

**Processing of Onboard Images to Assist  
Automatic Forward Motion Compensation for  
Micro-Satellites**

**Christiaan J. Mouton**

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

**Studyleader:**

**Prof. A. Schoonwinkel**

**Co-studyleader:**

**Prof. G.W. Milne**

April 2003

## 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 the degree.

Signature

Date

## **Abstract**

South Africa's first micro satellite, SUNSAT, was operational in orbit for 23 months since its NASA-sponsored launch on February 23, 1999. SUNSAT is a graduate student-developed satellite from the University of Stellenbosch in South Africa. Research work is in progress on improving SUNSAT's 15m multi-spectral imager to a 2.5m-resolution multi-sensor imager. This will require the use of Forward Motion Compensation for exposure control.

This thesis presents Automatic Forward Motion Compensation for Micro Satellites using a new earth sensor that measures the bore-sight projection's speed on the earth directly. This sensor will have no drift and will make use of a series of images and cross-correlation of them. The high-resolution imager's bore-sight motion can be controlled by this technique to ensure high quality stereo images.

A control system based on the DLR-TUBSAT was designed and is simulated in MATLAB. This technique of measuring the bore-sight projection's speed on the earth directly was tested on a 2-axis telescope and used to measure random movement of a satellite.

## Samevatting

SUNSAT, Suid Afrika se eerste satelliet was vir 23 maande in werking in 'n wentelbaan na sy NASA-geborgde lasering op 23 Februarie 1999. SUNSAT is 'n satelliet wat ontwikkel is deur nagraadse studente aan die Universiteit van Stellenbosch.

Navorsingswerk is aan die gang om SUNSAT se 15m resolusie multi-spektrale kamera te verbeter tot 'n 2.5m resolusie multi-sensor kamera. Die navorsing sal die gebruik van Voorwaartse Bewegingskompensasie benodig.

Die tesis handel oor automatiese voorwaartse bewegingskompensasie vir mikrosatelliete deur gebruik te maak van 'n nuwe aardgerigte sensor, wat die siglyn projeksie se snelheid op die aarde direk sal meet. Die sensor sal gebruik maak van 'n reeks foto's wat gekruiskorreleer word en sal geen tempo-wegdrywing hê nie. Hoë resolusie kameras se siglyn kan deur middel van hierdie tegniek beheer word om hoë kwaliteit stereo foto's te verseker.

'n Beheerstelsel wat gebaseer is op dié van DLR-TUBSAT, is ontwerp en is gesimuleer in MATLAB. Die tegniek om die siglyn projeksie se snelheid op die aarde direk te meet, is getoets op 'n 2-as teleskoop en is gebruik om onverwille keurige beweging van 'n satelliet te meet.

# Index

<b>List of Symbols.....</b>	<b>vii</b>
<b>List of Abbrevations.....</b>	<b>x</b>
<b>List of Figures .....</b>	<b>xi</b>
<b>1 Introduction .....</b>	<b>1</b>
<b>2 Orbit and Attitude Orientation Geometry .....</b>	<b>4</b>
2.1 Introduction.....	4
2.2 Coordinate System.....	4
2.2.1 Spacecraft-Fixed Coordinates.....	4
2.2.2 Orbit-Defined Coordinates.....	5
2.2.3 Inertial Coordinates.....	6
2.3 Geometrical Model Analyses.....	7
2.3.1 Geometrical Model .....	8
2.3.2 Bore-Sight Projection Model .....	10
2.4 Simulations .....	10
2.5 Conclusion .....	11
<b>3 Satellite Control System.....</b>	<b>13</b>
3.1 Introduction.....	13
3.2 Satellite Equations of Motion .....	13
3.2.1 Euler Dynamics.....	13
3.3 System Design .....	14
3.3.1 PD Control .....	14
3.3.2 DLR-TUBSAT Interface Unit .....	19
3.3.3 DLR-TUBSAT Control System.....	22
3.4 Conclusion .....	24
<b>4 Bore-sight Projection Model .....</b>	<b>25</b>
4.1 Introduction.....	25
4.2 Fourier Definitions.....	26
4.3 Model Analyses .....	27
4.3.1 FFT Processing .....	27
4.3.2 High Pass Filtering.....	28

4.3.3	Fourier-Mellin Transform.....	29
4.3.4	Effects of the Image Size .....	32
4.4	Simulations .....	33
4.5	Fourier Transform Limitations due to Flat Surfaces.....	37
4.6	Conclusion .....	41
<b>5</b>	<b>Simulation and Measurements.....</b>	<b>42</b>
5.1	Introduction.....	42
5.2	Implementation .....	42
5.3	Meade LX200 Measurements .....	47
5.4	Measurement Errors.....	51
5.5	DLR-TUBSAT Measurements .....	53
5.6	Is this technique worthwhile? .....	55
5.7	Conclusion .....	57
<b>6</b>	<b>Conclusion .....</b>	<b>58</b>
6.1	Summary .....	58
6.2	Recommendations.....	59
<b>7</b>	<b>Bibliography.....</b>	<b>61</b>
<b>Appendix A:</b>	<b>Equipment description.....</b>	<b>63</b>
	Euresys Picolo Frame Grabbing .....	64
	Meade LX200 Telescope Control.....	65
<b>Appendix B:</b>	<b>Keyboard Encoder source code.....</b>	<b>67</b>
<b>Appendix C:</b>	<b>Keyboard Encoder Schematics and PCB.....</b>	<b>75</b>
<b>Appendix D:</b>	<b>SINav Interface Source Code:.....</b>	<b>79</b>
<b>Appendix E:</b>	<b>Log-files of DLR-TUBSAT Overhead Pass .....</b>	<b>91</b>
	DLR-TUBSAT Control Software Log-file.....	92
	SINav Measuring Software Log-file.....	93
<b>Appendix F:</b>	<b>Radix-2 Fast Fourier Transform.....</b>	<b>97</b>

## List of Symbols

$\delta_S, L_S$	Latitude and longitude coordinates of satellite's position on earth's surface
$\delta_T, L_T$	Latitude and longitude coordinates of target's position on earth's surface
$\Delta L$	Difference between satellite and target longitude
$p_1, p_2$	Poles of a control system
$x_0, y_0, z_0$	Orbit defined coordinates
$A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$	Direction Cosine Matrix or Transformation matrix
$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix}$	Satellite Body Coordinates
$P_{xyz}^\alpha$	Position vector of object $\alpha$ in the $xyz$ -axis coordinates
$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$	Inertia Tensor
$\omega_B^I = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$	Inertial referenced body angular vector
$h = \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix}$	Reaction wheel angular momentum vector
$\theta(s)$	Euler output angle of a control system
$T(s)$	Reference input torque of a control system
$G_{CL}(s)$	Closed-loop control system model

$\theta_{ref}(s)$	Reference Euler input angle of a control system
$\dot{\theta}_i$	Angular rate at a specific time-step $i$
$f(x_1, x_2)$	Continuous function in 2 dimensions
$F(k_1, k_2)$	Fast Fourier Transform of a continuous function in 2 dimensions
$\mu_{ref}$	Reference mean pixel value
$\sigma_{ref}$	Reference standard deviation
$M_{i,j}$	Magnitude of Image Intensity value at grid point (i,j)
$\phi$	Azimuth angle in a spherical body coordinate system
$\lambda$	Elevation angle in a spherical body coordinate system; Angle formed at earth's center between a satellite and a target
$\Phi$	Rotation about Euler axis
$\zeta$	Damping factor of a second order control system
$\mu$	Mean Pixel value
$\sigma$	Standard deviation
$\alpha, \beta$	Scaling factors
$\phi, \theta, \psi$	Roll, Pitch and Yaw angle
$\omega_n$	Natural frequency of a control system
$\omega_0$	Orbit mean motion
$Az, \eta$	Azimuth and Elevation angle in a satellite orbit coordinate system
D	Distance between a satellite and a target on the earth's surface
$e$	Orbit eccentricity
H	Height of a satellite about the earth's surface
J	Moment of Inertia
$K_d$	Derivative Gain of a PD-control system
$K_p$	Proportional Gain of a PD-control system

$M_0$	Orbit mean anomaly at epoch
$\mathbf{N}_D$	External disturbance torque vector
$\mathbf{N}_{GG}$	Gravity gradient torque vector
$\mathbf{N}_M$	Magnetic torque vector
$R_E$	Earth's radius
$T$	End time of a sample period
$t_s$	Settling time of a step response of a control system

## List of Abbreviations

ADCS	Attitude Determination and Control System
CCD	Charge Coupled Device
CMOS	Charge Metal Oxide Semiconductor
DCM	Direction Cosine Matrix
FFT	Fast Fourier Transform
FMC	Forward Motion Compensation
FMT	Fourier-Mellin Transform
P-control	Proportional control
PD-control	Proportional-Derivative control
RST	Rotation, Scaling and Translation
SINav	Satellite Image Navigation
SUNSAT	Stellenbosch UNiversity SATellite

## List of Figures

Figure 1: Graphic Satellite System for Automatic FMC .....	1
Figure 2: Flow Diagram of Thesis .....	2
Figure 3: Spacecraft-Fixed Coordinate System .....	5
Figure 4: Orbit-Defined Coordinate System.....	5
Figure 5: Inertial Coordinates System (Celestial coordinates) .....	7
Figure 6: Relationship between Sub-satellite Point and Target.....	8
Figure 7: Relationship between Satellite, Target and Earth's Centre .....	8
Figure 8: Attitude Orientation Simulation Results for Various Target Longitudes.....	11
Figure 9: P-Control System .....	15
Figure 10: Poles of a P-Control Closed-Loop System.....	16
Figure 11: PD-Control System.....	17
Figure 12: Poles of a PD Closed-Loop System .....	17
Figure 13: Processing time of a 512 x 512 pixel image.....	18
Figure 14: Hardware description for a keyboard encoder .....	19
Figure 15: Keyboard-to-Host Communication .....	20
Figure 16: Host-to-Keyboard Communication .....	21
Figure 17: Keyboard-to-Host example .....	22
Figure 18: Host-to-Keyboard sample.....	22
Figure 19: Graphic Overview of Image Processing.....	26
Figure 20: 6-Point DFT using Radix-2 Algorithm .....	28
Figure 21: High Pass Filter for emphasising high frequencies .....	29
Figure 22: Computing of New Intensity Value.....	31
Figure 23: Fourier-Mellin Transform .....	32
Figure 24: Processing Time for Various Tasks.....	32
Figure 25: Image taken for Simulation .....	33
Figure 26: Cartesian and Log-Polar FFT of the Images in Figure 25.....	34
Figure 27: Translation Results of the Images in Figure 25.....	34
Figure 28: Rotation and Scaling Results of the Images in Figure 25 .....	34

Figure 29: Images of Saturn translated and rotated by 15 pixels right, 10 pixels down, - 15,5° .....	36
Figure 30: Translation Results of Figure 29 .....	36
Figure 31: Rotation and Scaling Results of Figure 29 .....	36
Figure 32: Images of damaged oil fields in Kuwait.....	37
Figure 33: Images of Santiago, Chile .....	37
Figure 34: Examples of Flat Surface Images .....	38
Figure 35: Examples of Image Normalisation with Various deviations.....	39
Figure 36: Normalisation Algorithm for Images .....	39
Figure 37: Example 1 of Image Normalisation.....	40
Figure 38: Example 2 of Image Normalisation.....	40
Figure 39: Flow Diagram of a Satellite Implementation .....	43
Figure 40: Flow Diagram of Meade LX200 Test Case Implementation .....	43
Figure 41: SINav Software Interface .....	44
Figure 42: Radix-2 FFT Code.....	45
Figure 43: High Pass Filter Code.....	45
Figure 44: Log-Polar Conversion Code.....	46
Figure 45: 0.008 deg/sec Rate Measurements .....	48
Figure 46: 0.133 deg/sec Rate Measurements .....	48
Figure 47: 2 deg/sec Rate Measurements .....	48
Figure 48: 4 deg/sec Rate Measurements .....	49
Figure 49: 4 deg/sec Rate Measurements of Experiment 3 .....	49
Figure 50: Zero Crossing Response.....	51
Figure 51: Environmental Distance Effect on Measurements .....	53
Figure 52: Enviromental Surface Effect on Measurements.....	53
Figure 53: DLR-TUBSAT Pitch Rate Measurement.....	54
Figure 54: DLR-TUBSAT Roll Rate Measurement .....	54
Figure 55: Euresys Picolo Frame Grabbing Card.....	64
Figure 56: Picolo Card Setup Code .....	65
Figure 57: PULNiX Camera .....	65
Figure 58: Meade LX200 Telescope.....	66

Figure 59: Schematic diagram .....	76
Figure 60: Component Placement Layer .....	77
Figure 61: Solder Top Layer.....	77
Figure 62: Solder Bottom Layer .....	78

# 1 Introduction

With the introduction of new high-resolution satellite imagery, *Petrie* [1] has highlighted the importance of providing value-added services or products from space imagery. Some of these services or products include cartography, environmental monitoring, exploratory surveys and mass media. Forward motion compensation needs to be provided to ensure these value-added services, with particular reference to the acquisition of still stereo video coverage for mass media, which is a feature of many new commercial high-resolution satellites.

*Gottzein* [2] points out that some challenges for future commercial satellites will be the minimisation of equipment, recurring costs and the provision of fairly autonomous control systems. The drivers behind the ways to address these challenges will be performance, quality and cost.

Developing a control system for a micro satellite, like SUNSAT, that semi-autonomously controls a high-resolution imager by using data from the images forms the basis of this study. Figure 1 gives a graphic illustration of the satellite system to be implemented in this thesis.

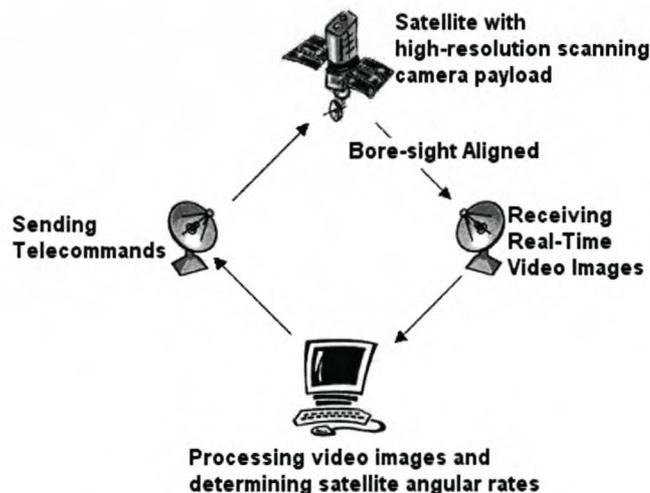


Figure 1: Graphic Satellite System for Automatic FMC

The need for forward motion compensation as it is mentioned earlier may rise from at least three independent requirements:

- The need for a line-scan CCD or CMOS imager to scan the earth at a rate lower than the orbit's ground track speed to increase the pixel integration time. This enables exposure control;
- The need for a video area imager to zero the *bore-sight pointing vector's speed* for a short time for a still or video images called dwelling on an area;
- The need to line-up the targets of two or more satellites. Satellites using this technique of flying are known as co-operative formation flight satellites.

To be able to control the bore-sight vector's track speed on the earth, either the satellite's inertial rates plus the spherical earth geometry model are needed, or the motion of actual bore-sight projection on the earth has to be measured. The latter control is used in this thesis and is illustrated in Figure 2.

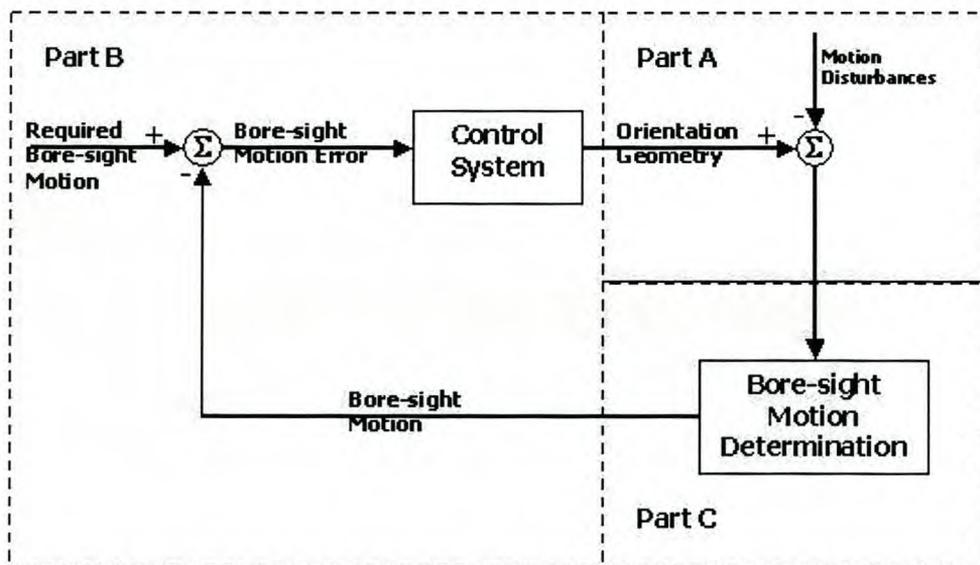


Figure 2: Flow Diagram of Thesis

Part A in Figure 2 is the environmental and motion disturbances that can affect the satellite commanded motion. It also consists of orientation geometry that forms part of this satellite and its environment. The orientation geometry is the connection between all the coordinate systems that ensures that forward motion compensation can be

implemented successfully. Part B is the satellite control system. This part covers the control algorithm to be used and the error-in-the-motion-determination. It also contains the hardware and software changes to be made to be able to execute this control system in the available ground station. Part C is the determination of the bore-sight projection motion from a series of successive images and includes other effects that can have an influence on this determination.

In short this thesis will cover the following topics:

- Chapter 2 is a review of coordinate systems used in satellite systems. In this chapter few models for performing an automatic forward motion control will be discussed of which some of them would be discussed in detail, with simulation results that show the behaviour of one of these models. (Part A of Figure 2)
- Chapter 3 reviews the control model of a spacecraft. A proportional-derivative (PD) control algorithm is designed for a satellite with the same hardware characteristics as those of the DLR-TUBSAT. An interface unit is designed to interface the DLR-TUBSAT control software by an external computer. (Part B of Figure 2)
- Chapter 4 defines and discusses translation, rotation and scaling identification using a series of successive images and their cross-correlation by means of Fourier transform properties. Effects that influence this identification will be discussed and possible solutions will be suggested. (Part C of Figure 2)
- Chapter 5 contains the tested results of the technique defined in Chapter 4. The software used to retain these measurements will be explained. A few experiments were performed to see how accurate the measurements are and if they were not accurate, to what extent they differ from the correct values. Errors and offsets will also be analysed with possible reasons for these effects discussed. Measurements were also made using the DLR-TUBSAT and will also discussed. This chapter will be concluded with the question if it is feasible to include this technique of measuring in a satellite control system.
- Chapter 6 concludes this thesis with a summary. Future research suggestions and recommendations will be given in this chapter.

## 2 Orbit and Attitude Orientation Geometry

### 2.1 Introduction

Forward motion control requires that the bore-sight motion speed needs to be known for various purposes. Some of the methods available to determine the speed of the bore-sight vector are to determining the satellite's inertial rates or measuring the bore-sight motion speed directly. Thus a clear understanding of the coordinate systems used in space technology is necessary.

This chapter will review the space coordinate systems for satellites. Two possible models for carry out FMC will be presented. A geometric model of the target, relative to the satellite body coordinates, will be defined using an orbit-defined coordinate system. The second model namely the bore-sight projection model will be discussed in detail in Chapter 4. This chapter will be concluded with simulations to confirm the geometric model.

### 2.2 Coordinate System

There are three major types of coordinate systems, namely the inertial, orbit-defined and the spacecraft-fixed coordinate systems, which define the attitude of a spacecraft.

#### 2.2.1 Spacecraft-Fixed Coordinates

The body coordinates of a spacecraft (Figure 3) are referred to as the spacecraft-fixed coordinate system. This system is used to define the orientation of ADCS hardware and is used to make attitude measurements. The rectangular body coordinates are composed by the  $x$ ,  $y$  and  $z$  components, while the spherical body coordinates are composed by azimuth ( $\phi$ ) and the elevation ( $\lambda$ ) coordinates.



Figure 3: Spacecraft-Fixed Coordinate System

### 2.2.2 Orbit-Defined Coordinates

The orbit-defined coordinate system maintains its orientation relative to its orbit and to the earth. The angles of the satellite body axes relative to this coordinate system are also referred to as the Euler angles. The  $z_o$ -axis (yaw-axis) is directed towards the centre of the earth (*nadir*), the  $x_o$ -axis (roll-axis) runs in the direction of the velocity vector and the  $y_o$ -axis (pitch-axis) completes the orthogonal set as seen in Figure 4, Wertz [3].

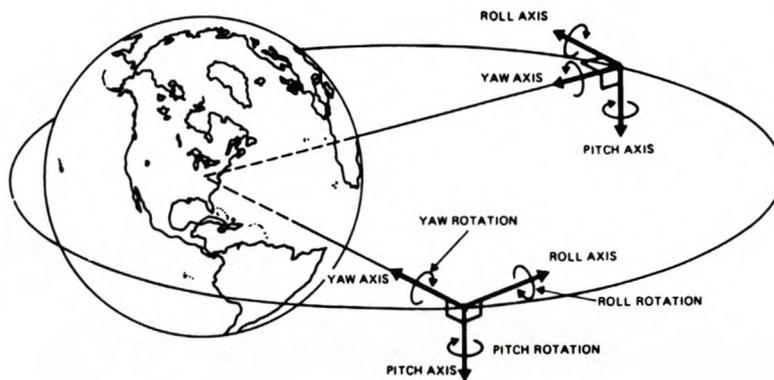


Figure 4: Orbit-Defined Coordinate System

The *direction cosine matrix* (DCM),  $A$  is used to transform the inertial coordinates between the body- and orbit- coordinate systems as seen in equation 2.1. The Euler angles, which are needed to perform a Euler transformation, can be expressed in a DCM as seen in equation 2.2.

$$\begin{bmatrix} x_B \\ y_B \\ z_B \end{bmatrix} = A \begin{bmatrix} x_O \\ y_O \\ z_O \end{bmatrix} \quad (2.1)$$

$$A = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi + S\psi C\phi & -C\psi S\theta C\phi + S\psi S\phi \\ -S\psi C\theta & -S\psi S\theta S + C\psi C\phi & S\psi S\theta C\phi + C\psi S\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix} \quad (2.2)$$

where,

$C$  = cosine function

$S$  = sine function

$\phi$  = roll angle

$\theta$  = pitch angle

$\psi$  = yaw angle

Equation 2.2 (an Euler 1-2-3 transformation) can be used to determine the angles through which the satellite needs to turn if the DCM is known. It is clear that this specific coordinate system plays an important role in the orientation geometry.

### 2.2.3 Inertial Coordinates

This coordinate system is used to define the motion of the spacecraft in inertial space.

The celestial coordinates (Figure 5) are commonly used by spacecrafts as the inertial coordinate system. The celestial  $z$ -axis is parallel to the rotation axis of the earth, with the positive  $z$ -axis in the direction of the geometric North Pole. The celestial  $x$ -axis is parallel to the line connecting the *vernal equinox* and the centre of the earth. The vernal equinox is the point where the plane of the earth's orbit around the sun crosses the equator moving from south to north. The celestial  $y$ -axis completes the orthogonal set.

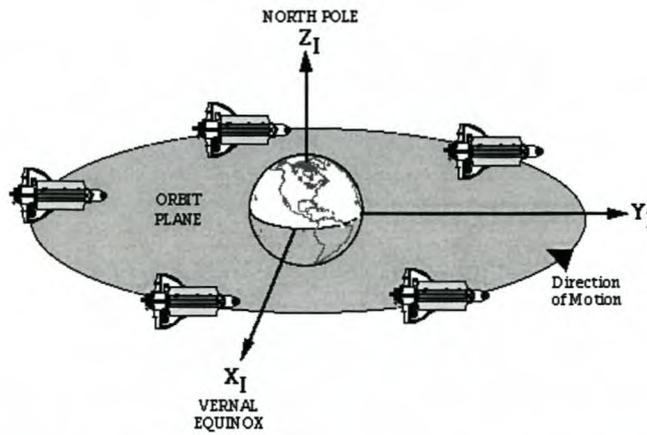


Figure 5: Inertial Coordinates System (Celestial coordinates)

### 2.3 Geometrical Model Analyses

Automatic forward motion compensation is defined as the focusing of the satellite's bore-sight (which normally contains the camera's field of view) on a specific area in space, whether it is a star constellation or a landscape on earth, while moving over this area. The satellite can focus on one specific point or scan an area while passing this point and/or area. This requirement can be conceptualised by means of a few possible models of which two are:

- **The Geometrical Model**, where the inertial positions and/or rates are available with a geometrical model which describes the positions and/or rates of the target relative to the satellite's reference frame. Matching the two models (inertial frame and geometrical model) will lead to the focusing of the satellite's bore-sight on the specific area as commanded;
- **The Bore-Sight Projection Model**, where the motion of the bore-sight is directly measured by means of image processing. This image processing consists of correlating two successive bore-sight images and using the peak in the correlated image to determine the bore-sight motion. These bore-sight images are such that there is a time difference between the two successive images and this time difference that can varies from image to image. As the current bore-sight motion and the required bore-sight motion is known, the bore-sight error motion can be determined and corrected for;

### 2.3.1 Geometrical Model

The geometrical model used in this study is defined by the relationship between a sub-satellite point and a target on the earth's surface as seen in Figure 6 and 7 from *Wertz* [3]. The coordinate system was chosen so that the z-axis is nadir (in the direction of the earth's centre), the x-axis is in the direction of the North Pole and the y-axis is orthogonal to the other axes. The angles  $Az$  and  $\eta$  through which the satellite has to turn so that its bore-sight vector is in the direction of the target can be determined by using *oblique spherical triangle* formulas.



