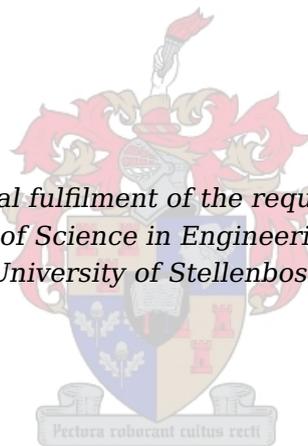


# The development of Sun and Nadir sensors for a solar sail CubeSat

by

Hanco Evert Loubser

*Thesis presented in partial fulfilment of the requirements for the degree of  
Master of Science in Engineering at the  
University of Stellenbosch*



Supervisors: Prof W.H. Steyn

Faculty of Engineering  
Department Electrical and Electronic Engineering

March 2011

# **Declaration**

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

March 2011

Copyright © 2011 Stellenbosch University

All rights reserved

# **Abstract**

This thesis describes the development of attitude sensors required for the Attitude Determination and Control System (ADCS) for a Cubesat. The aim is to find the most suitable sensors for use on a small picosatellite by implementing miniaturised sensors with available commercial-off-the-shelf (COTS) technology. Specifically, the algorithms, hardware prototypes, software and filters required to create accurate sensors to determine the 3-axis orientation of a CubeSat are discussed.

# Opsomming

Hierdie tesis beskryf die ontwikkeling van oriëntasiesensors wat benodig word vir die oriëntasiebepaling en -beheerstelsel (Engels: ADCS) van 'n *CubeSat*. Die doelwit is om sensors te vind wat die geskikste is om in 'n klein picosatelliet te gebruik, deur miniatuursensors met kommersiële maklik verkrygbare tegnologie (Engels: COTS technology) te implementeer. Daar word in die bespreking veral aandag geskenk aan die algoritmes, hardewareprototipes, programmatuur en filters wat benodig word om akkurate sensors te skep wat op hul beurt 3-as oriëntasie van die *CubeSat* kan bepaal.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>Opsomming</b>	<b>iv</b>
<b>List of Figures</b>	<b>viii</b>
<b>List of Tables</b>	<b>x</b>
<b>Nomenclature</b>	<b>xi</b>
<b>Acknowledgements</b>	<b>xiv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Background . . . . .	1
1.2 Problem formulation . . . . .	2
1.3 Thesis layout . . . . .	2
<b>2 Literature</b>	<b>3</b>
2.1 CanX-2 . . . . .	3
2.2 SwissCube . . . . .	4
2.3 DTUSat-2 . . . . .	4
2.4 Nanosatellites . . . . .	5
2.4.1 BRITE . . . . .	5
2.4.2 MOMENT . . . . .	6
2.5 CAPE-1 . . . . .	6
<b>3 Hardware choices and designs for sensors</b>	<b>7</b>
3.1 Camera modules . . . . .	7
3.1.1 CMOS vs CCD . . . . .	8
3.1.2 Lenses and optical filters . . . . .	9
3.1.2.1 Fisheye lens . . . . .	9
3.1.2.2 Neutral density filter . . . . .	10
3.2 Hardware controlling the sensors and processing images from the sensors . . . . .	11
3.2.1 Memory . . . . .	11
3.2.1.1 Power switch and current sensors . . . . .	12
3.2.2 Microcontrollers and FPGAs . . . . .	12
3.2.3 Other hardware choices . . . . .	14
3.3 Layout . . . . .	14
3.3.1 Hardware failure . . . . .	15
3.3.2 PCB layout . . . . .	16
<b>4 Methodology: Algorithms for sun and nadir sensors</b>	<b>17</b>
4.1 Thresholding of images . . . . .	18
4.1.1 Fixed vs dynamic threshold . . . . .	18
4.1.2 Choosing a fixed threshold . . . . .	19
4.2 Edge detection . . . . .	20
4.2.1 Searching for edge pixels . . . . .	20
4.2.1.1 Searching algorithm for the nadir sensor . . . . .	20
4.2.1.2 Searching algorithm for the sun sensor . . . . .	21

4.2.2	Edge detection for the nadir sensor . . . . .	21
4.2.3	Edge detection for the sun sensor . . . . .	22
4.3	Distortion . . . . .	22
4.3.1	Distortion model . . . . .	23
4.4	Centroid calculation . . . . .	25
4.4.1	Centroid calculation for the nadir sensor . . . . .	25
4.4.1.1	Least squares estimation . . . . .	25
4.4.1.2	Equation of a circle from three points . . . . .	26
4.4.1.3	Least squares circle . . . . .	27
4.4.2	Centroid calculation for the sun sensor . . . . .	28
<b>5</b>	<b>Experimental setups and calibrations</b>	<b>29</b>
5.1	Camera calibration . . . . .	29
5.1.1	Boresight . . . . .	29
5.1.2	Distortion centre point . . . . .	30
5.1.3	Focus . . . . .	31
5.1.3.1	Nadir sensor: focal length . . . . .	31
5.1.3.2	Sun sensor: the Sun's radius . . . . .	32
5.1.4	Exposure time . . . . .	33
5.2	Distortion . . . . .	33
5.2.1	Distortion model . . . . .	33
5.2.2	Distortion correction model . . . . .	36
5.3	Resolution . . . . .	37
5.4	Testbenches . . . . .	38
5.4.1	Nadir sensor testbench . . . . .	38
5.4.2	Sun sensor testbench . . . . .	39
5.5	Angular relationship . . . . .	40
5.6	Rotation point (optic point calculation) . . . . .	42
5.7	Threshold determination . . . . .	43
5.8	Sampling factor for the nadir sensor search algorithm . . . . .	44
<b>6</b>	<b>Software implementation</b>	<b>45</b>
6.1	Overview . . . . .	45
6.1.1	Data type definitions and algorithm complexity . . . . .	45
6.2	Distortion correction lookup table . . . . .	46
6.3	Interpolation . . . . .	47
6.4	Controlling the camera module . . . . .	48
6.4.1	Microcontroller . . . . .	49
6.4.2	FPGAs . . . . .	50
6.4.3	I <sup>2</sup> C protocol for the OBC . . . . .	50
6.4.4	Memory allocation . . . . .	51
6.5	Nadir sensor . . . . .	51
6.5.1	Search algorithm . . . . .	52
6.5.2	Edge detection . . . . .	53
6.5.2.1	Binary search . . . . .	54
6.5.2.2	Partial profile edge detection . . . . .	55
6.5.3	Centroid calculation . . . . .	56
6.5.3.1	Least squares circle . . . . .	56
6.6	Sun sensor . . . . .	56
6.6.1	Search algorithm . . . . .	57
6.6.2	Area search . . . . .	58
6.6.3	Centroid calculation . . . . .	58
<b>7</b>	<b>Results</b>	<b>60</b>
7.1	Nadir sensor measurement results . . . . .	60
7.2	Sun sensor measurement results . . . . .	62
7.3	Power consumption . . . . .	64
7.4	Time requirements . . . . .	65
7.5	Mass measurements . . . . .	66

