the longer distance of the two intersects the actual moon border. The point at this longest distance is selected as on the desired side of the moon since the shadow side of the moon will lie closer to the bisecting line.

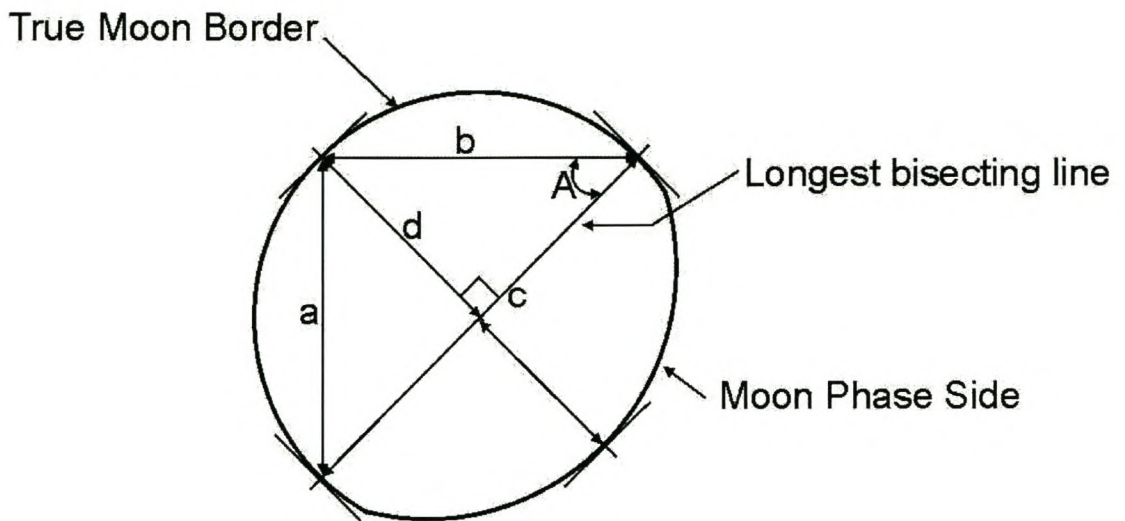


Figure 3.18 Calculation and Selection of Point on Actual Moon Border

Below is demonstrated how one of the two distances is calculated, the same procedure is followed to calculate the other distance.

From the figure above using the *cosine* rule:

$$a^2 = b^2 + c^2 - 2bc \cos A \quad (3.9)$$

Solve for the angle A :

$$A = \cos^{-1} \left(\frac{-a^2 + b^2 + c^2}{2bc} \right) \quad (3.10)$$

The required perpendicular distance, d , is calculated:

$$d = c \sin A \quad (3.11)$$

3. Now that one point on the actual moon border has been acquired, two more points need to be selected. These two points are obtained by selecting the

nearest points from either side of the point obtained in step 2. (See Figure 3.19)

Three Selected Points

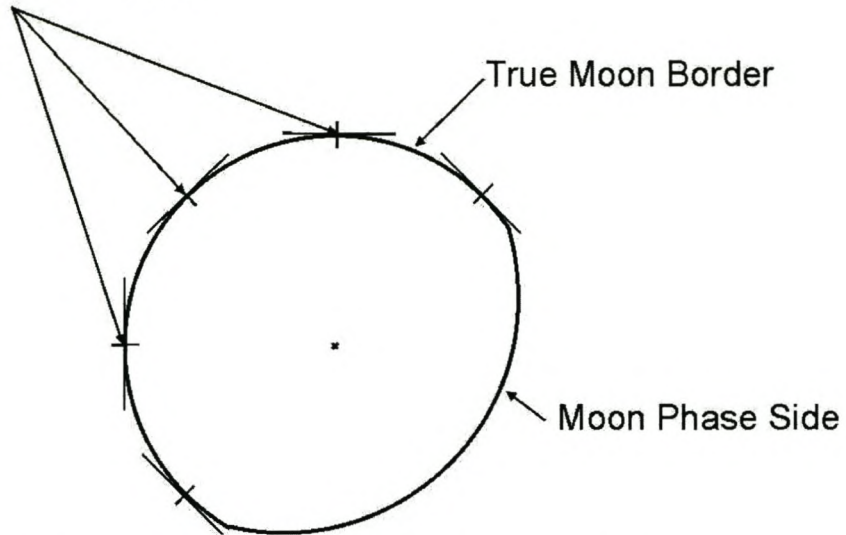


Figure 3.19 *Final Selection of Three points on Actual Moon Border*

These two points are used instead of the points subtending the longest line (step 1). The points from step 1 only signify the maximum distance on the given moon image and do not necessarily both lie on the correct part of the moon curvature.

4. The coordinates of these three points are now applied in the three linear Equations (2.14) – (2.16) which are in turn solved for the center coordinate of the moon (Figure 3.20).

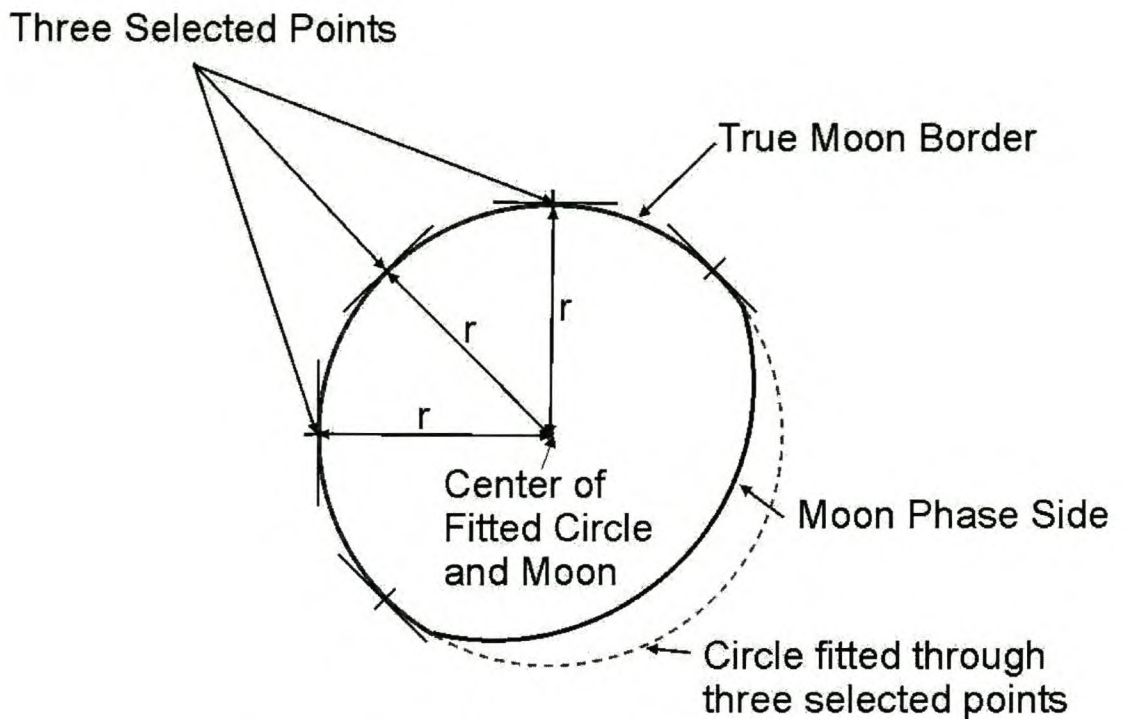


Figure 3.20 *Calculation of Moon Center*

3.3.2 Moon Position Algorithm

The satellite orbits the earth approximately 444.6 times in the time that it takes the moon to complete 1 orbit around the earth. Therefore their positions relative to each other and so the LOS from the satellite to the moon changes constantly.

When considering a FOV for a camera to acquire images of the moon there are two basic options, a large FOV or a narrow FOV. Each option is discussed below.

1. Camera with a large FOV

For a camera with a large FOV, the moon appears very small in the image, resulting in a decrease in accuracy of the sensor. The advantage is that the camera could remain fixed on the satellite and if a few correctly placed fixed cameras are used there is no need to develop a pointing algorithm to move the camera, drastically decreasing the complexity of the sensor.

2. Camera with a narrow FOV

A camera with a narrow FOV requires an algorithm to direct it to look at the moon, increasing the complexity of the sensor system. The advantage is that the moon image within the FOV is much larger, allowing calculations to be done on the image and increasing the accuracy of the sensor.

The camera used in this project has a narrow FOV, resulting in a high resolution. But this requires that the coordinates of the moon's position be known accurately so the satellite knows where to point the sensor. The position of the moon must be calculated relative to the satellite. This position should be sufficiently accurate to ensure that the moon falls within the sensor's FOV each time an unobstructed view of the moon is available.

The moon position algorithm implemented is discussed in Chapter 2.2.2 and was taken from the Astronomical Almanac. It offers approximate positional accuracy of 10 arc minutes. The moon's diameter is approximately 30 arc minutes when viewed from the earth, thus the 10 arc minute (0.1667°) accuracy is sufficient. Any other refinements to the algorithm would be more resource intensive with little to no gain in this application. The resultant coordinates as given by the algorithm are in the GCI system. The coordinates of the satellite are also supplied in this system and the resultant satellite moon vector can be calculated. The assumption is made that the satellite will supply its coordinate position in its orbit to the sensor.

Chapter 4

Hardware and Software Set-Up

4.1 Introduction

In this chapter the hardware as well as the software used in the implementation of the moon sensor will be discussed. These include:

- Hardware
 - Sensor Processor Board
 - Sensor Camera
 - Physical Test Setup (Incl. Pictures)
- Software
 - Hardware Initialization
 - Basic Sensor Software Structure
 - Image Processing - Software Implementation
 - Communications Software Structure (PC <->Sensor)

4.2 Hardware Overview (Implementation)

4.2.1 DSP Evaluation Board (Sensor Processor Board)

During the preparation for this project, the ADSP-BF533 EZKit-Lite evaluation board by Analog Devices was provided and deemed more than adequate to carry the load of all the sensor's tasks.

The basic functions required of the evaluation board were:

- To convert analog video to digital
- To store a full frame of video to memory
- To process any given image sufficiently fast
- To communicate with a PC

The evaluation board has many components and capabilities, only those relevant to this project are mentioned. A more complete description of all the evaluation board's components and possible uses is included in the datasheets (Analog Devices [3], [4]). The major functional components put to use are:

- The onboard DSP is used for all the processing tasks required by the sensor, (it is the processor on which the code for the moon sensor runs). The DSP also handles all communications with the PC (the PC acts as a satellite). The maximum clock frequency that the DSP is capable of running at is 600MHz. However, the default clock frequency of 270MHz was used since it was the easiest to implement and was sufficient for the sensor's processing requirements.
- The onboard Video Decoder reads in an entire frame of analog video from the sensor's camera, converts the input to digital YCrCb format and outputs this image - using DMA - via the Parallel Peripheral Interface (PPI) to the evaluation board's SDRAM where it is stored.
- The onboard SDRAM is used to store the image from the video decoder. This image is read in and stored as two fields: one field contains the data of all the odd numbered lines and the other the data of the even numbered lines of the frame of video. These two fields are written in succession to memory (refer to Figure 4.1)

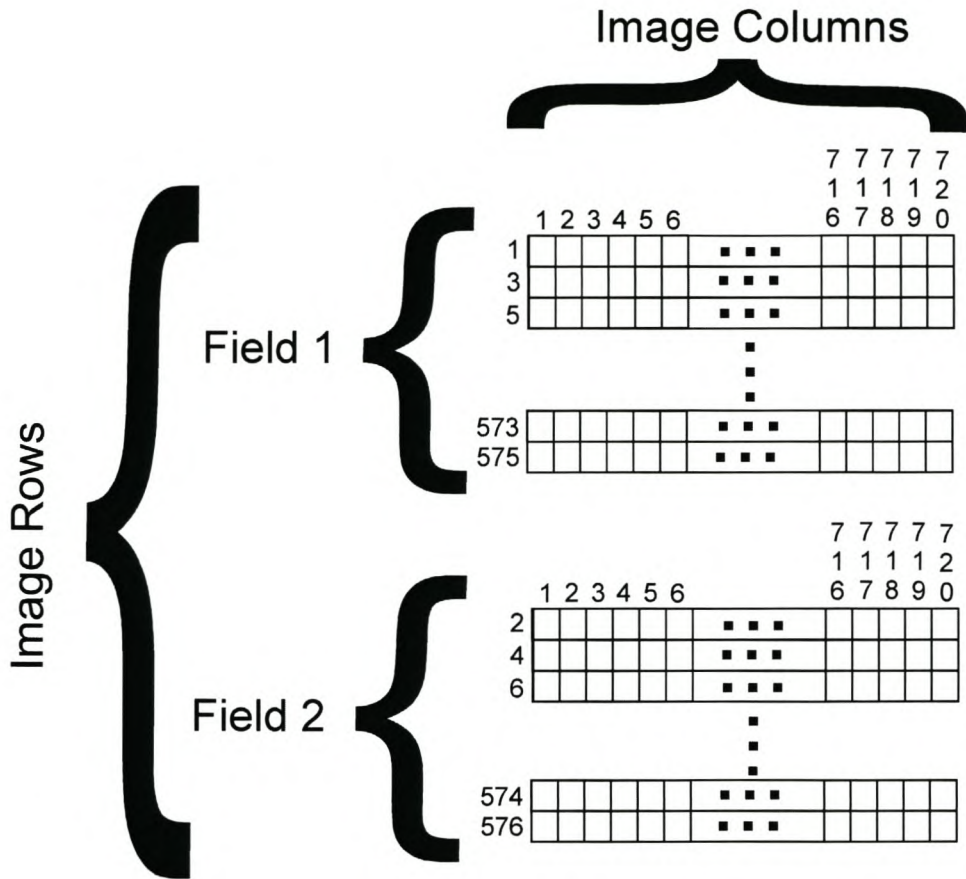


Figure 4.1 *Format of Image as Downloaded to SDRAM*

- The UART Port (serial in/output port) is used for all communications between the PC and the evaluation board, i.e. transfer of requests/responses and data between PC and evaluation board.

The DSP was capable (used at less than ½ of its maximum clock speed) of taking, processing and returning the results of 8 pictures per second. If the requirement of this sensor was one picture per second, as is the case with many sun sensors, the current speed would allow time for other necessary steps and algorithms to be implemented. It would be possible to implement this sensor on a slower and possibly more affordable platform and still obtain satisfactory results.