Figure 6: Relationship between Sub-satellite Point and Target

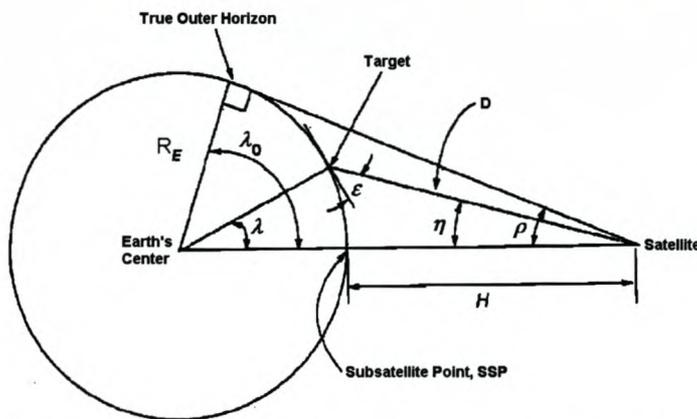


Figure 7: Relationship between Satellite, Target and Earth's Centre

Assume that the latitude ( $\delta_S, \delta_T$ ) and longitude ( $L_S, L_T$ ) of the satellite and target are known. The azimuth angle,  $Az$ , can then be determined as seen in equations 2.3 and 2.4.

$$\begin{aligned} \cos \lambda &= \sin \delta_S \sin \delta_T + \cos \delta_S \cos \delta_T \cos \Delta_L \\ \Delta_L &= |L_S - L_T| \end{aligned} \quad (2.3)$$

$$\cos Az = \frac{\sin \delta_T - \cos \lambda \sin \delta_S}{\sin \lambda \cos \delta_S} \quad (2.4)$$

The elevation angle  $\eta$  can be defined as:

$$\begin{aligned} \tan \eta &= \frac{\sin \rho \sin \lambda}{1 - \sin \rho \cos \lambda} \\ \sin \rho &= \frac{R_E}{R_E + H} \end{aligned} \quad (2.5)$$

The distance,  $D$ , can also be determined from the equations above, but the distance is not important for the control of the satellite.

$$D = \frac{R_E \sin \lambda}{\sin \eta} \quad (2.6)$$

Combining the above equations into a position unit vector, it would have the form of equation 2.7,

$$P_{xyz}^{Satellite} = \begin{bmatrix} r \sin \eta \cdot \cos Az \\ r \sin \eta \cdot \sin Az \\ r \cos \eta \end{bmatrix} \quad (2.7)$$

and also a rate unit vector as seen in equation 2.8.

$$\dot{P}_{xyz}^{Satellite} = \begin{bmatrix} \dot{r} \sin \eta \cdot \cos Az + r (\cos \eta \cdot \cos Az \cdot \dot{\eta} - \sin \eta \cdot \sin Az \cdot \dot{Az}) \\ \dot{r} \sin \eta \cdot \sin Az + r (\cos \eta \cdot \sin Az \cdot \dot{\eta} + \sin \eta \cdot \cos Az \cdot \dot{Az}) \\ \dot{r} \cos \eta - r \sin \eta \cdot \dot{\eta} \end{bmatrix} \quad (2.8)$$

Using these two reference unit vectors as a model, the start position and the inertial rates needed for the forward motion control process are known. This model forms the command reference input to the control system of the satellite. The simulation results of this model are shown in section 2.4.

### **2.3.2 Bore-Sight Projection Model**

The bore-sight projection model defines the movements of the satellite around its reference frame axis by means of using bore-sight images as a source for determining these movements. Most commercial satellites always have some sort of camera on board either as part of the payload or as part of the attitude determination system. Using these images from these cameras and processing two of these images, taken shortly after each other, information about the satellite's inertial behavior can be determined. This model will be discussed in detail in Chapter 4 about the image processing to be undertaken.

## **2.4 Simulations**

Simulations for the geometrical model were performed for various target longitudes. The satellite's orbit in the simulation was a polar orbit moving from South to North. The target was situated on the equator at various longitudes so that the differences between the satellite and target longitudes were  $0.1^\circ$ ,  $1^\circ$ ,  $2^\circ$  and  $5^\circ$ .

The geometrical simulation model can be seen in Figure 6 and 7. The result, which contains the position and angular rate values, of this simulation is given in Figure 8. As the distance between the satellite and target's latitude increases, the change rate in the azimuth angle decreases as the satellite moves over the equator. The elevation angular rate also decreases as the distance in between the satellite and target's latitudes increases. The more important results are those of the rate. A peak in the azimuth rate is expected when the satellite is passing directly over the target. This peak can be removed by multiplying the elevation with  $-1$  from time instance 100 seconds to 200 seconds. This will remove the peak in the azimuth rate and the sign change in the elevation rate. The

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

graph also shows the maximum rate through which the satellite, at a height of 700 km, is turning.

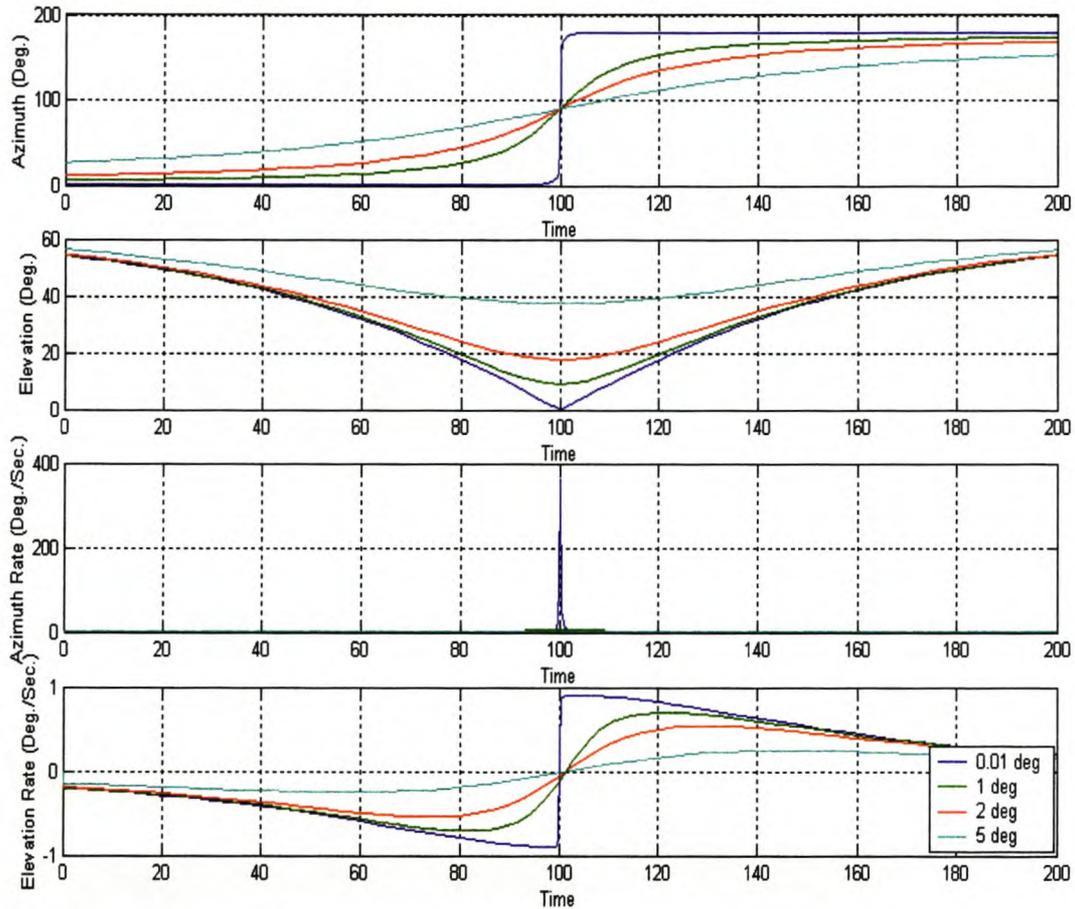


Figure 8: Attitude Orientation Simulation Results for Various Target Longitudes

## 2.5 Conclusion

In this chapter the coordinate systems were reviewed and two models were defined to perform automatic forward motion compensation, namely a geometrical model and a bore-sight projection model. The geometrical model would need the inertial rates to be ascertained while the bore-sight projection model would need a series of successive images to perform the automatic FMC. A simulation was performed using the

geometrical model while the bore-sight projection model will be discussed in detail later. The results were what were expected and they also ensured that the rates to be detected were also within reach of the bore-sight projection model.

## 3 Satellite Control System

### 3.1 Introduction

The control system is an important subsystem of a satellite that will ensure the stability of images. The system consists mainly of a few sensors (star tracker, magnetometer) to measure various parameters (attitude, magnetic field of earth, angular rates) and actuators (reaction wheels, propulsion system) to control these parameters.

In this chapter the satellite model in space will be reviewed and a controller will be designed for the specific model for 3-axis stabilisation. The model will represent the DLR-TUBSAT satellite, as this satellite will form part of the testing equipment.

### 3.2 Satellite Equations of Motion

Mathematical models are required to represent the motion of the spacecraft in space and the space environment itself. Euler's equations of motion govern the *dynamics* and *kinematics* of a spacecraft in inertial space. Only Euler dynamic equations are important for this study.

#### 3.2.1 Euler Dynamics

The influence of a gravity gradient boom reaction wheel angular momentum will affect Euler's dynamic equations such that they can be expressed as:

$$I\dot{\omega}_B^I = N_{GG} + N_M + N_D - \omega_B^I \times (I\omega_B^I + h) - \dot{h} \quad (3.1)$$

where  $I$  is the moment of inertia (MOI) tensor,

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix},$$

$\omega_b^I$  the inertial referenced body angular rate vector,

$$\omega_B^I = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix},$$

$h$  the reaction wheel angular momentum vector,

$$h = \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix},$$

$N_{GG}$  is the gravity gradient torque vector,  $N_M$  the magnetic torque vector and  $N_D$  the external disturbance torque vector.

### 3.3 System Design

The system design mainly consists of two parts. These parts are the design of the PD Control algorithm and the interface between the two computers where one contain the control software and the other the processing software and are needed to close the loop in real-time and semi-removing the man in the control loop. The man in the loop only handles the beginning phase of the control by initialises the satellite so that the camera on board the satellite focus on the area that is required. As soon as the target area is in sight, the algorithm explained in Chapter 4 takes over the control by keep focusing or dwelling on the target area. This is shortly what will be discussed in this section.

#### 3.3.1 PD Control

Most commercial imaging satellites' attitude control systems require 3-axis stabilisation. For the sake of simplicity, the control design will be done for a single axis and then implemented for all three axes.

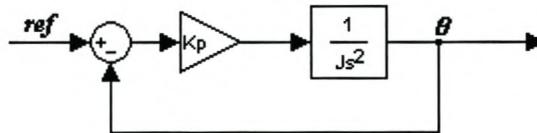
Combining all the control torques of equation 3.1 and ignoring the disturbance torque, the Laplace transform for a single axis would simplify to

$$\frac{\theta(s)}{T(s)} = \frac{1}{Js^2} = G_s(s) \quad (3.2)$$

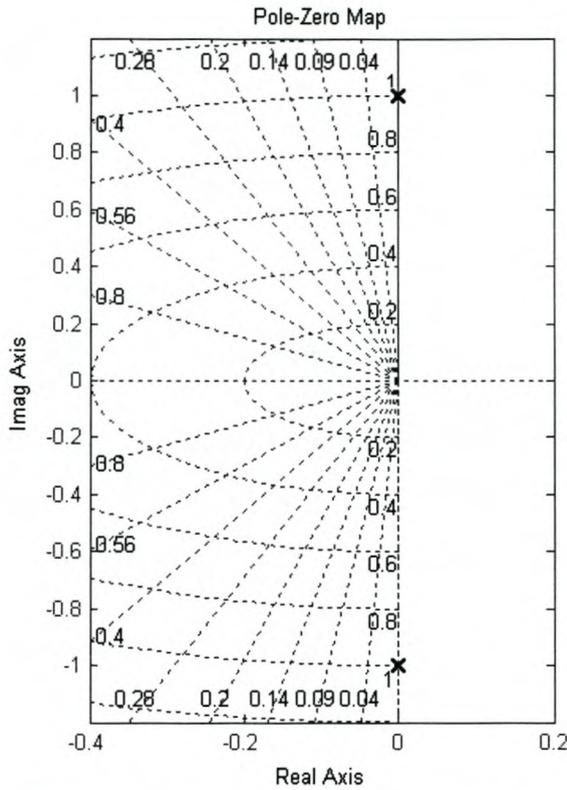
where  $\theta$  is the Euler angle,  $T$  the reference input torque to the satellite control system and  $J$  the moment of inertia of the satellite.

Closing the control system as seen in Figure 9 so that it forms a proportional gain control system (P-control system) will move the closed-loop poles up with the imaginary axes away from the centre of the graph as seen in Figure 10, which is confirmed by equation 3.3.  $K_P$  is a proportional gain in the backward path of the closed-loop system seen in Figure 9. This movement of the poles causes the satellite to exhibit a small oscillation on a command, which is not acceptable.

$$\frac{\theta(s)}{\theta_{ref}(s)} = \frac{K_P/J}{s^2 + K_P/J} = G_{CL}(s) \quad (3.3)$$



**Figure 9: P-Control System**



**Figure 10: Poles of a P-Control Closed-Loop System**

The closed-loop poles for the system defined in equation 3.3 are at

$$p_1, p_2 = \pm \sqrt{\frac{K_P}{J}}$$

Implementing a proportional-derivative control system (PD-control system) as seen in Figure 11 will bring a damping factor into the close-loop system. In this case the closed-loop poles had moved away from imaginary axis, as seen in Figure 12. The closed-loop system transfer function is defined as

$$\frac{\theta(s)}{\theta_{ref}(s)} = \frac{K_P/J}{s^2 + K_D/J s + K_P/J} = G_{CL}(s) \quad (3.4)$$

where  $K_D$  is a integral gain in the backward path of the closed-loop system.

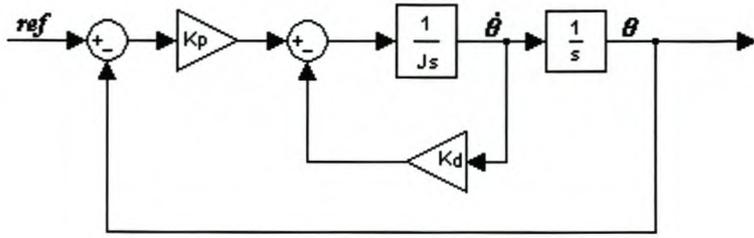


Figure 11: PD-Control System

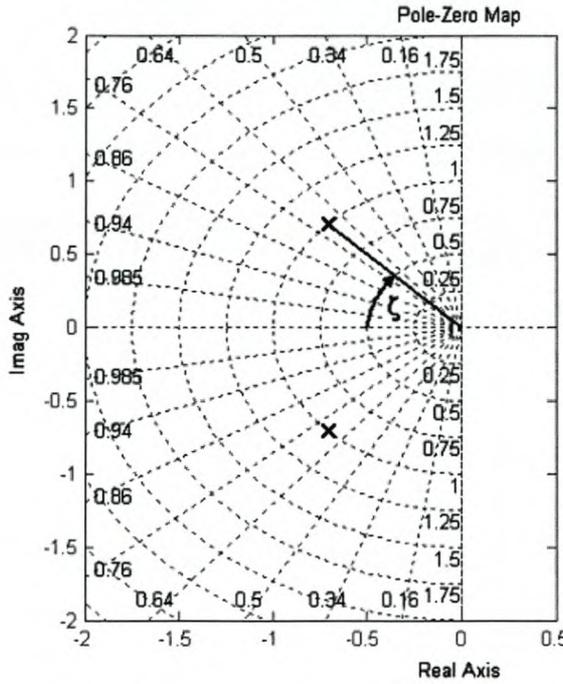


Figure 12: Poles of a PD Closed-Loop System

For the closed-loop system as defined in equation 3.4 the poles are at

$$p_1, p_2 = \frac{-1}{2J} \left( K_D \mp \sqrt{K_D^2 - 4JK_P} \right)$$

It can be seen that should the rate motion measurements not be detected, that the closed-loop control system could go into oscillation. In other words the angular rate measurement in Figure 11 would fall away and the remaining of the control system is then the same as that of Figure 9 if the motion is not detected and this can place the satellite in an oscillation mode. Thus the detection of the rate motion plays an important role in the success of automatic FMC.

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

Experiments were done on various image sizes to determine the average processing time of these images, using a Pentium 2.2GHz computer, so that the bandwidth of the control system could be determined. These experiments will be discussed in Chapter 5. Figure 13 shows the average processing time for various sizes of images in order to identify RST.

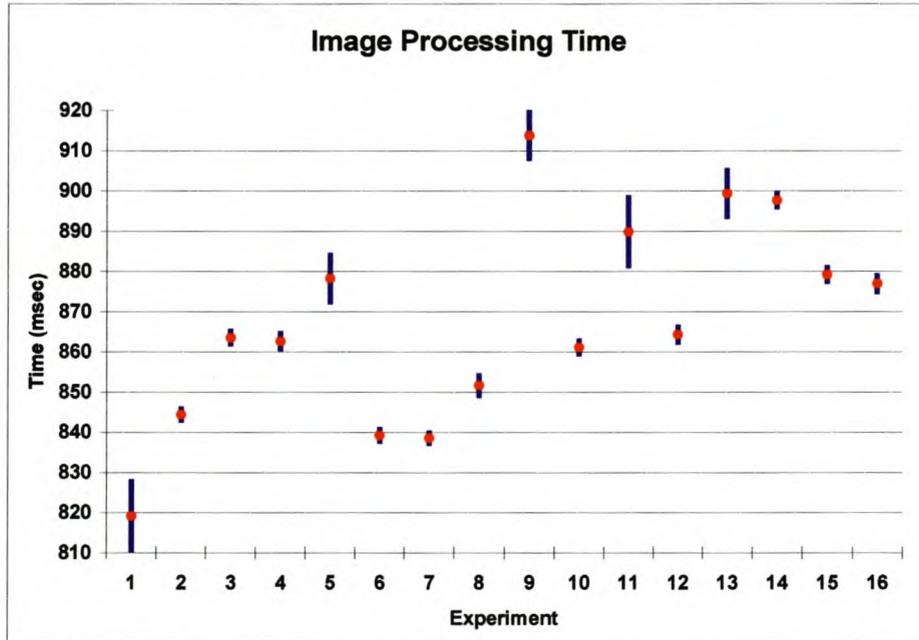


Figure 13: Processing time of a 512 x 512 pixel image

It can be seen clearly from the graph that the average processing time for an image, with a size of 512 x 512 pixels, is  $\pm 870$  milliseconds. The sample time of the images will be chosen a few milliseconds higher than the average processing time; in this case the time chosen was 1 second. The bandwidth of the control system will be 6-10 times lower than the sample period in order to prevent aliasing. In this case the bandwidth will be chosen as 0.1 Hz, which is 10 times lower than the sample bandwidth, that is 1 Hz. With a sample bandwidth of 0.1 Hz and a damping factor of 0.6, the closed-loop PD-control system would be

$$G_{CL}(s) = \frac{0.395}{s^2 + 0.754s + 0.395} \quad (3.5)$$

and with the chosen bandwidth the settling time would be  $\pm 12.2$  seconds. The reason for the chosen damping factor will be discussed along with the Meade LX200 measurements in Chapter 5. The settling time for the control system is about 0.2% of 100 minute orbital period.

### 3.3.2 DLR-TUBSAT Interface Unit

The control software of the DLR-TUBSAT is designed so that the satellite can be controlled via a keyboard and/or joystick that implements a man-in-the-loop control system. Removing the man-in-the-loop from the controlling-interface of the satellite can be done in two ways without changing the original source code. It can be done by:

- **Software** that takes the data that is dumped in the serial port buffer and dumps them in the keyboard buffer. The advantage is that it is easy to implement, but the disadvantage is that not all operating systems allow the user to have access to their keyboard buffer;
- **Hardware** that generates the necessary keyboard signals from the data received from the external computer and then switches between the keyboard and the external computer, as it is required, as seen in Figure 14. The switch of the modes can be done by an external interrupt switch or by sending a command via the serial port to switch the modes. The advantage is that it can be used for almost all operating systems and is also easy to implement.

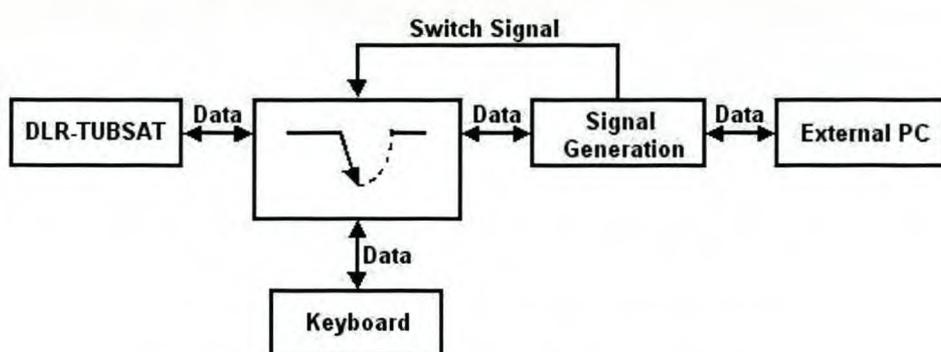


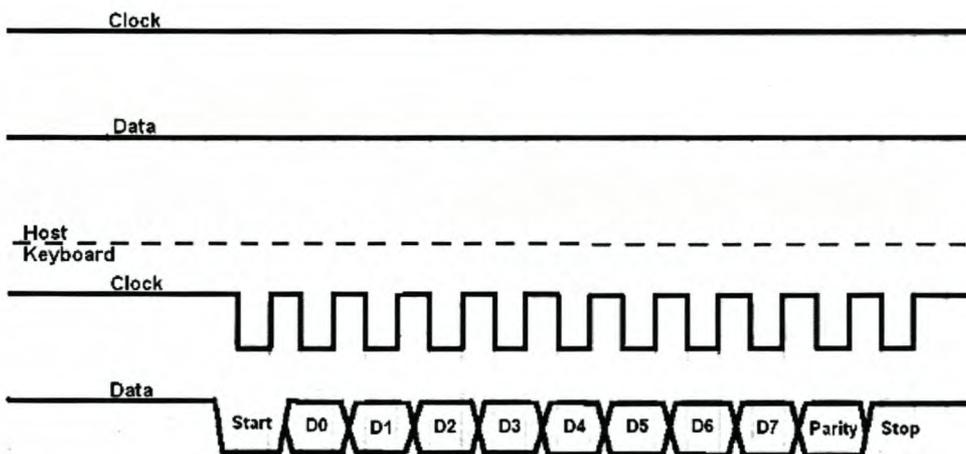
Figure 14: Hardware description for a keyboard encoder

The second option was chosen to semi-control the DLR-TUBSAT and thereby test the technique discussed in Chapter 4. The reasons why this interface is necessary are that the current computer that has the control software is not fast enough for the kind of

processing needed (as will be discussed in Chapter 4 and 5) and permission was not granted to have access to the source code of the control software. So an interface unit have to be designed that can change the signals from the external computer (on which the processing is done) and transform it into the equivalent keyboard signals.

The keyboard has two bi-directional control lines namely a clock and data line. The keyboard generates the clock signal, but is controlled by the host (the computer) as it has the highest priority. The data are then clocked through at a rate of  $\pm 15$  kHz as most devices' clocks vary between 10 kHz and 20 kHz. An AVR-processor is used to control this interface.

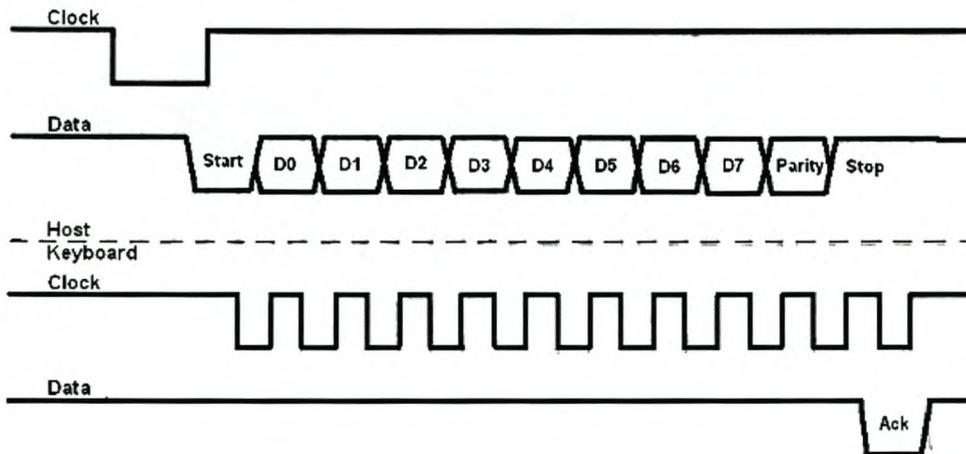
Data from the keyboard can be transmitted if both the clock and data signals are high (logic 1), which means that the host is ready to receive data from the keyboard. The package to be send consists of a start-bit that has to be logic 0, the eight bits data, one odd parity bit and a stop-bit that has to be logic 1. Figure 15 shows the signals generated by the host and the keyboard graphically.



**Figure 15: Keyboard-to-Host Communication**

The host lets the keyboard know that the host wants to send data by pulling the clock line low for more than one clock cycle and then pulling the data line low thereafter. The clock line can only be pulled high again after the data line has been pulled low. As soon as the

host releases the clock line, the keyboard will start generating the clock signal. The host will clock out the data using the clock signal generated by the keyboard. The first bit that will be sent by the host will be a start-bit (logic 0), eight bits data, a parity bit (odd) and ends with logic 1 stop-bit. After the keyboard receives a stop-bit, it will acknowledge the data by pulling the data line low for one clock cycle. This communication between the host and keyboard is shown graphically in Figure 16.



**Figure 16: Host-to-Keyboard Communication**

The source code for the keyboard encoder can be seen in Appendix B. Figures 17 and 18 show a keyboard-to-host example and a host-to-keyboard example that were taken during a real-time operation of the unit. The blue graph shows the data signal, while the red graph is the clock signal.

Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

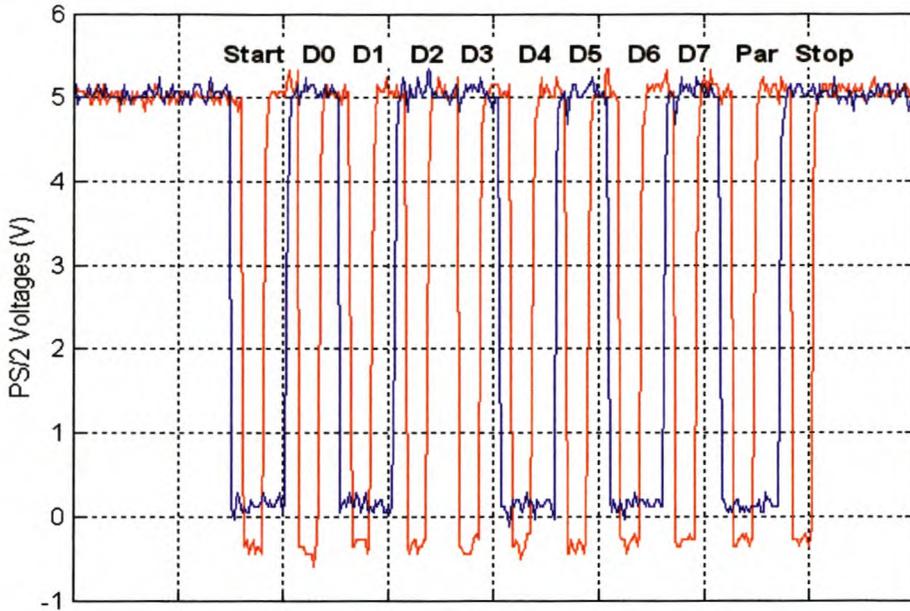


Figure 17: Keyboard-to-Host example

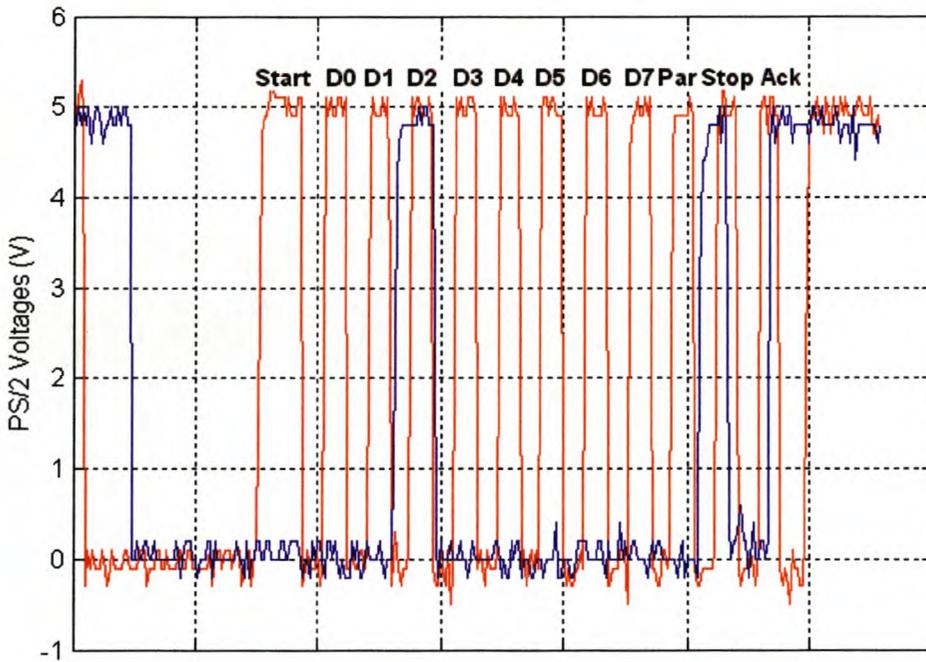


Figure 18: Host-to-Keyboard sample

### 3.3.3 DLR-TUBSAT Control System

Satellites at a height of about 700 km above the earth's surface have a period of about 100 minutes. This gives a ground speed of about 0.06 degrees per second. Thus for the

test measurements, the range that is to be covered is from 0 – 0.1 degrees per second. It is therefore necessary to use an image size where small changes in the angular rate can be detected; the maximum possible size was 512 x 512 pixels, because the camera on board the satellite has a video output. The size of these video outputs is 768 x 576 pixels. The reason why the full image is not taken is that Fourier transforms are available that can process images of the size  $2^n$  much faster than an image that is not a  $2^n$  size image.

The onboard hardware of the DLR-TUBSAT consists of the following:

- **3 reaction wheels** with low power consumption, low mass and small volume. The maximum torque and angular momentum were 0.02 Nm and 0.24 Nms respectively. Each reaction wheel has its own processor unit to control the operating modes (current control, wheel speed control and torque control) of the wheel.
- **3 fiber optic gyroscopes** that were connected via serial communication to the processor unit of each reaction wheel. The bias drift of these gyroscopes was  $<6^\circ/\sqrt{h}$  and the noise  $<0.6^\circ/\sqrt{h}$ . The velocity rate was read out four times per second. The resolution of these gyroscopes was 8 bits. The purpose of these units was to provide 3-axis stabilization;
- **Magnetorquer** used for the reduction of the angular momentum of the satellite.

The control of the DLR-TUBSAT is very simple. The control system on board the DLR-TUBSAT was of such a design that the command station which in this case is the ground station, just needs to command an angular rate to the control system of the satellite and the satellite will make a movement as indicated by the commanded angular rate.

This technique as discussed in Chapter 4, requires a different kind of control algorithm. The algorithm on board the satellite is such that the angular rate needed, is command and applied. The technique discussed in Chapter 4 needs an algorithm where the error between the measured angular rate and the reference angular rate are commanded, in other words, the *angular error rate* or *angular rate increment* needs to be commanded.

Thus the current control software is not applicable to the current situation needed to close the DLR-TUBSAT loop.

### **3.4 Conclusion**

This chapter reviewed Euler's motion equations from which a simplified satellite model was created. There has been looked into the design of a P-control and PD-control system and their effects for a satellite like the DLR-TUBSAT using the RST identification technique. It were shown that if the identification would fail during operation, the PD-control system would change to a P-control system and this put the satellite into an oscillation mode. A unit that would interface the DLR-TUBSAT Control Software was design as the RST identification was done on an external computer. This chapter was concluded by why the DLR-TUBSAT was not feasible to test this technique by closing the loop in real-time and semi-removing the man in the loop.

## 4 Bore-sight Projection Model

### 4.1 Introduction

The bore-sight projection model defines the satellite bore-sight motion by means of a series of images. These series of images can be any images of the same objects taken a few seconds after each other. These objects can be a constellation of stars and/or a landscape on the earth's surface. Processing these series of images could reveal the motion of the satellite.

If it is assumed that the yaw axis is normal to the camera's view surface and the  $x$ -axis of the image is normal to the satellite velocity vector, then a turn around the roll-axis will cause a translation in a left-right direction on the image; a turn around the pitch-axis will cause a translation in the forward-backward direction; a yaw movement will turn the image clock- or anti-clockwise. A positive turn in the satellite axes will be a shift to the right, downwards or anti-clockwise between two successive images and this configuration will be used throughout this chapter.

Matching two image frames requires rotation, scaling and translation (RST) identification between two successive images. The identification of RST can be done using cross-correlation between the two image frames.

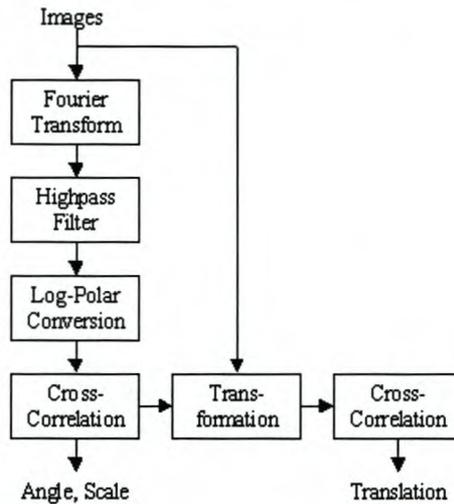
A computationally effective way to calculate a cross-correlation between two functions is by means of Fourier transform algorithms as given in equations 4.1, 4.2 and 4.3.

$$\varphi_{xy}(\tau) = x(t) * y(t) = \int x(t) \cdot y(t - \tau) dt \quad (4.1)$$

$$\Phi_{xy}(\omega) = X(\omega) \cdot Y(\omega) \quad (4.2)$$

$$\therefore \varphi_{xy}(\tau) = F^{-1}\{\Phi_{xy}(\omega)\} \quad (4.3)$$

where  $*$  in equation 4.1 defines the correlation between  $x(t)$  and  $y(t)$ .



**Figure 19: Graphic Overview of Image Processing**

Figure 19 gives a flow chart of the image processing to be made in order to identify RST. A Fourier spectrum of the images is needed on which a high pass filter is being implemented. Next this Fourier spectrum is changed from their original Cartesian grid to a log-polar grid. A cross-correlation would reveal the rotation and scaling factor between these images that is used to transform one of the images. Another cross-correlation would reveal the last unknown factors needed namely the translation between these images.

In this chapter some properties of the Fourier transform that will be used in the image processing will be reviewed. Furthermore, the effects of the image size and the existence of flat surface images on the identification characteristics will also be discussed. Simulations are then done on a very simple image to show the principle with their RST identification.

## 4.2 Fourier Definitions

Let an image be a continuous function  $f(x_1, x_2)$  with  $0 \leq x_1 \leq N$ ,  $0 \leq x_2 \leq M$  in the spatial domain. Let  $F(k_1, k_2)$  be the fast Fourier transform(FFT) with  $0 \leq k_1 \leq N$ ,  $0 \leq k_2 \leq M$  in the frequency domain.  $F(k_1, k_2)$  is defined as:

$$F(k_1, k_2) = \sum_{x_1=0}^{N-1} \sum_{x_2=0}^{M-1} f(x_1, x_2) e^{-j2\pi(x_1 k_1 / N + x_2 k_2 / M)} \quad (4.4)$$

$$f(x_1, x_2) = \frac{1}{MN} \sum_{k_1=0}^{N-1} \sum_{k_2=0}^{M-1} F(k_1, k_2) e^{j2\pi(k_1 x_1 / N + k_2 x_2 / M)} \quad (4.5)$$

Making a shift in the spatial domain will cause a phase shift in the frequency domain as defined in equation 4.6.

$$e^{-j(a k_1 + b k_2)} F(k_1, k_2) \leftrightarrow f(x_1 - a, x_2 - b) \quad (4.6)$$

A rotation of an image through an angle  $\theta$  in the spatial domain will cause a rotation through the same angle in the frequency domain as given in equation 4.7.

$$F(k_1 \cos \theta + k_2 \sin \theta, -k_1 \sin \theta + k_2 \cos \theta) \leftrightarrow f(x_1 \cos \theta + x_2 \sin \theta, -x_1 \sin \theta + x_2 \cos \theta) \quad (4.7)$$

Scaling the axes in the spatial domain will lead to an inverse scaling of the axes in the frequency domain (equation 4.8).

$$\frac{1}{\alpha\beta} F\left(\frac{k_1}{\alpha}, \frac{k_2}{\beta}\right) \leftrightarrow f(\alpha x_1, \beta x_2) \quad (4.8)$$

Correlating two images  $f_1, f_2$  in the spatial domain will give the same results as the inverse Fourier transform of the multiplication of the two images in the frequency domain.

$$F_1(k_1, k_2) \cdot F_2(k_1, k_2) \leftrightarrow f_1(x_1, x_2) * f_2(x_1, x_2) \quad (4.9)$$

## 4.3 Model Analyses

### 4.3.1 FFT Processing

Processing time is of great importance during the image processing and the Fourier transform processing time takes the largest fragment of it. Various algorithms are

available that reduce the processing time. The algorithm that has been used is an FFT Radix-2 algorithm, because of its fast processing speed and the various sizes of images it can process. A flow diagram of 6-point FFT processing using the radix-2 algorithm can be seen in Figure 20. A detail description of the working can be seen in Appendix F. This FFT algorithm can process any image of the size  $2^N$  where N is a natural number.

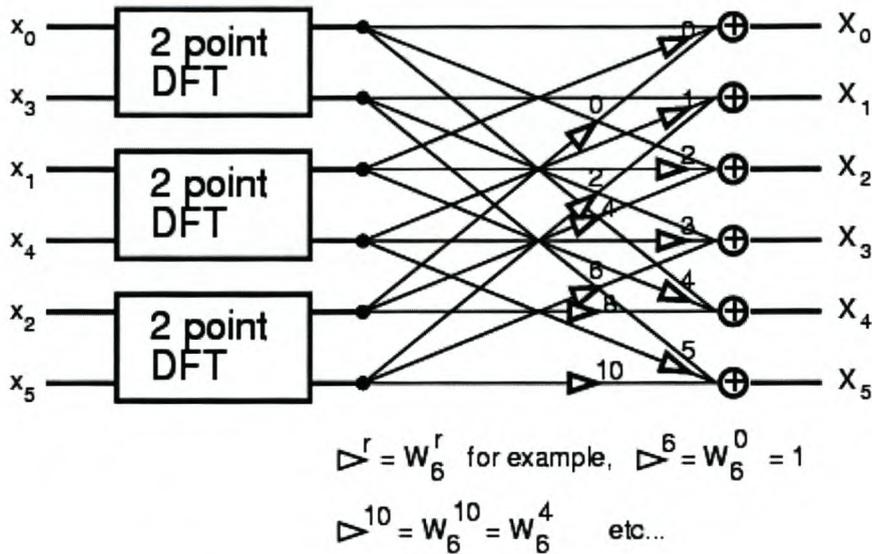
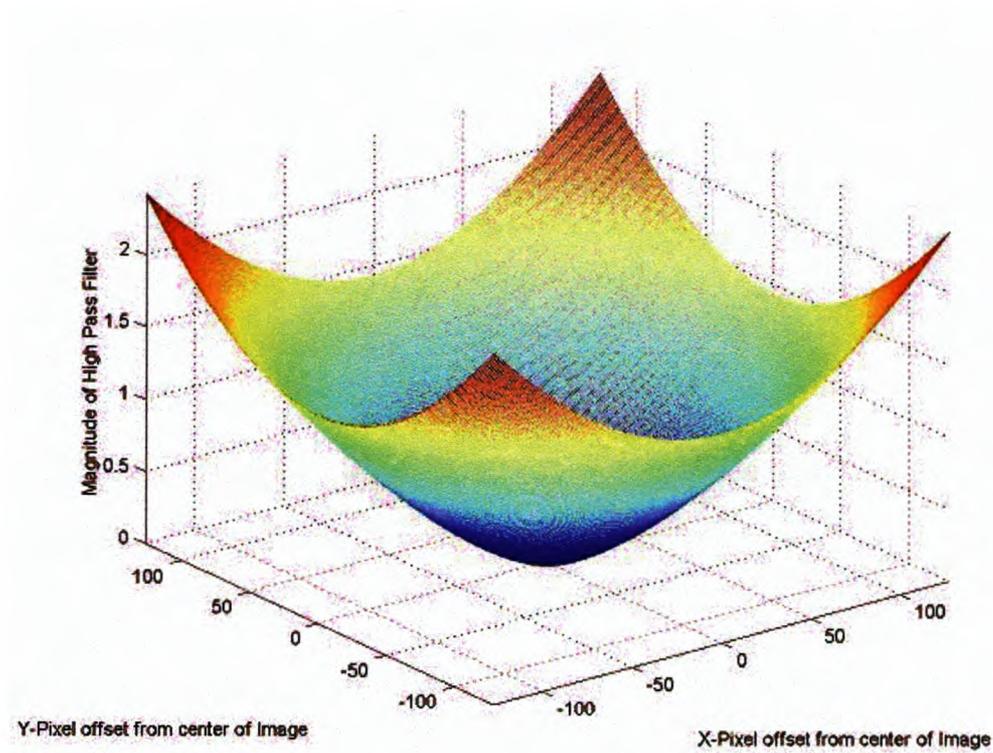


Figure 20: 6-Point DFT using Radix-2 Algorithm

### 4.3.2 High Pass Filtering

A high pass filter is implemented to emphasise the higher frequencies. Emphasising the higher frequency will ensure more accurate correlation of small rotations of the images. The filtering of the high frequencies is done just before converting the Fourier spectra in the cartesian plane to the Fourier spectra in a log-polar plane.



**Figure 21: High Pass Filter for emphasising high frequencies**

The simple high pass filter seen in equation 4.10 was used to do the emphasising of the frequencies,

$$H(\xi, \nu) = \kappa \left( \sqrt{\xi^2 + \nu^2} \right)^\eta \quad (4.10)$$

where  $\eta$  is a degree of emphasis and  $\kappa$  is a scaling factor. Figure 21 shows the high pass filter (mask) as seen in equation 4.10 with  $\kappa = 0.001$  and  $\eta = 1.5$ .

### 4.3.3 Fourier-Mellin Transform

The Fourier-Mellin transform (FMT) (Lin[11]) is an FFT in a log-polar plane. Suppose the Fourier spectrum  $F_2$  is rotated and scaled from the Fourier spectrum  $F_1$  in such a way that equation 4.11 is true.

$$F_2(k_1, k_2) = \frac{1}{\alpha^2} F_1\left(\alpha^{-1}(k_1 \cos \theta_0 + k_2 \sin \theta_0), \alpha^{-1}(k_2 \cos \theta_0 - k_1 \sin \theta_0)\right) \quad (4.11)$$

Rewriting equation 4.11 in the log-polar configuration yields

$$M_2(\rho, \theta) = M_1(\rho - \log \alpha, \theta - \theta_0) \quad (4.12)$$

where  $M_1$  and  $M_2$  are the magnitudes of  $F_1$  and  $F_2$  at each grid point,  $\rho$  is the log-function of the radius of the grid point and is plotted along the vertical axis, and  $\theta$  is the angle of the grid point and plotted along the horizontal axis. The reason for the log-polar format is to enable one to determine the scaling in the images by changing the scaling factor from a gain in equation 4.11 to an offset in 4.12. Doing a cross-correlation between  $M_1$  and  $M_2$  reveals the scaling factor,  $\text{Log } \alpha$ , and the rotation angle,  $\theta_0$ . The formulae for revealing the scaling factor and the rotation angle are

$$\text{Scale} = 10^{\frac{y \log 256}{256}} \quad (4.13)$$

$$\text{Angle}(\theta_0) = \frac{180x}{256} \quad (4.14)$$

if the assumption is made that an image size is 256 x 256 pixels,  $x$  is the difference between the peak value position and centre position along the angle axis, and  $y$  is the difference between the peak value position and the centre position along the log-polar axis. The angle  $\theta_0$  is the turn of the satellite around its yaw axis, if the view direction of the imager is in the yaw axis.

Knowing the scale factor and rotation, the most recent image in time is rotated and scaled by these values. The amount of translation can be found by doing a cross-correlation between the new rotated, scaled image and the unmodified image. The turn around the roll and pitch axes can then be determined using the formulae in equation 4.15 and 4.16.

$$\text{Roll\_Rate} = \arctan\left(\frac{X\_Shift.d}{f}\right) \quad (4.15)$$

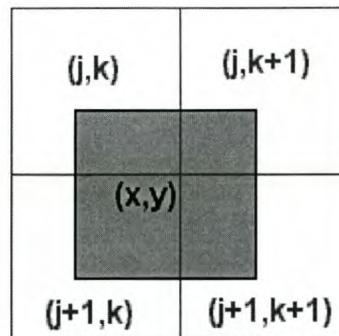
$$\text{Pitch\_Rate} = \arctan\left(\frac{Y\_Shift.d}{f}\right) \quad (4.16)$$

where  $X\_Shift$  and  $Y\_Shift$  are the difference between the peak value position and the centre position of the cross-correlated image. The focal length and the pixel diameter are represented by  $f$  and  $d$  respectively.

The conversion of the Cartesian grid to a log-polar grid leads to computing an average intensity of four grid points, as seen in Figure 22. The new intensity value is computed by taking the same fraction of the grid point intensity as that of the overlapping part of the grid point. Equation 4.17 shows the computation formula for the new intensity values.

$$M(x, y) = M_{j,k} \cdot t \cdot u + M_{j+1,k} \cdot (1-t) \cdot u + M_{j,k+1} \cdot t \cdot (1-u) + M_{j+1,k+1} \cdot (1-t) \cdot (1-u) \quad (4.17)$$

where  $t$  and  $u$  are the fraction of overlapping of  $x$  and  $y$  respectively. Figure 23 shows the Fourier-Mellin transform that is used to identify the rotation and scaling between images. The rotation of the image (top images) around the center is plotted such that this rotation changes to a shift in the images (bottom images) along the horizontal axis. If we know the difference in time between two successive images and also the change in degrees in these images, then the angular rate for each axis as well as the bore-sight's projection speed on the earth can be determined. The latter can be used directly for FMC.



**Figure 22: Computing of New Intensity Value**

Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

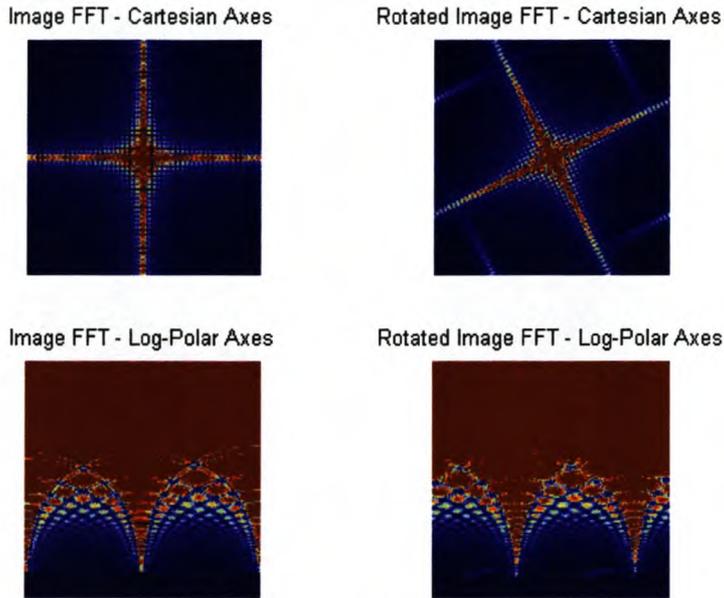


Figure 23: Fourier-Mellin Transform

4.3.4 Effects of the Image Size

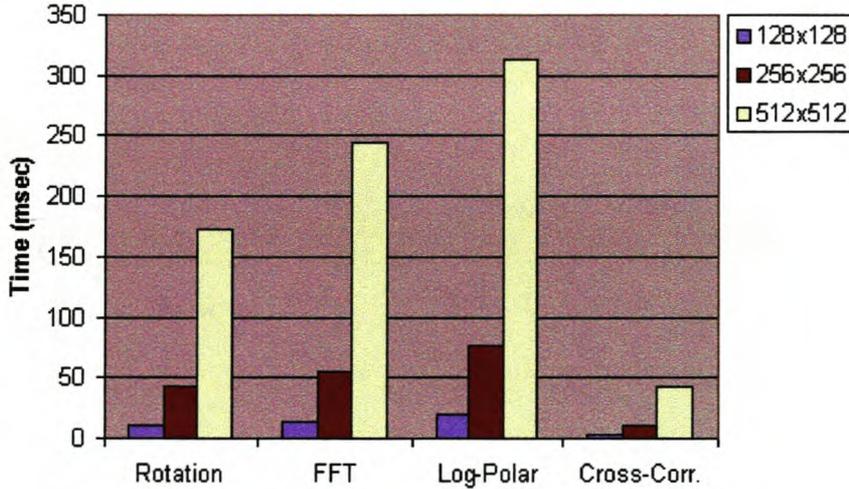


Figure 24: Processing Time for Various Tasks

The size of the images that need to be processed should be carefully selected. The size of the images has a direct effect on the image processing time as well as on the size and accuracy of the rates that can be measured. A large image can measure smaller rates accurately, but a huge amount of processing time will be needed. The inverse is that with

a small image the rates cannot be measured very accurately, but the amount of processing time is not that huge as with a large image. Figure 24 gives the processing time of various tasks for various sizes of images during an image-processing run. The processing was done in a C++ Builder environment on a Pentium 2.2 GHz computer.

#### 4.4 Simulations

Running a simulation with a very simple image (as seen in the left hand picture of Figure 25) tested this technique. The same image was rotated and translated arbitrarily by the computer. The test image was rotated through  $15.23^\circ$  anti-clockwise, translated 8 pixels to the left (-8) and 4 pixels down (4). The new rotated and translated image can be seen in right hand picture of Figure 25 and the results of the cross-correlation can be seen in Figures 27 and 28. The translation of the images in the x-direction is shown on the horizontal axis and the y-direction on the vertical axis as can be seen in figure 27. Figure 28 shows the rotation and scaling results on the horizontal axis and vertical axis respectively. The results after processing show that the image was shifted 8 pixels left, 4 pixels down and rotated through an angle of  $14.41^\circ$  anti-clockwise, which is very near to the original rotation and translation of the test image.

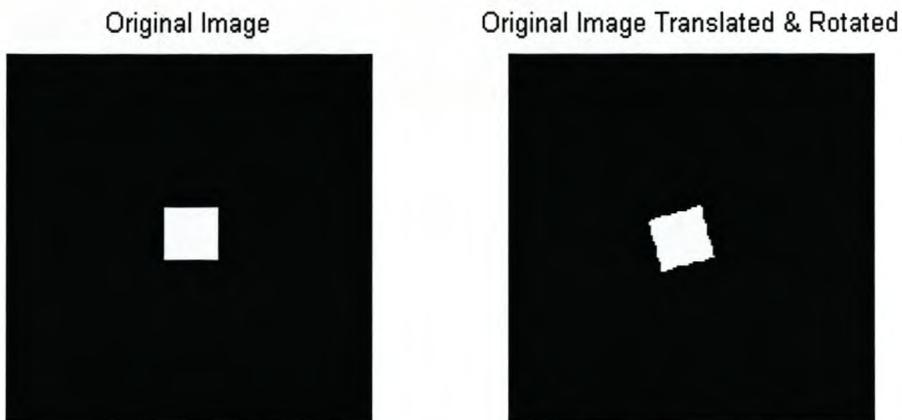


Figure 25: Image taken for Simulation

Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

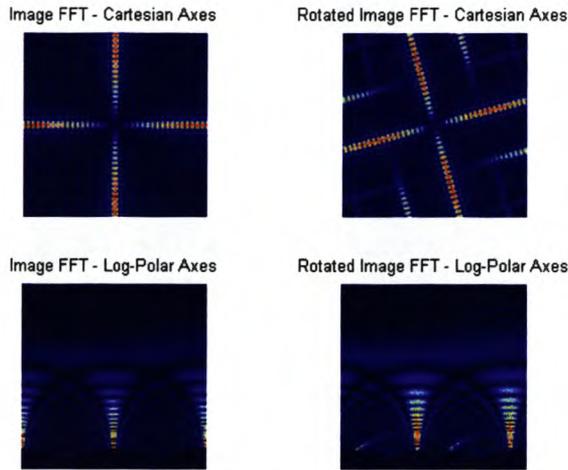


Figure 26: Cartesian and Log-Polar FFT of the Images in Figure 25

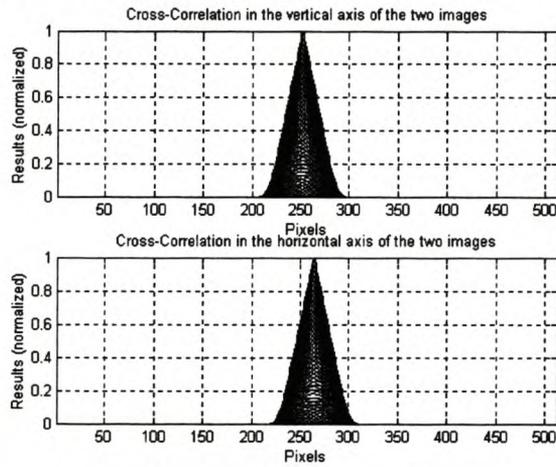


Figure 27: Translation Results of the Images in Figure 25

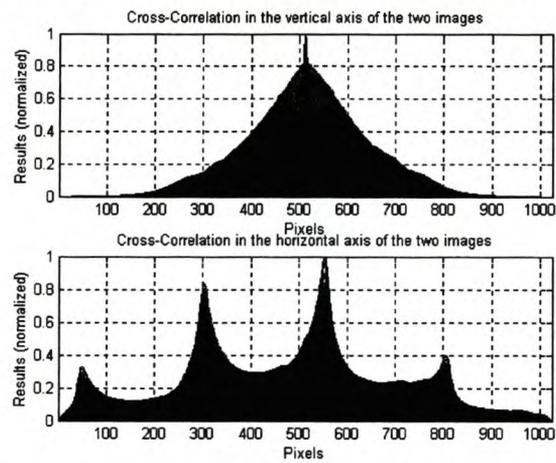
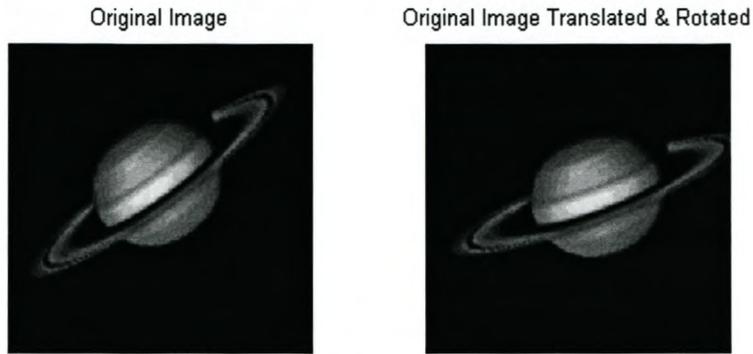


Figure 28: Rotation and Scaling Results of the Images in Figure 25

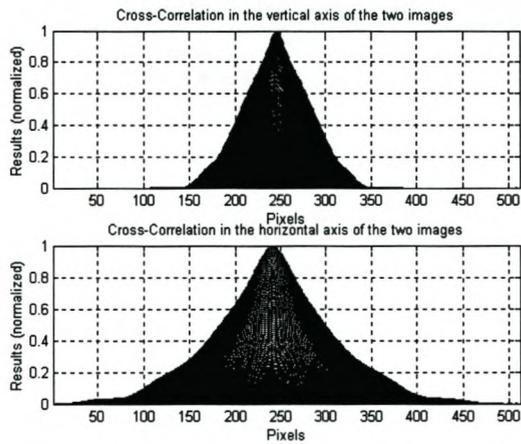
Figure 29 shows images of Saturn. The image on the right hand side of Figure 29 was arbitrarily shift by 15 pixels up, 10 pixels left and rotated by  $15,5^\circ$  clockwise. The results from the simulation of these Saturn images reveal a translation of 15 pixels shift up in the vertical axis, 10 pixels shift left on the horizontal axis and a  $14.76^\circ$  rotation clockwise around the center of the image. The difference in the rotation is due to a 1-pixel error. The possible reasons for this error will be discussed in the next chapter. Each pixel is equivalent to a  $0.703^\circ$  rotation around the centre in the case of an image size of  $256 \times 256$ .