<b>8 Conclusion</b>	<b>67</b>
<b>9 Summary and Recommendations</b>	<b>69</b>
9.1 Summary . . . . .	69
9.2 Recommendations and improvements . . . . .	70
9.2.1 I <sup>2</sup> C on FPGA . . . . .	70
9.2.2 Address control from microcontroller . . . . .	71
9.2.3 Alternative hardware layout . . . . .	72
<b>A Datasheets</b>	<b>73</b>
A.1 C3188A camera module . . . . .	73
A.2 OV7620 colour image sensor . . . . .	74
A.3 ORIFL190-3 fisheye lens . . . . .	77
A.4 AS7C34096A SRAM . . . . .	79
A.5 IRF7210PbF power MOSFET . . . . .	80
A.6 INA169 high-side measurement shunt monitor . . . . .	81
A.7 PIC18F45K20 microcontroller . . . . .	82
A.8 IGLOO Nano AGLN030 FPGA . . . . .	85
<b>B Hardware Layout</b>	<b>86</b>
B.1 Schematics . . . . .	86
<b>C Source code for microcontroller</b>	<b>89</b>
C.1 Software I2C . . . . .	89
C.2 Interpolation . . . . .	90
C.3 Edge detection for Nadir sensor . . . . .	90
C.4 Binary search . . . . .	93
C.5 Centroid calculation for Nadir sensor . . . . .	93
C.6 Centroid calculation for Sun sensor . . . . .	94
<b>D Source code for FPGA</b>	<b>96</b>
D.1 Gray code . . . . .	96
D.2 Memory read and write . . . . .	98
D.2.1 Write . . . . .	98
D.2.2 Read . . . . .	98
D.3 Search algorithm for nadir sensor . . . . .	99
D.4 First Sun pixel and area search for sun sensor . . . . .	100
<b>E Python source code</b>	<b>102</b>
E.1 Source code to download image from microcontroller . . . . .	102
<b>Bibliography</b>	<b>103</b>

# List of Figures

2.1	CanX-2 satellite [6]	3
2.2	SwissCube satellite [8]	4
2.3	DTUSat-2 satellite [10]	4
2.4	BRITE satellite [12]	5
2.5	MOMENT satellite [13]	6
3.1	Hardware part of the project	7
3.2	Difference between CMOS and CCD image sensors [16]	8
3.3	A lens that converges light [17]	9
3.4	Camera module with fisheye lens	9
3.5	Light intensity chart	10
3.6	Placement of neutral density filter (Appendix A.3)	10
3.7	Layout of the SRAM's power management	12
3.8	Pin count for one FPGA, one SRAM device and a microcontroller	13
3.9	Pseudo layout of the hardware	14
3.10	Hardware failure: memory failure	15
3.11	Hardware failure: memory and camera module failure	15
3.12	Sun and nadir sensor prototype	16
3.13	PCB layout	16
4.1	Algorithm development for the sun and nadir sensors	17
4.2	Image processing for the sun and nadir sensor	17
4.3	Errors occurring with dynamic thresholding	18
4.4	Errors occurring with fixed thresholding.	19
4.5	Calculating a fixed threshold [30]	19
4.6	The pixel coordinate axes	20
4.7	The "grid search" algorithm for the nadir sensor	20
4.8	Difference between background, edge and object pixels	21
4.9	The edge pixels of the Earth's profile	22
4.10	The area searched for Sun pixels	22
4.11	Radial distortions. The dashed rectangle indicates the original image. (a) Barrel distortion and (b) Pincushion distortion	23
4.12	Fisheye lens distortion	23
4.13	FOV distortion model [32]	24
4.14	Distortion correction	25
4.15	Equation of a circle with three points [35]	26
5.1	Determining the boresight of the nadir and sun sensor	29
5.2	Test for the distortion centre point	30
5.3	Distortion centre point at different points on an image	30
5.4	Calculating the focal length	31
5.5	Calculating the focal length at 103 cm and 150 cm respectively	31
5.6	Change in the Sun's radius between the boresight and the FOV edge	32
5.7	Maximum and minimum exposure time	33
5.8	Metric rotary stage used in tests	33
5.9	Images of horizontal distortion measurements	34
5.10	Radial distortion of fisheye lens	34
5.11	Error between simulated and measured Distortion Models	35

5.12 PFET distortion model . . . . .	35
5.13 Images of vertical distortion measurements . . . . .	36
5.14 Distortion correction . . . . .	37
5.15 Resolution over FOV . . . . .	37
5.16 Images of testbenches for the sun and nadir sensors . . . . .	38
5.17 Nadir Sensor testbench . . . . .	38
5.18 Images of the ball during rotation measurements . . . . .	39
5.19 Sun sensor testbench . . . . .	39
5.20 Images of the Sun during rotation measurements . . . . .	39
5.21 Angular relationship between the satellite, target and Earth [40] . . . . .	40
5.22 The ball at 3.52cm . . . . .	41
5.23 Error in body frame measurements . . . . .	42
5.24 Images captured during rotation measurements . . . . .	42
5.25 Optic point (Appendix A.3) . . . . .	43
5.26 Grayscale . . . . .	43
5.27 Edge profile of Earth . . . . .	44
5.28 Lines searched for edge pixels . . . . .	44
6.1 Data used for the distortion correction lookup table . . . . .	46
6.2 Using interpolation for sub-pixel accuracy . . . . .	47
6.3 Microcontroller control actions to the FPGAs and camera modules . . . . .	49
6.4 I <sup>2</sup> C time diagram . . . . .	49
6.5 Layout of FPGA modules . . . . .	50
6.6 Different arrangements of images in memory . . . . .	51
6.7 Search algorithm state machine . . . . .	52
6.8 Edge detection for nadir sensor . . . . .	53
6.9 Flow diagram of edge detection for the nadir sensor . . . . .	53
6.10 Pixels between two sample pixels . . . . .	54
6.11 False edge . . . . .	55
6.12 Calculating the Earth's centroid . . . . .	56
6.13 Inverse of a 3 x 3 matrix . . . . .	56
6.14 Extra module in the FPGA of the sun sensor . . . . .	57
6.15 Flow diagram for searching the first Sun pixel . . . . .	57
6.16 Area search flow diagram for sun sensor . . . . .	58
6.17 Calculating the Sun's centroid . . . . .	58
7.1 Nadir sensor body frame angles . . . . .	60
7.2 Nadir sensor RMS error . . . . .	61
7.3 Sun sensor body frame angles . . . . .	62
7.4 Sun sensor RMS error . . . . .	63
7.5 Power measurement . . . . .	64
7.6 Time required for the current to stabilise after an image is produced . . . . .	64
7.7 Processing time measurements for sun and nadir sensors . . . . .	65
8.1 Accuracy for nadir sensor . . . . .	67
8.2 Accuracy for sun sensor . . . . .	68
9.1 I <sup>2</sup> C module for FPGA . . . . .	70
9.2 Edge detection for nadir sensor with address control on microcontroller . . . . .	71
9.3 Difference in edge detection procedures . . . . .	71
9.4 Alternative hardware layout . . . . .	72