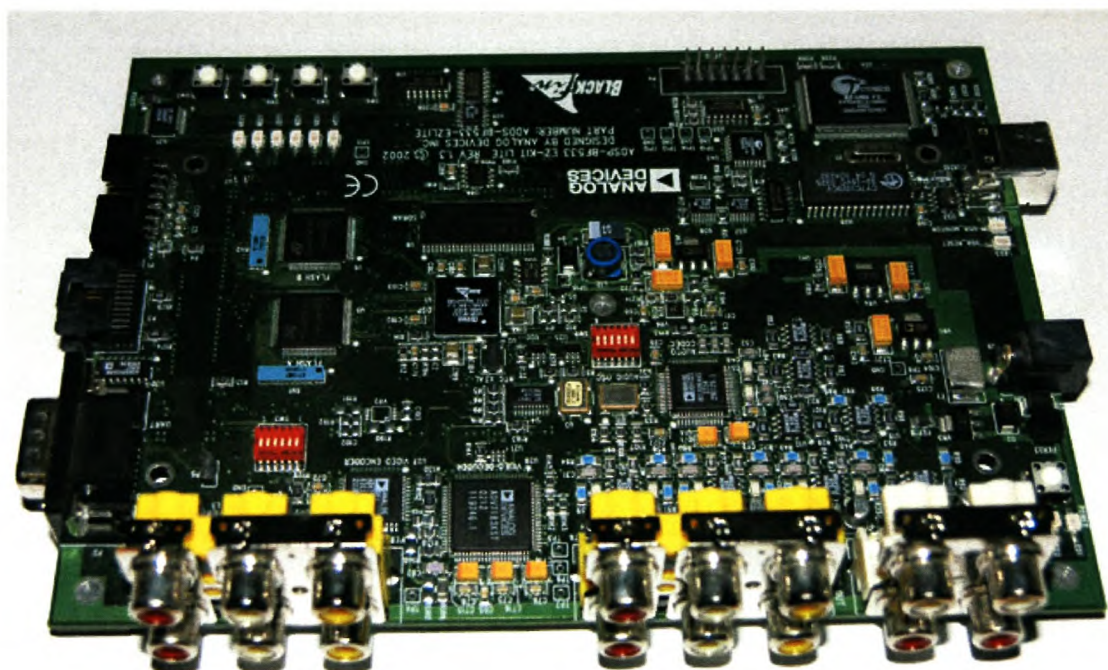


Figure 4.2 Picture of the ADSP-BF533 EZKit-Lite Evaluation Board

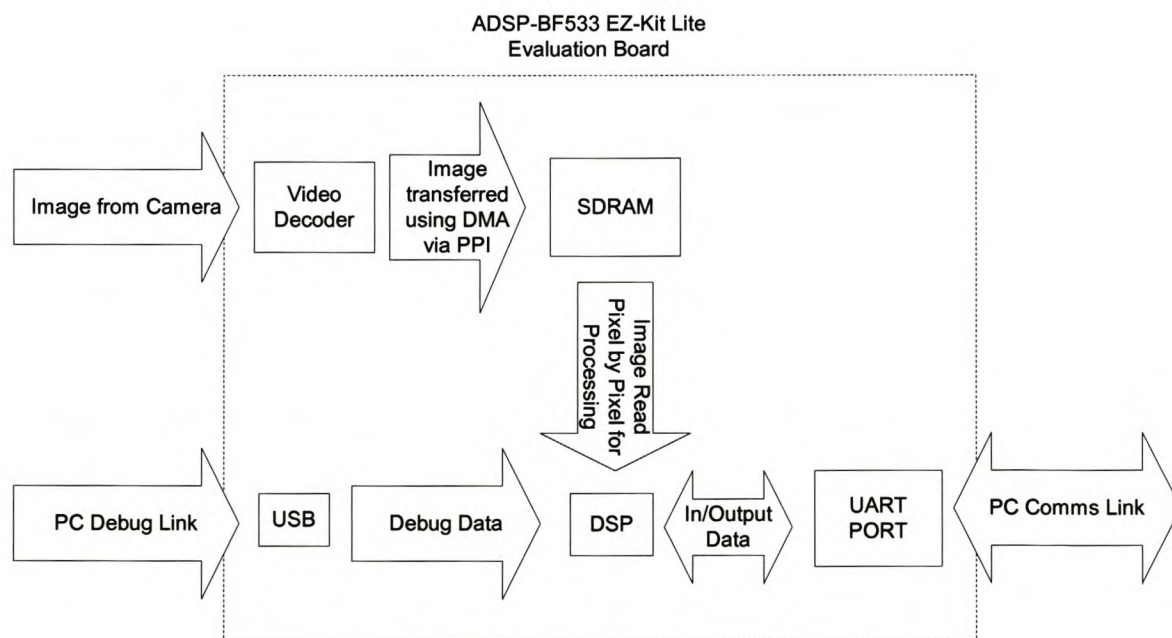


Figure 4.3 Flow Diagram of Evaluation Board

4.2.2 Sensor Camera

A standard black-and-white camera is used. It has analog video output and a resolution of 720x578 pixels. Not all the pixels of the camera are functional, as can be seen in images below (Figure 4.4), thus the processed part of the image is slightly smaller than the resolution of the camera (approx. 680x570 pixels). This camera was fitted with a lens which has manually adjustable aperture and focus. The FOV was approximated and calibrated by taking an image of an object (a single section of paving on a sidewalk), with known width/height at a known distance, and calculated in pixels/degree (see Figure 4.4 below). From the figure below the horizontal resolution is approximately 0.0234° resulting in a horizontal FOV of 14.65° . The vertical resolution is approximately 0.0189° resulting in a vertical FOV of 10.92° .

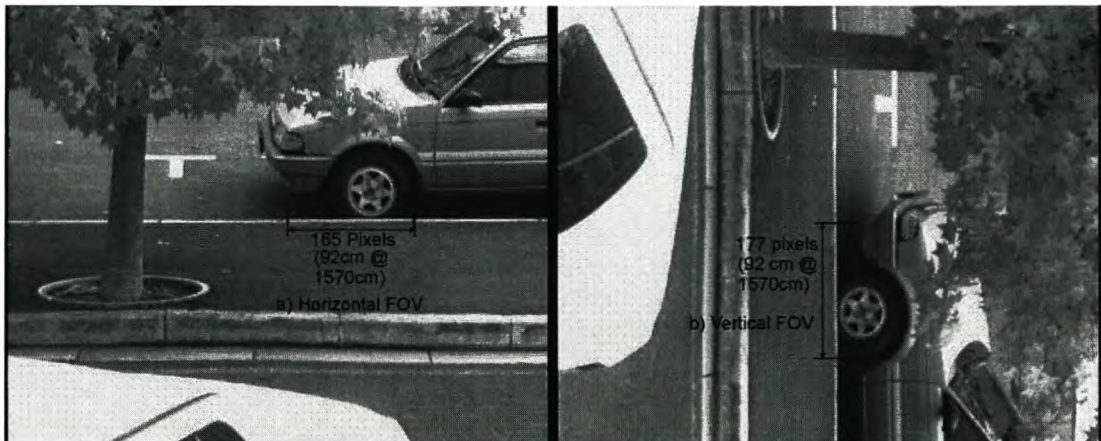


Figure 4.4 *Images Taken to Calculate Calibrate the Camera's FOV*

The camera has built in gain control, meaning that it will automatically adjust the intensity of the image taken by the sensor (higher gain in low light conditions and lower gain in lighter conditions). The moon is a relatively small light source on a large dark background. The camera compensates for the apparent low light conditions by raising the gain of the sensor and over-exposing the moon image (Figure 4.5).

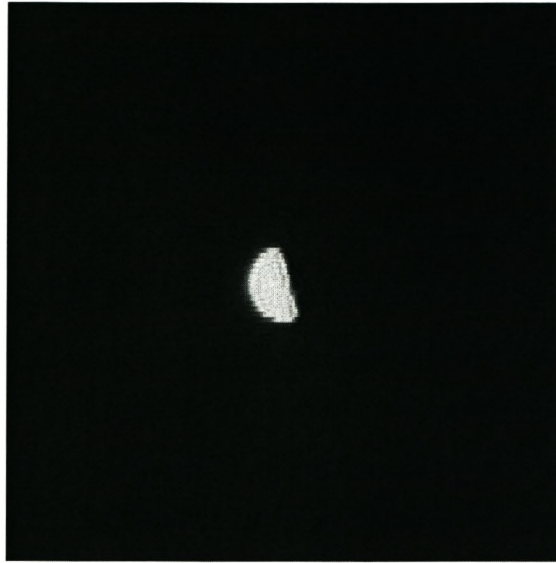


Figure 4.5 *Picture of Overexposed Moon*

To solve the overexposure problem the aperture had to be adjusted to decrease the amount of light allowed onto the sensor (Figure 4.6). (Although the overexposed moon image is larger than the correctly exposed image, the basic moon shape appears to be retained).



Figure 4.6 *Picture of Correctly Exposed Moon*

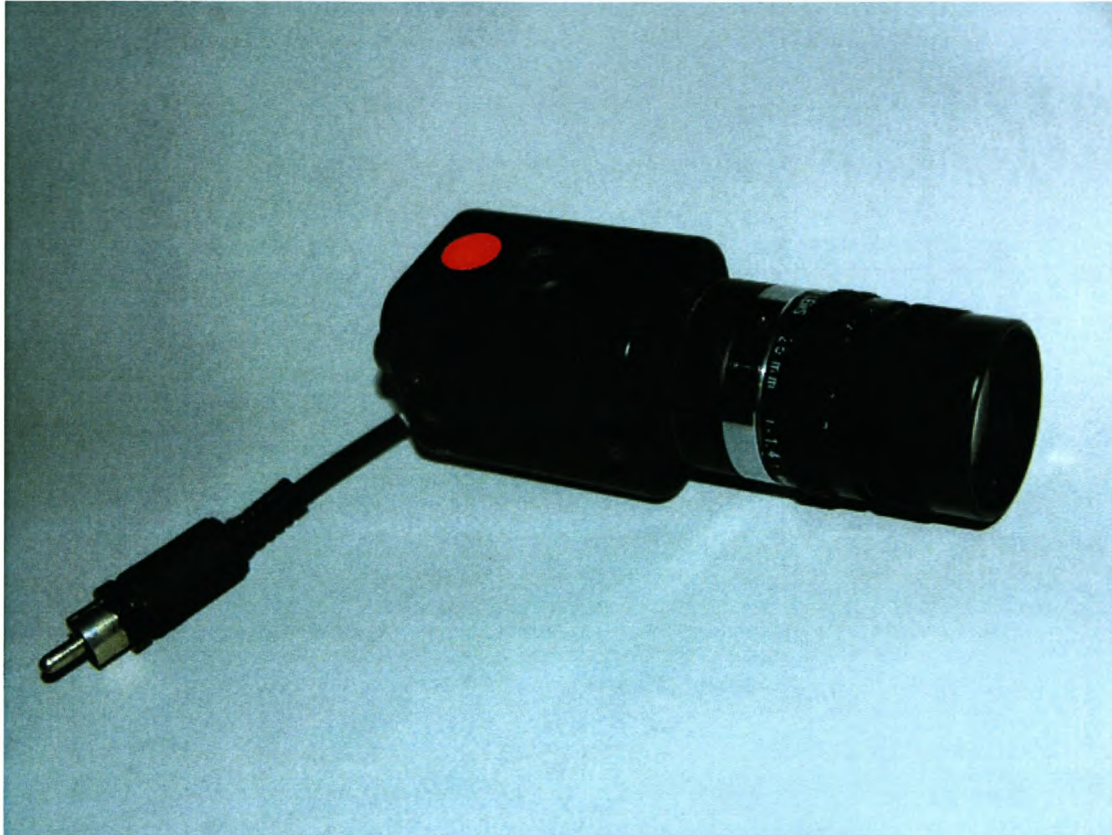


Figure 4.7 *Picture of Camera used for sensor*

4.2.3 Physical Test Configuration

The basic configuration of hardware is shown in a block diagram (Figure 4.8) and a photo of the entire configuration appears in Figure 4.9. The evaluation board is connected via 2 cables to the PC: a serial cable, for transfer of data and requests/responses to and from the PC and a USB debug cable for programming the evaluation board. The camera is connected to the evaluation board via a standard video input jack.

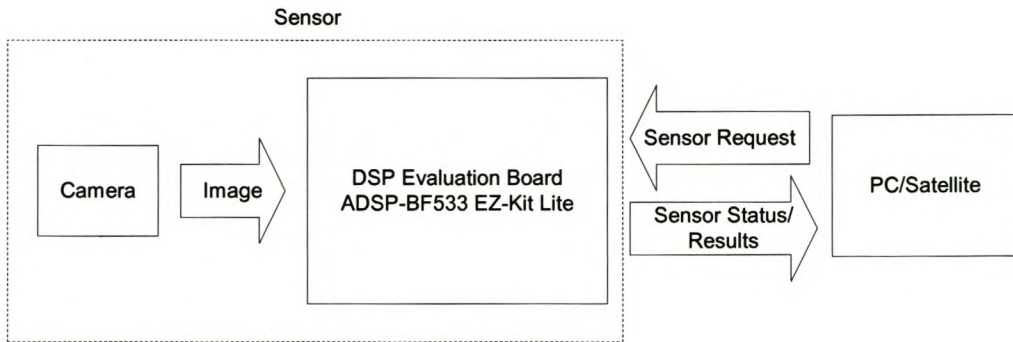


Figure 4.8 *Flow Diagram of Physical Test Configuration*

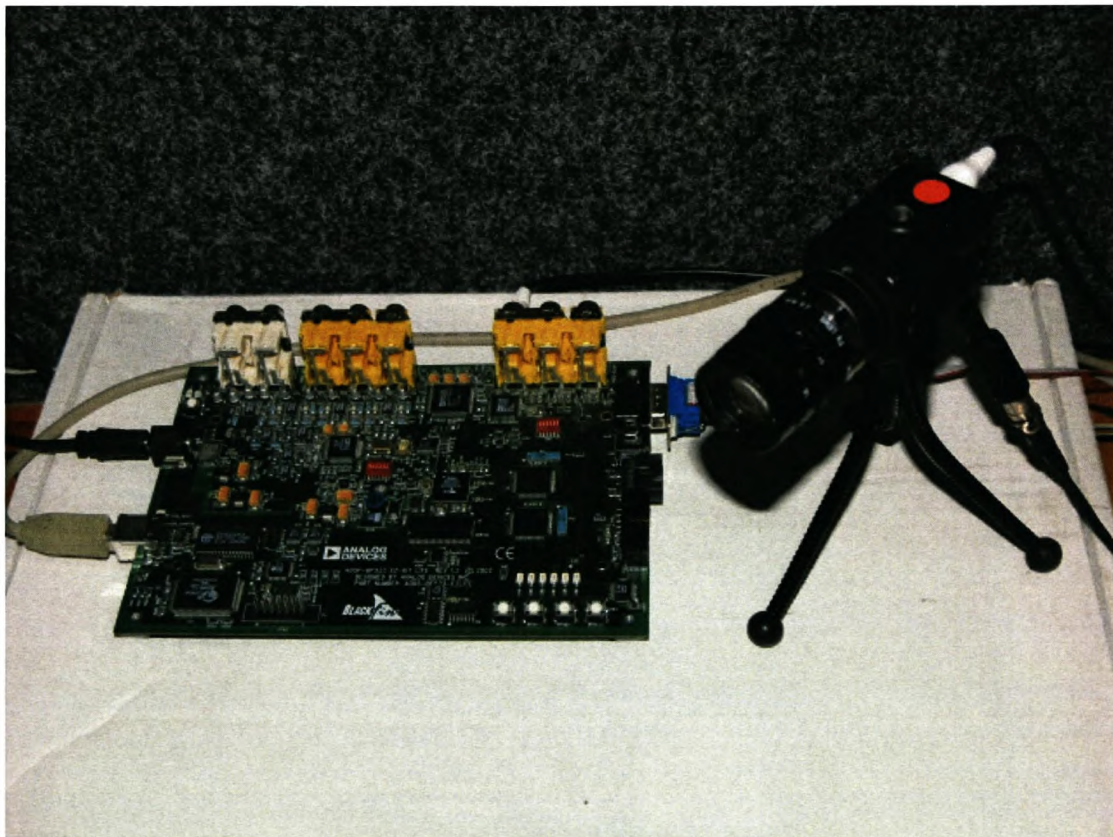


Figure 4.9 *Picture of Entire Configuration*

4.3 Software Implementation

4.3.1 Hardware Initialization

Initialization of the PC and DSP board was required before the actual processing was possible and the communications software could be implemented (refer to Analog devices [3], [4], [5]).