Real satellite images, taken by KITSAT-3 that had a camera with a GSD of  $<15\text{m}$ , were then used on which the same test were performed. The image of the damaged oil fields (left hand image of Figure 32) was originally shifted 50 pixels up and 113 pixels left to produce the image on the right hand side of Figure 32. When these images were shifted, the size of the images was  $400 \times 400$  pixels. These images were then resized to  $256 \times 256$  pixels in order to process them. The result of the process was that the image on the right-hand side of Figure 32 was shifted 72 pixels left and 31 pixels up. The results of the processing can be checked by multiplying them with  $1.563 \left(\frac{400}{256}\right)$  that reveal a shift of 112.5 pixels right and 50 pixels up, which is very close to the shift in the beginning. The value of 1.563 is the factor by which the image size of 400 was scaled down to end with an image size of 256.

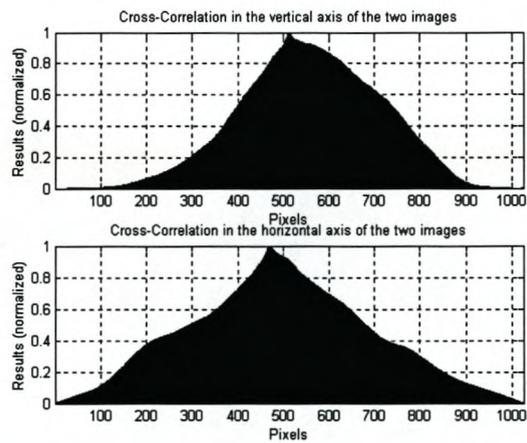
The same test was also performed on the satellite images of Santiago, Chile (Figure 33). The image (with a size of  $400 \times 400$  pixels) on the left hand side was shifted 32 pixels left and 112 pixels up to produce the image on the right hand side. This shifting was done before the processing of it. The processing results revealed that the images were shifted 20 (31,25) pixels left and 71 (110,94) pixels up. The value in the brackets is the shift in the images if the correction is made for the scaling down of the image size. It can then be seen that even images that have been resized for processing would reveal the same answer with a small error in some cases.



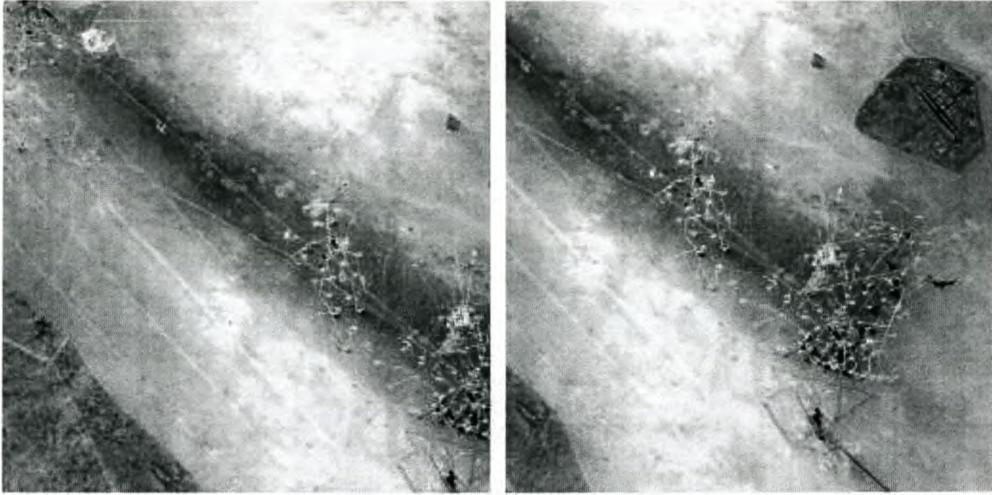
**Figure 29: Images of Saturn translated and rotated by 15 pixels right, 10 pixels down,  $-15,5^\circ$**



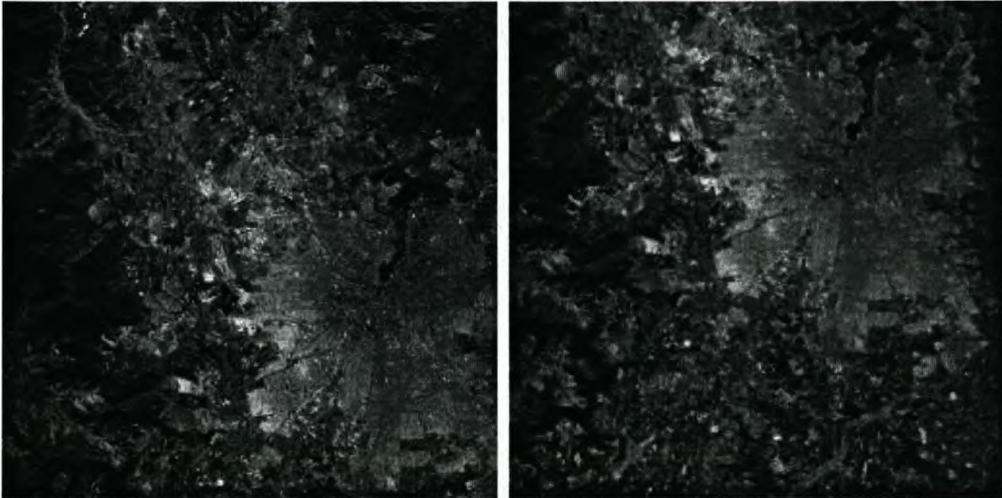
**Figure 30: Translation Results of Figure 29**



**Figure 31: Rotation and Scaling Results of Figure 29**



**Figure 32: Images of damaged oil fields in Kuwait**



**Figure 33: Images of Santiago, Chile**

#### ***4.5 Fourier Transform Limitations due to Flat Surfaces***

One of the many obstacles that limit the performance of cross-correlation is the existence of flat surface images. The unique characteristic of flat surface images is that their colour intensity is very close to each other. Examples of flat surface images are images that consist mainly of clouds, calm water surfaces and low-contrast landscapes. Some of these images with their histograms are shown in Figure 34. The mean and standard deviation values of these images correspond to the histograms seen in Figure 34. The mean value

and standard deviation for the image on the left are 131.1782 and 12.9760 respectively. The values of the image on the right are 210 and 30. Correcting and/or changing these values to more acceptable values of 128 and 64 would enhance the contrast and make motion detection possible.

A model for such enhancement is given in equation 4.18 (Schowengerdt[9]). This technique is known as *normalisation*.

$$\begin{aligned}
 \text{New\_Pixel\_Value} &= \frac{\sigma_{ref}}{\sigma} (\text{Old\_Pixel\_Value} - \mu) + \mu_{ref} \\
 \text{New\_Pixel\_Value} &= 0, \text{New\_Pixel\_Value} \leq 0 \\
 \text{New\_Pixel\_Value} &= 255, \text{New\_Pixel\_Value} \geq 255
 \end{aligned}
 \tag{4.18}$$

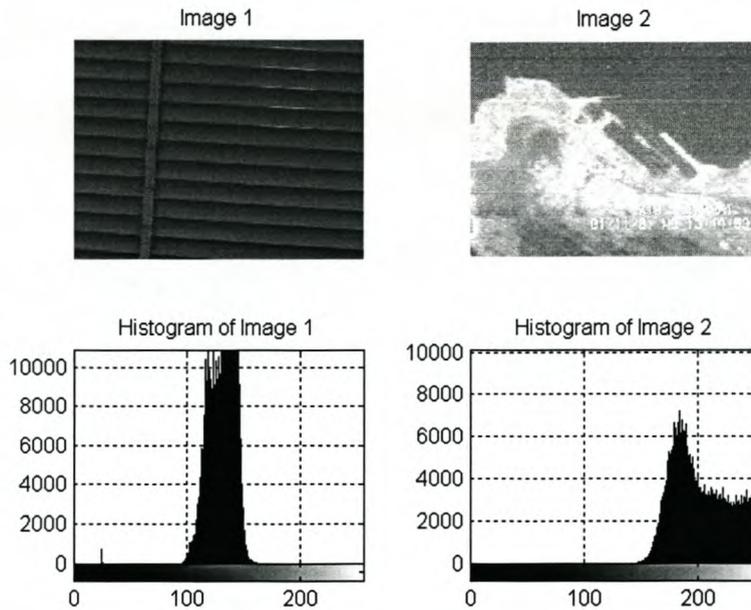
where

$\mu_{ref}$  = reference mean pixel value of the image

$\mu$  = mean pixel value of the image

$\sigma_{ref}$  = reference standard deviation of the image

$\sigma$  = standard deviation of the image



**Figure 34: Examples of Flat Surface Images**

Figure 35 shows an example of an image, taken by DLR-TUBSAT that is changed to the same mean value, but with different deviations. This contrast enhancement can also be done on colour images for each of the primary colours. Figure 36 shows the basic normalisation algorithm in general for images (*Schowengerdt*[9]).

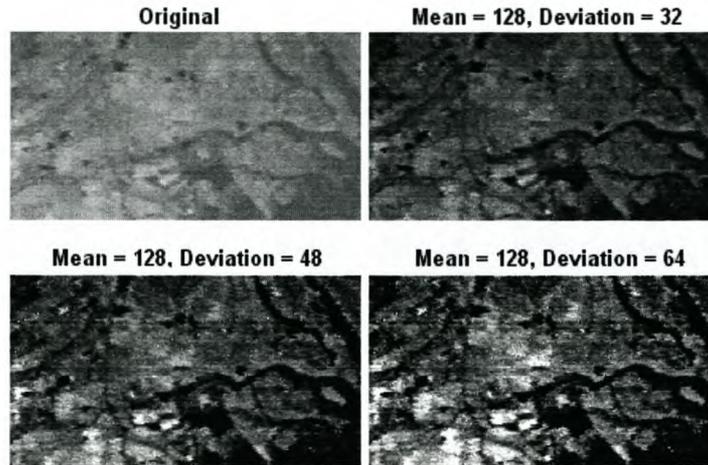


Figure 35: Examples of Image Normalisation with Various deviations

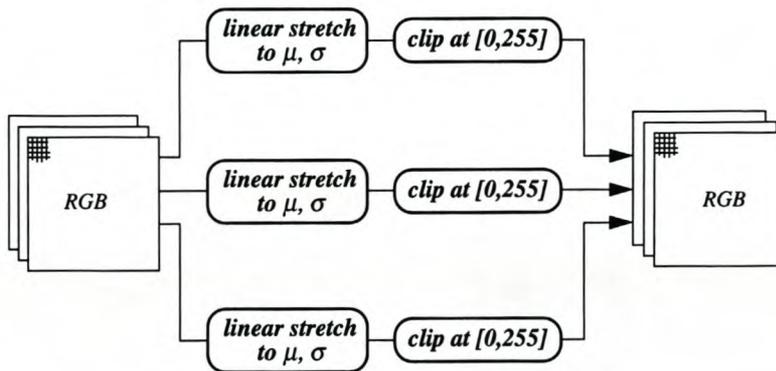
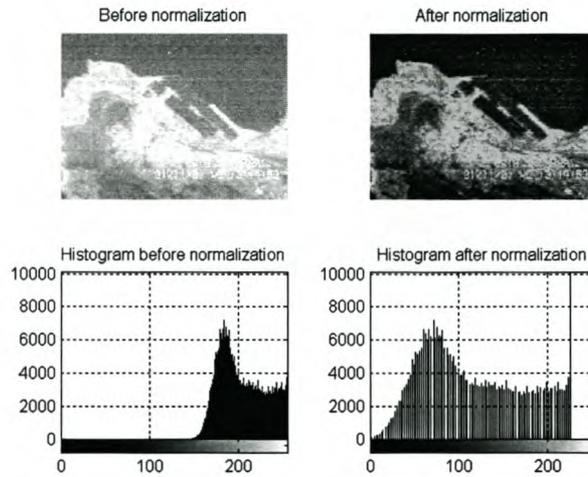


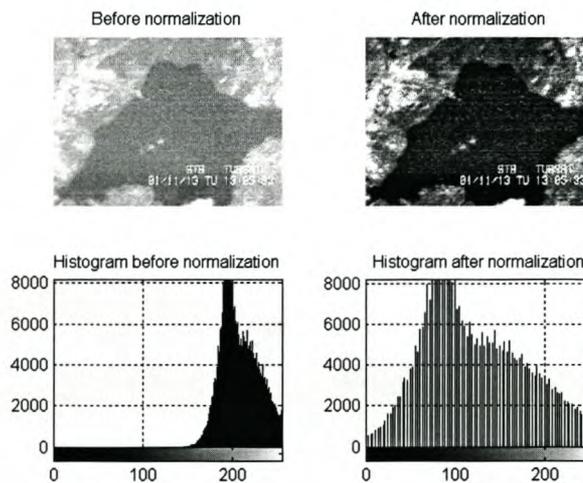
Figure 36: Normalisation Algorithm for Images

Although high pass filtering is applied to these images in the determination of RST, the mean value was still moved so that the colour spectrum coverage is optimised and does not have a concentration of pixels at the image ends (0 and 255). According to the algorithm used, if the new pixel value falls outside of these ends, it is changed to the nearest end value.

Figures 37 and 38 show images and their histograms before and after normalisation. The change in these images can be seen clearly and motion detection on these images is possible.



**Figure 37: Example 1 of Image Normalisation**



**Figure 38: Example 2 of Image Normalisation**

Various other contrast-enhancement algorithms are also available like minimum-maximum, histogram equalisation and threshold algorithms. The minimum-maximum algorithm will enhance the image, but is very sensitive to peaks at the ends of the colour range (0 to 255) and/or noise in the image. The threshold algorithm will not solve the problem very efficiently due to the variation of the threshold level. Histogram equalisation is a very common and easy-to-apply algorithm. It will stretch the image

histogram in order to cover the whole range, but is often very harsh in that the physical appearance of the image.

## **4.6 Conclusion**

In this section the use of the Fourier transform as a computationally effective way of implementing cross-correlation to determine the translation between two successive images was shown. It was also shown that using the Fourier transform of the log-polar grid of the frequency domain would reveal the rotation and scaling between the same two successive images. Simulations were done on simple and more complicated images. The accuracy of these measurements would depend on the size and quality of the images. Using a reasonable image size of 512 x 512 pixels will give good estimation of the real motion of the actual bore-sight's projection on earth. The effects of flat surface images were also discussed and an image normalisation technique was shown to solve the problem to a considerable degree.

## 5 Simulation and Measurements

### 5.1 Introduction

Tests on the accuracy of this technique of measuring the bore-sight motion speed needed to be conducted before implementing it using the DLR-TUBSAT. These tests were done using the Meade LX200 telescope. The telescope is able to turn very accurately at four main speeds, namely 30 arcsec/second ( $0.008^\circ/\text{second}$ ), 480 arcsec/second ( $0.133^\circ/\text{second}$ ),  $2^\circ/\text{second}$  and  $8^\circ/\text{second}$ . These tests will also give an indication of the performance of the processing algorithm.

In this chapter the implementation of the RST identification analyses as presented in Chapter 4 will be discussed. The Meade telescope will be turned at various speeds in various environments to test if the actual rate at which the telescope is turning differs from that measured by the software. The measurements will be analysed in order to determine the reliability of the technique to determine very slow speeds to relatively fast speeds. Thereafter the same measurements would be conducted via DLR-TUBSAT satellite

### 5.2 Implementation

For the processing of the images a Pentium IV 2.2 GHz computer was used for fast processing. The implementation of the software, as seen in Figure 41, can be broken into three phases namely:

- Frame grabbing using the Euresys Picolo card;
- Rotation, scaling and translation Identification via FFT;
- Control of the satellite.

Figure 39 shows the flow diagram for the satellite case if the implementation is done successfully on a satellite while Figure 40 shows the flow diagram of the implementation done for the Meade telescope and satellite test case where the purpose was only to verify

the measurements of the correlation technique on both the telescope and the satellite. The latter flow diagram was implemented successfully. The telescope was turned at various speeds in various directions while bore-sight images are taken with the use of a video camera that is mounted on the telescope. These images are sent to the processing computer and are captured with the Euresys Picolo card. It has to be emphasizing that the telescope loop was not closed as the telescope was mainly used to verify the measurements of the technique. The equipment description can be found in Appendix A. The image processing is followed as described in Figure 19.

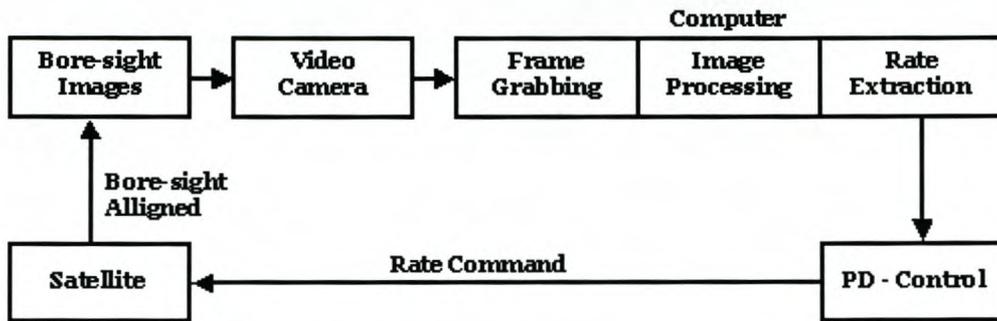


Figure 39: Flow Diagram of a Satellite Implementation

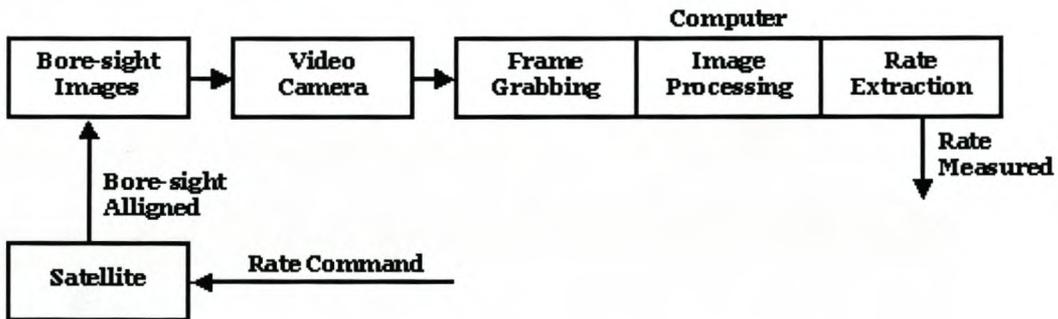


Figure 40: Flow Diagram of Meade LX200 Test Case Implementation



Figure 41: SINav Software Interface

The rotation, scaling and translation identification are done using a Radix-2 FFT algorithm. For the purposes of the Meade case study, the rotation and scaling identification are not undertaken, although they will also be discussed in this section. The reason for this is that the processing time for these two factors is very long relative to the processing time of the translation. Also only the processing of the translation is relevant because of the mobility of the Meade LX200 telescope.

As mentioned earlier, the Radix-2 FFT algorithm was used in the software. 2-D FFT is the same as performing 1-D FFT in the horizontal and vertical direction. This algorithm, however, limits the size of the image to  $2^N$ . Images whose size is not  $2^N$  are zero padded to a size of  $2^N$ . Figure 42 shows the code for 1-D FFT.

```
int TSINav::FFT(int dir, int m, double *x, double *y)
{
    long nn,i,i1,i2,j,k,l,l1,l2;
    double c1,c2,tx,ty,t1,t2,u1,u2,z;

    nn = 1;
    for (i = 0;i < m;i++)
        nn *= 2;
    // Bit reversal //
    i2 = nn >> 1;
    j = 0;
    for (i = 0;i < (nn-1);i++){
        if (i < j){
            tx = x[i];
            ty = y[i];
            x[i] = x[j];
            y[i] = y[j];
            x[j] = tx;
            y[j] = ty;
        }
    }
}
```

```

        }
        k = i2;
        while (k <= j){
            j -= k;
            k >>= 1;
        }
        j += k;
    }
    // Start FFT process //
    c1 = -1.0;
    c2 = 0.0;
    l2 = 1;
    for (l = 0; l < m; l++){
        l1 = l2;
        l2 <<= 1;
        u1 = 1.0;
        u2 = 0.0;
        for (j = 0; j < l1; j++){
            for (i = j; i < nn; i+=l2){
                i1 = i + l1;
                t1 = u1*x[i1] - u2*y[i1];
                t2 = u1*y[i1] + u2*x[i1];
                x[i1] = x[i] - t1;
                y[i1] = y[i] - t2;
                x[i] += t1;
                y[i] += t2;
            }
            z = u1*c1 - u2*c2;
            u2 = u1*c2 + u2*c1;
            u1 = z;
        }
        c2 = sqrt((1.0 - c1)/2);
        if (dir == 1)
            c2 = -c2;
        c1 = sqrt((1.0 + c1)/2);
    }

    if (dir == 1){
        for (i = 0; i << nn; i++){
            x[i] /= (double)nn;
            y[i] /= (double)nn;
        }
    }

    return(TRUE);
}

```

**Figure 42: Radix-2 FFT Code**

The high pass filter of equation 4.10 was implemented. The filter was multiplied with the images spectra as it can be seen in the code in figure 43.

```

    for (j = 0; j < ny; j++)
        for (i = 0; i < nx; i++)
            c[i][j] = c[i][j]*0.1*pow(sqrt(pow((i-nx/2),2)+ pow((i-
            nx/2),2)),1.5);

```

**Figure 43: High Pass Filter Code**

Most of the log-polar conversion computation was pre-determined in the software. The conversion of equation 4.17 was used. The values  $t$  and  $u$  change from one grid point to

another. The variables  $t$  and  $u$  are an indication of the percentage overlapping in the image pixels, as it is describe in Chapter 4. The positional values  $M_{j,k}$ ,  $M_{j+1,k}$ ,  $M_{j,k+1}$  and  $M_{j+1,k+1}$  were also pre-determined in the setup of the software as these values change from image size to image size. During the processing of the images, these values are imported from the memory. New values and positions are determined to create a new image that has a log-polar grid. The code for the conversion is shown in Figure 44.

```

Center_Rows = (Rows+1)/2;
Center_Cols = (Cols+1)/2;
dTheta = PI/Cols;
dLogR = pow(10,log10(Rows)/Rows);

for (i = 0;i < Rows;i++){
    for (j = 0; j < Cols; j++){
        v = j*dTheta;
        w = pow(dLogR,i) - 1;
        XMatrix[i][j] = floor(w*cos(v) + Center_Cols);
        YMatrix[i][j] = floor(w*sin(v) + Center_Rows);
        XFU[i][j] = ceil(XMatrix[i][j]) - XMatrix[i][j];
        YFU[i][j] = ceil(YMatrix[i][j]) - YMatrix[i][j];
        XFL[i][j] = 1 - XFU[i][j];
        YFL[i][j] = 1 - YFU[i][j];
    }
}
.
.
float f1,f2,f3,f4;
int x1,y1;

for (j = 0;j < ny;j++){
    for (i = 0;i < nx;i++){
        x1 = int(XMatrix[j][i]);
        y1 = int(YMatrix[j][i]);
        if ((x1<0)|| (y1<0)|| (x1>=nx-1)|| (y1>=ny-1)){
            d[i][j] = complex<double>(0,0);
        }
        else{
            f1 = float(XFL[i][j]*YFL[i][j])*(abs(c[x1][y1]));
            f2 = float(XFU[i][j]*YFL[i][j])*(abs(c[x1+1][y1]));
            f3 = float(XFL[i][j]*YFU[i][j])*(abs(c[x1][y1+1]));
            f4 = float(XFU[i][j]*YFU[i][j])*(abs(c[x1+1][y1+1]));
            d[i][j] = complex<double>(f1+f2+f3+f4,0);
        }
    }
}

return(TRUE);

```

**Figure 44: Log-Polar Conversion Code**

The cross-correlation was done by multiplying 2 images' spectra with each other and doing an inverse FFT on the multiplied image. Then a search was done for the peak in the image. The inverse was incorporated into the processing of the FFT as this inverse configuration can be seen in the FFT code (Figure 42). The integer  $dir$  gives the direction

of the FFT process. A value of 1 will apply a forward FFT while a value of  $-1$  would apply an inverse FFT process.