# List of Tables

1.1	Satellite classifications . . . . .	1
3.1	CMOS vs CCD . . . . .	8
3.2	Characteristics of suitable memory types . . . . .	11
3.3	Total pins connected to the microcontroller . . . . .	13
3.4	Total pins connected to one FPGA . . . . .	13
5.1	Results from the boresight test . . . . .	30
5.2	Results from the focal length test . . . . .	32
5.3	Sun's radius at different positions . . . . .	32
5.4	Angular diameter of sun and nadir sensor at an altitude of 750 km . . . . .	41
5.5	The radius of the ball at 3.52 cm . . . . .	41
5.6	Average luminance values for sensor images . . . . .	43
6.1	Data types for the C18 compiler . . . . .	45
6.2	Computational complexity of mathematical operations . . . . .	45
6.3	Definitions of the control pins . . . . .	48
6.4	Truth table of commands . . . . .	48
7.1	Power consumption measured . . . . .	64
7.2	Total mass of sun and nadir sensor . . . . .	66

# Nomenclature

## Abbreviations and Acronyms

ADC	Analog-to-digital converter
ADCS	Attitude determination and control system
AMR	Anisotropic magnetoresistive
AOI	Area of interest
BRITE	BRiight Target Explorer
CAD	Computer-aided design
Cal Poly	California Polytechnic State University
CanX	Canadian Advanced Nanospace Experiment
CAPE	Cajun Advanced Picosatellite Experiment
CCD	Charge-coupled device
CMOS	Complementary metal oxide semiconductor
COTS	Commercial off-the-shelf
DRAM	Dynamic random access memory
DMA	Direct memory access
DTUSat	Danish Technical University Satellite
EEPROM	Electrically erasable programmable read-only memory
EPFL	École Polytechnique Fédérale de Lausanne
EUSART	Enhanced universal synchronous asynchronous receiver transmitter
FET	Fish-eye transform
FOV	Field of view
FPGA	Field programmable gate array
GPIO	General purpose input/output
GPPL	Ground primary payload
GPS	Global positioning system
IDE	Integrated development environment
I <sup>2</sup> C	Inter-integrated circuit
IWA	Inertia wheel assembly
LEO	Low Earth orbit
LSB	Least significant byte
LSC	Least squares circle

MCU	Microcontroller unit
MIPS	Million instructions per second
MOEMS	Micro-opto-electro-mechanical systems
MOMENT	Magnetic observations of Mars enabled by nanosatellite technology
MOSFET	Metal-oxide-semiconductor field-effect transistor
MSB	Most significant byte
MSSP	Master synchronous serial port
NDF	Neutral density filter
OBC	Onboard computer
PC	Personal computer
PCB	Printed circuit board
PFET	Polynomial FET
RMS	Root mean square
SEB	Single event burnout
SEE	Single event effects
SEGR	Single event gate rupture
SEL	Single event latch-up
SEU	Single event upset
SFL	Space flight laboratory
SNR	Signal-to-noise ratio
SNS	Sun and Nadir sensor
SRAM	Static random access memory
SSTL	Surrey satellite technology ltd
UTIAS	University of Toronto Institute for Aerospace Studies
VHDL	VHSIC hardware description language
VHSIC	Very high speed integrated circuits

**Greek Letters**

$\lambda$	Distortion factor
$\Delta$	Difference between an image's object and background luminance level
$\omega$	FOV of an ideal fisheye lens
$\sigma$	Standard deviation

**Lowercase Letters**

$s$             Scaling factor

$r$             Radius

**Subscripts**

$d$             Distorted radius

$u$             Undistorted radius

$e$             Distortion error

**Syntax and Style**

$v$             Algebraic variable (Italic)

$j$             Imaginary number

log          Mathematic functions (Normal)

**A**          The matrix **A** (usually uppercase)

$\frac{df}{dx}$         Derivative of function,  $f$ , with respect to  $x$

$\frac{\partial f}{\partial x}$         The partial derivative of function,  $f$ , with respect to  $x$

# Acknowledgements

I would like to thank and acknowledge the following people for their contributions during the process of this project:

- Prof. WH Steyn for his knowledge and guidance
- AM de Jager for his knowledge on cameras and VHDL coding
- Arno Barnard for his knowledge on coding of microcontrollers and FPGA
- Johan Arendse for his excellent soldering work
- My family for their support and knowing when not to bother me
- My friends inside and outside the lab: Those outside the lab for being patient with me when I had to say no, because I'm busy writing my thesis and those inside the lab for the social gatherings, the lunch times and a little bit of gaming every now and then.

# Chapter 1

## Introduction

### 1.1 Background

On October 4, 1957, the first artificial satellite, Sputnik 1, was launched [1]. Sputnik had a mass of 83.46 kg and had a diameter of 58 cm. The satellite had four antennas between 2.4 -and 2.9 m long that transmitted radio signals between 20.005 and 40.002 MHz [2]. On October 26, 1957, Sputnik 1's signal stopped transmitting, as the transmitter batteries were depleted. The launch of Sputnik 1 began the age of space science and exploration.

Since Sputnik 1, satellites became smaller (depending on their application), more power efficient and able to process more information as technology advanced in the past 53 years. Table 1.1 shows how satellites are classified by their mass, including fuel [3].

<b>Group name</b>	<b>Wet Mass</b>
Large satellite	>1000 kg
Medium satellite	500-1000 kg
Mini satellite	10-100 kg
Nano satellite	1-10 kg
Pico satellite	0.11-1 kg
Femto satellite	<100 g

**Table 1.1** – Satellite classifications

In 1999, Prof. Jordi Puig-Suari at California Polytechnic State University (Cal Poly) and Prof. Bob Twiggs at Stanford University began a project to standardise the design of a picosatellite to reduce the cost and the development time [4]. This standard is called a CubeSat. A Cubesat is a cube shaped satellite with measurements 10 cm x 10 cm x 10 cm and has a maximum mass of 1.33 kg. This is the smallest form of a CubeSat and is called a 1U (1 unit) CubeSat. Bigger CubeSats such as a 3U CubeSat, will have measurements of 10 cm x 10 cm x 30 cm.

The first five CubeSats were launched together in June 2003 [5]:

- AAU CubeSat
- DTUSat
- CUTE-1
- CanX-1
- Quakesat

The CubeSats had different mission objectives, for example Earth observation, earthquake detection from space and technology validation for future technology demonstrators. Independent of the mission objectives, a common factor was that the CubeSats were all developed by students of different universities.