On the DSP the following had to be initialized (refer to DSPMoonSensor.c in Appendix B):

- The Clock frequencies required for the timing of all operations on DSP board needed to be set up. The DSP Core Clock (CCLK) frequency was set at the default of 270MHz. The System Peripheral Clock (SCLK) was set to 90MHz, a factor 3 smaller than then DSP CCLK.
- The UART port was then activated for communications between the DSP board and the PC. The UART details are discussed further in section 4.3.4.
- The Video Decoder was setup in its default setting: to automatically detect the input video type (NTSC/PAL) and then output the data in ITU-R 656 format (see Analog Devices [5, pages 22-35]).
- The Parallel Peripheral Interface (PPI) was set up as an input to transfer only the active fields from the video decoder (ITU-R 656) output to the SDRAM via DMA.
- The DMA controller was set up to transfer data from the PPI to SDRAM upon activation of the controller.
- The SDRAM was configured and a block of random data written to it to initialize the memory.

The PC used in this project was used to open and setup the serial port (COM1) for communications between the DSP board and the PC.

4.3.2 Basic Sensor Software Structure

The basic structure and flow of the actual sensor is as follows:

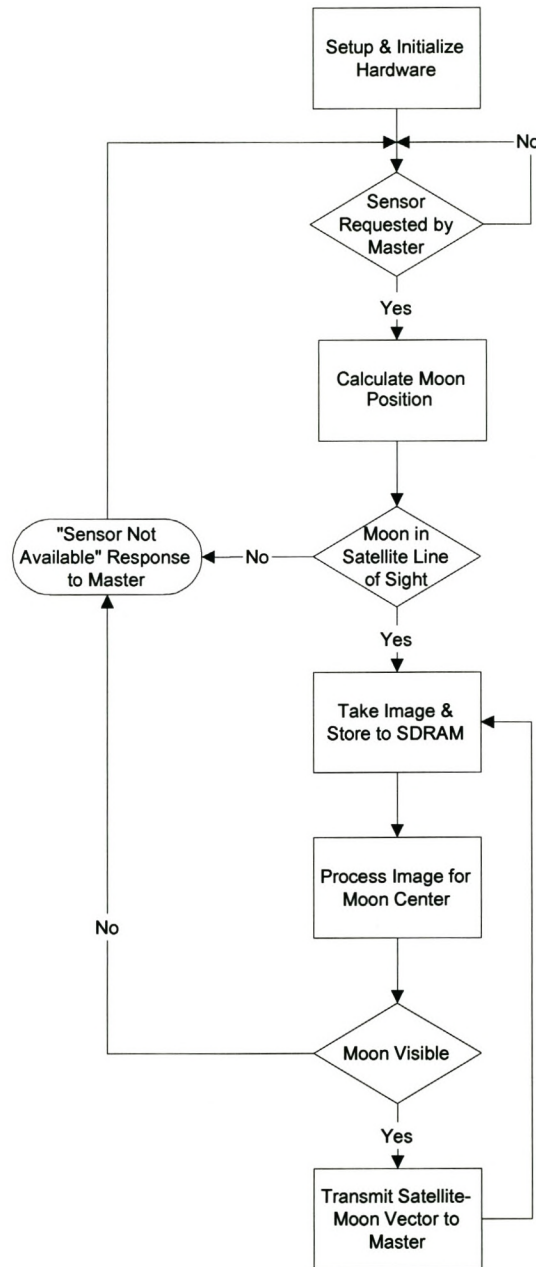


Figure 4.10 *Flow Diagram of Sensor Software Structure*

As shown in Figure 4.10, after initialization, the sensor waits for the Master (PC/Satellite) to request its use. When the request signal is received, the sensor runs the moon position algorithm to see if the moon is visible to the satellite at that time. If the moon is not visible the sensor sends a “moon not visible” result to the PC/satellite.

If the moon is visible, the sensor takes an image and processes it for the center coordinate of the moon. This coordinate is sent to the PC and is used along with the satellite’s position to calculate the vector from satellite to moon. This process continues until the moon is no longer visible to the sensor. The sensor then again sends a “moon not visible” result to the PC/satellite.

4.3.3 Image Processing - Software Implementation

The algorithm for locating the moon’s center was developed, implemented and tested in Matlab (Appendix A). It was then re-coded to C and transferred to the DSP board (Appendix B). A single refinement, as discussed below, was made to the code during this process stemming from the specific structure of the image taken and stored in SDRAM:

The image stored in the SDRAM is stored exactly as it is read from the video decoder: a serial stream of data separated into 2 fields which together form the entire frame. It is then processed field by field. The first field is processed and then, using the acquired values, only the relevant section in the second field is processed. This effectively decreases the number of pixels processed per image by somewhat less than half (The example in Figure 4.11 below only indicates the concept not specific values).

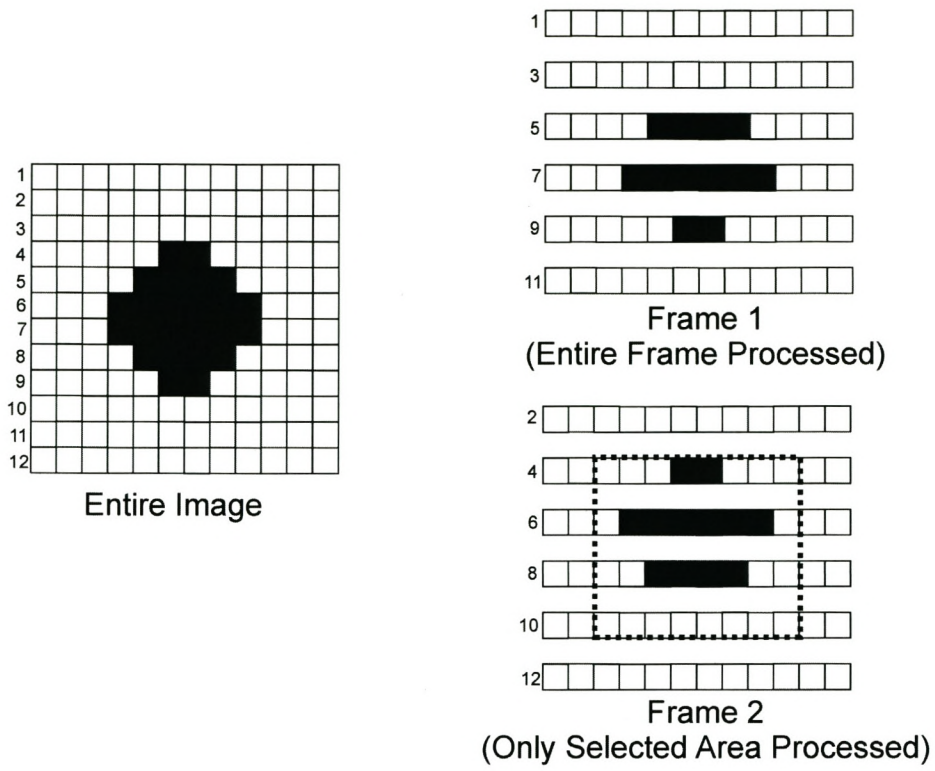


Figure 4.11 *Optimization of Image Processing Code*

The basic structure of the software algorithm can be seen in the following flow diagram:

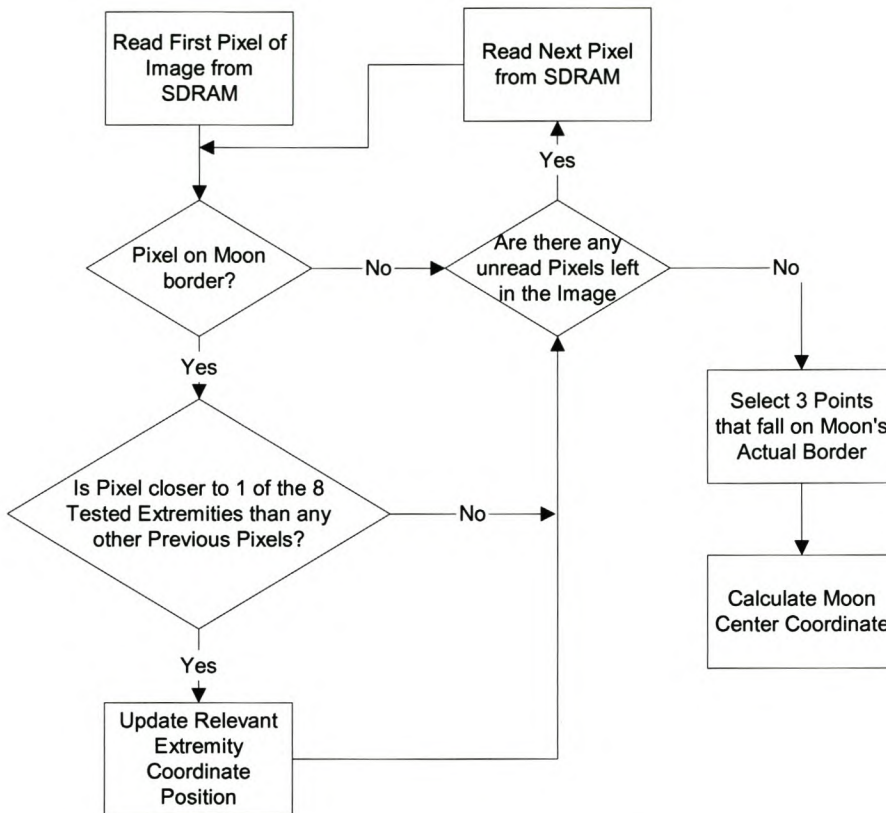


Figure 4.12 Basic Image Processing Algorithm Flow Diagram

The image pixel values are stored in memory as 1-dimensional array and are read serially from the memory. The position of the pixel in the array is converted to an equivalent x-y coordinate as in the actual image. This mapping process is handled by the processing software as each pixel is read and processed.

[*DSPMoonSensor.c*]

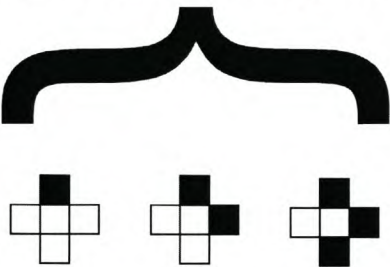
```

for (x = 3; x < (xBound-4); x = x+2)
{
    b = (x-1)/2;
    for (y = 20; y < ((720-20)); y++)
    {
        a = b*yBound + y*2 - 1;
    }
}
  
```

Here 'x' and 'y' specify the coordinate position of the pixel in the image. 'xBound' and 'yBound' specify the x and y resolutions respectively and 'a' the resultant position of the pixel in the linear array in memory.

Whether or not a pixel lies on the boundary of the moon can be tested according to the brightness of the pixel: if the level is above a certain threshold, it might be a boundary pixel. All adjacent pixels of the possible boundary pixel are now tested to determine whether it is above or below the given threshold. For a pixel to qualify as a boundary pixel, at least one of the four adjacent pixels tested must be above the specified threshold, and at least one below. Figure 4.13 illustrates the different cases as tested by the algorithm (Each case rotated by a multiple 90° is also taken into account but not shown in the figure)

Center Pixel is an Edge Pixel



Center Pixel is not an Edge Pixel

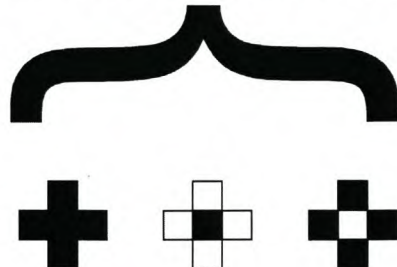


Figure 4.13 *Evaluation of Possible Edge Pixels*

The implementation of this test is shown in the code extract below.

[MatlabMoonSensor.m]

```

if (Im1(x,y) > Level)
  && ((Im1(x,y-1) <= Level)
      || (Im1(x,y+1) <= Level)
      || (Im1(x-1,y) <= Level)
      || (Im1(x+1,y) <= Level))
  && ((Im1(x,y-1) > Level)
      || (Im1(x,y+1) > Level))

```

```

|| (Im1(x-1,y) > Level)
|| (Im1(x+1,y) > Level))

```

'Im1' is the image in x-y plane. 'x' and 'y' are the coordinates of the pixel under inspection. Testing a pixel on the perimeter will result in an error because it would require that a (non-existent) pixel outside of the bounds of the image be tested. Thus the boundary pixels of the image are excluded during the processing of the image.

The process of calculating the moon's center coordinate is explained in four steps in Section 3.3.1 of this thesis. The implementation of this process is described below separated into the same 4 steps. For a full explanation of each step refer to the corresponding step in section 3.3.1.

1. The process of acquiring eight points on the moon border, as mentioned in section 3.3.1, is applied as in the following code:

[Pseudo-code describing selection of above mentioned points]

```

if (MaxDistance < CurrentPixelDistance)
MaxCoordinate = CurrentPixelCoordinate

if (MinDistance > CurrentPixelDistance)
MinCoordinate = CurrentPixelCoordinate

```

'MaxDistance' refers to the maximum perpendicular distance found, from the axis in question, out of all boundary pixels tested thus far. 'CurrentPixelDistance' refers to the current pixel being tested. If the current pixel betters the previous best pixel in the tested criteria, the coordinate of the previous best pixel is updated with the current pixel coordinate. The same reasoning follows for the 'MinDistance'. These criteria are tested for all 4 axes.

The following code is used to calculate the four bisecting line lengths:

[MatlabMoonSensor.m]

```

xDist = round(sqrt((xMin(1) - xMax(1))^2 + (xMin(2)
    - xMax(2))^2));
yDist = round(sqrt((yMin(1) - yMax(1))^2 + (yMin(2)
    - yMax(2))^2));

xyDist = round(sqrt((xyMin(1) - xyMax(1))^2 + (xyMin(2)
    - xyMax(2))^2));
yxDist = round(sqrt((yxMin(1) - yxMax(1))^2 + (yxMin(2)
    - yxMax(2))^2));

```

'xMin' and 'xMax' refer to the nearest and furthest points from the x axis respectively. 'xDist' refers to the distance between these two points (x-bisecting line), rounded to the nearest pixel. The same follows for yDist, xyDist and yxDist. The two points subtending the longest of these four lines is selected.

2. The perpendicular distance from the longest of the four bisecting lines to a point furthest from it, to either side is calculated using the cosine rule. (Refer to procedure: SideSelect in DSPMoonSensor.c for implementation)
3. Two more points are now selected according to the procedure described in Step 3 of section 3.3.1. The example code below is for the case of xMin and xMax being selected in Step 1 and the side selection procedure returning the point at yMin.

[MatlabMoonSensor.m]

```

x1 = yMin(1);
x2 = xyMax(1);
x3 = yxMin(1);
y1 = yMin(2);
y2 = xyMax(2);
y3 = yxMin(2);

```

(x_1, y_1) , (x_2, y_2) , (x_3, y_3) refer to the 3 points used in the calculation of the moon's center.

4. These three points are used to solve the three linear equations given in section 3.3.1 and as result the coordinate for the center of the moon is obtained. The linear equations were solved as follows in the Matlab code:

[MatlabMoonSensor.m]

```
res = inv([x1 y1 1; x2 y2 1; x3 y3 1])
      * (-[x1^2+y1^2; x2^2+y2^2; x3^2+y3^2]);
center = [-res(1)/2 -res(2)/2];
```

The procedure followed in the C-code (on the DSP) for Step 4, looks as follows:

[DSPMoonSensor.c]

```
void getCenter()
{
    float detA = x1*(y2 - y3) - y1*(x2 - x3)
                + (x2*y3 - y2*x3);
    float invA[3][3] = {{(y2 - y3)/detA,
                        -(y1 - y3)/detA,
                        (y1 - y2)/detA},
                       {-(x2 - x3)/detA,
                        (x1 - x3)/detA,
                        -(x1 - x2)/detA},
                       {(x2*y3 - y2*x3)/detA,
                        -(x1*y3 - y1*x3)/detA,
                        (x1*y2 - y1*x2)/detA}};

    float xRes = (invA[0][0]*(-(pow(x1,2)+pow(y1,2)))
                  + invA[0][1]*(-(pow(x2,2)+pow(y2,2)))
                  + invA[0][2]*(-(pow(x3,2)+pow(y3,2))))/(-2);
    float yRes = (invA[1][0]*(-(pow(x1,2)+pow(y1,2)))
                  + invA[1][1]*(-(pow(x2,2)+pow(y2,2)))
                  + invA[1][2]*(-(pow(x3,2)+pow(y3,2))))/(-2);
```

Where 'detA' is the determinant of the matrix and 'invA' the inverse. Then 'xRes' and 'yRes' are the x and y coordinates of the moon center respectively.