### **5.3 Meade LX200 Measurements**

The Meade telescope can turn at four main speeds, as mentioned earlier in this chapter. For the testing or calibration of this technique, the telescope was moved at various speeds in different environments. The environments were different, so the contrast of each test differs from the others. The reason for this was to see how accurate the measurements are if the quality and detail of the images change. The sample time of the images was also changed, so that the shift in the images occurs in the boundaries of detection (as explained in the example below).

Example:

Moving the telescope at a rate of  $0.008^\circ/\text{second}$  and sampling these images every second would mean an angle increment of  $0.008^\circ$  per second. The camera that is used has the specifications that one pixel shift in the images is equal to a turn of  $0.0198^\circ$  and thus an increment of  $0.008^\circ$  would not even be recognised. Sampling these images every ten seconds, while the telescope is moving at a rate of  $0.008^\circ/\text{second}$ , would mean an increment of  $0.08^\circ$  in the angle, which would be equal to a shift of about 4 pixels in these images and this would be recognised by the software.

The graphs in Figure 45, 46, 47 and 48 show the mean values (red dot) as well as the standard deviation (blue line) of the experiments taken using the Meade telescope as a test bed.

Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

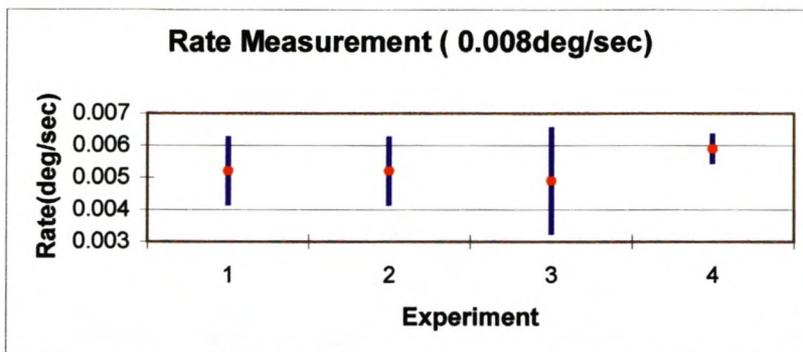


Figure 45: 0.008 deg/sec Rate Measurements

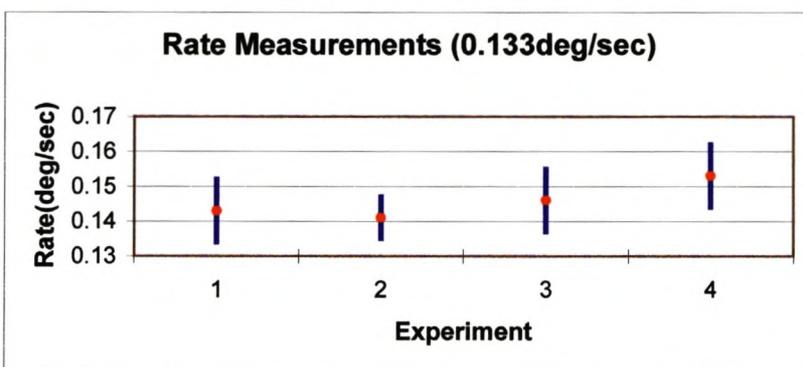


Figure 46: 0.133 deg/sec Rate Measurements

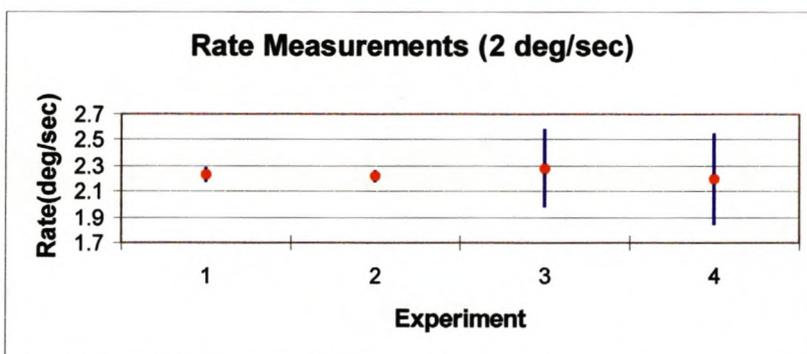


Figure 47: 2 deg/sec Rate Measurements

Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

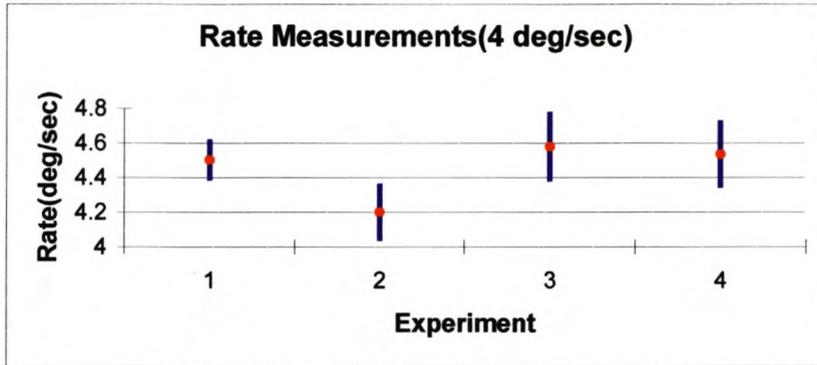


Figure 48: 4 deg/sec Rate Measurements

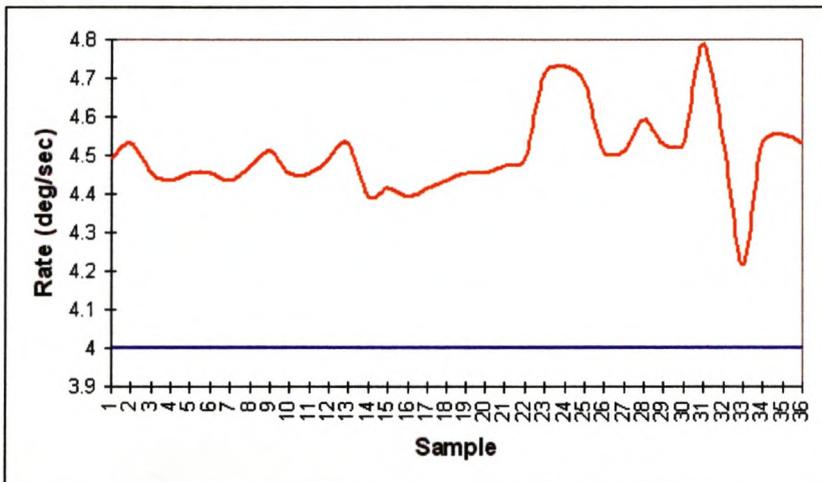


Figure 49: 4 deg/sec Rate Measurements of Experiment 3

Analysing the first four graphs, it can be seen that the measurements are relatively constant over the four samples taken. The standard deviation shows that during the experiment the measured rate did not vary very much.

It can be argued that the error is too large and that it is not acceptable to use this technique for FMC purposes. The first thing to keep in mind is the quantization offset or error, that is

$$Q(\Delta) = \arctan\left(\frac{d}{f}\right)$$

where

$Q(\Delta)$  = quantisation offset or error

$d$  = pixel width of the CCD camera

$f$  = focal length of the CCD camera

The offset or error in the measurements in Figures 45 and 46 are then equal or sometimes smaller than the quantisation error. The measurements were taken with a camera with a focal length of 26 mm and pixel width of  $9\mu\text{m}$ . This gives a quantisation offset or error of 19.8 mdeg/sec. It can be seen that as the angular rate increases, this quantisation effect grows very small to such an extent where it cannot be seen anymore as an effect that has an influence on the error in the measurements. Other possible effects that can cause errors in the measurements will be discussed in the next section. Figure 49 shows the rate measurements (red line) and commanded rate (blue line) versus time where every second a sample was taken for processing. It can be seen that during first 20 samples the variation was not that large and change then rapidly. These changes can be due to environmental effects that will be discussed in a later section

In a case where the command was given to zero the bore-sight motion speed, the software will automatically zero itself over a period of a few seconds even with the offset errors. The reason is that the error becomes smaller over a period of time as the measured rate approaches to zero due to the command given. It can also be seen that in the case of a zeroing command, the overall response can have an over-shoot that varies between 10% and 20% and this is equal to a damping factor of about 0.6. In the case where the command is not to zero the bore-sight motion speed, but to turn at a certain rate, an offset error between 10% and 20%, relative to the commanded rate, can be expected in the measured rate. This error can be corrected for. It can be seen clearly that this technique is ideal for determining zero crossing, as zero crossing can be used for dwelling. A typical zero crossing response based on the measurements undertaken, can look like that seen in Figure 50 where the graph has an overshoot of 10%.

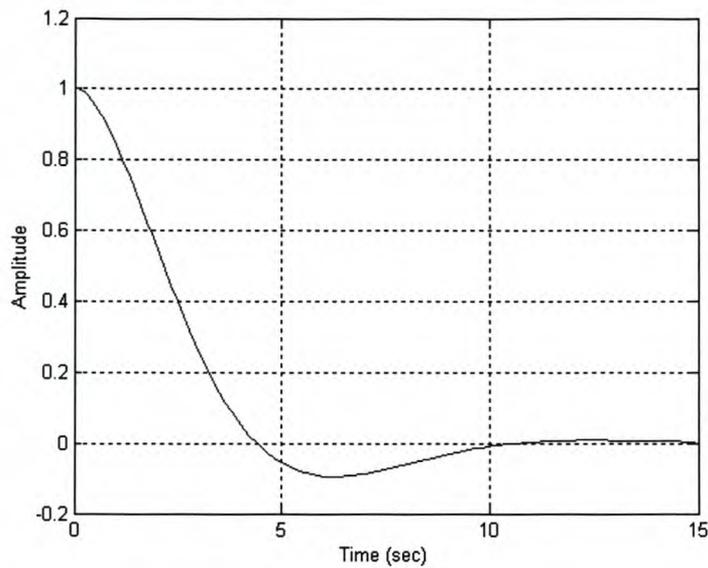


Figure 50: Zero Crossing Response

## 5.4 Measurement Errors

There are a few possible reasons for the errors in the measurements apart from the quantisation offset. They are:

- **Offset errors due to shifting in the FFT images:** In a normal FFT image, the low frequencies are on the edges, while the higher frequencies are at the centre of the image. In the process of the identification the image has been shifted, so the lower frequencies are at the centre of the image and the higher frequencies on the edges. The effect of the shifting causes an error of maximum 2 pixels. Subtracting 1 or 2 pixels can correct for this error in the offset.
- **Noisy images and noisy grabbing of the images:** Noise in the image can cause incorrect measurements and this can be corrected by applying a noise filter to the image before doing the image processing. The possibility also exists that the Pico card grabbing is a bit noisy on the sharp edges if the target to be grabbed, is a fast-moving image. This effect can cause that the peaks to become flat and an error can occur in the measurement.
- **Boundary conditions:** In the RST identification process zero padding is not performed on the edges of the image to decrease the processing time. More accurate result can be achieved if the shift between the two successive images

does not exceed more than  $\frac{1}{4}$  of the image size. If the shift does exceed  $\frac{1}{4}$  of the image size, the reliability of the measurement decreases.

- **Quality of the images:** It has been experienced that the quality of the image has a direct effect on the measurement. In the measurements enough light was available so that a clear, high-contrast image could be produced in order to ensure a high-quality image and this ensured a reliable measurement. Out of focused images could also lead to flat peaks in the correlation results. Images where the so-called flat surface images appear can also lead to a flat peak in the correlation. The latter was reduced and/or removed by cubing the pixels intensity value. This change had a positive influence on the quality and detection ability to quite an extent.
- **Environmental Geometry Effects:** During the measurement, the environment had also change during the experiment. These effects can be described with the help of Figure 51 and 52. In the left hand image of Figure 51 the distance between the surface and the camera stays constant; not matter what the angle is. The surface is viewed from its best direction that is normal to the image surface as it can be seen on the left hand image of Figure 52. Thus a small shift will be recognised and the shift measured would also be the true shift; the surface area viewed by the camera is the same size as that of the real surface. This image represents the ideal case. In the right hand image of Figure 51 the distance between the camera and the surface does not stay constant but is a function of the angle it is viewed by. The surface area viewed by the camera is smaller than that of the real surface, thus the shift measure would also be smaller than the real shift as it can be seen on the image on the right hand side of Figure 52.

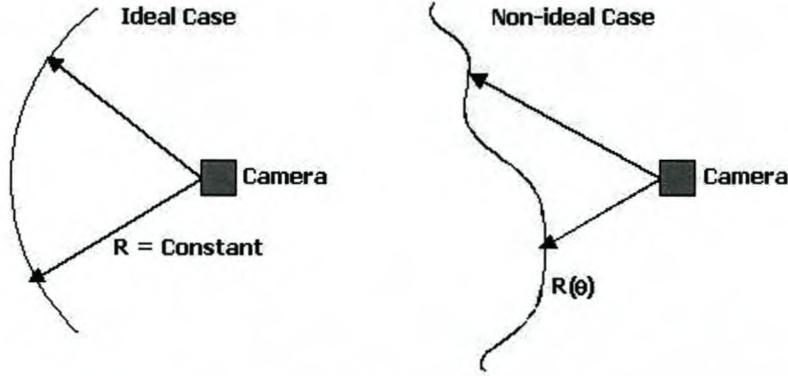


Figure 51: Environmental Distance Effect on Measurements

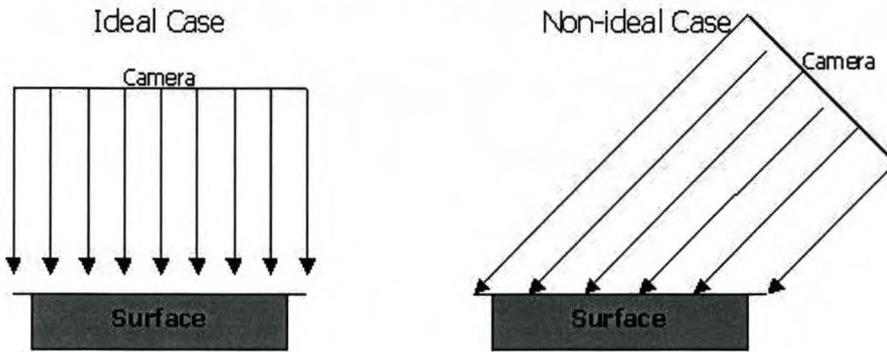


Figure 52: Environmental Surface Effect on Measurements

## 5.5 DLR-TUBSAT Measurements

The measurements of the control systems are done at the ground station in the Electronic System Laboratory at the University of Stellenbosch. The purpose of the experiment is to correlate the applied angular rate with the measured angular rate of the satellite. At the ground station during an overhead pass, various commands were given to the satellite. Figures 51 and 52 shows the commanded (blue line) and measured rate (red line) during the overhead pass of the DLR-TUBSAT. The log-files of the SINav measuring software and the DLR-TUBSAT control software can be viewed in Appendix E.

There are errors in the measurements. These errors are because the pixel width of the camera on board the DLR-TUBSAT is not known exactly and some processing errors because of the lack of a high quality image. The pixel width (assumed to be  $15\mu\text{m}$ ) is

Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

used to determine the angular rate, as it is shown in equation 4.15. The average of the measurements over the short period of samples are relatively constant although the measurement value changes quite frequently. In Figure 53 at sample 11 the measurements are zero and that is possible the result of flat surface images. Image enhancements can remove this feature and possible ways of removing it was discussed in Chapter 4.

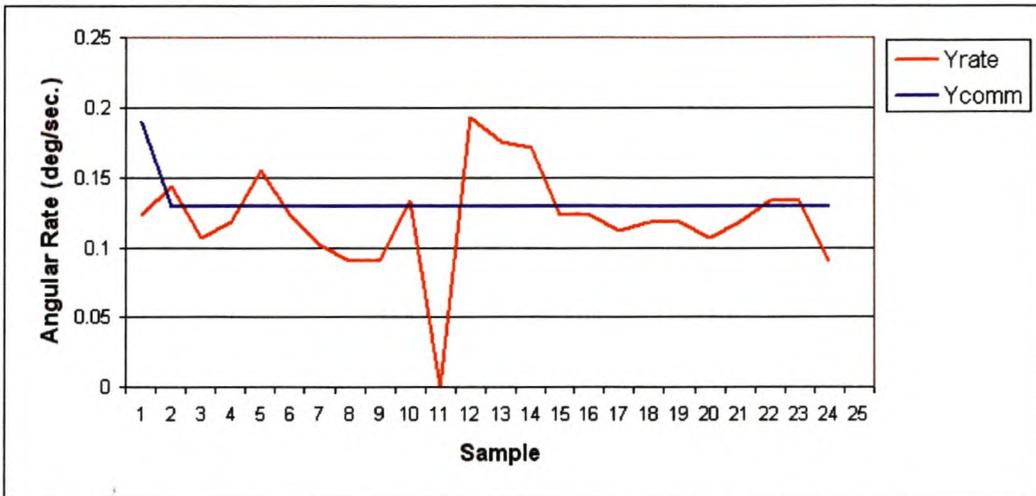


Figure 53: DLR-TUBSAT Pitch Rate Measurement

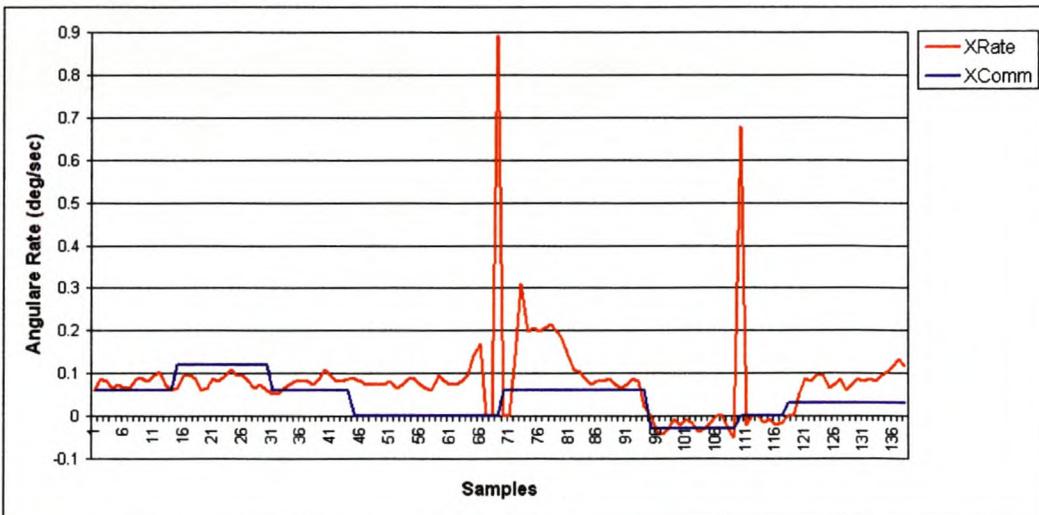


Figure 54: DLR-TUBSAT Roll Rate Measurement

The peaks in Figure 54 can be the result of mainly one feature, namely the sudden lost of signals. On a television screen it be a screen with a lot of noise on it. During samples 45 to 68 the difference between the measured and commanded rate is quite large. A command that did not go through to the satellite or was not been executed on the satellite could have the same effect as seen during the mentioned period of samples. The relative steadiness of the measured rate confirms the effect of a command that was not executed.

From the experiments that have been taken it can be seen that this technique works with the limitations it had. These limitations were that

- That sample rate was 300ms and ideally it need to be around 100ms,
- The size of the image was 256 x 256 pixels so that a 300ms sample rate could be achieved while an image size of above 512 x 512 pixels would be better,
- The minimum image enhancements were done on the images. Some of the enhancements as mentioned in Chapter 4 would have increase the accuracy of these measurements,
- The camera specifications like the width of the camera pixels are not known exactly. This can cause a small variation in the determined angular rates.

The technique of implementing automatic FMC can be successfully accomplished via correlation through Fourier transforms if these limitations are removed from the system.

## ***5.6 Is this technique worthwhile?***

The question arises if it is worth to use this technique as part of the satellite system. The answer to this question depends on if the requirements for such a technique are added to the control system and if theses requirements are reachable with the available hardware. Theoretically this technique is working perfect, but practical this technique is worthless if the few requirements are not achieved. Meeting these requirements makes this technique reliable, economical and an effective way of measuring angular rates on a satellite and to use it as a way of implementing forward motion control easily. The requirements are:

- **Low Noise:** Noise in the image affects the results directly and it affects the results even at times it is not expected. It was experienced during measurements that the best image to use is the raw image itself. The results were very stable over a period of time using the raw data. Using images that have some noise, like video clips, ends up with measurements that are not stable and in most cases incorrect. With the term *incorrect* it is meant that the measurements is not even within 20% of the real value. The reason for this incorrectness is that noise can cause flat peaks in the correlation results, and as the software is searching for the highest peak, it can lead to an incorrect search. Conclusion: the more raw the data, the better and more correct the results.
- **Fast Processing:** Quite some processing is needed with this technique of measuring. The processing-time-growth is directly equivalent to the size of the image. In this case, where forward motion control is performed, at least 5 samples per second are needed. It can be concluded that if the processing power it requires to operate effectively, is available, it is worthwhile to be included into a satellite control system, otherwise the results could lead to incorrect and/or ineffective measurements.
- **Large Images:** In the previous section, it was emphasize that the image size plays an important role in this measurement technique although it evens plays a more important role as it was described. In section 3.3.3 it was said that the angular rate range to be covered, is  $0^\circ/\text{second}$  to around  $0.1^\circ/\text{second}$ . Thus an angular rate of  $0.1^\circ/\text{second}$  is equivalent to a shift of 193 pixels. Thus the image size need to be minimum 3 times the maximum shift that is about 600 pixels. It can be seen that the larger the rates are it has to measure, the larger the image size has to be, and with that in mind, the processing time is longer. This leads to the case, as previously discussed, of fast processing needed.
- **Quality Images:** In Chapter 4 the quality of the images were emphasise. Effects that could influence the quality of the images as discussed in Chapter 4 had been highlighted. Quality-enhancements should also be included if the processing power can accommodate it. The results of such enhancements ensure to quite an extent that motion detection will be possible. The enhancements can be left out

though, but then the chance of motion detection decreases as it were experienced during measurements.

To summarize and answer the question above – yes, it is worthwhile to add this control technique to a satellite control system if most of the requirements are achieved. Should the specifications of the hardware fall outside these requirements; this technique is not worth the effort to add.

## **5.7 Conclusion**

In this chapter the algorithm presented in Chapter 4 was tested at various speeds in the roll and pitch direction of the Meade LX200 telescope. The C++ code is discussed for the various parts of the measuring software. The measured data were processed and shown. Data had shown some offset errors for which some possible reasons were given. Flat surface image measurements were part of the data taken and the results had shown the absence of motion detection in these data regions. Correcting these phenomena, some algorithms were discussed in Chapter 4. DLR-TUBSAT was also used to confirm this technique. The satellite was moved arbitrarily while this technique determined the angular rate at which it was moved. These measurements were confirmed with the log-files of the specific satellite pass. All the results correspond to what was expected for the various movements.

## 6 Conclusion

### 6.1 Summary

In conclusion, it has been shown that cross-correlation by using fast Fourier transform properties could measure the angular rates of satellites, which are needed for various control modes.

Two models were defined that could be used for forward motion compensation. A geometrical model of the target's position relative to that of a spacecraft was defined. This model is used to determine the actual bore-sight vector's speed from the inertial rates of the satellite. The other model is a bore-sight projection model, where the bore-sight's motion speed is measured directly by means of Fourier properties.

Images, which were taken by high-resolution cameras, formed the source of these measurements. The end product of the image processing revealed the translation, rotation and scaling that the images had undergone. The processing consists of performing cross-correlations on images on two levels. The first level revealed the translation of the two successive images. The second level, which consists of performing a cross-correlation on the Fourier transform of the images, revealed the rotation and scaling of the two successive images. The second-level processing was left out where the rotation and scaling of the two images were very small, as this second level's processing time is very long relative to that of the first level. It was shown that the quality of these images plays an important role in the accuracy of the final results. The quality of the images was negatively influenced by problems like flat surfaces, images that were slightly out of focus and images that had been over-exposed. To ensure accuracy in this regard, some solutions to this problem of quality were given.

Tests were performed to determine the accuracy of these measurements via images. A Meade LX200 telescope was turned at a rate of  $0.008^\circ/\text{second}$ ,  $0.133^\circ/\text{second}$ ,  $2^\circ/\text{second}$

and  $4^\circ/\text{second}$ . The measurements were mostly within 20% of the real value. This showed clearly that the satellite would be able to perform zero-crossing measurements.

The final test was conducted using the DLR-TUBSAT satellite. The real-time video clips were processed to reveal RST. These measurements consist of turning the satellite at various speeds and measuring them using the correlation technique. The results showed that this technique reveals RST to a certain degree of accuracy. Limitations that did have an effect on the accuracy of the measurements were identified and discussed throughout this study.

Future research should include the design of an ADCS system that uses cross-correlation and fast Fourier techniques to control remote-sensing equipment and also the auto-focusing of CCD cameras by means of the auto-correlation of an image.

Furthermore, co-operative formation flight control between two or more satellites in the same orbit, but separated in local time, can be achieved by means of the same cross-correlating process, provided that the same imaging targets are visible simultaneously to them. This will allow real-time stereo imaging and also enhance the resolution capabilities of the imager.

This technique can also be used in the military and avionics projects where target controlling is needed. Pointing a missile onto a point of contact and ensuring that it focuses on that point can be done using CCD area cameras and this technique. Another area of research is the use of this technique in micro spy aeroplanes to determine their motion movements.

## **6.2 Recommendations**

This thesis mainly looked at testing the RST identification technique and not optimising it. This process would benefit from splitting the processing unit into a parallel processing unit where one unit, performs coarse matching (finding the area of the peak) while the

other unit is conducting fine matching (finding the peak). Coarse matching would consist of reducing the size of the images and taking a pixel average to define the peak area. After the coarse matching, a fine matching in the area of the peak would carry out the accurate matching. This would increase the speed of the processing of large images.

The problem with most of the images used in this thesis is their quality. Having a unit that could enhance the images for processing would ensure the quality of images needed for processing, and this would decrease the problem of flat surface images. Optimising the contrast quality of images during processing could improve the final results significantly. Using FFT techniques can ensure a very high contrast level.

It is also recommended that a suitable satellite is needed to test this technique. By loading this software for implementing this technique onto a suitable satellite, would reveal the best and most reliable results. Most of the problems experienced will also be removed by this recommendation.

All these methods recommended would make the technique discussed in this thesis a very reliable source for a satellite's ADCS, where reducing equipment size and cost, and providing autonomous controlling are priorities.