## 1.2 Problem formulation

The attitude determination and control system (ADCS) is an important part of any satellite. The ADCS utilises various sensors and actuators to determine and control the attitude of the satellite. The problem defined for this thesis is to develop the most suitable minimum set of sensors to determine the attitude of a CubeSat in 3 axes. The research will focus on available technology to implement these sensors in a miniaturised form. Practical hardware prototypes must be developed and calibration models must be developed.

The following was found to be the most common ADCS sensors available:

- Sun sensors – determine the Sun’s position relative to a satellite
- Star sensors – use stars to determine a satellite’s attitude
- Horizon/Nadir sensors – determine the Earth’s position relative to a satellite
- Magnetometers – determine the satellite’s position by measuring the Earth’s magnetic field
- Gyroscopes and accelerometers – measure a satellite’s angular velocities and accelerations

The two sensors that were given to be developed for a CubeSat, are a nadir sensor and a sun sensor. Both will be optical sensors, consisting of CMOS cameras.

## 1.3 Thesis layout

The development of the sun and nadir sensors for a solar sail CubeSat is discussed in the following chapters:

- Chapter 2 provides an overview of CubeSat projects around the world.
- Chapter 3 shows the hardware designs implemented for the sun and nadir sensors.
- Chapter 4 develops the algorithms implemented to calculate a centroid from the images of the sensors.
- Chapter 5 investigates the experimental setup and calibration procedures applied to the sensors.
- Chapter 6 discusses the software development for the Sun and Nadir sensors.
- Chapter 7 shows the practical results obtained from the Sun and Nadir sensors.
- Chapter 8 summarizes the results obtained from the Sun and Nadir sensors.
- Chapter 9 concludes this thesis with a summary, as well as recommendations and suggestions for improvement for future development.

## Chapter 2

# Literature

In this chapter a number of recent projects of CubeSats are summarized and the sensors implemented for the specific projects are discussed. There are ongoing projects with respect to CubeSats at different universities, which is contributing to the field of space technology research and development. It is the combination of attitude sensors used that is of importance for this project.

### 2.1 CanX-2

The CanX-2 is a satellite developed by the Space Flight Laboratory (SFL) at the University of Toronto Institute for Aerospace Studies (UTIAS) [6] and was launched in April 2008 [7]. It is a 3U CubeSat with dimensions 10 x 10 x 34 cm and a mass of 3.5 kg. Figure 2.1 is an image of the CanX-2 satellite.



**Figure 2.1** – CanX-2 satellite [6]

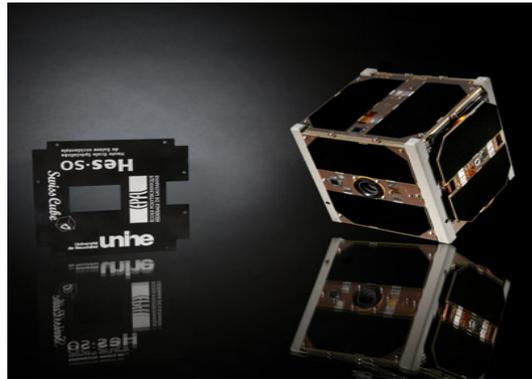
The main objectives for this satellite are to demonstrate the technology that will be implemented on the CanX-4/5 formation flight mission and to provide cost-effective access to space for the research and development community in Canada [6]. The payloads on the CanX-2 are the following:

- a miniature atmospheric spectrometer
- a GPS atmospheric occultation experiment
- a surface material experiment
- a dynamic spacecraft networking protocol experiment

The sensors implemented for the ADCS of this satellite are high precision sun sensors and a magnetometer. Both fine and coarse sun sensors are used. The six fine sun sensors are CMOS detectors with an accuracy of  $1^\circ$  and measure the Sun's body frame vector. The coarse sun sensors with an accuracy of between  $5^\circ$  and  $10^\circ$  determine which of the fine sun sensors are required to be sampled. When both sun sensors and a magnetometer are being sampled, it is estimated that the accuracy of the attitude determination is between  $1^\circ$  and  $2^\circ$  in sunlight. The accuracy decreases during Sun sensor drop-out, because only the magnetometer is being sampled.

## 2.2 SwissCube

SwissCube is the first satellite entirely built in Switzerland and was mainly built by students from different universities under the supervision of the Space Center EPFL [8]. It is a 1U CubeSat with dimensions 10 x 10 x 10 cm and a mass of less than 1 kg. SwissCube was launched in October 2009 [7]. Figure 2.2 is an image of the SwissCube CubeSat.



**Figure 2.2** – SwissCube satellite [8]

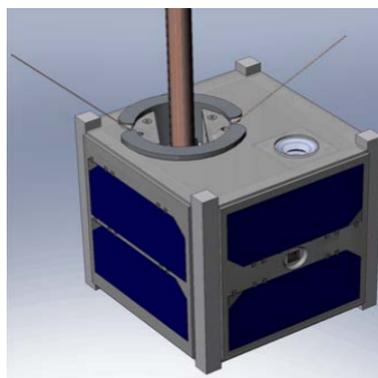
The aim of the SwissCube project was to develop an Inertia Wheel Assembly (IWA) with minimum mass and power consumption [9]. The mission for the SwissCube is to measure the nightglow, which is a luminescence phenomenon of the atomic oxygen at high altitudes of the atmosphere. The nightglow will be measured by taking images of the atmosphere with a custom camera.

The ADCS of SwissCube determines its velocity, orientation and position of the CubeSat. It is important for the payload, as the payload is required to be pointed and oriented precisely to take images of the nightglow. The ADCS implements the following sensors [8]:

- a 3-axis magnetometer to measure the intensity and direction of the Earth's magnetic field
- six sun sensors. The sun sensors do not function in eclipse
- a 3-axis gyroscope to measure the angular velocity of the satellite

## 2.3 DTUSat-2

DTUSat-2 is a project of the National Space Institute at the Technical University of Denmark [10] and was developed by students of DTU [11]. This CubeSat has dimensions of 10 x 10 x 11.35 cm with a mass of 1 kg. Figure 2.3 is a CAD drawing of DTUSat-2 fully deployed.



**Figure 2.3** – DTUSat-2 satellite [10]

In November 2005, a conference was held to select a payload for the DTUSat-2. The mission for this CubeSat is to track the migration patterns of smaller birds over different continents. The CubeSat implements a Ground Primary Payload (GPPL), which tracks the small radio transmitters placed on small birds [11].