4.3.4 Communications Software Structure

The communications between the PC and the Sensor was implemented on a 1-to-1 (request-response) basis.

To Sensor:

1. Sensor Data Request.
2. Satellite position coordinates.

To Satellite:

1. Response as to whether or not the moon is available.
2. If moon is available and located: moon coordinate position within image is returned.

The relationship between satellite and sensor is illustrated in Figure 4.8. The satellite sends a request to the sensor. If the sensor detects the moon, it begins to calculate and send the required data to the satellite and will continue to do so until the moon is no longer visible to the sensor. If the moon is not visible to the sensor, a message is sent to the satellite that the sensor cannot be used at that point. (Note that in this study, the visibility of the moon was determined by eye and the acquisition by hand, thus the moon position algorithm and satellite coordinate position were not used).

The actual physical communications interface was set up as follows (7N1 @ 115k2 baud):

- 115k2 baud rate.
- 7 data bits
- 1 stop bit
- No parity bits

The communications structure was implemented as above and functioned correctly for the purposes of this study. The exact format of the data package sent to and from the satellite can be changed to suit specific application needs.

Chapter 5

Simulations and Test results

5.1 Evaluation Procedure Overview

The moon center and position algorithms were implemented and tested in Matlab. The moon center algorithm code was then translated to C to run in real-time on the DSP development board. It was confirmed from numerous tests that the algorithm returned the same results in Matlab as from the DSP, thus all simulations and tests for this algorithm were performed in Matlab and these results are shown in this chapter. This chapter discusses the simulation and test results of the moon center algorithm, the aim being to investigate its functionality.

The moon center algorithm was analyzed for a number of scenarios to evaluate its functionality. Each scenario was tested using the moon in a number of different phases. The following tests were performed:

1. Algorithm evaluation using ideal moon images varying in size to determine consistency with regards to moon image size
2. The moon within the image is rotated through 360° to assess consistency with regards to rotation
3. Moon position within image was shifted to different positions within the image to ensure the algorithm functions regardless of moon position
4. Actual images are processed and the results are discussed

An approximate error evaluation was performed and discussed using the data as obtained from the test cases.

5.2 Moon Center Algorithm

5.2.1 Results for Varying Moon Image Size (Using Matlab)

Ideal moon images were acquired (Cayless [8]) and used in the testing of the moon center algorithm. Image dimensions are 480x480 pixels with the full-moon diameter slightly smaller at approximately 456 pixels. The pictures were progressively scaled down until the resultant moon diameter was approximately equivalent to that of an actual moon image as expected from the sensor (Radius of approximately 13 pixels).

In the figures below, only the largest (Figure 5.1) and smallest (Figure 5.2) images that were tested are shown. It can be seen from these figures that the moon center appears to be calculated successfully for all tested moon images. An error evaluation for both sets of moon sizes, as shown below, appears in Section 5.3.

The functionality of the algorithm for varying moon size was first tested for the large images (Radius 227.24 pixels). The results are shown in Figure 5.1, for phases as seen on day 1 to day 15 of the moon phase cycle.



Figure 5.1 *Large Moon Images with Average Radius of 227.24 pixels*

The smaller images (Radius 13.08 pixels) tested are shown in Figure 5.2. The images are enlarged to show the result more accurately. These images also reflect the phase of the moon from day1 to day 15 in the moon orbit.

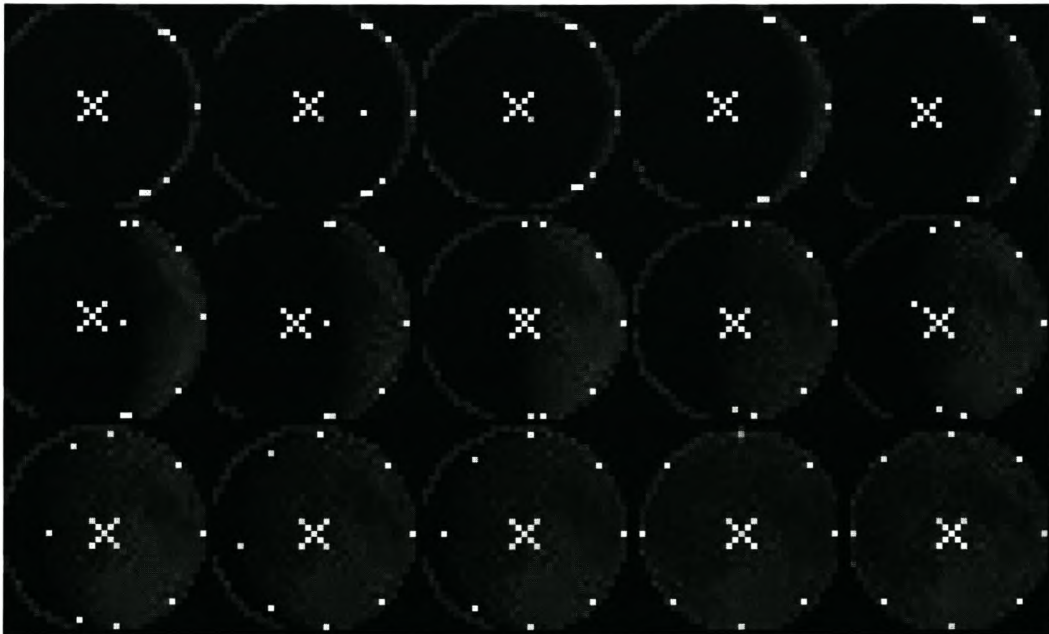


Figure 5.2 *Small Moon Images with Average Radius of 13.08 pixels*

5.2.2 Results for Varying Moon Image Rotation (Using Matlab)

The algorithm is developed in such a way that if the axis system in which the algorithm functions is rotated by any multiple of 45° , the resultant points on the moon's border will remain the same, as if it had not been rotated. Conversely, if the moon is rotated within the image by any multiple of 45° , the points selected on the moon border will remain the same. Therefore, to indicate the algorithm's rotation independence, it is only necessary to rotate and process the moon image between 0° and 45° . A moon image of approximately 75% phase and rotated in 5° increments was used for this study. This reflects the behavior of the algorithm throughout a full 360° rotation. This evaluation is applied to only the smallest moon images from Section 5.2.1.

In Figure 5.3 below, it is indicated that the moon center is calculated successfully regardless of the rotation of the moon image between 0° and 45° .

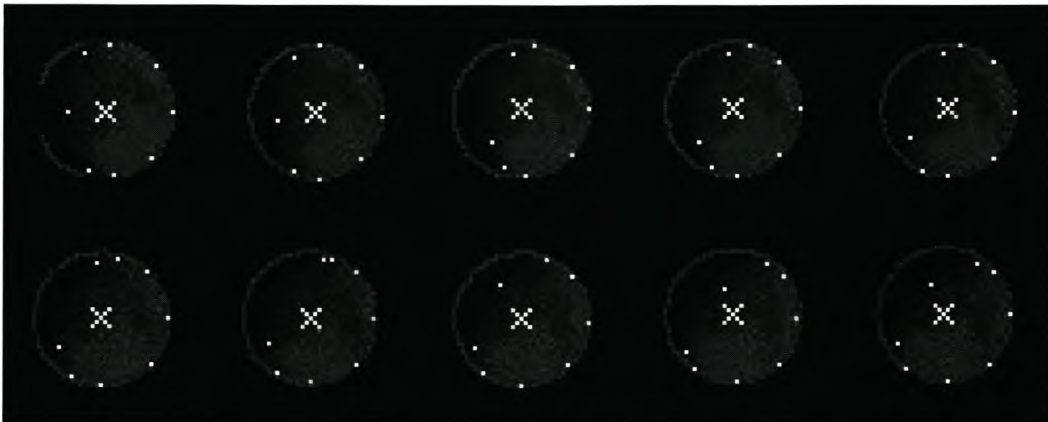


Figure 5.3 Moon Image Rotated in 5° Increments from 0° to 45°

As illustration that the algorithm does function correctly for any rotation of the moon, an image rotated from 0° to 315° in 45° increments is shown in Figure 5.4. From both the figure above and below it follows that the algorithm is rotation independent.

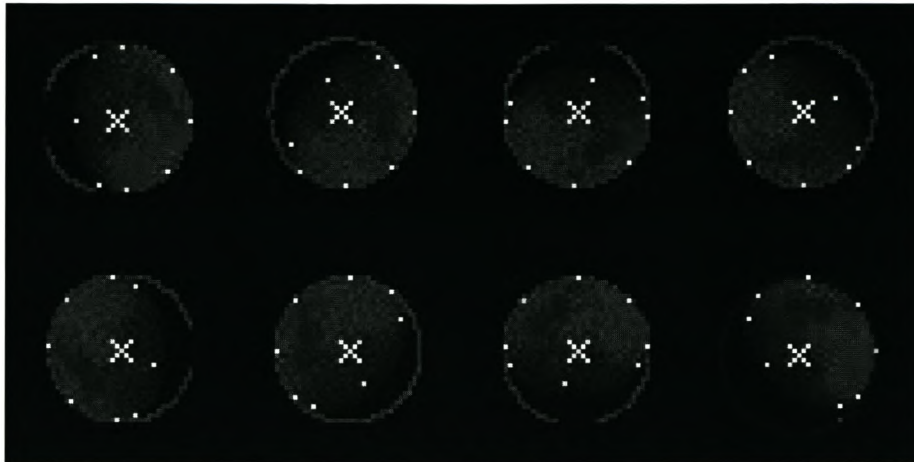


Figure 5.4 *Moon Image Rotated in 45° increments from 0° to 315°*

5.2.3 Results for Varying Moon Position within Image (Using Matlab)

From the previous test cases it is apparent that the moon, when in the center of the image, processes correctly. In this test case the moon is shifted to different positions within an image, the results are indicated in Figure 5.5 below.



Figure 5.5 *All Positions of Moon within Image Tested*

From Figure 5.5, the moon center is calculated correctly for all evaluated positions (Note: the results of each separate image have been combined to form Figure 5.5, each moon image was processed individually at the positions as indicated in the figure).

The axes relative to which the moon's extremities are calculated are situated outside of the borders of the image (see Figures 3.14 and 3.16) Therefore the moon's position within the image does not influence the results. The algorithm is position independent.

5.2.4 Algorithm Results for Actual Moon Images (Using Matlab)

Pictures of a perfectly round circle were taken by the sensor camera and it was observed that the image downloaded to the computer is distorted along the horizontal axis. The horizontal diameter was approximately 6-7% smaller than the vertical diameter. To demonstrate the distortion, a picture of a perfect circle is shown in Figure 5.6, as taken by the sensor.

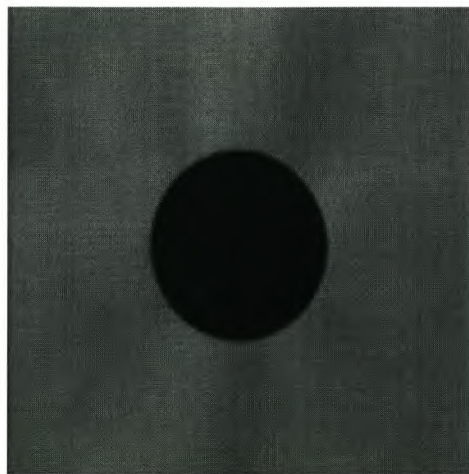


Figure 5.6 *Image of Perfect Circle Showing Distortion by Sensor*

Images of the moon in a few different phases were acquired and processed. Due to the distortion of the image the resultant moon centers were expected to have a lesser accuracy. The error margin will be discussed and evaluated in section 5.3.3.



Figure 5.7 *Actual Moon Images as Taken by Sensor (Average Radius 14.79 pixels)*

As can be seen in the images above (Figure 5.7), the algorithm calculates a center for each actual image. This center falls approximately where it would be expected (as judged by the naked eye). The distortion of the image does thus not appear to have a far reaching effect on the result, the actual error is discussed in section 5.3.3. Also evident in the image above is that the moon becomes more overexposed as it nears full-moon. The full-moon image however does not differ from the other phases by too great a margin. The effects of the overexposure will be discussed in more detail in section 5.3.

5.3 Moon Center Error and Accuracy Evaluation

A consideration for error evaluation was constructed to approximate the moon center calculation's error margin. Error evaluation was applied first to ideal images of a few selected phases of the moon. For the case of full moon, the center of the image was predicted, then the actual center was calculated, these values were then compared and possible error was discussed. For moon images of different phases the radius obtained for each phase is compared to an average radius over all the test cases. An error evaluation for images taken by the sensor was then formulated and the results discussed

5.3.1 Algorithm Error Evaluation on a Full Moon

In the case of a full moon, the point of intersection of the x-bisecting and y-bisecting lines was taken as the predicted center of the moon. This value is compared to the result of the moon center algorithm. This comparison is done for ideal moon images with radius equal to that of the largest and smallest tested images. The results are indicated and summarized in the following table.

<u>Moon</u> <u>Radius</u>	<u>Predicted</u> <u>Center</u>	<u>Algorithm</u> <u>Center</u>	<u>Difference</u>
226.88	(241.0; 241.5)	(241.3; 241.9)	0.5 pixels
13.08	(14; 14)	(14.1; 14.1)	0.1 pixels

Table 5.1 *Full Moon Center Predictions vs. Algorithm Center Calculation*

As can be seen in Table 5.1 above, the difference between the predicted and calculated centers of the moon in the case of the large moon image was 0.5 pixels and the smaller moon image, 0.1 pixels. This indicates that the algorithm has a high accuracy in calculating the full-moon center regardless of size. Although the difference is small for both cases tested, it must be noted that, relative to each moon image size, the accuracy is quite different. For the larger of the two moons the 0.5 pixel difference is 0.22% of the radius of that particular image, which equates to a high accuracy. Alternatively for the smaller of the two images the difference of 0.1 pixels is 0.765% of the radius of that image, which equates to a slightly lesser accuracy. The drop in accuracy as the image decreases in size is as a result of the resolution of the image decreasing.

5.3.2 Algorithm Error Evaluation on Other Phases

The accuracy of the algorithm for the other phases of the moon is evaluated by comparing the resultant radii of all phases including full-moon to an average radius calculated across all phases. This evaluation was applied for the largest and smallest moon images as processed in Section 5.2.1, the results are displayed in Figures 5.8 and 5.9 respectively.