## 7 Bibliography

- [1] Petrie, G.: "High Resolution Space Imagery" *Survey Ireland*, University of Glasgow, Winter 1999
- [2] Gottzein, E., Fichter, W., Jablonski, O., Juckenhöfel, O., Mittnacht, M., Müller C., Surauer, M.: "Challenges in the control and autonomy of communication satellites" *Control Engineering Practice*, Vol. 8, Year 2000, pp. 406-427
- [3] Wertz, J.R., Larson, W.J.: "*Space Mission Analysis and Design*", Third Edition, Kluwer Academic Publishers, 1999, pp. 95 – 130
- [4] Oosthuizen, P.J.: "*Development of a Custom Machine Vision Camera for Froth Flotation*", University of Stellenbosch, June 1999, pp. 74 – 91
- [5] Reddy, B.S., Chatterji, B.N.: "An FFT-Based Technique for Translation, Rotation and Scale-Invariant Image Registration" *IEEE Transactions on Image Processing*, Vol. 5, No.8, August 1996, pp. 1266 – 1271
- [6] Xie, H., Hicks, N., Keller, G.R., Huang, H., Kreinovich, V.: "*Automatic Image Registration based on a FFT Algorithm and IDL/ENVT*" Pan America Center for Earth and Environmental Studies and Department of Computer Science, University of Texas
- [7] Schulz, S., Renner, U.: "*DLR-TUBSAT: a Microsatellite for Interactive Earth Observation*", Technical University of Berlin and Institute of Aerospace
- [8] Engelbrecht, J.A.A.: "*A Hardware-in-the-Loop Simulation Facility for the Attitude Determination and Control System of SUNSAT*", University of Stellenbosch, December 1999, pp. 27 – 37

- [9] Schowengerdt, R.A.: “*Remote Sensing – Models and Methods for Image Processing*”  
Second Edition, Academic Press, 1997, pp. 202 – 226
- [10] Bourke, P.: “*2 Dimensional FFT*”,  
<http://astronomy.swin.edu.au/~pbourke/analysis/fft2d/>
- [11] Lin, C.Y., Wu, M., Bloom, J.A., Cox, I.J., Miller, M.L., Lui, Y.M.: “Rotation, Scale and Translation Resilient Watermarking for Images” *IEEE Transactions on Image processing*, Vol.10, No. 5, May 2001, pp. 767 - 782
- [12] Peacock, C.: “*Interfacing the AT keyboard*”,  
<http://www.beyondlogic.org/keyboard/keybrd.htm>

## **Appendix A: Equipment description**

## Euresys Pico Frame Grabbing

The Euresys Pico Frame grabbing card (Figure 55) was used in the software. The card can receive various types of video signals including PAL, CCIR and NTSC, and can also be interfaced using most of the commonly used programming languages like C/C++, Visual Basic and Java. The analogue video signals are sent via a BNC, DB9 or mini DIN4 port. The DB9 port can accommodate two different video signals at the same time. The interfaced code in C++ used to set up the card is shown in Figure 56. The card can grab up to 25 frames/sec and takes 60 – 70 msec to grab a video frame and store it in the memory assigned to it.

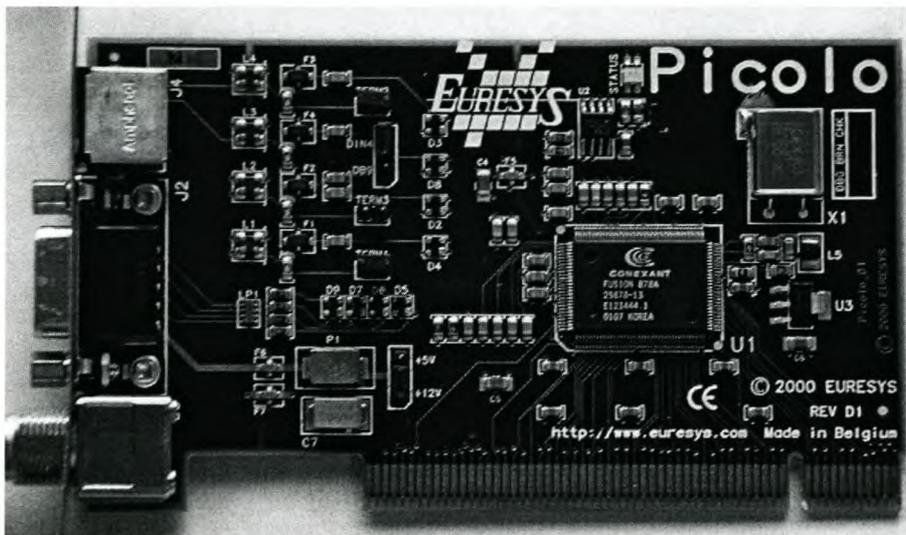


Figure 55: Euresys Pico Frame Grabbing Card

```
void TSINav::CAMERASETUP(void)
{
  EImageBW8 BW8Image1;
  EImageBW8 BW8Image2;
  EMCImageBW8 EMCBW8Image;

  EMCBoard* pEMCBoard = NULL;
  EmcGetBoardByIndex(0,pEMCBoard);
  pEMCBoard->SetParam(MC_BoardTopology,MC_BoardTopology_1_1);

  switch (VideoStandard->ItemIndex) {
    case 0:
      m_Image.AssignSourceByIndex("CCIR", "PICOLO_0", 0);
      break;
    case 1:
      m_Image.AssignSourceByIndex("EIA", "PICOLO_0", 0);
      break;
    case 2:
      m_Image.AssignSourceByIndex("PAL", "PICOLO_0", 0);
      break;
    case 3:
      m_Image.AssignSourceByIndex("NTSC", "PICOLO_0", 0);
      break;
    case 4:

```

```
        m_Image.AssignSourceByIndex("SECAM", "PICOLO_0", 0);  
        break;  
    }  
    m_Image.SetParam(EC_PARAM_VideoGain, 65536 * (CameraColorGain->Position) / 100);  
    m_Image.SetParam(EC_PARAM_VideoOffset, 65536 * (CameraColorGain->Position) / 1000);  
  
    ImageFrame1.SetSize(&m_Image);  
    ImageFrame2.SetSize(&m_Image);  
    ImageFrame3.SetSize(&m_Image);
```

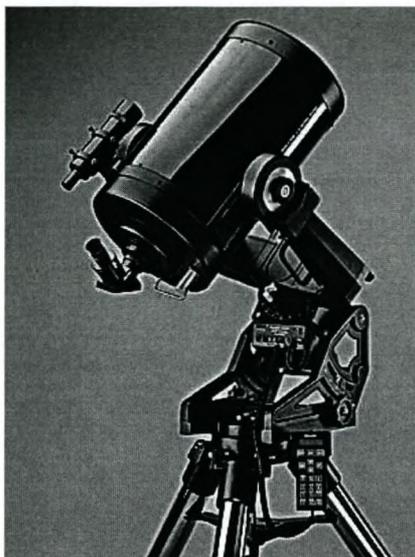
**Figure 56: Picolo Card Setup Code**

## Meade LX200 Telescope Control

The Meade LX200 telescope (Figure 58) can be controlled through an RS232-port at a rate of 9600kbytes/sec. The telescope is reset to its home position that will serve as a reference point for controlling it. The home position is defined as the altitude and azimuth coordinates in the memory of the telescope at start up. These positions can be change at any time using some of the commands given in Appendix A. The PULNiX camera (Figure 57) is connected to the telescope. This +12DC black and white camera has a CCIR output signal that is connected to the Picolo card.



**Figure 57: PULNiX Camera**



**Figure 58: Meade LX200 Telescope**

The maximum rate of the telescope is  $8^\circ/\text{second}$  and have four movement speeds namely  $30\text{arcsec}/\text{sec}$ ,  $480\text{arcsec}/\text{sec}$ ,  $2^\circ/\text{sec}$  and  $8^\circ/\text{sec}$ . Yaw measurement was not taken because the Meade LX200 telescope does not accommodate yaw movements.

## **Appendix B: Keyboard Encoder source code**

```

*****
,*
,*
,*   Keyboard Encoder
,*   C.J. Mouton
,*   January 2003
,*
,*
,*   Ver. 1.0 10/01/2003
,*
*****

.include "2313def.inc"

*****

.equ ClockIn      =1      ;
.equ DataIn       =0      ;
.equ ClockOut     =3      ;
.equ DataOut      =2      ;
.equ Select_pin   =5      ;
.equ Serial_pin   =6      ;

.def tmp          = r16   ;
.def DATATemp     = r17   ;
.def KBOutData    = r18   ;
.def KBInData     = r19   ;
.def Parity       = r20   ;
.def Delay_Counter = r22   ;
.def KBDDataSafe  = r23   ;

***** Main Program *****

.cseg
.org $0000
rjmp RESET

.org $0001
rjmp SWITCH_MODES

.org $0002
rjmp HOST_INTERRUPT

.org $0007
rjmp UART_Receive

RESET:
ldi tmp,RAMEND
out SPL,tmp
rcall INIT

START:
rcall HOST_PULL_LOW
rjmp START

***** Initialize Routine *****

```

```

INIT:
    clr DATATemp          ;
    clr Parity           ;

    ldi tmp,$68
    out DDRD,tmp         ; D.3,D.5 & D.6 = Outputs
    cbi PORTD,Select_pin ; Set in Keyboard - mode
    sbi PORTD,Serial_pin ;

    ldi tmp,$fc          ; 11111100b
    out DDRB,tmp        ; Port B.0 & B.1 = Inputs, Rest = Output

    ldi tmp,$00
    out PORTB,tmp       ; Make the KBClock and Data lines high

    ldi tmp,$98
    out UCR,tmp         ;
    ldi tmp,71
    out UBRR,tmp       ; Baudrate = 9600kb

    ldi tmp,$a
    out MCUCR,tmp      ;

    ldi tmp,$C0
    out GIMSK,tmp     ;

    ldi tmp,$c0
    out GIFR,tmp      ;

    sei                 ; Set interrupt flag (Enable)

    ret

;
;***** Switch Modes Routine *****
;
SWITCH_MODES:
    sbis PORTD,Select_pin ; Check mode
    rjmp DISABLE_KB      ; Was in KB_Mode
    rjmp ENABLE_KB       ; Was in Serial Mode

DISABLE_KB:
    sbi PORTD,Select_pin ; Disable KB
    cbi PORTD,Serial_pin ; Enable Serial
    rjmp EXIT_INT0

ENABLE_KB:
    sbi PORTD,Serial_pin ; Disable Serial
    cbi PORTD,Select_pin ; Enable KB

EXIT_INT0:
    rcall delay20        ; Wait 20usec
    rcall delay20        ; Wait 20usec
    ldi tmp,$7d
    out UDR,tmp
    sei

```

```

    reti                                ; Return Interrupt

;***** UART-Receive Routine *****
UART_Receive:
    in KBOutData,UDR                    ; Read valid data
    cpi KBOutData,$01                   ; Check if Data = 0x01H
    breq SWITCH_MODES                   ; if true = Switch modes
    mov KBDDataSafe,KBOutData           ; Safe KBOutData
    rcall PARITY_DET                     ; Determine Parity bit
    rcall CONFIRM_TEST                   ; Check if host will receive data
    rcall delay100                       ; Delay +/- 100usec
    com KBOutData                         ; Invert Data
    rcall START_KB_SEND                  ; Send KBDData
    rcall delay100                       ; Delay +/- 100usec
    ldi KBOutData,$F0                   ; Load KBDData for Key_release
    rcall PARITY_DET                     ; Determine Parity bit
    rcall CONFIRM_TEST                   ; Check if host will receive data
    rcall delay100                       ; Delay +/- 100usec
    rcall delay100                       ; Delay +/- 100usec
    com KBOutData                         ; Invert KB Data
    rcall START_KB_SEND                  ; Send KB Data to Host
    rcall delay100                       ; Delay +/- 100usec
    mov KBOutData,KBDDataSafe           ; Load KB Data to identify released key
    rcall PARITY_DET                     ; Determine Parity bit
    rcall CONFIRM_TEST                   ; Check if host will receive data
    rcall delay100                       ; Delay +/- 100usec
    rcall delay100                       ; Delay +/- 100usec
    com KBOutData                         ; Invert Data
    rcall START_KB_SEND

EXIT_UART_Receive:
    rcall delay100                       ; Delay +/- 100usec
    out GIFR,tmp
    sei                                    ; Reset Interrupt
    reti

PARITY_DET:
    mov tmp,KBOutData                    ; Move KBOutData to tmp
    mov DATATemp,KBOutData               ; Move KBOutData to DataTemp
    ror DATATemp                          ; Rotate DataTemp 1 bit right through carry
    eor tmp,DATATemp                      ; Exclusive OR tmp & DataTemp
    ror DATATemp                          ; Rotate DataTemp 1 bit right through carry
    eor tmp,DATATemp                      ; Exclusive OR tmp & DataTemp
    ror DATATemp                          ; Rotate DataTemp 1 bit right through carry
    eor tmp,DATATemp                      ; Exclusive OR tmp & DataTemp
    ror DATATemp                          ; Rotate DataTemp 1 bit right through carry
    eor tmp,DATATemp                      ; Exclusive OR tmp & DataTemp
    ror DATATemp                          ; Rotate DataTemp 1 bit right through carry

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

eor tmp,DATAtemp      ; Exclusive OR tmp & DataTemp
ror DATAtemp         ; Rotate DataTemp 1 bit right through carry
eor tmp,DATAtemp      ; Exclusive OR tmp & DataTemp
ror DATAtemp         ; Rotate DataTemp 1 bit right through carry
eor tmp,DATAtemp      ; Exclusive OR tmp & DataTemp
andi tmp,$01
mov Parity,tmp
ret

CONFIRM_TEST:
sbis PINB,DataIn      ; check if D.3 is high
rjmp CONFIRM_TEST     ; not high
sbis PINB,ClockIn     ; check if D.4 is high
rjmp CONFIRM_TEST     ; not high
ret                   ; both are high

START_KB_SEND:
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
sec                   ; Clr carry flag
rcall SENDDATA        ; Send Start-bit (0)
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate LSB into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate bit-1 into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate bit-2 into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate bit-3 into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate bit-4 into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate bit-5 into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate bit-6 into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
ror KBOutData         ; Rotate MSB into carry
rcall SENDDATA        ; Send bit
rcall HOST_PULL_LOW   ; Check if Clock is pulled low
rcall SENDPARITY      ; Send Parity bit
clc                   ; Set Carry flag
rcall SENDDATA        ; Send Stop-bit (1)
ret

SENDPARITY:
sbrs Parity,0         ; Check Parity status
rjmp Parity_Clr

Parity_Set:
sec                   ; Parity set
rcall SENDDATA        ; Set carry flag
rcall SENDDATA        ; Send bit

```

```

        rjmp EXIT_SENDDATA
Parity_Clr:                ; Parity clr
        clc                ; Clr carry flag
        rcall SENDDATA    ; Send bit
EXIT_SENDDATA:
        ret

SENDDATA:
        brcs High_bit     ; Check carry flag status
Low_bit:                  ; Carry low (0)
        cbi PORTB,DataOut ; Clr D.4
        rcall delay20     ; Wait 20 usec
        sbi PORTB,ClockOut ; Clr D.3
        rcall delay20     ; Wait 20 usec
        rcall delay20     ; Wait 20 usec
        cbi PORTB,ClockOut ; Set D.3
        rcall delay20     ; Wait 20 usec
        rjmp EXIT_SENDDATA
High_bit:                 ; Carry high (1)
        sbi PORTB,DataOut ; Set D.4
        rcall delay20     ; Wait 20 usec
        sbi PORTB,ClockOut ; Clr D.3
        rcall delay20     ; Wait 20 usec
        rcall delay20     ; Wait 20 usec
        cbi PORTB,ClockOut ; Set D.3
        rcall delay20     ; Wait 20 usec
EXIT_SENDDATA:
        ret

HOST_PULL_LOW:
        sbis PINB,ClockIn ; Check if ClockIn is low
        rjmp EXIT_UART_Receive ; If True - abort transmittion
        ret

;***** Host Interrupt *****
HOST_INTERRUPT:
        clr KBInData      ; clear Keyboard IN-Data Register
        sbic PINB, ClockIn ; Check if Clock-line low
        rjmp HOST_INTERRUPT ; Clock-line not low

Clock_low:
        sbis PINB, DataIn ; Check if Data-line high
        rjmp Clock_low    ; Data-line low

Data_low:
        sbic PINB, ClockIn ; Check if Clock-line still low
        rjmp EXIT_HOST_INTERRUPT ; Abort if not

Start_KB_Read:
        rcall KB_Read     ; Read Bit 0 (LSB)
        ror KBInData      ; Shift carry into Keyboard Register
        rcall HOST_PULL_LOW ; Check if Clock is pulled low
        rcall KB_Read     ; Read Bit 1
        ror KBInData      ; Shift carry into Keyboard Register
        rcall HOST_PULL_LOW ; Check if Clock is pulled low
        rcall KB_Read     ; Read Bit 2

```

```

    ror KBIInData          ; Shift carry into Keyboard Register
    rcall HOST_PULL_LOW   ; Check if Clock is pulled low
    rcall KB_Read         ; Read Bit 3
    ror KBIInData          ; Shift carry into Keyboard Register
    rcall HOST_PULL_LOW   ; Check if Clock is pulled low
    rcall KB_Read         ; Read Bit 4
    ror KBIInData          ; Shift carry into Keyboard Register
    rcall HOST_PULL_LOW   ; Check if Clock is pulled low
    rcall KB_Read         ; Read Bit 5
    ror KBIInData          ; Shift carry into Keyboard Register
    rcall HOST_PULL_LOW   ; Check if Clock is pulled low
    rcall KB_Read         ; Read Bit 6
    ror KBIInData          ; Shift carry into Keyboard Register
    rcall HOST_PULL_LOW   ; Check if Clock is pulled low
    rcall KB_Read         ; Read Bit 7
    ror KBIInData          ; Shift carry into Keyboard Register
    rcall HOST_PULL_LOW   ; Check if Clock is pulled low
    rcall KB_Read         ; Read Parity Bit
    rcall KB_Read         ; Read Stop-bit
    rcall SEND_ACK        ; Send Acknowledge-bit
    rcall UART_SEND       ; Send Keyboard data via Serial Port
EXIT_HOST_INTERRUPT:
    rcall delay100        ; Delay 100 usec
    ldi tmp,$c0
    out GIFR,tmp
    sei                   ; Reset interrupt routine
    reti

SEND_ACK:
    rcall delay15         ; Delay 15usec
    sbi PORTB, DataOut    ; Set bit for data-line
    rcall delay5          ; Delay 5usec
    sbi PORTB, ClockOut   ; Set bit for clock-line
    rcall delay20         ; Delay 20usec
    rcall delay20         ; Delay 20usec
    cbi PORTB, ClockOut   ; Clear bit for clock-line
    rcall delay5          ; Delay 5usec
    cbi PORTB, DataOut    ; Clear bit for Data-line
    rcall delay15         ; Delay 15usec
    ret

KB_Read:
    rcall delay20         ; Delay 20usec
    sbi PORTB, ClockOut   ; Set bit for Clock-line
    rcall delay20         ; Delay 20usec
    rcall delay20         ; Delay 20usec
    cbi PORTB, ClockOut   ; Clear bit for Clock-line
    rcall delay20         ; Delay 20usec
    sbis PINB, DataOut    ; Check the status of Data-line
    rjmp ClrCarry         ; Jump if cleared
    sec                   ; Set carry flag for read-in
    ret

ClrCarry:
    clc                   ; Clear carry flag for read-in
    ret

```

```

UART_SEND:
    sbis USR,UDRE                ; Check if port is ready to send data
    rjmp UART_SEND
    out UDR,KBInData            ; Send KB-Data serial
    ret

```

```

;***** Delay Routines *****

```

```

delay5:
    ldi Delay_Counter,56        ; Load Counter
not5:
    dec Delay_Counter          ; Decrement Counter
    cpi Delay_Counter,0        ; Compare with zero
    brne not5                  ; Stop Timer 0
    ret

```

```

delay15:
    ldi Delay_Counter,166       ; Load Counter
not15:
    dec Delay_Counter          ; Decrement Counter
    cpi Delay_Counter,0        ; Compare with zero
    brne not20                 ; Stop Timer 0
    ret

```

```

delay20:
    ldi Delay_Counter,53        ; Load Counter
not20:
    dec Delay_Counter          ; Decrement Counter
    cpi Delay_Counter,0        ; Compare with zero
    brne not20                 ; Stop Timer 0
    ret

```

```

delay100:
    ldi Delay_Counter,250       ; Load Counter
not100:
    dec Delay_Counter          ; Decrement Counter
    cpi Delay_Counter,0        ; Compare with zero
    brne not100                ; Stop Timer 0
    ret

```

## **Appendix C: Keyboard Encoder Schematics and PCB**

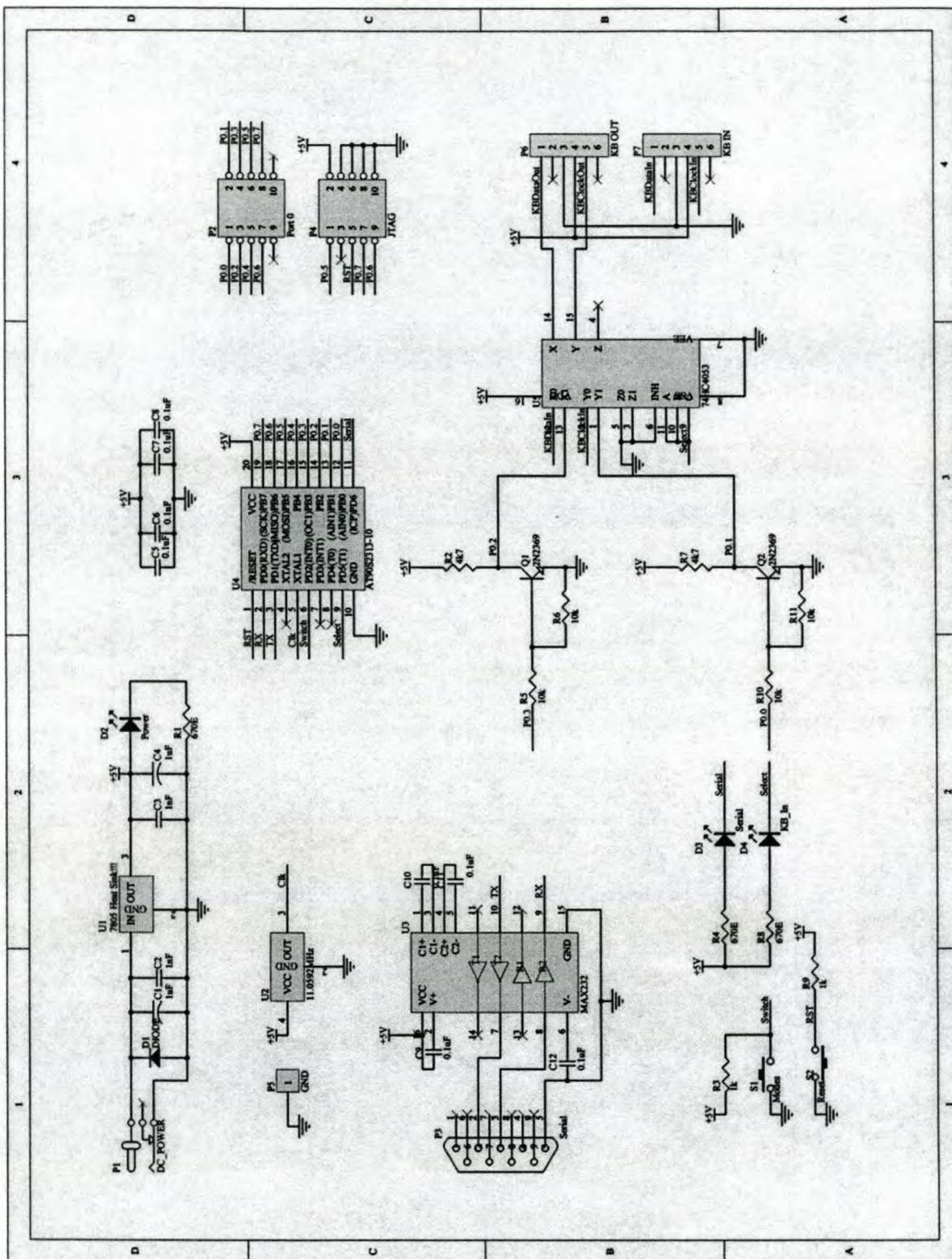


Figure 59: Schematic diagram

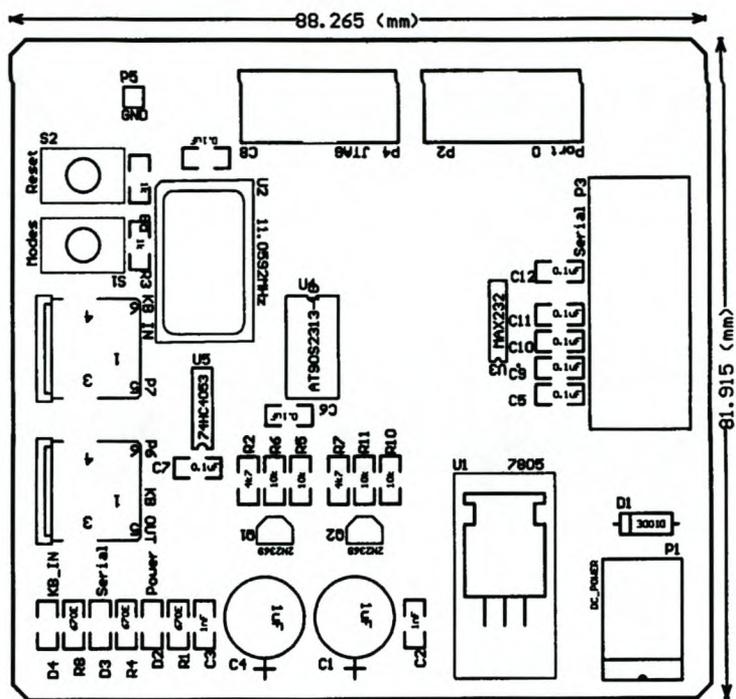


Figure 60: Component Placement Layer

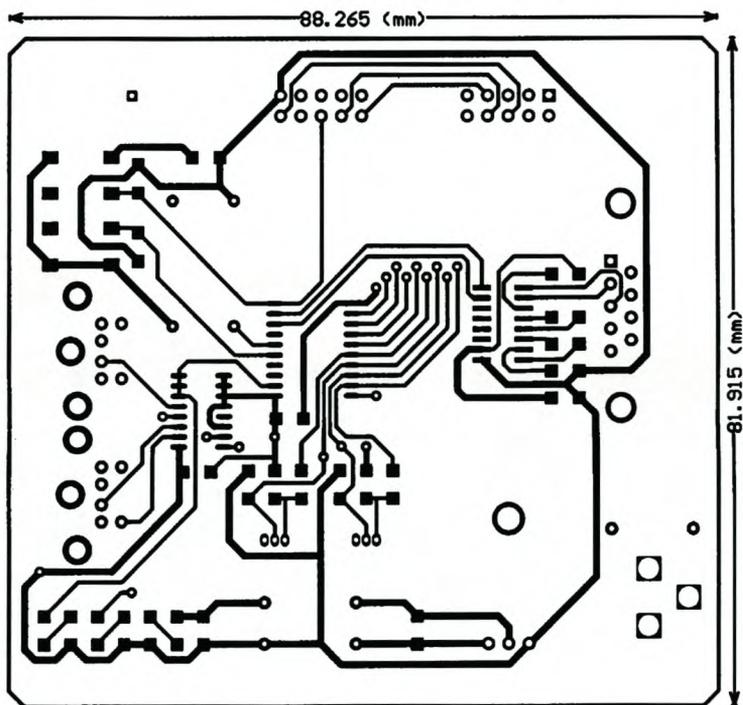


Figure 61: Solder Top Layer

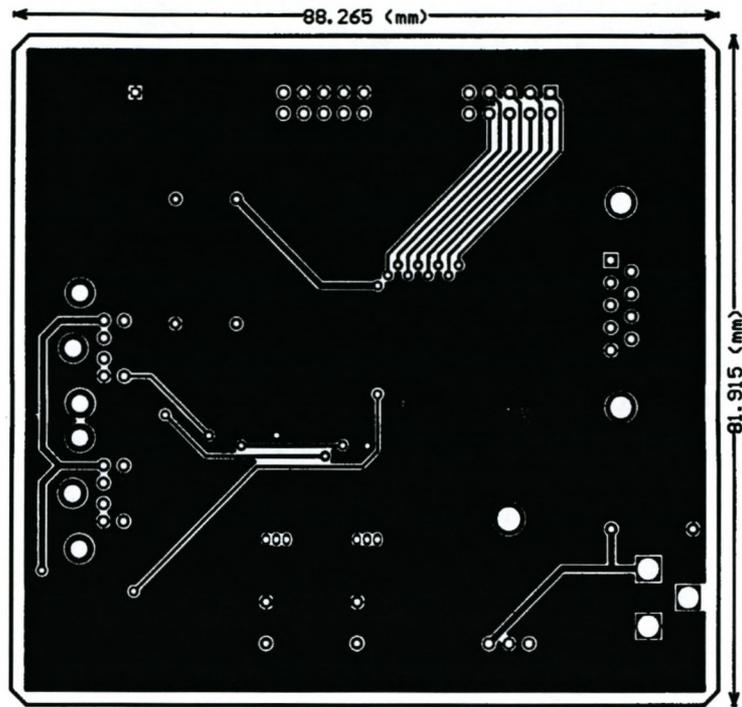


Figure 62: Solder Bottom Layer

## **Appendix D: SINav Interface Source Code:**

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

#include <vcl.h>
#include <time.h>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <dos.h>
#include <alloc.h>
#include <complex.h>
#include <algorith.h>
#include <ctype.h>
#include <EImage.h>
#include <Easy.h>
#include <EMultiCam.h>
#pragma hdrstop