The ADCS implements the following sensors:

- a custom built four axis AMR magnetometer
- a 2-axis sun sensor
- an infrared coarse attitude sensor (horizon sensor)
- a PICOAM for offline stellar reference sensing (star sensor)

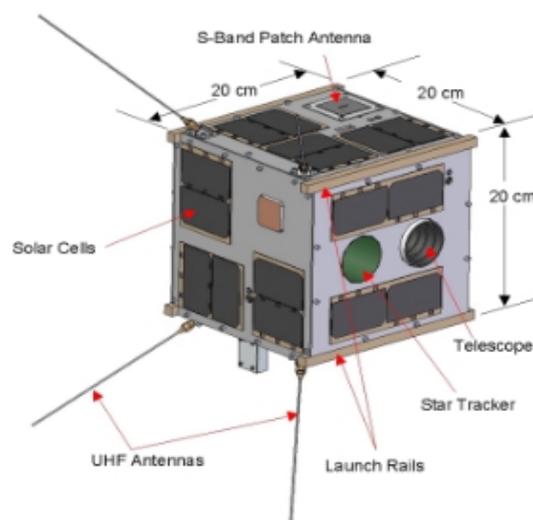
The primary attitude sensor is the magnetometer. The magnetometer delivers 3-axis attitude measurements with an accuracy of  $0.5^\circ$ . The PICOAM is optimized for applications with limited power capabilities [11]. The CCD camera is implemented as a deployment monitor and a star sensor. The MOEMS sun sensor is the smallest dual axis sun sensor developed for attitude determination and has an accuracy of less than  $1^\circ$ .

## 2.4 Nanosatellites

Two significant nanosatellite that have been developed are the BRITE and MOMENT. Although they do not comply with the CubeSat standards, the technologies implemented on these satellites were initially developed and implemented on CubeSats.

### 2.4.1 BRITE

The BRiGht Target Explorer (BRITE) is a project consisting of a constellation of four nanosatellites. This is a project of UTIAS/SFL [12]. Each of the four nanosatellites has dimensions of 20 x 20 x 20 cm, has a mass of about 5 kg and implements technologies from the CanX-2 and CanX-4/-5 CubeSats. Figure 2.4 illustrates one of the nanosatellites in the constellation.



**Figure 2.4** – BRITE satellite [12]

The aim of the BRITE-Constellation is to observe the luminous oscillations of stars that have the most affect on the ecology of the Universe [12]. This is done by implementing a small-lens telescope with a FOV of  $25^\circ$  on the nanosatellites and observing the brightest stars in the sky.

Attitude determination is provided by the following sensors:

- a magnetometer
- six sun sensors (coarse and fine) developed by SFL
- a nanosatellite star tracker

The sensors enable attitude determination to 10 arcseconds ( $0.003^\circ$ ) [12].

### 2.4.2 MOMENT

The Magnetic Observations of Mars Enabled by Nanosatellite Technology (MOMENT) was developed by SFL to measure the magnetic field of Mars [13]. Figure 2.5 illustrates the 16 kg MOMENT nanosatellite.



**Figure 2.5** – MOMENT satellite [13]

The design of the MOMENT satellite is based on the generic nanosatellite bus developed during the BRITE and CanX-4 and -5 missions [13]. To measure the magnetic field of Mars, a 3-axis magnetometer is used as payload, with a range of  $\pm 4000$  nT and a resolution of 0.5 nT.

The sensors used for attitude determination are:

- a star tracker (This is the primary attitude sensor, as it provides the best accuracy)
- six sun sensors
- rate sensors

The star tracker implemented in the MOMENT nanosatellite is based on the star tracker from the BRITE mission [13]. The attitude of the magnetometer is required to be accurate to  $\pm 1$  arcminute ( $0.02^\circ$ ). Because MOMENT has its heritage from BRITE, the attitude determination is well equipped to follow this accuracy.

## 2.5 CAPE-1

The above mentioned satellites all use attitude sensors to implement an active attitude control, but it is also worth mentioning that CubeSats can be controlled passively. The CAPE-1 CubeSat is passively controlled by magnetic stabilisation through permanent magnets on the CubeSat. The satellite spin is controlled by hysteresis rods. In October 2005 the University of Louisiana started plans to develop this CubeSat [14] and it was launched in April 2007 [15]. The payload of the CAPE-1 CubeSat are the sensors that measure temperatures, battery life and solar power collection on the satellite. CAPE-1 transmits and receives data from sensors in the Gulf of Mexico that measure the saltwater erosion on the wetlands.

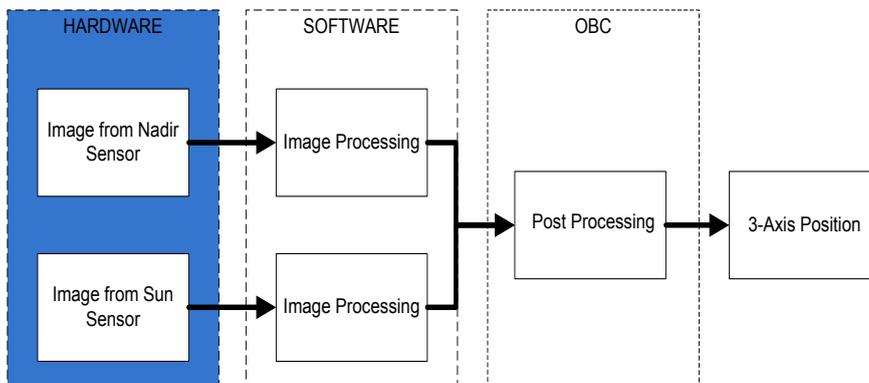
## Chapter 3

# Hardware choices and designs for sensors

The sun and nadir sensors consist of two camera modules. The hardware required to store and retrieve images produced by these camera modules, as well as the hardware containing the image processing software, are discussed in this chapter. The choices of hardware is determined by a few criteria set by the goal of this project:

- The hardware components' power consumption should be taken into consideration when a choice is made, as the only source of power is solar power, which is limited.
- The space environment should also be taken into consideration when choosing hardware components. Extreme temperatures and radiation are expected in space. These conditions can damage hardware components and prevent them from performing nominally.
- The size of a PCB that slots into a CubeSat has maximum dimensions of 100 x 100 mm. The size of the components have to be considered as well, because the space for hardware placement is limited.
- The sensors are required to be able to communicate with the rest of the satellite, or at least with the OBC, to receive commands, or to report errors observed through sensors. The hardware therefore requires the necessary peripherals to communicate with the rest of the satellite.

The goal of the hardware is to control the two camera modules, store and retrieve images, contain the software required to process the information to be extracted from the stored images and connect to the other parts of the satellite to transmit and receive data and commands.