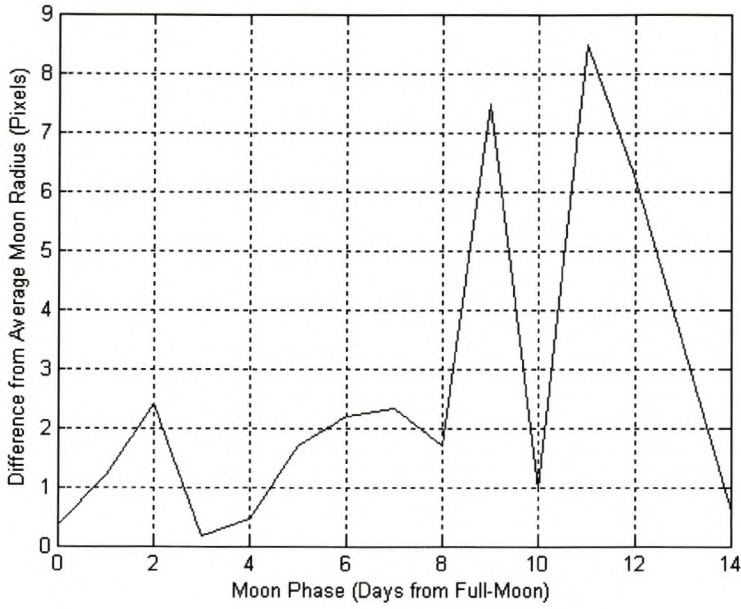


Figure 5.8 *Large Moon Images (Avg. Radius 227.24 pixels)*

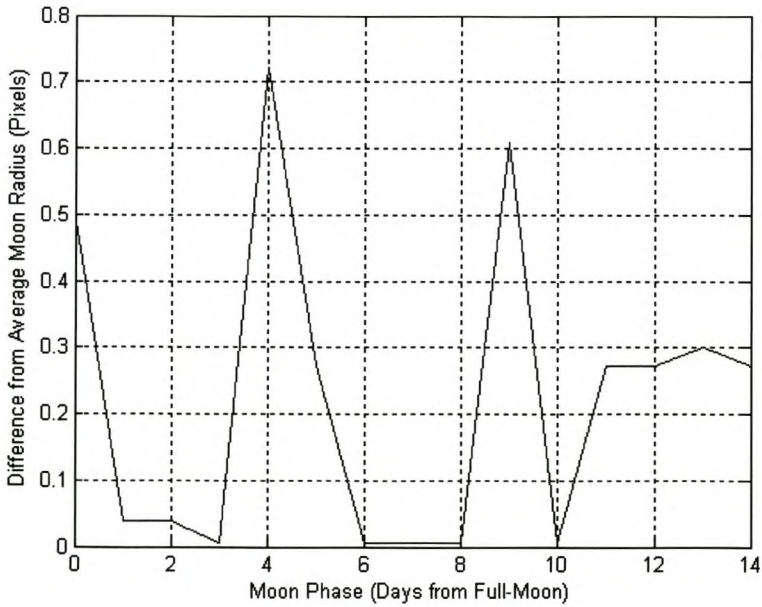


Figure 5.9 *Small Moon Images (Avg. Radius 13.08 pixels)*

The worst case result for the large moon is approximately 8.49 pixels from the average radius (3.74% of avg. radius) and the worst case from the smaller moon is approximately 0.721 pixels (5.51% of avg. radius). To put the previous values into some perspective it is assumed that the moon has a diameter of 0.5° resulting in the error for the larger moon of approximately 0.009° and the smaller approximately 0.014° .

Judging by the results from the above two figures and from further inspection of the processed images, it was found that the moon images did not all have the same radius. The difference in radius is possibly due to the fact that the conditions of capturing each phase were not precisely the same (different zoom on the camera, moon in different position in the sky). The results above are therefore not a very accurate indication of the Algorithm's accuracy over varying phases of the moon but do give an indication of its functionality. Moon images of the exact same radius would be required for a more accurate result. The algorithm should in theory be more accurate than indicated.

As illustration of an ideal case, a single full-moon image was used and the phase was implemented by shading out the relevant section of the moon manually. The algorithm results in the same center value for each image, thus no error is measurable. The result is shown in the following figure (Figure 5.10).

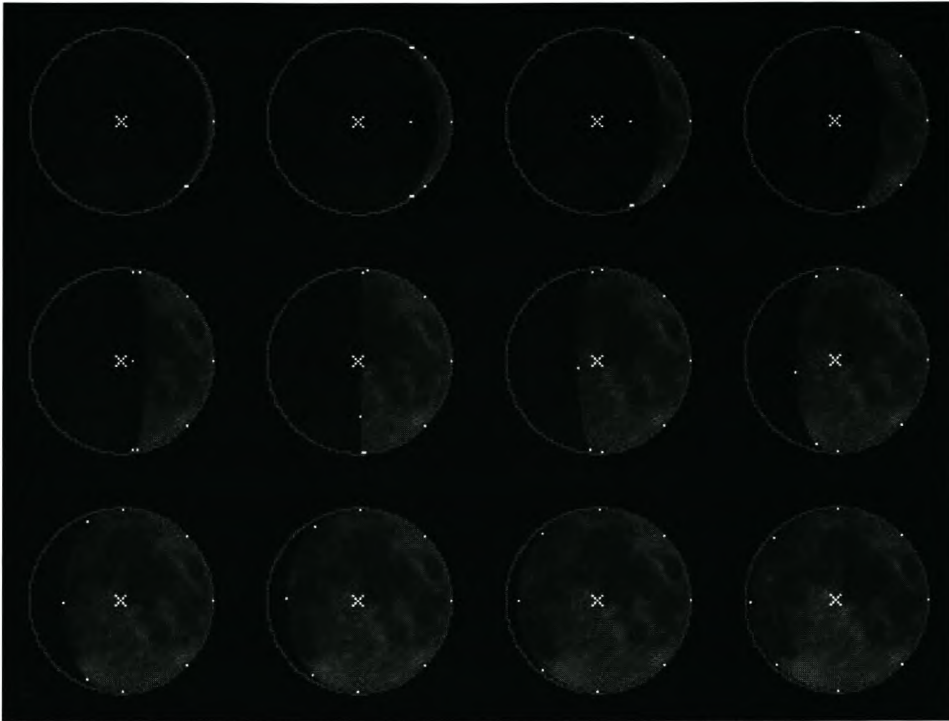


Figure 5.10 *Ideal Moon Images with Exact Same Radius (Radius 38.64 pixels)*

5.3.3 Algorithm Error Evaluation for Images Taken by Actual Sensor

The effects of the distortion of actual moon images is evaluated by comparing the calculated center of a half-circle (representing a half-moon) with the predicted center of a perfect circle with same position and radius. The half-circle is captured first with its flat edge lying parallel to the x-axis, second with its flat edge lying parallel to the y-axis and lastly with its flat edge lying parallel to the xy-axis. The center of the perfect circle is predicted by the same process as followed in section 5.3.1.



Figure 5.11 *Perfect Circle Predicted and Calculated Centers*

<u>Moon Image</u>	<u>Predicted Center</u>	<u>Calculated Center</u>	<u>Difference</u>
Perpendicular flat edge	(96; 93)	(95.66; 97.87)	4.88 pixels
Horizontal flat edge	(96; 93)	(91.72; 93.06)	4.28 pixels
45° flat edge	(96; 93)	(95.82; 95.82)	2.83 pixels

Table 5.2 *Distortion Results of Calculated center of Perfect circle*

The first case results in a too-large circle being fitted around the moon image and thus the moon center will be offset slightly in the y-direction. In the second case, the fitted circle is too small, also resulting in the calculated moon center being off-set, this time in the x-direction (see Figure 5.11). For the image at a 45° angle the result is closest to that predicted for the circle. The results of these evaluations are summarized in Table 5.2.

The diameter of the distorted circle is approximately 69 pixels along the y-axis and 74 pixels along the x-axis. An average radius was calculated at 71.5 pixels to use in calculating the approximate error. The maximum error caused by the distorted image is 4.88 pixels (approximately 6.83% of the average radius). This result is within a similar range as the results obtained with ideal moon images of varying size and phase. It would appear that the distortion of the moon image does not influence the accuracy of the algorithm too drastically since the difference in x and y diameter is in the region of 6.76%.

Due to the camera used in this project having a specific minimum aperture the exposure of the moon was difficult to control when near and equal to full moon. As a result these phases are slightly overexposed. This overexposure results in a larger calculated radius than for the other phases. The radii as measured for the images in Figure 5.7 are depicted in the graph below compared to an average of all the radii:

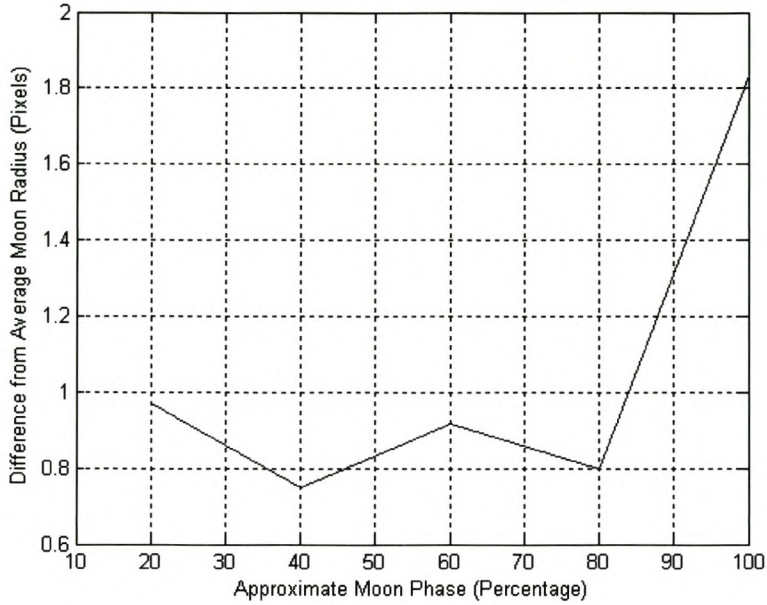


Figure 5.12 *Actual Moon Images (Avg. Radius 14.79 pixels)*

The largest difference in radius is at full-moon due to the glow added by overexposure. Because this slight overexposure only occurs near and on full-moon, it basically only enlarges the existing borders of the moon and should not affect the general shape of the moon or the result of the moon center algorithm greatly.

Chapter 6

Conclusions and Recommendations

6.1 Summary

Three major areas are investigated in this thesis:

1. The feasibility of a moon sensor was considered with regards to the visibility of the moon from the satellite. Refer to Section 6.1.1.
2. A moon position algorithm was implemented to calculate the position of the moon relative to that of the sun and the satellite. Refer to Section 6.1.2.
3. An algorithm to calculate the center of the moon, regardless of phase or rotation, was developed and evaluated by means of tests and simulation. Refer to Section 6.1.3.

6.1.1 Sensor Feasibility with Moon as Target (Summary)

The visibility of the moon from the satellite was evaluated for a number of positions of the moon in its orbit and the satellite in its orbit. The only obstructions to the satellite's view of the moon are the earth and the sun. Taking all the above into account, a number of specific scenarios were constructed and evaluated. The results were compared to the visibility of the sun.

The visibility of the moon proved to be on average either equal to, or better when compared to that of the sun and this sensor option was decided a viable research topic. The moon sensor offers the following three advantages when the sun is not visible or when used in conjunction with a sun sensor.

- When the moon is visible during the eclipse period of the satellite orbit. An extra point of reference is now available to the satellite since the sun is not visible.
- There are periods when the moon is visible throughout the satellite orbit. The moon offers a consistent point of reference during the entire satellite orbit while the sun is only visible for a part of it.
- There are also times when both the moon and sun are visible simultaneously, but not both fall within the sensor FOV. This results in the satellite having two points of reference to make use of.

A disadvantage of the moon sensor is that, during the time that the sun and moon fall in the sensor FOV simultaneously, the sensor is of no use. This fact leads to the conclusion that the moon sensor would not be a replacement for a sun sensor, but rather an additional sensor, to be used in conjunction with a sun sensor. The advantage of this is that, a single sensor always has an axis about which it cannot determine rotation, thus with the addition of a second sensor, this axis of un-surety is removed.

6.1.2 Moon Position Algorithm

The moon and sun position algorithms implemented were sufficiently accurate to predict the position of the moon and sun as required in this thesis. The moon position algorithm is specified to be accurate to 10arc minutes (approximately 1/3 of the moon diameter), and the sun position to 0.01°.

6.1.3 Moon Center Algorithm

A moon center algorithm was developed to calculate the center of the moon within an image, regardless of the moon's phase or rotation. The required result of the algorithm was to increase the accuracy of a sensor that uses the moon as point of reference. The analysis of this algorithm was done for a number of different cases and an approximate-error evaluation was formulated.

The algorithm proved to be consistently accurate over a range of moon phases, sizes, positions and rotations within the image. There was a slight decrease in accuracy for actual moon images taken by the sensor itself due to slight distortion of the image. The typical error in accuracy calculated for this algorithm lay between 5.51% and 6.83% of the given moon radius. It is reasonable to say that a projected accuracy for the moon center algorithm would be within 10% of the moon's radial length from the true center.

A factor that influences the image processing and as a result the moon center processing is the brightness of other stellar bodies (i.e. the sun, planets and stars). The influence of this factor is largely decided by the sensitivity of the sensor-camera. Since even the brightest planet and star are much less bright than the moon, a sensor that is calibrated to image the moon would not be sensitive enough to 'see' planets or stars. This implies that there is no need of the extra expense of purchasing a high sensitivity camera. Alternatively, the sun is much brighter than the moon; this would pose a problem if the sun falls in the sensor's FOV. This problem is easily bridged by implementing software that calculates the angle between the sun and moon and ensures the camera is not used when the sun falls within its FOV.

6.2 Conclusion, Considerations and Recommendations

From the evaluation of this moon sensor, it was found to have reasonable accuracy (within approximately 10% of the moon's radius). The visibility of the moon, using simplified orbit assumptions, proved to be mostly greater than that of the sun, resulting in a consistent reference to be used by the sensor. A more accurate orbit specification for

the moon orbit should be implemented in future studies. The fact that there are times when the moon is not visible at all, results in the consideration of this sensor as a supplemental sensor and not a replacement sensor.

The hardware supplied for this sensor proved to be more than sufficient for the purposes of this study. The DSP evaluation board (Running at 270MHz not the maximum possible of 600MHz) was able to photograph and completely process 1 image every 125ms (8 images per second), sending the result to the computer via the serial port. The camera used has a narrow FOV and could supply a moon image sufficiently large to be recognized and processed by the moon center algorithm.

The components used in this sensor did not require any special functionality or sensitivity for the sensor to function correctly, hence by implementing the sensor on less complex hardware the cost could be minimized. In conclusion the moon sensor is an accurate and affordable supplemental sensor that could be seriously considered for application in a satellite system in conjunction with a sun sensor.

The nature of the moon center algorithm (calculating the center of a round object) allows the sensor, with a few minor changes, to be implemented for a sun sensor or even a type of star sensor. The former would require a filter for the lens to accommodate the brightness of the sun. In the latter case, a camera with higher sensitivity should be used and the sensor could make use of a bright polar star or a star that is always visible from the satellite.

With regards to the physical positioning of the sensor on the satellite, this can be selected from a number of configurations. A few possibilities include:

- A single-fixed wide FOV sensor which results in a less accurate sensor with function limited to when the moon falls in the sensor's FOV. Other problems such as the earth or sun falling within in the sensor FOV also need to be considered.

- Multiple-fixed wide FOV sensors increase the amount of time the moon is visible to the sensors. Similar problems as with the previous option exist with the added complexity of active sensor selection (only the sensor with the moon in its FOV must be active).
- A narrow FOV sensor that can be pointed, using servo-motors or other means, in the direction of the moon each time the moon is visible. This sensor will be the most accurate and is usable whenever the moon is visible. The accuracy of the sensor will increase by decreasing the FOV of the camera.

The satellite-moon vector is calculated relative to the direction in which the sensor is pointing. This allows for the sensor to be placed anywhere on the satellite as long as the satellite is aware of the sensor's position and orientation it can calculate the required vector from the supplied center-coordinate.

The approach with regards to image processing followed in this study was simple yet effective. It is possible to develop an algorithm which would be faster in its acquisition of the moon center and more accurate, by taking into account and compensating for image distortion as seen in the test images.