#include "Main.h"
//-----
#pragma package(smart_init)
#pragma link "CSPIN"

#pragma link "MULTICAMLib_OCX"
#pragma link "PERFGRAP"
#pragma resource "*.dfm"
TSINav *SINav;

#define PI 3.14159265359
#define DegToRad 17.453293e-3
#define RadToDeg 57.29578

HANDLE hComm = NULL;
COMMTIMEOUTS ctmoNew = {0},ctmoOld;

bool CameraConnectStatus = false,PortConnectStatus = false;
bool SaveStatus = false;

char str[18];
char InBuff[22];
complex<double> **ImageData1, **ImageData2, **ImageData3, **ImageData4;

float v,w,x,y,z;
float ElapseTime,XRate,YRate,ZRate;
float dTheta,dLogR;
float SampleTime,FocalLength,PixelWidth,PixelHeight;
float RollAverage[6] = {0,0,0,0,0,0};
float PitchAverage[6] = {0,0,0,0,0,0};
FILE *DATAStream;

int Baudrates[13] = {110,300,1200,2400,4800,9600,19200,38400,57600,115200,
                    230400,460800,921600};
int Sizes[6] = {64,128,256,512,1024,2048};
int KBLookUp[128] = {0,0,0,0,0,0,0,0,102,13,90,0,0,90,0,0,
                    0,0,0,0,0,0,0,0,0,0,118,0,0,0,0,
                    41,22,82,38,37,46,61,82,70,69,124,121,16,123,73,74,
                    112,105,114,122,107,115,116,108,117,125,76,76,65,85,73,74,
                    30,28,50,33,35,36,43,52,51,67,59,66,75,58,49,68,
                    77,21,45,27,44,60,42,29,22,53,26,84,93,91,54,78,
                    14,28,50,33,35,36,43,52,51,67,59,66,75,58,49,68,
                    77,21,45,27,44,60,42,29,22,53,26,84,93,91,14,0};

int i,j,k,l;
int CommandTime = 0,DelayCounter = 0;
int XPixelOffset,YPixelOffset;
int XPosition,YPosition,RotationPosition;
int Rows,Cols,CenterRows,CenterCols;
int FrameWidth,FrameHeight;
int ImageXGain,ImageYGain,ImageXOffset,ImageYOffset;
int Counter = 0;
time_t secs_now;
tm *time_now;

//-----

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

__fastcall TSINav::TSINav(TComponent* Owner)
    : TForm(Owner)
{
}
//-----

void __fastcall TSINav::FormCreate(TObject *Sender)
{
  ESetAngleUnit(E_ANGLE_UNIT_DEGREES);
  EmcOpen();

  SINav->Left = 30;
  SINav->Top = 30;
  CommandHistory->Clear();

  for (i=0;i<=3;i++)
    VideoSource->Items->Add("Source "+IntToStr(i));
    VideoSource->ItemIndex = 0;

    VideoStandard->Items->Add("CCIR");
    VideoStandard->Items->Add("EIA");
    VideoStandard->Items->Add("PAL");
    VideoStandard->Items->Add("NTSC");
    VideoStandard->Items->Add("SECAM");
    VideoStandard->ItemIndex = 2;

  for (i=1;i<=4;i++)
    PortID->Items->Add("COM"+IntToStr(i));
    PortID->ItemIndex = 0;

  for (i=0;i<=11;i++)
    PortBaudrate->Items->Add(IntToStr(Baudrates[i]));
    PortBaudrate->ItemIndex = 5;

  for (i=0;i<=5;i++)
    ImageSizeBox->Items->Add(IntToStr(Sizes[i]));
    ImageSizeBox->ItemIndex = 2;

}
//-----

void __fastcall TSINav::SetupButtonClick(TObject *Sender)
{
  DATASETUP();
  PORTSETUP();
  CAMERASETUP();
  ImageXGain = floor(FrameWidth/Cols);
  ImageYGain = floor(FrameHeight/Rows);
  ImageXOffset = (FrameWidth - Cols*ImageXGain)/2;
  ImageYOffset = (FrameHeight - Rows*ImageYGain)/2;
  if (ImageXGain == 0 || ImageYGain == 0) {
    Application->MessageBoxA("Image size too Large","Error",MB_OK);
    SetupButton->Enabled = true;
    RunButton->Enabled = false;
    StopButton->Enabled = false;
  }
  else{
    SetupButton->Enabled = true;
    RunButton->Enabled = true;
    StopButton->Enabled = false;
    CommandHistory->Lines->Add("Setup Completed");
  }
}
//-----

void __fastcall TSINav::RunButtonClick(TObject *Sender)
{
  CommandTime=0;
  SetupButton->Enabled = false;
  RunButton->Enabled = false;
  StopButton->Enabled = true;
}

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

ExitButton->Enabled = false;

if (PortConnectStatus && CameraConnectStatus){
    ProcessStatus->Checked = true;
    ProcessStatus->Color = clLime;
    Timer1->Enabled = true;

    DATAstream = fopen("TeleData.xls", "w+");
    fprintf(DATAstream, "Date Time Sample Duration XRate YRate\n ");
    CommandHistory->Lines->Add("Process started");
}
else{
    ProcessStatus->Checked = false;
    ProcessStatus->Color = clRed;
    Timer1->Enabled = false;
}
}
//-----

void __fastcall TSINav::StopButtonClick(TObject *Sender)
{
    SetupButton->Enabled = true;
    RunButton->Enabled = true;
    StopButton->Enabled = false;
    ExitButton->Enabled = true;
    ProcessStatus->Checked = false;
    ProcessStatus->Color = clRed;
    Timer1->Enabled = false;
    if (TrackEnable->Checked)
        TransmitCommChar(hComm, char(1));
    TrackEnable->Checked = false;
    fclose(DATAstream);

    if (SaveImage->Checked) {
        m_Image.Save("c:/SINav_Img1.bmp");
    }

    CommandHistory->Lines->Add("Process Stopped");
}
//-----

void __fastcall TSINav::ExitButtonClick(TObject *Sender)
{
    if (TrackEnable->Checked) {
        TransmitCommChar(hComm, char(1));
        TrackEnable->Checked = false;
    }
    if (PortConnectStatus) {
        SetCommTimeouts(hComm, &ctmoOld);
        CloseHandle(hComm);
    }
    EmcClose();
}
//-----

void __fastcall TSINav::Timer1Timer(TObject *Sender)
{
    char Imgchar[20];

    EStartTiming();
    m_Image.Draw(Image2->Canvas->Handle, 0.5, 0.5);
    sprintf(Imgchar, "c:/Img%3i.bmp", Counter);
    m_Image.Save(Imgchar);
    Counter++;
    EBW8* pPixell;

    for(i=0; i<Rows; i++){
        pPixell = (EBW8*)m_Image.GetImagePtr(0, ImageYGain*i);
        for(j=0; j<Cols; j++){

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

        ImageData2[j][i] = complex<double>(pPixel1[ImageXGain*j+ImageXOffset],0);
    }
}

m_Image.Grab();
Invalidate();
m_Image.Draw(Image1->Canvas->Handle,0.5,0.5);
Image1->Refresh();
Image2->Refresh();

for(i=0;i<Rows;i++){
    pPixel1 = (EBW8*m_Image.GetImagePtr(0,ImageYGain*i);
    for(j=0;j<Cols;j++){
        ImageData1[j][i] = complex<double>(pPixel1[ImageXGain*j+ImageXOffset],0);
    }
}
IMAGEPROCESS();

PerformanceGraph1->DataPoint(clRed,50+10*XRate);
PerformanceGraph1->DataPoint(clBlue,50+10*YRate);
PerformanceGraph1->DataPoint(clWhite,50);
PerformanceGraph1->Update();

ElapseTime = EStopTiming(1000000);
TimeDisplay->Text = FloatToStrF(ElapseTime/1000,ffFixed,8,3);

time(&secs_now);
time_now = localtime(&secs_now);
strftime(str, 18, "%c",time_now);
fprintf(DATAStream, "%18s %7.3f %7.3f %7.3f %7.3f\n ",
str,SampleTime,ElapseTime/1000,XRate,YRate);
}
//-----

void __fastcall TSINav::VideoSourceChange(TObject *Sender)
{
if (StopButton->Enabled){
    Application->MessageBoxA("Stop process to enable changes","Warning!",MB_OK);
}
RunButton->Enabled = false;
if (PortConnectStatus){
    SetCommTimeouts(hComm,&ctmoOld);
    CloseHandle(hComm);
    PortConnect->Checked = false;
    PortConnect->Color = clRed;
    PortConnectStatus = false;
}
}
//-----

void __fastcall TSINav::VideoStandardChange(TObject *Sender)
{
if (StopButton->Enabled){
    Application->MessageBoxA("Stop process to enable changes","Warning!",MB_OK);
}
RunButton->Enabled = false;
if (PortConnectStatus){
    SetCommTimeouts(hComm,&ctmoOld);
    CloseHandle(hComm);
    PortConnect->Checked = false;
    PortConnect->Color = clRed;
    PortConnectStatus = false;
}
}
//-----

void __fastcall TSINav::PortIDChange(TObject *Sender)
{
if (StopButton->Enabled){
    Application->MessageBoxA("Stop process to enable changes","Warning!",MB_OK);
}
}

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

RunButton->Enabled = false;
if (PortConnectStatus){
    SetCommTimeouts(hComm,&ctmoOld);
    CloseHandle(hComm);
    PortConnect->Checked = false;
    PortConnect->Color = clRed;
    PortConnectStatus = false;
}
}
//-----

void __fastcall TSINav::PortBaudrateChange(TObject *Sender)
{
if (StopButton->Enabled){
    Application->MessageBoxA("Stop process to enable changes","Warning!",MB_OK);
}
RunButton->Enabled = false;
if (PortConnectStatus){
    SetCommTimeouts(hComm,&ctmoOld);
    CloseHandle(hComm);
    PortConnect->Checked = false;
    PortConnect->Color = clRed;
    PortConnectStatus = false;
}
}
//-----

void __fastcall TSINav::SampleTimeBoxChange(TObject *Sender)
{
Timer1->Interval = SampleTimeBox->Value;
SampleTime = 1e-3*float(SampleTimeBox->Value);
}
//-----

void __fastcall TSINav::PixelWidthBoxChange(TObject *Sender)
{
PixelWidth = 1e-6*float(PixelWidthBox->Value);
}
//-----

void __fastcall TSINav::PixelHeightBoxChange(TObject *Sender)
{
PixelHeight = 1e-6*float(PixelHeightBox->Value);
}
//-----

void __fastcall TSINav::ImageSizeBoxChange(TObject *Sender)
{
if (StopButton->Enabled){
    Application->MessageBoxA("Stop process to enable changes","Warning!",MB_OK);
}
RunButton->Enabled = false;
if (PortConnectStatus){
    SetCommTimeouts(hComm,&ctmoOld);
    CloseHandle(hComm);
    PortConnect->Checked = false;
    PortConnect->Color = clRed;
    PortConnectStatus = false;
}
}
//-----

void __fastcall TSINav::FocalLengthBoxChange(TObject *Sender)
{
FocalLength = 1e-3*float(FocalLengthBox->Value);
}
//-----

void __fastcall TSINav::SaveImageClick(TObject *Sender)
{
if (SaveImage->Checked)
    SaveImage->Color = clLime;
}
}

```

```

else
    SaveImage->Color = clRed;
}
//-----

void __fastcall TSINav::XPixelOffsetBoxChange(TObject *Sender)
{
XPixelOffset = XPixelOffsetBox->Value;
}
//-----

void __fastcall TSINav::YPixelOffsetBoxChange(TObject *Sender)
{
YPixelOffset = YPixelOffsetBox->Value;
}
//-----

void TSINav::DATASETUP(void)
{
Timer1->Interval = SampleTimeBox->Value;
SampleTime = 1e-3*float(SampleTimeBox->Value);
FocalLength = 1e-3*float(FocalLengthBox->Value);
PixelWidth = 1e-6*float(PixelWidthBox->Value);
PixelHeight = 1e-6*float(PixelHeightBox->Value);
Rows = StrToInt(ImageSizeBox->Text);
Cols = Rows;
CenterRows = (Rows+1)/2;
CenterCols = (Cols+1)/2;

ImageData1 = new complex<double>*[Rows];
ImageData2 = new complex<double>*[Rows];
ImageData3 = new complex<double>*[Rows];
for (i=0;i<Rows;i++){
    ImageData1[i] = new complex<double>[Cols];
    ImageData2[i] = new complex<double>[Cols];
    ImageData3[i] = new complex<double>[Cols];
}
}
//-----

void TSINav::PORTSETUP(void)
{
char PortSpec[14], PortName[4];

sprintf(PortSpec, "%s,N,8,1", PortBaudrate->Text);
sprintf(PortName, "%s", PortID->Text);

if(PortConnectStatus){
    SetCommTimeouts(hComm, &ctmoOld);
    CloseHandle(hComm);
}

DCB dcbCommPort;

hComm = CreateFile(PortName,
    GENERIC_READ | GENERIC_WRITE,
    0,
    0,
    OPEN_EXISTING,
    0,
    0);

if(hComm == INVALID_HANDLE_VALUE){
    PortConnect->Checked = false;
    PortConnect->Color = clRed;
    PortConnectStatus = false;
}
else{
    GetCommTimeouts(hComm, &ctmoOld);
    ctmoNew.ReadTotalTimeoutConstant = 20;
    ctmoNew.ReadTotalTimeoutMultiplier = 0;
    ctmoNew.WriteTotalTimeoutMultiplier = 0;
    ctmoNew.WriteTotalTimeoutConstant = 0;
}
}

```

```

    SetCommTimeouts (hComm, &ctmoNew);

    dcbCommPort.DCBLength = sizeof(DCB);
    GetCommState (hComm, &dcbCommPort);
    BuildCommDCB (PortSpec, &dcbCommPort);
    SetCommState (hComm, &dcbCommPort);

    PortConnect->Checked = true;
    PortConnect->Color = clLime;
    PortConnectStatus = true;
}
}
//-----
void TSINav::CAMERASETUP (void)
{
    EImageBW8 BW8Image1;
    EMCImageBW8 EMCBW8Image;

    EMCBoard* pEMCBoard = NULL;
    EmcGetBoardByIndex (0, pEMCBoard);
    pEMCBoard->SetParam (MC_BoardTopology, MC_BoardTopology_1_1);

    switch (VideoStandard->ItemIndex) {
        case 0:
            m_Image.AssignSourceByIndex ("CCIR", "PICOLO_0", 0);
            break;
        case 1:
            m_Image.AssignSourceByIndex ("EIA", "PICOLO_0", 0);
            break;
        case 2:
            m_Image.AssignSourceByIndex ("PAL", "PICOLO_0", 0);
            break;
        case 3:
            m_Image.AssignSourceByIndex ("NTSC", "PICOLO_0", 0);
            break;
        case 4:
            m_Image.AssignSourceByIndex ("SECAM", "PICOLO_0", 0);
            break;
    }
    m_Image.SetParam (EC_PARAM_VideoGain, 65536 * (CameraColorGain->Position) / 100);
    m_Image.SetParam (EC_PARAM_VideoOffset, 65536 * (CameraColorOffset->Position) / 1000);

    FrameWidth = m_Image.GetWidth();
    FrameHeight = m_Image.GetHeight();
    char BoardImageSize [20];

    if (m_Image.IsAssigned()) {
        CameraConnect->Checked = true;
        CameraConnect->Color = clLime;
        CameraConnectStatus = true;
        sprintf (BoardImageSize, "Image Width = %4i", FrameWidth);
        CommandHistory->Lines->Add (BoardImageSize);
        sprintf (BoardImageSize, "Image Height = %4i", FrameHeight);
        CommandHistory->Lines->Add (BoardImageSize);
    }
    else {
        CameraConnect->Checked = false;
        CameraConnect->Color = clRed;
        CameraConnectStatus = false;
    }
}
//-----
void TSINav::IMAGEPROCESS (void)
{
    FFT2D (ImageData1, Cols, Rows, 1);
    DCChange (ImageData1, Cols, Rows);
    Change180 (ImageData2, Cols, Rows);
    FFT2D (ImageData2, Cols, Rows, 1);
    DCChange (ImageData2, Cols, Rows);
    for (i=0; i<Cols; i++) {

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

        for (j=0;j<Rows;j++){
            v = real(ImageData1[i][j])*real(ImageData2[i][j]);
            w = imag(ImageData1[i][j])*imag(ImageData2[i][j]);
            x = real(ImageData1[i][j])*imag(ImageData2[i][j]);
            y = imag(ImageData1[i][j])*real(ImageData2[i][j]);
            ImageData3[i][j] = complex<double>(v-w,x+y);
        }
    }
    FFT2D(ImageData3, Cols, Rows, -1);
    DCChange(ImageData3, Cols, Rows);
    v = 0;
    for (i = 0; i < Cols; i++){
        for (j = 0; j < Rows; j++){
            if (abs(ImageData3[i][j])>v){
                XPosition = i;
                YPosition = j;
                v = abs(ImageData3[i][j]);
            }
        }
    }
    XRate = -RadToDeg*(atan((ImageXGain*(XPosition-
    CenterCols+XPixelOffset))*PixelWidth/FocalLength))/SampleTime;
    YRate = -RadToDeg*(atan((ImageYGain*(YPosition-
    CenterRows+YPixelOffset))*PixelHeight/FocalLength))/SampleTime;
    RollRate->Text = FloatToStrF(XRate, ffFixed, 9, 3);
    PitchRate->Text = FloatToStrF(YRate, ffFixed, 9, 3);

}
//-----
int TSINav::FFT2D(complex<double> **c,int nx,int ny,int dir)
{
    int m,twopm;
    double *Real, *Imag;

    Real = (double *)malloc(nx * sizeof(double));
    Imag = (double *)malloc(nx * sizeof(double));
    if (Real == NULL || Imag == NULL)
        return(FALSE);
    if (!Powerof2(nx, &m, &twopm) || twopm != nx)
        return(FALSE);
    for (j=0;j<ny;j++){
        for (i=0;i<nx;i++){
            Real[i] = real(c[i][j]);
            Imag[i] = imag(c[i][j]);
        }
        FFT(1,m,Real,Imag);
        for (i=0;i<nx;i++)
            c[i][j] = complex<double>(Real[i],Imag[i]);
    }
    free(Real);
    free(Imag);

    Real = (double *)malloc(ny * sizeof(double));
    Imag = (double *)malloc(ny * sizeof(double));
    if (Real == NULL || Imag == NULL)
        return(FALSE);
    if (!Powerof2(ny, &m, &twopm) || twopm != ny)
        return(FALSE);
    for (i=0;i<nx;i++){
        for (j=0;j<ny;j++){
            Real[j] = real(c[i][j]);
            Imag[j] = imag(c[i][j]);
        }
        FFT(1,m,Real,Imag);
        for (j=0;j<ny;j++)
            c[i][j] = complex<double>(Real[j],Imag[j]);
    }
    free(Real);
    free(Imag);

    return(TRUE);
}

```

```

}
//-----
int TSINav::FFT(int dir, int m, double *x, double *y)
{
long nn,i,i1,i2,j,k,l,l1,l2;
double c1,c2,tx,ty,t1,t2,u1,u2,z;

nn = 1;
for (i = 0;i < m;i++)
    nn *= 2;
// Bit reversal //
i2 = nn >> 1;
j = 0;
for (i = 0;i < (nn-1);i++){
    if (i < j){
        tx = x[i];
        ty = y[i];
        x[i] = x[j];
        y[i] = y[j];
        x[j] = tx;
        y[j] = ty;
    }

    k = i2;
    while (k <= j){
        j -= k;
        k >>= 1;
    }

    j += k;
}
// Start FFT process //
c1 = -1.0;
c2 = 0.0;
l2 = 1;
for (l = 0;l < m;l++){
    l1 = l2;
    l2 <<= 1;
    u1 = 1.0;
    u2 = 0.0;
    for (j = 0;j < l1;j++){
        for (i = j;i < nn;i+=l2){
            i1 = i + l1;
            t1 = u1*x[i1] - u2*y[i1];
            t2 = u1*y[i1] + u2*x[i1];
            x[i1] = x[i] - t1;
            y[i1] = y[i] - t2;
            x[i] += t1;
            y[i] += t2;
        }

        z = u1*c1 - u2*c2;
        u2 = u1*c2 + u2*c1;
        u1 = z;
    }

    c2 = sqrt((1.0 - c1)/2);
    if (dir == 1)
        c2 = -c2;
    c1 = sqrt((1.0 + c1)/2);
}

if (dir == 1){
    for (i = 0;i < nn;i++){
        x[i] /= (double)nn;
        y[i] /= (double)nn;
    }
}

return(TRUE);
}
//-----
int TSINav::Powerof2(int n,int *m,int *twopm)
{

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

```

if (n <= 1){
    *m = 0;
    *twopm = 1;
    return(FALSE);
}
*m = 1;
*twopm = 2;
do {
    (*m)++;
    (*twopm) *= 2;
} while (2*(twopm) <= n);

if (*twopm != n)
    return(FALSE);
else
    return(TRUE);

}
//-----
int TSINav::DCChange(complex<double> **c,int nx,int ny)
{
for (i = 0;i < nx;i++){
    for (j = 0;j < ny/2;j++){
        if (i < nx/2)
            swap(c[i][j],c[i+nx/2][j+ny/2]);
        else
            swap(c[i][j],c[i-nx/2][j+ny/2]);
    }
}

return(TRUE);
}
//-----
int TSINav::Change180(complex<double> **c,int nx,int ny)
{
for(i=0;i<nx;i++){
    for(j=0;j<ny/2;j++){
        swap(c[i][j],c[nx-i-1][ny-j-1]);
    }
}

return(TRUE);
}
//-----
void TSINav::CONTROLCOMMANDS(void)
{
//for(i=0;i<4;i++){
//    RollAverage[i]=RollAverage[i+1];
//    PitchAverage[i]=PitchAverage[i+1];
//}
//RollAverage[5]=0;
//PitchAverage[5]=0;
//for(i=0;i<5;i++){
//    RollAverage[5]+=RollAverage[i];
//    PitchAverage[5]+=PitchAverage[i];
//}
//RollAverage[5] /= 5;
//PitchAverage[5] /= 5;
}
//-----

void __fastcall TSINav::ResetButtClick(TObject *Sender)
{
for (i=0;i<6;i++){
    RollAverage[i]=0;
    PitchAverage[i]=0;
}
}
//-----

void __fastcall TSINav::TrackEnableClick(TObject *Sender)

```

```

{
TransmitCommChar(hComm, char(1));
if (TrackEnable->Checked){
    CommandHistory->Lines->Add("Start Tracking!!!");
    Timer2->Enabled = true;
}
else{
    CommandHistory->Lines->Add("Stop Tracking!!!");
    Timer2->Enabled = false;
}
}
//-----

void __fastcall TSINav::Timer2Timer(TObject *Sender)
{
float ROLLRATE, PITCHRATE;
ROLLRATE = -XRate;
PITCHRATE = -YRate;
sprintf(InBuff, "%6.3f%c%c%c%c%6.3f%c%c%c%c%c%c",
ROLLRATE, 13, 9, 9, 9, PITCHRATE, 13, 9, 9, 9, 9);
i = 0;
Timer3->Enabled = true;
}
//-----

void __fastcall TSINav::Timer3Timer(TObject *Sender)
{
DelayCounter++;
if (DelayCounter == 5){
    if (i <= 21){
        TransmitCommChar(hComm, char(KBLookUp[char(InBuff[i])]));
    }
    else{
        Timer3->Enabled = false;
        i = 0;
    }
    i++;
    DelayCounter = 0;
}
}
//-----

void __fastcall TSINav::CameraColorGainChange(TObject *Sender)
{
m_Image.SetParam(EC_PARAM_VideoGain, 65536*(CameraColorGain->Position)/100);
}
//-----

void __fastcall TSINav::CameraColorOffsetChange(TObject *Sender)
{
m_Image.SetParam(EC_PARAM_VideoOffset, 65536*(CameraColorOffset->Position)/1000);
}
//-----