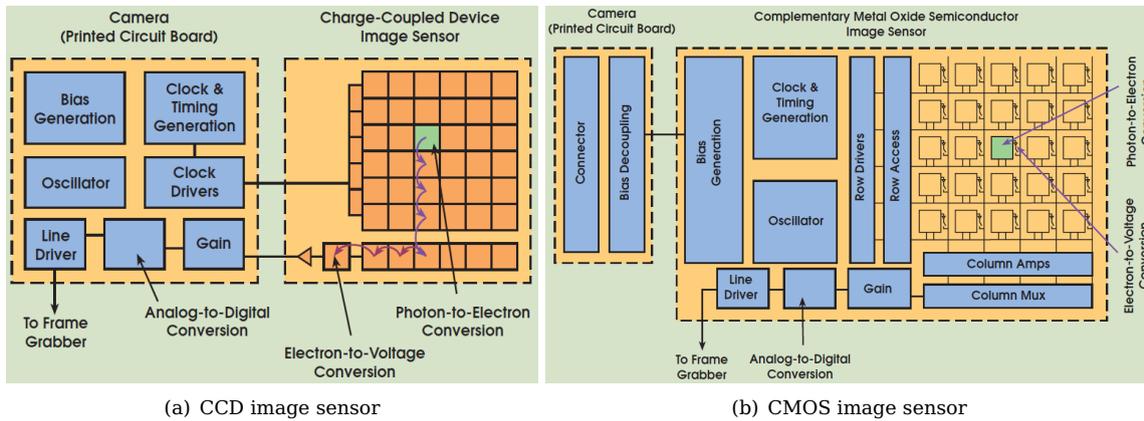
**Figure 3.1** – Hardware part of the project

### 3.1 Camera modules

Each of the sun and nadir sensors contains a camera module that enables it to capture images. Aspects considered when choosing the sun and nadir sensors include the type of image sensor and the lenses and filters required for the specific implementations.

### 3.1.1 CMOS vs CCD

CMOS and CCD are the most commonly used image sensor technologies for still imagery applications. Choosing between these two technologies are depended mainly on the application it will be used for. Figure 3.2 shows a simple way of distinguishing between CCD and CMOS image sensors.



**Figure 3.2** – Difference between CMOS and CCD image sensors [16]

Litwiller [16] explains that both image sensor types are pixelated metal oxide semiconductors that accumulate signal charge in each pixel proportional to the local illumination intensity. The CCD sensor transfers the pixel charges sequentially to a common ADC that converts the electrical charges to voltages. The CMOS sensor converts each electrical charge to a voltage on the specific pixel. Litwiller also lists eight attributes for characterizing an image sensor. Table 3.1 lists these attributes compares CCD and CMOS image sensors.

ATTRIBUTE	CCD	CMOS
Responsivity	Uses more power	Uses less power
Dynamic range	Lower noise levels	Higher noise levels
Uniformity	Better	Good
Shuttering	Better	Good
Speed	Slow	High
Windowing	Limited	Unique to CMOS
Anti blooming	Limited	Generally immune
Biasing and clocking	Multiple levels	Single level

**Table 3.1** – CMOS vs CCD

By comparing the attributes in table 3.1, it is evident that CCD is better in dynamic range, uniformity and shuttering. These three attributes ensure that CCD image sensors have a better SNR and contrast ratio than that of CMOS image sensors. Dynamic range shows that CCD image sensors have lower noise levels and the uniformity attribute confirms this, as uniformity entails the consistent performance of pixels under constant illumination and especially in near-dark illumination. Shuttering allows CCD image sensors to start and stop exposure at any time, giving them the ability to prevent saturation of pixels and give the best contrast between shades of light.

However, the advantages that CCD image sensors have over CMOS image sensors, are not primary attributes considered when choosing an image sensor for this project. This is because the difference between light and dark is large due to the nature of the objects being tracked and the darkness of the background behind them. The SNR advantage of CCD image sensors will give a better accuracy than CMOS image sensors, but the accuracy can be improved with software. CMOS image sensors use less power, has single voltage level clocking and are faster than CCD image sensors. Most of the post processing components, for example ADC, are implemented on-chip and therefore the implementation of CMOS image sensors into hardware is more compact than that of CCD image sensors.

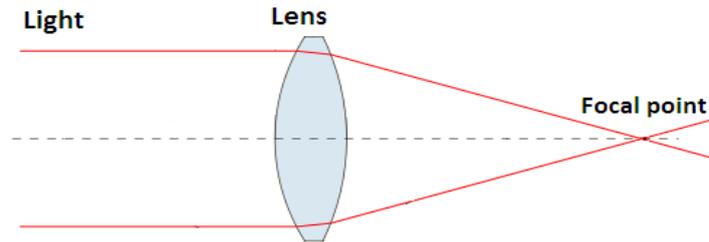
The camera module given for both the nadir and sun sensors is the OMNIVISION C3188A camera module, which uses the OMNIVISION OV7620 CMOS colour image sensor, as shown in Appendix A.1 and A.2 respectively. The OV7620 is a low power image sensor. The power requirements for the image sensor is less than 120 mW when active and less than 10  $\mu$ W when in standby mode.

### 3.1.2 Lenses and optical filters

Besides the image sensor required to capture an image, lenses and optic filters are also necessary to complete the camera module.

#### 3.1.2.1 Fisheye lens

A lens is an optical device with perfect or approximate axial symmetry which transmits and refracts light, which converges or diverges the beam [17]. The lenses used in this project converge the light that falls onto the lens to a proportional point on the pixel array of the image sensor. Figure 3.3 illustrates how a lens converges light to a focal point.



**Figure 3.3** – A lens that converges light [17]

For the sun and nadir sensors in this project, fisheye lenses are used. A fisheye lens is a wide-angle lens that takes in an extremely wide, hemispherical image [18]. When a hemispherical lens is used to capture images of spherical objects, such as the Sun and the Earth, the image will display a perfect circle at the boresight of the lens. This is useful when calculating the centroid of a spherical object. The wide-angle FOV of fisheye lenses enables the sensors to track an object over a larger FOV. It is especially useful in the application of a nadir sensor. The Earth can be tracked over a larger distance with the larger FOV, while still having the full profile of the Earth in view. Fisheye lenses, however, have a significant amount of distortion that will affect the shape of the object when the object is moving away from the boresight.

The fisheye lens given is the OMNIVISION ORIFL190-3, a 1/3" lens with a field of view of 190° and F-number of 2.8, as shown in Appendix A.3. Figure 3.4 is an image of the C3188A camera module with the ORIFL190-3 fisheye lens.



**Figure 3.4** – Camera module with fisheye lens

### 3.1.2.2 Neutral density filter

Figure 3.5 shows the lux value of the Sun. The minimum lux value for a lens with F-number 1.4, detected by the OV7620 image sensor, is 2.5 lux. Lux is a unit used to measure the amount of visible light. The fisheye lens has F-number 2.8, which means that the aperture size is 25% of a lens with F-number 1.4 [19]. Therefore, the light gathering area of the lens is smaller. If the same ADC gain and exposure time are implemented for the lens with F-number 2.8 as for the lens with F-number 1.4, the minimum lux that is required to be detected by the image sensor, is 10 lux.

The Sun has an illumination of about 100 000 lux on the surface of the Earth. Because the light from the Sun is so intense, the image sensor will be saturated without using a filter. The Sun's light is therefore filtered through a neutral density filter. Neutral density filters are filters that reduce the intensity of light that passes through it without disturbing the relative spectral content [20].