Bibliography

- [1] J. R. Wertz and W. J. Larson, *Space Mission Analysis and Design*. Microcosm Press and Kluwer Academic Publishers, 1999.
- [2] M. Bellezza, D. Mortari and R. Perfetti, *Moon Image Processing for Spacecraft Attitude Estimation*.
- [3] Analog Devices, Inc., *ADSP-BF533 EZ-KIT Lite™ Evaluation System Manual*. Analog Devices, Inc., 2003.
- [4] Analog Devices, Inc., *ADSP-BF533 Blackfin® Processor Hardware Reference*. Analog Devices, Inc., 2003.
- [5] Analog Devices, Inc., *Advanced Video Decoder with 10-Bit ADC and Component Input Support*. Analog Devices, Inc., 2002.
- [6] J. Meeus, *Astronomical Algorithms* Willman-bell, Inc. Richmond Virginia 23235, ISBN 0-943396-35-2
- [7] J. R. Wertz, *Spacecraft Attitude Determination and Control*. D. Reidel Publishing Company, 1978.
- [8] A. Cayless, *Moon Phase Project*. Stirling Astronomical Society Webpage, 2003
<http://www.stirlingastronomicalsociety.org.uk/moonphase%20project/moonphase.html>
- [9] J. Kaler, *Sirius (Alpha Canis Majoris)*. Webpage.
<http://www.astro.uiuc.edu/~kaler/sow/sirius.html>

Appendix A

Matlab Implementation of Moon Center Algorithm

The moon center algorithm, as discussed in Sections 3.3.1 and 4.3.3, can be implemented in Matlab as shown in **MatlabMoonSensor.m** and **SideSelect.m**.

MatlabMoonSensor.m:

```
%=====
% Implementation of Moon Center Algorithm in Matlab %
%=====

clear all;

%-----

% Load Image from File
Im1 = imread('MoonImage','jpg');
Imi = Im1(:,:,1);
Im1 = double(Imi);
Im3 = 0.5*double(Imi);
Im4 = double(Imi);

% Display Image
imshow(Imi);
figure;

% Acquire x and y Bounds for Image
xBound = length(Im1(:,1));
yBound = length(Im1(1,:));
Level = 120;
```

```
if (xBound > yBound)
    ScaleFac = xBound;
else
    ScaleFac = yBound;
end;

% Initialize all Values to 0
xMinCnt = 0;
xMaxCnt = 0;
yMinCnt = 0;
yMaxCnt = 0;

xyMinCnt = 0;
xyMaxCnt = 0;
yxMinCnt = 0;
yxMaxCnt = 0;

xMin = [0, 0];
xMax = [0, 0];
yMin = [0, 0];
yMax = [0, 0];

xMin1 = [0, 0];
xMax1 = [0, 0];
yMin1 = [0, 0];
yMax1 = [0, 0];

xMin2 = [0, 0];
xMax2 = [0, 0];
yMin2 = [0, 0];
yMax2 = [0, 0];

xyMin = [0 0];
xyMax = [0 0];
yxMin = [0 0];
yxMax = [0 0];

xyMin1 = [0 0];
xyMax1 = [0 0];
yxMin1 = [0 0];
yxMax1 = [0 0];
```

```

xyMin2 = [0 0];
xyMax2 = [0 0];
yxMin2 = [0 0];
yxMax2 = [0 0];

%=====
% Locate Edge Pixels at x, y ,xy and yx Extremities

for x = 1:xBound
    for y = 1:yBound
        if (Iml(x,y) > Level)
            && ((Iml(x,y-1) <= Level)
            || (Iml(x,y+1) <= Level)
            || (Iml(x-1,y) <= Level)
            || (Iml(x+1,y) <= Level))
            && ((Iml(x,y-1) > Level)
            || (Iml(x,y+1) > Level)
            || (Iml(x-1,y) > Level)
            || (Iml(x+1,y) > Level))

            if (xMin1(1) == 0) && (xMax1(1) == 0) && (yMin1(1) == 0)
                && (yMax1(1) == 0)

                xMin1 = [x y];
                xMax1 = [x y];
                yMin1 = [x y];
                yMax1 = [x y];

                xyMin1 = [x y];
                xyMax1 = [x y];
                yxMin1 = [x y];
                yxMax1 = [x y];

            end;

            xyNewLength = round(sqrt(2*(x + ScaleFac - y)^2));
            xyMinLength = round(sqrt(2*(xyMin1(1) + ScaleFac
                - xyMin1(2))^2));
            xyMaxLength = round(sqrt(2*(xyMax1(1) + ScaleFac
                - xyMax1(2))^2));

```

```
yxNewLength = round(sqrt(2*(x+y)^2));
yxMinLength = round(sqrt(2*(yxMin1(1)
    + yxMin1(2))^2));
yxMaxLength = round(sqrt(2*(yxMax1(1)
    + yxMax1(2))^2));

if (xMin1(1) > x)
    xMin1(1) = x;
    xMin1(2) = y;
end;
if (xMin1(1) >= x)
    xMin2(1) = x;
    xMin2(2) = y;
end;

if (xMax1(1) < x)
    xMax1(1) = x;
    xMax1(2) = y;
end;
if (xMax1(1) <= x)
    xMax2(1) = x;
    xMax2(2) = y;
end;

if (yMin1(2) > y)
    yMin1(2) = y;
    yMin1(1) = x;
end;
if (yMin1(2) >= y)
    yMin2(2) = y;
    yMin2(1) = x;
end;

if (yMax1(2) < y)
    yMax1(2) = y;
    yMax1(1) = x;
end;
if (yMax1(2) <= y)
    yMax2(2) = y;
    yMax2(1) = x;
```

```

end;

%-----

if (xyMinLength > xyNewLength)           % Bottom Left
    xyMin1 = [x y];
end;

if (xyMinLength >= xyNewLength)          % Bottom Left
    xyMin2 = [x y];
end;

if (xyMaxLength < xyNewLength)          % Top Right
    xyMax1 = [x y];
end;

if (xyMaxLength <= xyNewLength)         % Top Right
    xyMax2 = [x y];
end;

if (yxMinLength > yxNewLength)          % Top Left
    yxMin1 = [x y];
end;

if (yxMinLength >= yxNewLength)         % Top Left
    yxMin2 = [x y];
end;

if (yxMaxLength < yxNewLength)          % Bottom Right
    yxMax1 = [x y];
end;

if (yxMaxLength <= yxNewLength)         % Bottom Right
    yxMax2 = [x y];
end;

    Im2(x, y) = 0;
else
    Im2(x, y) = 0;
end;
end;

```



```

end;

xMin = [xMin1(1) round((xMin2(2) + xMin1(2))/2)];
xMax = [xMax1(1) round((xMax2(2) + xMax1(2))/2)];
yMin = [round((yMin2(1) + yMin1(1))/2) yMin1(2)];
yMax = [round((yMax2(1) + yMax1(1))/2) yMax1(2)];

xyMin = [round((xyMin2(1) + xyMin1(1))/2) round((xyMin2(1)
+ xyMin1(1))/2) - (xyMin2(1) - xyMin2(2))];
xyMax = [round((xyMax2(1) + xyMax1(1))/2) round((xyMax2(1)
+ xyMax1(1))/2) - (xyMax2(1) - xyMax2(2))];
yxMin = [round((yxMin2(1) + yxMin1(1))/2) (yxMin2(1) + yxMin2(2))
- round((yxMin2(1) + yxMin1(1))/2)];
yxMax = [round((yxMax2(1) + yxMax1(1))/2) (yxMax2(1) + yxMax2(2))
- round((yxMax2(1) + yxMax1(1))/2)];

%=====
% Calculate Longest Line

xDist = round(sqrt((xMin(1) - xMax(1))^2 + (xMin(2) - xMax(2))^2));
yDist = round(sqrt((yMin(1) - yMax(1))^2 + (yMin(2) - yMax(2))^2));

xyDist = round(sqrt((xyMin(1) - xyMax(1))^2 + (xyMin(2)
- xyMax(2))^2));
yxDist = round(sqrt((yxMin(1) - yxMax(1))^2 + (yxMin(2)
- yxMax(2))^2));

%=====
% Select Points on Actual Moon Border

if ((xDist >= yDist) && (xDist >= xyDist) && (xDist >= yxDist))

    Side = SideSelect(xMin, xMax, yMin, yMax);
    if (Side == 1)

        x1 = yMin(1);
        x2 = xyMax2(1);
        x3 = yxMin1(1);
        y1 = yMin(2);
        y2 = xyMax2(2);
        y3 = yxMin1(2);

```

```
    xyRad = yMin;

else

    x1 = yMax(1);
    x2 = xyMin1(1);
    x3 = yxMax2(1);
    y1 = yMax(2);
    y2 = xyMin1(2);
    y3 = yxMax2(2);

    xyRad = yMax;

end;

elseif ((yDist > xDist) && (yDist >= xyDist) && (yDist >= yxDist))

Side = SideSelect(yMin, yMax, xMin, xMax);
if (Side == 1)

    x1 = xMin(1);
    x2 = xyMin2(1);
    x3 = yxMin2(1);
    y1 = xMin(2);
    y2 = xyMin2(2);
    y3 = yxMin2(2);

    xyRad = xMin;

else

    x1 = xMax(1);
    x2 = xyMax1(1);
    x3 = yxMax1(1);
    y1 = xMax(2);
    y2 = xyMax1(2);
    y3 = yxMax1(2);

    xyRad = xMax;
```

```
end;

elseif ((xyDist > xDist) && (xyDist > yDist) && (xyDist >= yxDist))

    Side = SideSelect(xyMin, xyMax, yxMin, yxMax);
    if (Side == 1)

        x1 = yxMin(1);
        x2 = xMin2(1);
        x3 = yMin2(1);
        y1 = yxMin(2);
        y2 = xMin2(2);
        y3 = yMin2(2);

        xyRad = yxMin;

    else

        x1 = yxMax(1);
        x2 = xMax1(1);
        x3 = yMax1(1);
        y1 = yxMax(2);
        y2 = xMax1(2);
        y3 = yMax1(2);

        xyRad = yxMax;

    end;

elseif ((yxDist > xDist) && (yxDist > yDist) && (yxDist > xyDist))

    Side = SideSelect(yxMin, yxMax, xyMin, xyMax);
    if (Side == 1)

        x1 = xyMin(1);
        x2 = yMax2(1);
        x3 = xMin1(1);
        y1 = xyMin(2);
        y2 = yMax2(2);
        y3 = xMin1(2);
```

```

xyRad = xyMin;

else

    x1 = xyMax(1);
    x2 = yMin1(1);
    x3 = xMax2(1);
    y1 = xyMax(2);
    y2 = yMin1(2);
    y3 = xMax2(2);

    xyRad = xyMax;

end;
end;

%=====
% Calculate Moon Center From 3 Points on Edge

res = inv([x1 y1 1; x2 y2 1; x3 y3 1])*(-[x1^2+y1^2; x2^2+y2^2; x3^2+y3^2]);
center = [-res(1)/2 -res(2)/2];
rad = (sqrt((xyRad(1)+res(1)/2)^2 + (xyRad(2)+res(2)/2)^2));

```

%=====

SideSelect.m:

```

%=====
% Function to Select True Side of Moon Implementation of Cos Rule%
%=====

```

```

function [SecSelect] = SideSelect(PrimIn1, PrimIn2, SecIn1, SecIn2)
    Prim1_Prim2 = (sqrt((PrimIn1(1) - PrimIn2(1))^2 + (PrimIn1(2)
        - PrimIn2(2))^2));
    Prim1_Sec1 = (sqrt((PrimIn1(1) - SecIn1(1))^2 + (PrimIn1(2)
        - SecIn1(2))^2));
    Prim1_Sec2 = (sqrt((PrimIn1(1) - SecIn2(1))^2 + (PrimIn1(2)
        - SecIn2(2))^2));
    Prim2_Sec1 = (sqrt((PrimIn2(1) - SecIn1(1))^2 + (PrimIn2(2)
        - SecIn1(2))^2));
    Prim2_Sec2 = (sqrt((PrimIn2(1) - SecIn2(1))^2 + (PrimIn2(2)
        - SecIn2(2))^2));

```

```
if (Prim1_Sec1 < 2) && (Prim1_Sec2 > Prim1_Sec1)
    SecSelect = 2;
elseif (Prim1_Sec2 < 2) && (Prim1_Sec1 > Prim1_Sec2)
    SecSelect = 1;
else
    Sec1Dist = Prim1_Sec1*sin(acos((Prim1_Prim2^2 + Prim1_Sec1^2
        - Prim2_Sec1^2)/(2*Prim1_Prim2*Prim1_Sec1)))
    Sec2Dist = Prim1_Sec2*sin(acos((Prim1_Prim2^2 + Prim1_Sec2^2
        - Prim2_Sec2^2)/(2*Prim1_Prim2*Prim1_Sec2)))
    if (Sec1Dist >= Sec2Dist)
        SecSelect = 1;
    else
        SecSelect = 2;
    end;
end;
```

Appendix B

DSP Implementation of Moon Center Algorithm

The moon center algorithm, as discussed in Sections 3.3.1 and 4.3.3, was translated from Matlab to C so as to run on the DSP. All the hardware initialization and required software for the moon image processing can be implemented on the DSP as shown in **DSPMoonSensor.c**.

DSPMoonSensor.c:

```
//=====//
// The DSP initialization and Implementation of the //
// Moon Processing Algorithm //
//=====//

#include "cdefBF533.h"
#include "defBF533.h"

#include "Control.h"
#include "stdio.h"
#include "string.h"
#include "stdlib.h"
#include "math.h"

#define SDRAM_START_ADDR 0x00000000;

float x1, x2, x3, y1, y2, y3;
```

```
const int xBound = 576;
const int yBound = 1440;
const int PosF1_F2Offset = 576*1440/2;

// Pixel Intensity Level Criteria for Edge Pixel
int Level = 90;

//-----
// Reset All points to be found on Moon Border

float xmin[2] = {0, 0};
float xmax[2] = {0, 0};
float ymin[2] = {0, 0};
float ymax[2] = {0, 0};
float xymin[2] = {0, 0};
float xymin[2] = {0, 0};
float yxmin[2] = {0, 0};
float yxmin[2] = {0, 0};

float xmin1[2] = {0, 0};
float xmax1[2] = {0, 0};
float ymin1[2] = {0, 0};
float ymax1[2] = {0, 0};
float xymin1[2] = {0, 0};
float xymin1[2] = {0, 0};
float yxmin1[2] = {0, 0};
float yxmin1[2] = {0, 0};

float xmin2[2] = {0, 0};
float xmax2[2] = {0, 0};
float ymin2[2] = {0, 0};
float ymax2[2] = {0, 0};
float xymin2[2] = {0, 0};
float xymin2[2] = {0, 0};
float yxmin2[2] = {0, 0};
float yxmin2[2] = {0, 0};

float xyNewLength = 0;
float xyMinLength = 0;
float xyMaxLength = 0;
```

```

float yxNewLength = 0;
float yxMinLength = 0;
float yxMaxLength = 0;
//-----

unsigned short xMinNew;
    unsigned short xMaxNew;
    unsigned short yMinNew;
    unsigned short yMaxNew;

long i, j, c, k, l, cnt =0;
unsigned short Field, Rx, Tx, x, y, LSR = 0;
unsigned short EbiuSDStatus, AsyncMemGlobContReg = 0;

unsigned short pos[16] = {1, 2, 4, 8, 16, 32, 64, 128, 256, 512,
                        1024, 2048, 4096, 8192, 16384, 32768};

int ScaleFac;

//=====
// Rounding Function

int rnd(float input)
{
    if ((input - floor(input)) < 0.5)
        return floor(input);
    else
        return ceil(input);
}

//=====
//Functions that Transmit Different Data Types to PC

void charTX(unsigned short Text)
{
    LSR = 0x0000;
    //Wait for Data to be Cleared from Write Buffer
    while (LSR != 0x0020)
    {
        LSR = *pUART_LSR & 0x0020;
    }
}