```

## **Appendix E: Log-files of DLR-TUBSAT Overhead Pass**

**DLR-TUBSAT Control Software Log-file**

```

13:16:26 >Get Electrical Telemetry
Solar +X (Panel 2): 183.38 mA $003B
Solar +Y (Panel 4): 423.94 mA $0121
Solar -Y (Panel 1): 118.36 mA $000D
Solar -Z (Panel 3): 678.59 mA $012C
Solar +Z ( 27-4 ) : 254.65 mA $00E2
Solar +X (Panel 2): 12.52 V $01FE
Solar +Y (Panel 4): 12.57 V $0200
Solar -Y (Panel 1): 12.50 V $01FD
Solar -Z (Panel 3): 12.67 V $0204
Solar +Z ( 27-4 ) : 1.14 V $00E2
Temperature +X : 4.64 °C $0238
Temperature +Y : 1.71 °C $0232
Temperature +Z : 1.71 °C $0232
Temperature -X : 3.17 °C $0235
Temperature -Y : 2.19 °C $0233
Temperature -Z : 3.17 °C $0235
Batterie Bus : 12.06 V $02B2
Power Bus : 185.15 mA $0034
US Power Bus : 111.32 mA $0075
Wheel Z (1) : 0.00 mA $0005
Wheel Y (2) : 0.00 mA $0005
Wheel X (3) : 0.00 mA $0006
5V Bus/Gyro : 0.00 mA $0005
Spule/ TTC2 : 42.28 mA $0079
S-Band/Torq : 0.00 mA $0005
Camera : 0.00 mA $0005

```

```

-----
13:16:28 >Pic-Mode ON
13:16:30 >Start -Z To Sun
13:19:34 >Switch Head To 1000mm
13:19:47 >Z Angle +90°
13:20:12 >Switch Head To 50mm
13:20:22 >Switch Head To 1000mm
13:20:35 >Set Omega Y To 0.460°/s
13:20:42 >Set Omega Y To 0.490°/s
13:20:46 >Set Omega Y To 0.520°/s
13:20:50 >Set Omega X To -0.030°/s
13:20:54 >Set Omega Y To 0.550°/s
13:20:58 >Set Omega Y To 0.490°/s
13:21:08 >Set Omega Y To 0.430°/s
13:21:27 >Set Omega Y To 0.370°/s
13:21:34 >Set Omega Y To 0.310°/s
13:21:41 >Set Omega Y To 0.250°/s
13:21:53 >Set Omega X To 0.000°/s
13:21:56 >Set Omega X To -0.030°/s
13:22:02 >Set Omega Y To 0.190°/s
13:22:17 >Set Omega Y To 0.130°/s
13:22:45 >Set Omega Y To 0.460°/s
13:22:47 >Set Omega Y To 0.580°/s
13:22:50 >Set Omega Y To 0.490°/s
13:22:53 >Set Omega Y To 0.460°/s
13:22:56 >Set Omega Y To 0.400°/s
13:22:58 >Set Omega X To 0.030°/s
13:23:01 >Set Omega Y To 0.340°/s
13:23:03 >Set Omega X To 0.090°/s
13:23:06 >Set Omega Y To 0.280°/s
13:23:12 >Set Omega X To 0.150°/s
13:23:22 >Set Omega Y To 0.220°/s
13:23:24 >Set Omega Y To 0.220°/s
13:23:28 >Set Omega X To 0.060°/s
13:23:31 >Set Omega X To 0.000°/s
13:23:35 >Set Omega X To -0.060°/s
13:23:38 >Set Omega X To -0.030°/s
13:23:42 >Set Omega Y To 0.280°/s
13:23:44 >Set Omega X To -0.090°/s

```

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

13:23:47 >Set Omega X To 0.030°/s  
 13:23:53 >Set Omega X To 0.090°/s  
 13:23:55 >Set Omega Y To 0.220°/s  
 13:23:59 >Set Omega X To 0.030°/s  
 13:24:02 >Set Omega X To -0.030°/s  
 13:24:06 >Set Omega X To 0.060°/s  
 13:24:12 >Set Omega Y To 0.190°/s  
 13:24:16 >Set Omega Y To 0.160°/s  
 13:24:21 >Set Omega Y To 0.130°/s  
 13:24:25 >Set Omega Y To 0.100°/s  
 13:24:29 >Set Omega X To 0.120°/s  
 13:24:32 >Set Omega Y To 0.160°/s  
 13:24:35 >Set Omega X To 0.060°/s  
 13:24:40 >Set Omega X To 0.000°/s  
 13:24:45 >Set Omega Y To 0.250°/s  
 13:24:49 >Set Omega X To 0.060°/s  
 13:24:51 >Set Omega Y To 0.160°/s  
 13:24:54 >Set Omega Y To 0.100°/s  
 13:24:58 >Set Omega X To -0.030°/s  
 13:25:03 >Set Omega X To 0.000°/s  
 13:25:06 >Set Omega X To 0.030°/s  
 13:25:09 >Set Omega Y To 0.040°/s  
 13:25:16 >Set Omega Y To 0.100°/s  
 13:25:19 >Set Omega X To -0.030°/s  
 13:25:27 >Set Omega X To 0.030°/s  
 13:25:34 >Set Omega Y To 0.040°/s  
 13:25:38 >Set Omega Y To 0.100°/s  
 13:25:48 >Set Omega Y To 0.070°/s  
 13:25:52 >Set Omega X To 0.000°/s  
 13:25:54 >Set Omega Y To 0.010°/s  
 13:25:58 >Set Omega Y To 0.070°/s  
 13:26:01 >Set Omega Y To 0.130°/s  
 13:26:06 >Set Omega X To 0.060°/s  
 13:26:11 >Set Omega Y To 0.070°/s  
 13:26:21 >Set Omega X To 0.000°/s  
 13:26:25 >Set Omega X To -0.060°/s  
 13:26:30 >Set Omega Y To 0.010°/s  
 13:26:33 >Switch Head To 50mm  
 13:26:43 >Set Omega Y To 0.210°/s  
 13:26:46 >Set Omega Y To 0.410°/s  
 13:26:51 >Switch Head To 1000mm  
 13:26:55 >Set Omega Y To 0.350°/s  
 13:26:58 >Switch Head To 50mm  
 13:27:05 >Pic-Mode OFF

**SINav Measuring Software Log-file**

Date	Time	Sample	Duration	XRate	YRate
01/23/03	13:24:24	0.35	305.17	0.059	0.036
01/23/03	13:24:24	0.35	302.507	0.088	0.068
01/23/03	13:24:25	0.35	304.087	0.081	0.094
01/23/03	13:24:25	0.35	285.416	0.066	0.083
01/23/03	13:24:25	0.35	285.135	0.074	0.084
01/23/03	13:24:26	0.35	286.209	0.066	0.069
01/23/03	13:24:26	0.35	286.018	0.066	0.054
01/23/03	13:24:26	0.35	289.243	0.088	0.039
01/23/03	13:24:27	0.35	287.917	0.088	0.093
01/23/03	13:24:27	0.35	286.564	0.081	0.133
01/23/03	13:24:27	0.35	288.861	0.096	0.093
01/23/03	13:24:28	0.35	292.962	0.103	0.039

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

01/23/03	13:24:28	0.35	289.59	0.074	0.059
01/23/03	13:24:28	0.35	289.671	0.059	0.079
01/23/03	13:24:29	0.35	293.89	0.066	0.079
01/23/03	13:24:29	0.35	292.087	0.096	0.064
01/23/03	13:24:30	0.35	292.126	0.096	0.01
01/23/03	13:24:30	0.35	293.873	0.088	0
01/23/03	13:24:30	0.35	294.069	0.059	0
01/23/03	13:24:31	0.35	295.08	0.066	0.039
01/23/03	13:24:31	0.35	294.696	0.088	0.034
01/23/03	13:24:31	0.35	295.27	0.081	0.069
01/23/03	13:24:32	0.35	298.34	0.096	0.093
01/23/03	13:24:32	0.35	297.017	0.11	0.049
01/23/03	13:24:32	0.35	298.829	0.096	0.079
01/23/03	13:24:33	0.35	299.593	0.096	0.074
01/23/03	13:24:33	0.35	299.515	0.081	0.064
01/23/03	13:24:34	0.35	300.787	0.066	0.098
01/23/03	13:24:34	0.35	300.801	0.074	0.093
01/23/03	13:24:34	0.35	299.665	0.059	0
01/23/03	13:24:35	0.35	301.731	0.052	0
01/23/03	13:24:35	0.35	301.745	0.052	-0.069
01/23/03	13:24:35	0.35	303.07	0.066	-0.054
01/23/03	13:24:36	0.35	304.311	0.074	0
01/23/03	13:24:36	0.35	304.152	0.081	0
01/23/03	13:24:36	0.35	285.196	0.081	-0.034
01/23/03	13:24:37	0.35	286.602	0.081	0
01/23/03	13:24:37	0.35	284.756	0.074	0.054
01/23/03	13:24:37	0.35	286.891	0.088	0.015
01/23/03	13:24:38	0.35	288.983	0.11	-0.02
01/23/03	13:24:38	0.35	287.48	0.096	-0.025
01/23/03	13:24:39	0.35	288.604	0.081	-0.029
01/23/03	13:24:39	0.35	288.346	0.081	-0.044
01/23/03	13:24:39	0.35	288.493	0.088	-0.064
01/23/03	13:24:40	0.35	290.65	0.088	-0.049
01/23/03	13:24:40	0.35	291.071	0.081	-0.015
01/23/03	13:24:40	0.35	291.363	0.074	-0.02
01/23/03	13:24:41	0.35	294.275	0.074	-0.02
01/23/03	13:24:41	0.35	291.625	0.074	-0.02
01/23/03	13:24:41	0.35	295.483	0.074	0
01/23/03	13:24:42	0.35	296.14	0.081	-0.039
01/23/03	13:24:42	0.35	293.331	0.066	0
01/23/03	13:24:42	0.35	296.281	0.074	-0.049
01/23/03	13:24:43	0.35	297.807	0.088	-0.069
01/23/03	13:24:43	0.35	296.474	0.088	-0.064
01/23/03	13:24:44	0.35	297.004	0.074	-0.064
01/23/03	13:24:44	0.35	298.711	0.066	-0.079
01/23/03	13:24:44	0.35	298.275	0.059	-0.108

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

01/23/03	13:24:45	0.35	300.413	0.096	-0.138
01/23/03	13:24:45	0.35	298.667	0.081	-0.098
01/23/03	13:24:45	0.35	300.402	0.074	-0.064
01/23/03	13:24:46	0.35	302.976	0.074	-0.054
01/23/03	13:24:46	0.35	301.245	0.081	-0.054
01/23/03	13:24:46	0.35	301.027	0.096	-0.025
01/23/03	13:24:47	0.35	303.875	0.14	-0.049
01/23/03	13:24:47	0.35	302.517	0.169	-0.098
01/23/03	13:24:48	0.35	284.951	0	-0.103
01/23/03	13:24:48	0.35	285.683	0	-0.074
01/23/03	13:24:48	0.35	284.256	0.891	0
01/23/03	13:24:49	0.35	285.63	0	0
01/23/03	13:24:49	0.35	288.269	0	0
01/23/03	13:24:49	0.35	287.604	0.14	0
01/23/03	13:24:50	0.35	288.943	0.309	0
01/23/03	13:24:50	0.35	288.853	0.199	0.025
01/23/03	13:24:50	0.35	289.036	0.206	0.01
01/23/03	13:24:51	0.35	292.282	0.199	0
01/23/03	13:24:51	0.35	289.25	0.206	0.015
01/23/03	13:24:51	0.35	291.299	0.214	0.039
01/23/03	13:24:52	0.35	294.118	0.199	0.054
01/23/03	13:24:52	0.35	294.181	0.184	0.039
01/23/03	13:24:53	0.35	293.619	0.14	0.098
01/23/03	13:24:53	0.35	294.751	0.11	0.093
01/23/03	13:24:53	0.35	294.566	0.103	0.039
01/23/03	13:24:54	0.35	296.704	0.088	0.015
01/23/03	13:24:54	0.35	297.143	0.074	0.029
01/23/03	13:24:54	0.35	297.01	0.081	0.039
01/23/03	13:24:55	0.35	299.453	0.081	0.039
01/23/03	13:24:55	0.35	298.038	0.088	0.02
01/23/03	13:24:55	0.35	298.006	0.074	0
01/23/03	13:24:56	0.35	301.089	0.066	0
01/23/03	13:24:56	0.35	302.049	0.074	-0.015
01/23/03	13:24:57	0.35	300.949	0.088	-0.025
01/23/03	13:24:57	0.35	301.245	0.081	0
01/23/03	13:24:57	0.35	301.812	0.022	0.044
01/23/03	13:24:58	0.35	302.65	0	0.054
01/23/03	13:24:58	0.35	302.084	-0.037	0.093
01/23/03	13:24:58	0.35	302.401	-0.044	0.049
01/23/03	13:24:59	0.35	283.719	-0.029	-0.01
01/23/03	13:24:59	0.35	304.509	-0.007	-0.01
01/23/03	13:24:59	0.35	284.336	-0.022	0
01/23/03	13:25:00	0.35	287.707	-0.007	0.015
01/23/03	13:25:00	0.35	285.851	-0.015	0.039
01/23/03	13:25:00	0.35	287.021	-0.037	0.039
01/23/03	13:25:01	0.35	288.773	-0.029	0.02

## Processing of Onboard Images to Assist Automatic Forward Motion Compensation for Micro-Satellites

01/23/03	13:25:01	0.35	291.606	-0.015	0
01/23/03	13:25:02	0.35	289.632	0	0.049
01/23/03	13:25:02	0.35	289.849	0	0.054
01/23/03	13:25:02	0.35	290.241	-0.029	0.044
01/23/03	13:25:03	0.35	291.355	-0.052	0.128
01/23/03	13:25:03	0.35	311.694	0.678	0.133
01/23/03	13:25:03	0.35	292.581	-0.022	0.226
01/23/03	13:25:04	0.35	295.08	0	0.192
01/23/03	13:25:04	0.35	292.439	0	0.167
01/23/03	13:25:04	0.35	294.063	-0.015	0.182
01/23/03	13:25:05	0.35	298.486	-0.007	0.192
01/23/03	13:25:05	0.35	295.373	-0.022	0.236
01/23/03	13:25:05	0.35	296.492	-0.015	0.275
01/23/03	13:25:06	0.35	299.093	0	0.246
01/23/03	13:25:06	0.35	301.198	0	0.216
01/23/03	13:25:07	0.35	297.144	0.052	0.206
01/23/03	13:25:07	0.35	299.268	0.088	0.201
01/23/03	13:25:07	0.35	299.727	0.081	0.192
01/23/03	13:25:08	0.35	300.841	0.096	0.236
01/23/03	13:25:08	0.35	300.811	0.096	0.246
01/23/03	13:25:08	0.35	301.956	0.066	0.236
01/23/03	13:25:09	0.35	304.042	0.074	0.152
01/23/03	13:25:09	0.35	301.365	0.088	0.098
01/23/03	13:25:09	0.35	303.023	0.059	0.074
01/23/03	13:25:10	0.35	305.539	0.074	0.044
01/23/03	13:25:10	0.35	303.946	0.088	0.025
01/23/03	13:25:11	0.35	284.799	0.081	0.02
01/23/03	13:25:11	0.35	285.704	0.088	0.054
01/23/03	13:25:11	0.35	289.558	0.081	0.098
01/23/03	13:25:12	0.35	286.618	0.096	0.069
01/23/03	13:25:12	0.35	288.615	0.103	0
01/23/03	13:25:12	0.35	287.052	0.118	-0.025
01/23/03	13:25:13	0.35	290.436	0.133	-0.029
01/23/03	13:25:13	0.35	289.271	0.118	-0.044
01/23/03	13:25:13	0.35	289.927	0.088	-0.044

## **Appendix F: Radix-2 Fast Fourier Transform**

Source: [www.isip.msstate.edu/publications/courses/ece\\_4773/lectures/v1.0](http://www.isip.msstate.edu/publications/courses/ece_4773/lectures/v1.0)

### The Radix-2 Fast Fourier Transform

Under the constraint  $N = r^v$ , we have additional symmetry. Let us consider the case  $r = 2$ , and reexamine the divide and conquer algorithm with  $M = N/2$  and  $L = 2$ .

Define:

$$\begin{aligned} f_1(n) &= x(2n) \\ f_2(n) &= x(2n+1) \end{aligned}$$

This is referred to as a decimation-in-time approach. Why?

Let us derive a simplified expression for  $X(k)$ :

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n)W_N^{kn} \\ &= \sum_{n \text{ even}} x(n)W_N^{kn} + \sum_{n \text{ odd}} x(n)W_N^{kn} \\ &= \sum_{m=0}^{(N/2)-1} x(2m)W_N^{2mk} + \sum_{m=0}^{(N/2)-1} x(2m+1)W_N^{k(2m+1)} \end{aligned}$$

But,  $W_N^2 = W_{N/2}$ :

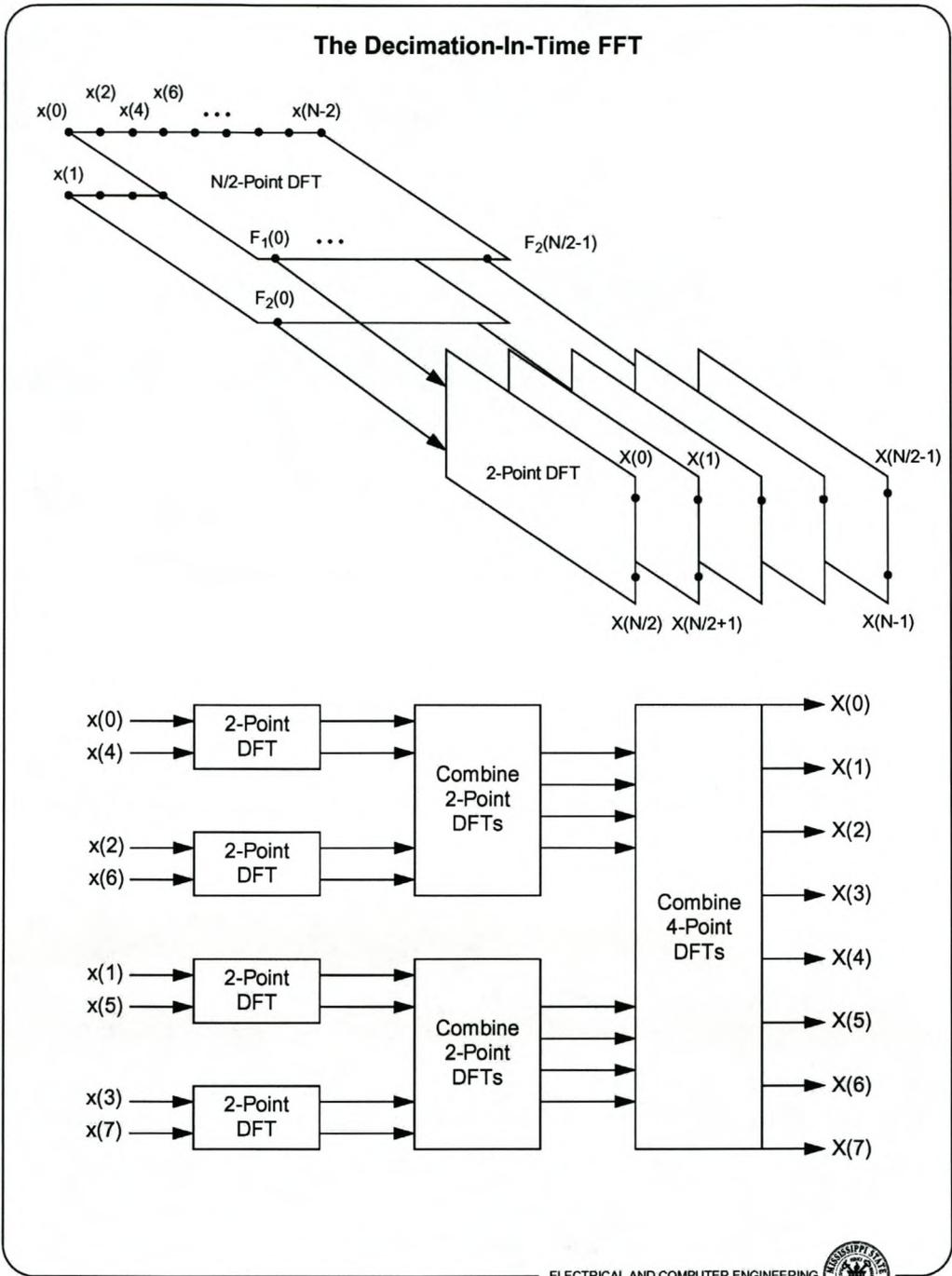
$$X(k) = \sum_{m=0}^{(N/2)-1} f_1(m)W_{N/2}^{km} + W_N^k \sum_{m=0}^{(N/2)-1} f_2(m)W_{N/2}^{km}$$

Note that  $F_1(k)$  and  $F_2(k)$  are  $N/2$ -point DFTs. This implies a reduction of a factor of 2 for large  $N$ . We can repeat the process by reducing  $F_1(k)$  (and  $F_2(k)$ ) from  $N/2$  to  $N/4$ -point transforms, and so on. This gives a complexity of  $O(N \log N)$  as opposed to  $O(N^2)$ .

This approach gives rise to a family of algorithms known as the Radix-2 Fast Fourier Transform. It has a simple graphical interpretation.

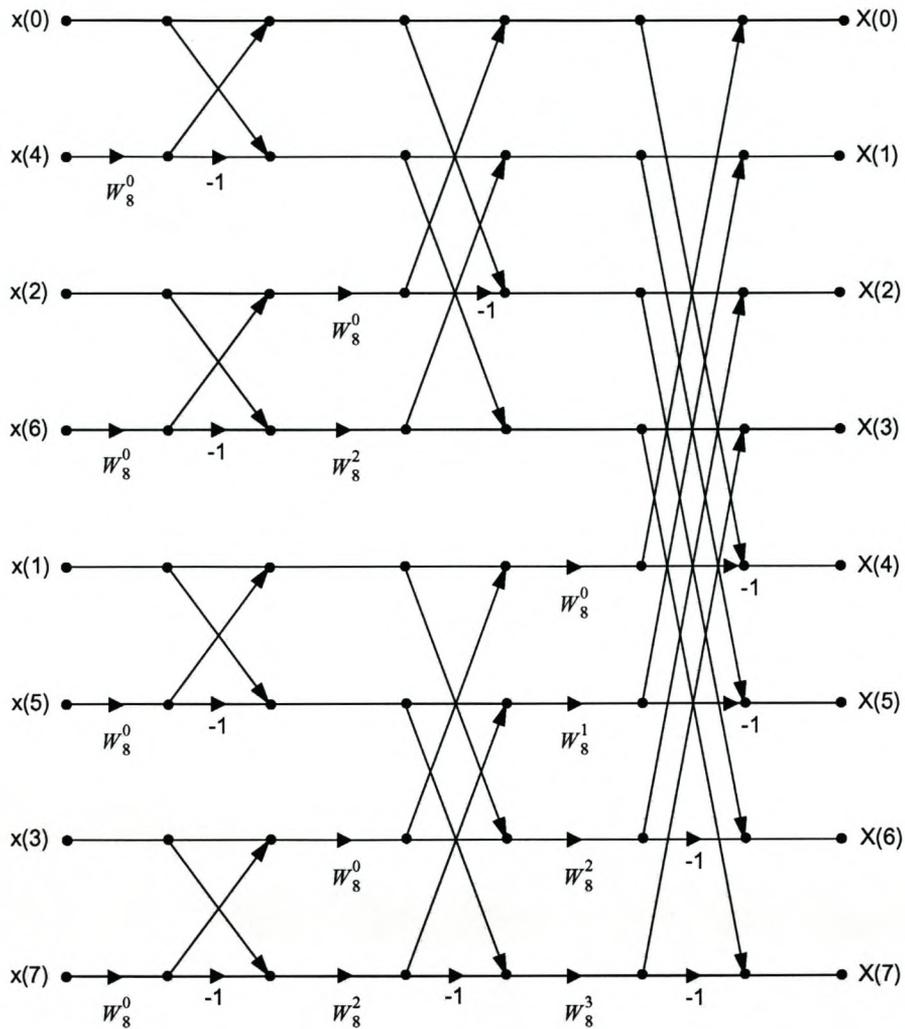
We can also do decimation-in-frequency.





### Signal Flow Graphs For Decimation-In-Time FFTs (Butterfly Diagrams)

An implementation of an 8-point DFT:



**DFT of Two Real Sequences**

Consider:

$$x(n) = x_1(n) + jx_2(n)$$

Because the DFT is a linear operator:

$$X(k) = X_1(k) + jX_2(k)$$

But,

$$x_1(n) = [x(n) + x^*(n)]/2$$

$$x_2(n) = [x(n) - x^*(n)]/2j$$

Therefore,

$$X_1(k) = \frac{1}{2}\{DFT[x(n)] + DFT[x^*(n)]\}$$

$$X_2(k) = \frac{1}{2j}\{DFT[x(n)] - DFT[x^*(n)]\}$$

But,  $DFT[x^*(n)] = X^*(N-k)$ , which implies:

$$X_1(k) = \frac{1}{2}[X(k) + X^*(N-k)]$$

$$X_2(k) = \frac{1}{2j}[X(k) - X^*(N-k)]$$

Strategy:

- (1) Compute the FFT of  $x(n)$ .
- (2) Use simple math to split the result apart.



**DFT of a 2N-Point Real Sequence**

Consider  $g(n)$  as a 2N-point real signal. Let us define two signals:

$$x_1(n) = g(2n)$$

$$x_2(n) = g(2n + 1)$$

and, of course,

$$x(n) = x_1(n) + jx_2(n)$$

We can show:

$$G(k) = X_1(k) + W_{2N}^k X_2(k)$$

$$G(k + N) = X_1(k) - W_{2N}^k X_2(k)$$

Strategy:

- (1) Form  $x_1(n)$  and  $x_2(n)$ .
- (2) Take N-point DFTs.
- (3) Combine

Note the reduction from length 2N to length N in complexity.



### Quantization Properties

One of the reasons the DFT/FFT is so popular is that it is very robust to quantization noise (because it is a linear operator).

We can show that the numerical properties for a DFT are:

$$\frac{\sigma_x^2}{\sigma_q^2} = \frac{2^{2b}}{N^2} \quad 0 \leq |x(n)| \leq \frac{1}{N}$$

Note that the SNR decreases with the length of the FFT.

If the input is scaled to have a maximum magnitude of 1, then:

$$\frac{\sigma_x^2}{\sigma_q^2} = 2^{2b} \quad |x(n)| \leq 1$$

For an FFT with scaling,

$$\frac{\sigma_x^2}{\sigma_q^2} = 2^{2b-v-1}$$

where  $v$  is the radix base of the FFT.

Note that the SNR in dB is linearly related to the number of bits (similar to what we observed with the uniform quantizer).