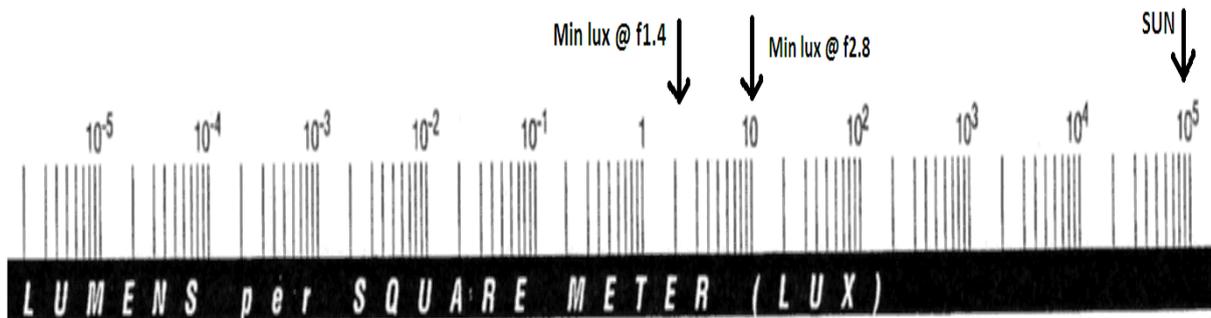


Figure 3.5 – Light intensity chart

The neutral density filter selected is the Kodak Wratten 2 ND 4.00 that has a transmission percentage of 0.01% [21]. This means that 99.99% of the Sun's light will be absorbed by, and only 0.01% will pass through the filter. The Sun's light will be filtered to 10 lux, which is more than or equal to the minimum lux value specified for the image sensor. This enables the Sun to be viewed as a small dot on an image from the sun sensor.

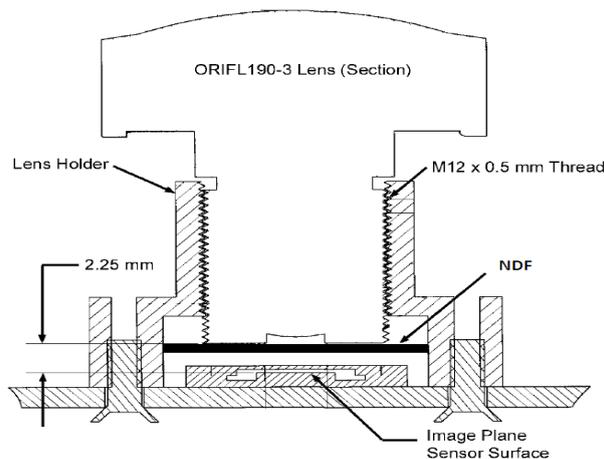


Figure 3.6 – Placement of neutral density filter (Appendix A.3)

Figure 3.6 illustrates the placement of the neutral density filter on the camera module. The filter is positioned between the image sensor and the fisheye lens.

### 3.2 Hardware controlling the sensors and processing images from the sensors

After selecting the hardware for the sensors, the hardware that supports these sensors is selected. Hardware is required for the following purposes:

- Data storage: the hardware design should contain memory to store and retrieve images.
- Data processing: the design requires hardware that store the image processing software.
- Power: the hardware must be powered efficiently, as well as support standby and shutdown modes to save power.

When choosing the specific components, especially for space applications, radiation tolerance must be considered. The effects of radiation on components are single event effects (SEEs)[22][23]. The SEEs commonly encountered are:

- Single Event Upsets (SEUs) - transient pulses in logic or bit flips in memory units. SEUs may cause permanent damage, such as stuck bits in memory.
- Single Event Latchups (SELs) - induced high current operation. SELs may cause permanent damage
- Single Event Gate Ruptures (SEGRs) - form a conducting path in the gate oxide
- Single Event Burnouts (SEBs) - induce high current state in power transistors. SEBs can cause device destruction of power transistors

If these SEEs are detected early enough, the damage can be minimized with a power cycle of the affected component.

#### 3.2.1 Memory

The camera modules are driven by a pixel clock at a rate equal to 13.5 MHz [24]. The storage space required per image is 300 KB. The microcontrollers that are able to process this amount of data at this clock rate are expensive and their power requirements are high. Memory is used instead to store the images in order for them to be retrieved by the microcontroller, at a speed more suitable for the requirements of this project. The memory types considered for this specific application are listed in table 3.2 [25].

ATTRIBUTE	SRAM	DRAM	EEPROM	FLASH
Volatile	Yes	Yes	No	No
Writable	Yes	Yes	Yes	Yes
Erase Size	Byte	Byte	Byte	Sector
Power consumption	Moderate	High	Low	Low
Read/write speed	Fast	Moderate	Fast read, slow erase/write	Fast read, slow erase/write
Storage density	Low	High	Low	High
SEE	Sensitive	Sensitive	Less sensitive	Less sensitive

**Table 3.2** – Characteristics of suitable memory types

EEPROM and FLASH memories seem to be good candidates. Both are non-volatile, retain their data when power is removed and are less power consuming. Both memory types are less sensitive to SEEs [26]. However, the slow write speeds of EEPROM and FLASH are to their disadvantage make them unsuitable for this application. FLASH also has the disadvantage of having to be erased before new data can be written to it.

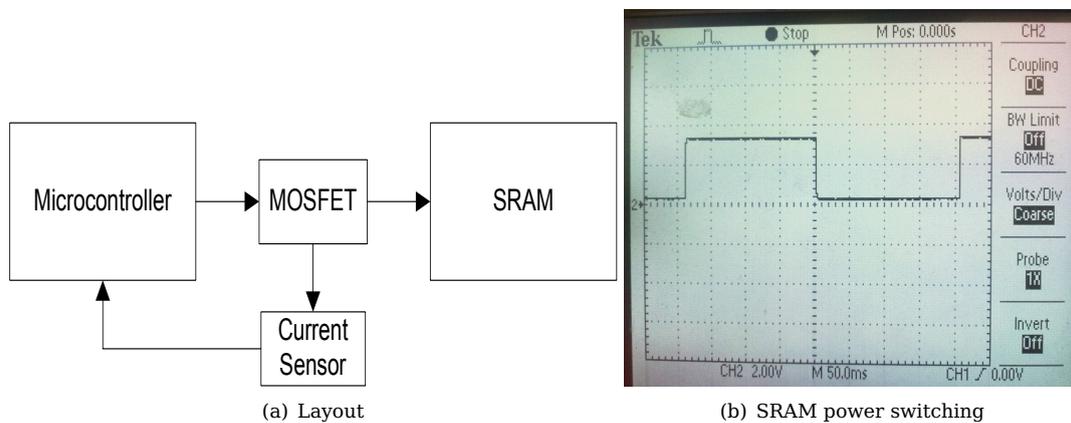
The memory selected has to be able to write with an access time of at least 74 ns per pixel, because of the pixel data rate of the camera module. SRAM and DRAM have fast read and write speeds. SRAM is selected above DRAM, because DRAM must be refreshed every few milliseconds [25]. This means extra hardware will be required to refresh the DRAM and consequently more power is required. SRAM is therefore the

best choice to store the images from the camera modules.