```



```

    LSR = 0x0000;
    //Transmit Data
    *pUART_THR = Text;

    //Wait for Data Available in Receive buffer
    while (LSR != 0x0001)
    {
        LSR = *pUART_LSR & 0x0001;
    }

    Rx = *pUART_RBR;
    LSR = 0x0000;
}

void bitTX(unsigned short Text)
{
    unsigned short Rxb;

    for (j = 15; j > -1; j--)
    {
        Rxb = (Text & pos[j])/(pos[j]);/// $+48$ ;
        charTX(Rxb);
    }
}

void decTX(unsigned short Text)
{
    unsigned short Rxb, RxbNum1, RxbNum2, RxbNum3;

    RxbNum1 = div(Text, 100).quot;
    RxbNum2 = div(Text - RxbNum1*100, 10).quot;
    RxbNum3 = Text - RxbNum1*100 - RxbNum2*10;

    charTX(RxbNum1 + 48);
    charTX(RxbNum2 + 48);
    charTX(RxbNum3 + 48);
}

//=====
// Function that Calculates Center Coordinate of Moon Image

```

```

// and Transmits it to PC

void getCenter()
{
    float detA = x1*(y2 - y3) - y1*(x2 - x3) + (x2*y3 - y2*x3);

    float invA[3][3] = {(y2 - y3)/detA, -(y1 - y3)/detA,
                        (y1 - y2)/detA, -(x2 - x3)/detA,
                        (x1 - x3)/detA, -(x1 - x2)/detA,
                        {(x2*y3 - y2*x3)/detA,
                        -(x1*y3 - y1*x3)/detA,
                        (x1*y2 - y1*x2)/detA}};

    float xRes = (invA[0][0]*(-(pow(x1,2)+pow(y1,2)))
                  + invA[0][1]*(-(pow(x2,2)+pow(y2,2)))
                  + invA[0][2]*(-(pow(x3,2)+pow(y3,2))))/(-2);
    float yRes = (invA[1][0]*(-(pow(x1,2)+pow(y1,2)))
                  + invA[1][1]*(-(pow(x2,2)+pow(y2,2)))
                  + invA[1][2]*(-(pow(x3,2)+pow(y3,2))))/(-2);

// Output Center Coordinate to PC
    decTX(rnd(xRes));
    decTX(rnd(yRes));
}

//=====
// Function that Selects a Point on the Actual Moon Border
// from Longest Subtending Line

int SideSelect(float PrimIn1[2], float PrimIn2[2],
               float SecIn1[2], float SecIn2[2])
{
    double Prim1_Prim2 = (sqrt(pow((PrimIn1[0] - PrimIn2[0]),2)
                               + pow((PrimIn1[1] - PrimIn2[1]),2)));
    double Prim1_Sec1 = (sqrt(pow((PrimIn1[0] - SecIn1[0]),2)
                               + pow((PrimIn1[1] - SecIn1[1]),2)));
    double Prim1_Sec2 = (sqrt(pow((PrimIn1[0] - SecIn2[0]),2)
                               + pow((PrimIn1[1] - SecIn2[1]),2)));
    double Prim2_Sec1 = (sqrt(pow((PrimIn2[0] - SecIn1[0]),2)
                               + pow((PrimIn2[1] - SecIn1[1]),2)));
    double Prim2_Sec2 = (sqrt(pow((PrimIn2[0] - SecIn2[0]),2)
                               + pow((PrimIn2[1] - SecIn2[1]),2)));
}

```

```

        + pow((PrimIn2[1] - SecIn2[1]),2));

if ((Prim1_Sec1 < 2) && (Prim1_Sec2 > Prim1_Sec1))
    return(2);
else if ((Prim1_Sec2 < 2) && (Prim1_Sec1 > Prim1_Sec2))
    return(1);
else
{
    double Sec1Dist = Prim1_Sec1*sin(acos((pow(Prim1_Prim2,2)
        + pow(Prim1_Sec1,2) - pow(Prim2_Sec1,2))
        /(2*Prim1_Prim2*Prim1_Sec1)));
    double Sec2Dist = Prim1_Sec2*sin(acos((pow(Prim1_Prim2,2)
        + pow(Prim1_Sec2,2) - pow(Prim2_Sec2,2))
        /(2*Prim1_Prim2*Prim1_Sec2)));

    if (Sec1Dist >= Sec2Dist)
        return(1);
    else
        return(2);
}
}

//=====
// Function which Updates the Values of the Points at Extremities

void ModEdgeVals()
{
    xyNewLength = rnd(sqrt(2*pow((x + ScaleFac - y),2)));
    xyMinLength = rnd(sqrt(2*pow((xyMin1[0] + ScaleFac
        - xyMin1[1]),2)));
    xyMaxLength = rnd(sqrt(2*pow((xyMax1[0] + ScaleFac
        - xyMax1[1]),2)));

    yxNewLength = rnd(sqrt(2*pow((x+y),2)));
    yxMinLength = rnd(sqrt(2*pow((yxMin1[0] + yxMin1[1]),2)));
    yxMaxLength = rnd(sqrt(2*pow((yxMax1[0] + yxMax1[1]),2)));

    if (xMin1[0] > x)
    {
        xMin1[0] = x;
        xMin1[1] = y;
    }
}

```

```

    }

//-----

    if (xyMinLength > xyNewLength)           // Bottom Left
    {
        xyMin1[0] = x;
        xyMin1[1] = y;
    }

    if (xyMinLength >= xyNewLength)          // Bottom Left
    {
        xyMin2[0] = x;
        xyMin2[1] = y;
    }

    if (xyMaxLength < xyNewLength)           // Top Right
    {
        xyMax1[0] = x;
        xyMax1[1] = y;
    }

    if (xyMaxLength <= xyNewLength)          // Top Right
    {
        xyMax2[0] = x;
        xyMax2[1] = y;
    }

    if (yxMinLength > yxNewLength)           // Top Left
    {
        yxMin1[0] = x;
        yxMin1[1] = y;
    }

    if (yxMinLength >= yxNewLength)          // Bottom Right
    {
        yxMin2[0] = x;
        yxMin2[1] = y;
    }

    if (yxMaxLength < yxNewLength)           // Top Left
    {

```

```

        yxMax1[0] = x;
        yxMax1[1] = y;
    }
    if (yxMaxLength <= yxNewLength)           // Bottom Right
    {
        yxMax2[0] = x;
        yxMax2[1] = y;
    }
}

//=====
// Function which Acquires Values on the Actual Moon Edge

void calcCenter()
{
    int xDist, yDist, xyDist, yxDist;
    int Side = 0;

    xMin[0] = xMin1[0];
    xMin[1] = rnd((xMin2[1] + xMin1[1])/2);

    xMax[0] = xMax1[0];
    xMax[1] = rnd((xMax2[1] + xMax1[1])/2);

    yMin[0] = rnd((yMin2[0] + yMin1[0])/2);
    yMin[1] = yMin1[1];

    yMax[0] = rnd((yMax2[0] + yMax1[0])/2);
    yMax[1] = yMax1[1];

    xyMin[0] = rnd((xyMin2[0] + xyMin1[0])/2);
    xyMin[1] = rnd((xyMin2[0] + xyMin1[0])/2) - (xyMin2[0]
        - xyMin2[1]);

    xyMax[0] = rnd((xyMax2[0] + xyMax1[0])/2);
    xyMax[1] = rnd((xyMax2[0] + xyMax1[0])/2) - (xyMax2[0]
        - xyMax2[1]);

    yxMin[0] = rnd((yxMin2[0] + yxMin1[0])/2);
    yxMin[1] = (yxMin2[0] + yxMin2[1]) - rnd((yxMin2[0]

```

```

        + yxMin1[0])/2);

yxMax[0] = rnd((yxMax2[0] + yxMax1[0])/2);
yxMax[1] = (yxMax2[0] + yxMax2[1]) - rnd((yxMax2[0]
        + yxMax1[0])/2);

xDist = rnd(sqrt(pow((xMin[0] - xMax[0]),2) + pow((xMin[1]
        - xMax[1]),2)));
yDist = rnd(sqrt(pow((yMin[0] - yMax[0]),2) + pow((yMin[1]
        - yMax[1]),2)));

xyDist = rnd(sqrt(pow((xyMin[0] - xyMax[0]),2) + pow((xyMin[1]
        - xyMax[1]),2)));
yxDist = rnd(sqrt(pow((yxMin[0] - yxMax[0]),2) + pow((yxMin[1]
        - yxMax[1]),2)));

if ((xDist >= yDist) && (xDist >= xyDist) && (xDist >= yxDist))
{
    Side = SideSelect(xMin, xMax, yMin, yMax);
    if (Side == 1)
    {
        x1 = yMin[0];
        x2 = xyMax2[0];
        x3 = yxMin1[0];
        y1 = yMin[1];
        y2 = xyMax2[1];
        y3 = yxMin1[1];
    }
    else
    {
        x1 = yMax[0];
        x2 = xyMin1[0];
        x3 = yxMax2[0];
        y1 = yMax[1];
        y2 = xyMin1[1];
        y3 = yxMax2[1];
    }
}

else if ((yDist > xDist) && (yDist >= xyDist)
        && (yDist >= yxDist))

```

```
{
    Side = SideSelect(yMin, yMax, xMin, xMax);
    if (Side == 1)
    {
        x1 = xMin[0];
        x2 = xyMin2[0];
        x3 = yxMin2[0];
        y1 = xMin[1];
        y2 = xyMin2[1];
        y3 = yxMin2[1];
    }
    else
    {
        x1 = xMax[0];
        x2 = xyMax1[0];
        x3 = yxMax1[0];
        y1 = xMax[1];
        y2 = xyMax1[1];
        y3 = yxMax1[1];
    }
}

else if ((xyDist > xDist) && (xyDist > yDist)
        && (xyDist >= yxDist))
{
    Side = SideSelect(xyMin, xyMax, yxMin, yxMax);
    if (Side == 1)
    {
        x1 = yxMin[0];
        x2 = xMin2[0];
        x3 = yMin2[0];
        y1 = yxMin[1];
        y2 = xMin2[1];
        y3 = yMin2[1];
    }
    else
    {
        x1 = yxMax[0];
        x2 = xMax1[0];
        x3 = yMax1[0];
        y1 = yxMax[1];
```

```

        y2 = xMax1[1];
        y3 = yMax1[1];
    }
}
else if ((yxDist > xDist) && (yxDist > yDist)
        && (yxDist > xyDist))
{
    Side = SideSelect(yxMin, yxMax, xyMin, xyMax);
    if (Side == 1)
    {
        x1 = xyMin[0];
        x2 = yMax2[0];
        x3 = xMin2[0];
        y1 = xyMin[1];
        y2 = yMax2[1];
        y3 = xMin1[1];
    }
    else
    {
        x1 = xyMax[0];
        x2 = yMin1[0];
        x3 = xMax2[0];
        y1 = xyMax[1];
        y2 = yMin1[1];
        y3 = xMax2[1];
    }
}
}

//=====
// Function which Steps Through the Image and Acquires the
// Values of The Extremities

void getBorder()
{
    float a1, a2;
    float b1, b2;
    int a, b;

    if (xBound > yBound)

```



```

        ScaleFac = xBound;
else
        ScaleFac = yBound;

x = 0;
y = 0;

xMin1[0] = 0;
xMin1[1] = 0;
xMax1[0] = 0;
xMax1[1] = 0;
yMin1[0] = 0;
yMin1[1] = 0;
yMax1[0] = 0;
yMax1[1] = 0;
xyMin1[0] = 0;
xyMin1[1] = 0;
xyMax1[0] = 0;
xyMax1[1] = 0;
yxMin1[0] = 0;
yxMin1[1] = 0;
yxMax1[0] = 0;
yxMax1[1] = 0;

for (x = 3; x < (xBound-4); x = x+2)
{
    b = (x-1)/2;
    for (y = 10; y < ((720-12)); y++)
    {
        a = b*yBound + y*2 - 1;
        if ((* (pSDRAM_BASE+a) > Level)
            && ((* (pSDRAM_BASE+(a-2)) <= Level)
                || (* (pSDRAM_BASE+(a+2)) <= Level)
                || (* (pSDRAM_BASE+(a+PosF1_F2Offset)) <= Level)
                || (* (pSDRAM_BASE+(a+PosF1_F2Offset-yBound))
                    <= Level))
            && ((* (pSDRAM_BASE+(a-2)) >= Level)
                || (* (pSDRAM_BASE+(a+2)) >= Level)
                || (* (pSDRAM_BASE+(a+PosF1_F2Offset)) >= Level)

```

```

|| (* (pSDRAM_BASE+(a+PosF1_F2Offset-yBound)
    >= Level)))
{
    if ((xMin1[1] == 0) && (xMax1[1] == 0)
        && (yMin1[1] == 0) && (yMax1[1] == 0))
    {
        xMin1[0] = x;
        xMin1[1] = y;
        xMax1[0] = x;
        xMax1[1] = y;
        yMin1[0] = x;
        yMin1[1] = y;
        yMax1[0] = x;
        yMax1[1] = y;
        xyMin1[0] = x;
        xyMin1[1] = y;
        xyMax1[0] = x;
        xyMax1[1] = y;
        yxMin1[0] = x;
        yxMin1[1] = y;
        yxMax1[0] = x;
        yxMax1[1] = y;
    }

    ModEdgeVals();
}

}

xMinNew = xMin1[0];
xMaxNew = xMax1[0];
yMinNew = yMin1[1];
yMaxNew = yMax1[1];

for (x = (xMinNew-1); x <= (xMaxNew+1); x = x+2)
{
    b = (x-2)/2;
    for (y = (yMinNew-1); y <= (yMaxNew+1); y++)
    {
        a = PosF1_F2Offset + b*yBound + y*2 - 1;
        if ((* (pSDRAM_BASE + a) > Level)

```

```

&& ((* (pSDRAM_BASE + a-2) <= Level)
|| (* (pSDRAM_BASE + a+2) <= Level)
|| (* (pSDRAM_BASE + a-PosF1_F2Offset+yBound)
<= Level)
|| (* (pSDRAM_BASE + a-PosF1_F2Offset) <= Level))
&& ((* (pSDRAM_BASE + a-2) >= Level)
|| (* (pSDRAM_BASE + a+2) >= Level)
|| (* (pSDRAM_BASE + a-PosF1_F2Offset+yBound)
>= Level)
|| (* (pSDRAM_BASE + a-PosF1_F2Offset) >= Level)))
{
    ModEdgeVals();
}
}
}

//=====
// Function which sets up all the Clock Values

void setupCLK(void)
{
    // setup CLKs
    *pPLL_CTL = 0x2800;
    *pPLL_DIV = 0x0003;
    *pUART_GCTL = 0x0001;
    *pUART_LCR = 0x0083;
    *pUART_DLL = 0x0031;
    *pUART_DLH = 0x0000;
    *pUART_LCR = 0x0000;
    *pUART_LCR = 0x0003;
}

//=====
// Function which sets up the Flash Memory

void setupFLASH(void)
{
    // Setup Flash Memory
    *pEBIU_AMBCTL0 = 0x7BB07BB0;
    *pEBIU_AMBCTL1 = 0x7BB07BB0;
}

```

```

AsyncMemGlobContReg = *pEBIU_AMGCTL;
*pEBIU_AMGCTL = AsyncMemGlobContReg | 0xF;

//Clear Flash A PortA Data Register
*pFlashA_PortA_DATA_OUT = 0x00;

//Flash A PortA Direction to Output
*pFlashA_PortA_DIR = 0xFF;

// Flash A PortA Data Register
// Reset Video Decoder: off
// Set PPI clk as Video decoder pixel clock
*pFlashA_PortA_DATA_OUT = 0x18;
}

//=====
// Function which Initializes SDRAM

void setupSDRAM(void)
{
    // Setup of SDRAM
    *pEBIU_SDSTAT = *pEBIU_SDSTAT & 0x0010;
    if (*pEBIU_SDSTAT == 0x0009)
    {
        *pEBIU_SDRRC = 0x02B6;
        *pEBIU_SDBCTL = 0x0013;
        *pEBIU_SDGCTL = 0x0091998d;
    }

    for (k = 0; k < 288; k++)
    {
        for (l = 0; l < 720; l++)
        {
            *(pSDRAM_BASE+l+(k*720)) = (l*255/719);
        }
    }
}

//=====
// Function which sets up the DMA

```

```

void setupDMA(void)
{
    // Setup DMA Registers
    *pDMA0_START_ADDR = SDRAM_START_ADDR; //0xFFC00400;
    *pDMA0_CONFIG = 0x00B2;
    *pDMA0_X_COUNT = 0x05A0; // 5A0 = 1440 decimal
    *pDMA0_X_MODIFY = 0x0001;// 1 decimal
    *pDMA0_Y_COUNT = 0x0240;// 576 decimal
    *pDMA0_Y_MODIFY = 0x0001;// 1 decimal
    *pDMA0_PERIPHERAL_MAP = 0x0000;
}

//=====
// Function which sets up the PPI

void setupPPI(void)
{
    // Setup of PPI
    // Set Programmable flags 0-2 as outputs
    // PF0 - serial clk for programming Video Decoder
    // PF1 - serial data for programming Video Decoder
    // PF2 - Video Decoder ~OE
    *pFIO_DIR = 0x0007;

    // Drive PF2 '0' so as to Enable Video Decoder output
    *pFIO_FLAG_C = 0x004;

    // Setup for RX Mode, ITU-R 656, Active Field Only
    *pPPI_FRAME = 0x020D; //0x0271 PAL //0x020D NTSC

    *pPPI_CONTROL = 0x0040; // 0040 no skipping
}

//=====
// Function which takes an Image and Downloads it to the SDRAM

void getImage(void)
{
    setupDMA();
    setupPPI();
}

```

```

// Initialize DMA
*pDMA0_CONFIG = *pDMA0_CONFIG | 0x0001;

// Initialize PPI
*pPPI_CONTROL = *pPPI_CONTROL | 0x0001;

//wait for DMA transfer to complete
while ((*pDMA0_IRQ_STATUS & 0x01) != 0x01);
}

//=====
// Function which Transmits the Image to the PC

void imageTX(void)
{
    LSR = 0x00;
    for (k = 0; k < 414720; k++)
    {
        charTX(*(pSDRAM_BASE+(k*2)+1));
    }
}

//=====
// Main Procedure from which all Commands are Handled

void main(void)
{
    bool halt = 0;
    setupCLK();
    setupFLASH();
    setupSDRAM();

    LSR = 0x0000;
    while (halt == 0)
    {
        //Wait for Data Available in Receive buffer
        while (LSR != 0x0001)
        {
            LSR = *pUART_LSR & 0x0001;
        }
        LSR = 0x0000;
    }
}

```

```

Rx = *pUART_RBR;

switch (Rx)
{
    case 0x01 : getImage(); break;
    case 0x02 : imageTX(); break;
    case 0x03 : while (LSR != 0x0001)
        {
            getImage();
            getBorder();
            calcCenter();
            getCenter();
            LSR = *pUART_LSR & 0x0001;
        }
        break;
    case 0x04 : halt = 1; break;
    case 0x05 : decTX(rnd(5.0/2));
                decTX(rnd((3.0+4.0)/2));
                decTX(rnd(7.0/2));break;
    default : halt = 0; break;
}
}
}

```

Appendix C

Matlab Implementation of Moon and Sun Position Algorithms

The moon and sun position algorithms, as discussed in Sections 2.2 and 3.3.2, can be implemented in Matlab as shown in **MoonSunPos.m**.

MatlabMoonSensor.m:

```
%===== %  
% Matlab Implementation of Moon and Sun Position Algorithms %  
%===== %  
  
clear all;  
  
Month = 3;  
Year   = 2005;  
Rvecsun = [0  
           0  
           0];  
Rvecmoon = [0  
            0  
            0];  
Rvecsat = [0  
           0  
           0];  
  
AngleSM = 0;
```



```

if ((Month == 1) || (Month == 2))
    Year = Year - 1;
    Month = Month + 12;
end;

A = Year/100;
B = A/4;
X = 2 - A + B;
Y = 365.25*(Year + 4716);
F = 30.6001*(Month + 1);

for Day = 1:31

    JD = X + Day + Y + F - 1524.5;

    T = (JD - 2451545)/36525;

    e = 0.016708617 - 0.000042037*T - 0.0000001236*T^2;

    Msun = 357.5291 + 35999.0503*T - 0.0001559*T^2
          - 0.00000048*T^3;

    Lsun = 280.46645 + 36000.76983*T + 0.0003032*T^2;

    Epsilon = 23.4392911 - 0.0130041*T - (1.64e-7)*T^2
              + (5.04e-7)*T^3;

    C = (1.9146 - 0.004817*T - 0.000014*T^2)*sin(pi*Msun/180)
        + (0.019993 - 0.000101*T)*sin(2*pi*Msun/180)
        + 0.00029*sin(3*pi*Msun/180);

    LambdaSun = Lsun + C;

    v = Msun + C;

    Rsun = 1.000001018*(1 - e^2)/(1 + e*cos(pi*v/180));

    %Vector to Sun in km for month of March 2005
    Rvecsun = [Rvecsun Rsun*149597870.691*[cos(pi*LambdaSun/180)
        cos(pi*Epsilon/180)*sin(pi*LambdaSun/180)
        sin(pi*Epsilon/180)*sin(pi*LambdaSun/180)]];

```

```

k = 360;

LambdaMoon = 218.3165 + 481267.8813*T;

Mmoon = 134.9729814 + (1325*k + 198.867398)*T + 0.0086972*T^2
        + (1.778e-5)*T^3;

uMmoon = 93.2719103 + (1342*k + 82.0175381)*T - 0.0036825*T^2
        + (3.06e-6)*T^3;

Dsun = 297.8503631 + (1236*k + 307.11148)*T - 0.00191417*T^2
        + (5.28e-6)*T^3;

Parallax = 0.9508 + 0.0518*cos(pi*Mmoon/180)
           + 0.0095*cos(pi*(Mmoon - 2*Dsun)/180)
           + 0.0078*cos(pi*2*Dsun/180)
           + 0.0028*cos(pi*2*Mmoon/180);

Rmoon = 1/(sin(pi*Parallax/180));

PhiEcliptic = 5.13*sin(pi*uMmoon/180) + 0.28*sin(pi*(Mmoon
           + uMmoon)/180) - 0.28*sin(pi*(uMmoon - Mmoon)/180)
           - 0.17*sin(pi*(uMmoon - 2*Dsun)/180);

LambdaEcliptic = LambdaMoon + 6.29*sin(pi*Mmoon/180)
                - 1.27*sin(pi*(Mmoon - 2*Dsun)/180)
                + 0.66*sin(pi*2*Dsun/180)
                + 0.21*sin(pi*2*Mmoon/180)
                - 0.19*sin(pi*Msun/180)
                - 0.11*sin(pi*2*uMmoon/180);

%Vector to Moon in km for month of March 2005
Rvecmoon = [Rvecmoon Rmoon*6371
            * [cos(pi*PhiEcliptic/180)*cos(pi*LambdaEcliptic/180)
              cos(pi*Epsilon/180)*cos(pi*PhiEcliptic/180)*sin(pi*LambdaEcliptic/180)
              - sin(pi*Epsilon/180)*sin(pi*PhiEcliptic/180)
              sin(pi*Epsilon/180)*cos(pi*PhiEcliptic/180)*sin(pi*LambdaEcliptic/180)
              + cos(pi*Epsilon/180)*sin(pi*PhiEcliptic/180)]];

```

```

IndexD = round(Day);

MoonD = sqrt(Rvecmoon(1, IndexD)^2
             +Rvecmoon(2, IndexD)^2+Rvecmoon(3, IndexD)^2);

SunD = sqrt(Rvecsun(1, IndexD)^2
            +Rvecsun(2, IndexD)^2+Rvecsun(3, IndexD)^2);

DotSM = (Rvecmoon(1, IndexD)*Rvecsun(1, IndexD)
         +Rvecmoon(2, IndexD)*Rvecsun(2, IndexD)
         +Rvecmoon(3, IndexD)*Rvecsun(3, IndexD));

AngleSM = [AngleSM 180*(acos(DotSM/(SunD*MoonD)))/pi];
end;

%=====
% Display all Results of Sun and Moon Vector Calculations

plot(AngleSM);
grid on;
xlabel('Days');
ylabel('Angle between Moon and Sun (Degrees)');
figure;
figure;

subplot(2,1,1);
plot3(Rvecmoon(1,:),Rvecmoon(2,:),Rvecmoon(3,:));grid on;
subplot(2,1,2);
plot3(Rvecsun(1,:),Rvecsun(2,:),Rvecsun(3,:));grid on;
xlabel('x')
ylabel('y')
zlabel('z')

distMoon = (sqrt(Rvecmoon(1,:).^2 + Rvecmoon(2,:).^2 + Rvecmoon(3,:).^2));
meanDistMoon = mean(distMoon)
distSun = (sqrt(Rvecsun(1,:).^2 + Rvecsun(2,:).^2 + Rvecsun(3,:).^2));
meanDistSun= mean(distSun)

meanSun = meanDistSun/6378;
meanMoon = meanDistMoon/6378;

```

Appendix D

C++ Implementation of PC Communication with DSP

The code implemented on the PC to communicate with the sensor, as mentioned in Section 4.3.4, can be implemented in C++ as shown in **PCComms.c**.

PCComms.c:

```
//-----//
// The Code which runs on the PC to coordinate all actions by the DSP //
//-----//
#include <vcl.h>
#include <string.h>
#include <dstring.h>
#include <stdio.h>
#include <math.h>
#pragma hdrstop
#include "PCComms.h"

//-----//
#pragma package(smart_init)
#pragma link "ZComm"
#pragma resource "*.dfm"
int i = 0;
String RXText = "";
FILE *fw, *fr;
```

```

TForm1 *Form1;
int FileCursPos = 0;
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//-----

void __fastcall TForm1::FormCreate(TObject *Sender)
{
    Edit1->Text = "";
    Edit2->Text = "";
    Edit3->Text = "Disconnected";
    Edit4->Text = "No File Open";
    Edit5->Text = "Enter Filename";
    Mem01->Text = "";
}

//-----
// Function to Open a File

String OpenFile(String Name, char FileOp)
{
    int k;
    char Filename[25] = "";
    for (k = 1; k <= (Name.Length()); k++)
    {
        Filename[k-1] = Name[k];
    }
    if (FileOp == 'w')
    {
        if ((fw = fopen(Filename, "w+")) == NULL)
        {
            return("Error");
        }
        else return("File Open");
    }
    else if (FileOp == 'r')
    {
        if ((fr = fopen(Filename, "r")) == NULL)

```

```

        {
            return("Error");
        }
        else return("File Open");
    }
}

//-----
// Function which Writes a String to the Open File

void WriteStr2File(String output)
{
    int k = 0;
    char c1 = 0;

    for (k = 1; k <= (output.Length()); k++)
    {
        c1 = output[k];
        fwrite(&c1, sizeof(c1), 1, fw); /* write struct s to file */
    }
}

//-----
// Function which Writes an Integer to the Open File

void WriteInt2File(String output)
{
    int k, l = 0;
    char c1 = 0;
    char c2 = 0;

    for (k = 1; k <= (output.Length()); k++)
    {
        c1 = output[k];
        String StrVal = String(int(c1));
        for (l = 1; l <= (StrVal.Length()); l++)
        {
            c2 = StrVal[l];
            fwrite(&c2, sizeof(c2), 1, fw); // write to file
        }
        c2 = ' ';
    }
}

```

```

        fwrite(&c2, sizeof(c2), 1, fw); // write to file
    }
}

//-----
// Function which Reads Data from the Open File

String ReadFromFile()
{
    int k, l = 0;
    String Input = "";
    char c = 0;
    int eof = 1;

    while (eof != 0)
    {
        Input = "";
        eof = fread(&c, 1, 1, fr); // read from file
        while ((c != ' ') & (eof != 0))
        {
            Input = Input + c;
            eof = fread(&c, 1, 1, fr); // read from file
        }
        WriteStr2File(Input);
        c = 0;
    }
    return Input;
}

//-----
// Function which Reads Data from the Com Port
// when it Becomes Available and Writes it to the Open File

void __fastcall TForm1::ZComm1DataAvailable(TObject *Sender)
{
    RXText = ZComm1->Text;
    WriteInt2File(RXText);
    RXText = "";
    ZComm1->Text = 0xFF;
}

```

```
//-----  
// Close the Open File  
  
void __fastcall TForm1::Button3Click(TObject *Sender)  
{  
    fclose(fw); /* close file */  
    Edit4->Text = ("File Closed");  
  
}  
  
//-----  
// Opens/Creates File withn Name in TextBox 5  
  
void __fastcall TForm1::Button2Click(TObject *Sender)  
{  
    Edit4->Text = OpenFile(Edit5->Text, 'w');  
}  
  
//-----  
// Set-up and Open the Comm Port  
  
void __fastcall TForm1::Button4Click(TObject *Sender)  
{  
    ZComm1->BaudRate = 115200;  
    ZComm1->MaxReadLength = 8192;  
    ZComm1->InputQueueSize = 8192;  
    ZComm1->Port = "Com1";  
    if (!ZComm1->OpenConnection())  
    {  
        Form1->Caption = "Serial Port 1 is NOT Open";  
        Edit2->Text = "Connection Failed";  
    }  
    else  
    {  
        Form1->Caption = "Connected to " + ZComm1->Port  
            + " : " + IntToStr(ZComm1->BaudRate)  
            + " baud";  
        Edit3->Text = "Connected";  
    }  
}
```



```
//-----  
// Close the Comm Port  
  
void __fastcall TForm1::Button5Click(TObject *Sender)  
{  
    int result = ZComm1->PurgeCommPort(PURGE_TXCLEAR);  
    result = ZComm1->PurgeCommPort(PURGE_RXCLEAR);  
  
    if (!ZComm1->CloseConnection())  
    {  
        Form1->Caption = "Serial Port 1 is STILL Open";  
        Edit2->Text = "Disconnection Failed";  
    }  
    else  
    {  
        Form1->Caption = "Disconnected from " + ZComm1->Port;  
        Edit3->Text = "Disconnected";  
    }  
}  
  
//-----  
// Issues the Command for an Image to be Taken by the DSP  
  
void __fastcall TForm1::Button6Click(TObject *Sender)  
{  
    ZComm1->Text = char(0x01);  
}  
  
//-----  
// Downloads the Image From the DSP  
  
void __fastcall TForm1::Button7Click(TObject *Sender)  
{  
    ZComm1->Text = char(0x02);  
}  
  
//-----  
// Send DSP into a Loop of Taking Images and Returning Moon Center Coordinate  
  
void __fastcall TForm1::Button8Click(TObject *Sender)
```

```
{
    ZComm1->Text = char(0x03);
}

//-----
// Halts the DSP Software

void __fastcall TForm1::Button9Click(TObject *Sender)
{
    ZComm1->Text = char(0x04);
}
//-----
```