The SRAM selected is the AS7C34096A from ALLIANCE SEMICONDUCTOR, which has 512 kB memory, 8-bit data words, and a maximum access time of 12 ns, as shown in Appendix A.4. This access time is six times faster than the access time that is required, giving the data from the camera module enough time to be stored in the memory. The 512 kB is enough memory to store one 300 kB image. Two memory chips are used, one for the nadir sensor and one for the sun sensor. If one memory should fail, the two sensors will be able to share the other memory unit by storing one sensor's image, processing the image and then storing the other sensor's image in order for it to be processed as well.

### 3.2.1.1 Power switch and current sensors

SRAM is susceptible to SEUs and SELs, where SELs are the more destructive of the two. As mentioned in section 3.2, a power cycle is required to attempt to correct these errors. The SRAMs' power inputs are controlled by two power MOSFETs and a microcontroller. By switching off the MOSFETs via the microcontroller, the SRAMs are switched off as well. The current flow to each SRAM is monitored by a current sensor. The current flow is monitored for possible latch-ups that may occur. Figure 3.7 illustrates the SRAMs' power management layout.



**Figure 3.7** – Layout of the SRAM's power management

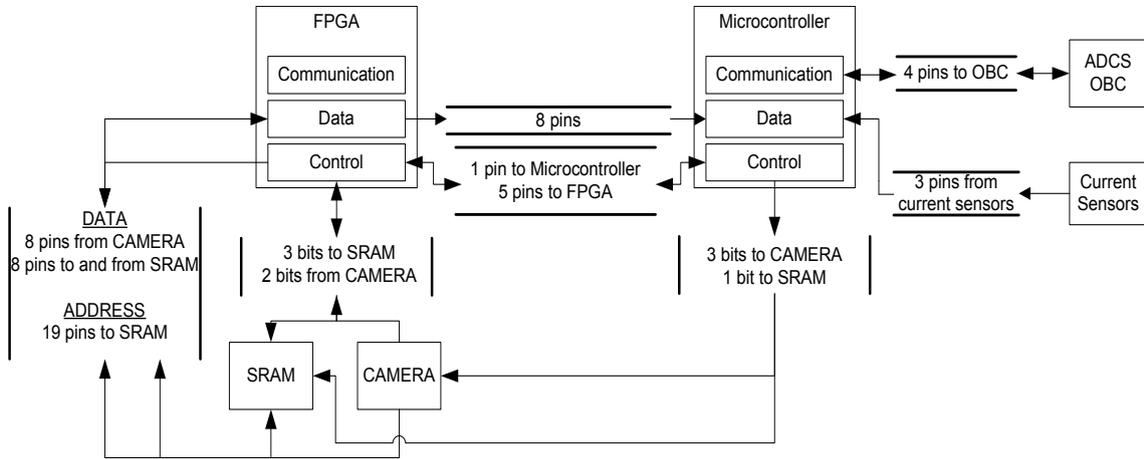
The power MOSFETs used are the IRF7210PbF P-Channel MOSFET from INTERNATIONAL RECTIFIER and the current sensors are the INA169 high-side measurement shunt monitors from TEXAS INSTRUMENTS.

### 3.2.2 Microcontrollers and FPGAs

A microcontroller is required to control the sensors, process the images into the necessary data and communicate with the ADCS OBC. The choice of microcontroller will, as with the choice of memory, depend on the data output rate of 13.5 MHz of the image sensor.

The image sensor's data output is in 8-bit format. A good choice for a microcontroller would therefore be an 8-bit microcontroller, as it will be able to process 8-bit data per clock cycle. Another advantage is that the power consumption of an 8-bit microcontroller is very low. A problem arises when using only a microcontroller with an image sensor with a data rate of 13.5 MHz. The microcontroller will be required to process data faster than 13.5 Mhz just to download an image from the image sensor and to store the image in the memory. At least 30 GPIO pins are necessary, because the memory unit has 19 address, three control and eight data pins. In such a case where many pins are required and data is processed at a high speed, the microcontroller will require more space on the PCB and will require more power. As an alternative, an FPGA is used in this project to download an image from the image sensor to the memory. An FPGA can work at higher frequencies, while using less power than microcontrollers working at the same frequencies.

Using an FPGA will allow the microcontroller to process at a lower frequency and therefore use less power. A microcontroller is still necessary, because of its dedicated arithmetic and communication units used for calculations and communication between itself and the OBC, respectively.



**Figure 3.8** – Pin count for one FPGA, one SRAM device and a microcontroller

Figure 3.8 illustrates the flow of data and control between the main components of the sun and nadir sensors. Note that the OBC is an off-board component. Figure 3.8 only shows one memory unit and one FPGA connected to the microcontroller, because the other memory unit and FPGA have the identical flow of data and control to and from the microcontroller. The choice of implementing two FPGAs will be discussed in section 3.3.

MCU	FPGAs	SRAMs	CAMERAs	Sensors	OBC
Data Inputs	8	-	-	3	-
Control Inputs	2	-	-	-	-
Control Outputs	6	2	6	-	-
Communication	-	-	-	-	4

**Table 3.3** – Total pins connected to the microcontroller

Table 3.3 indicates the assignment and total pins required to implement the data and control flow illustrated in figure 3.8. The assignments consist of 27 GPIOs, I<sup>2</sup>C and EUSART pins. The I<sup>2</sup>C and EUSART pins are the communication busses used to communicate with the OBC and the I<sup>2</sup>C is the primary and the EUSART the secondary bus. These busses receive commands from and send data to the OBC.

FPGA	SRAM	CAMERA	MCU	FPGA #2
Data Inputs	-	8	-	-
Data Outputs	-	-	8	-
Data Bidirectionals	8	-	-	8
Control Inputs	-	2	5	-
Control Outputs	22	-	1	-
Control Bidirectionals	-	-	-	2

**Table 3.4** – Total pins connected to one FPGA

Table 3.4 indicates the pin assignments for the FPGA to control the data flow between the memory, the microcontroller and the second FPGA. A total of 64 GPIOs are required.

The selected microcontroller and FPGAs are the PIC18F45K20 from MICROCHIP and the IGLOO NANO AGLN030 from ACTEL, respectively. The IGLOO NANO is the industry’s lowest power FPGA (see Appendix A.8) and the microcontroller uses MICROCHIP’s nanoWatt Technology (see Appendix A.7). With ACTEL’s

Flash Freeze technology and MICROCHIP's nanoWatt technology, the FPGAs and microcontroller only use a few microWatts when in idle mode.

The AGLN030 and PIC18F45K20 are selected specifically, because they have 77 and 36 GPIOs respectively, which is more than the 64 and 31 required for the hardware layout. The AGLN030 and PIC18F45K20 are both flash-based components. This makes them less sensitive to SEEs as well.

### 3.2.3 Other hardware choices

After the main components have been selected, the supporting hardware, such as voltage regulators, buffers, etc., are selected to incorporate the main components into the layout.

The CubeSat standard has a 5V supply to power the rest of the satellite. The main components need the following three voltage busses:

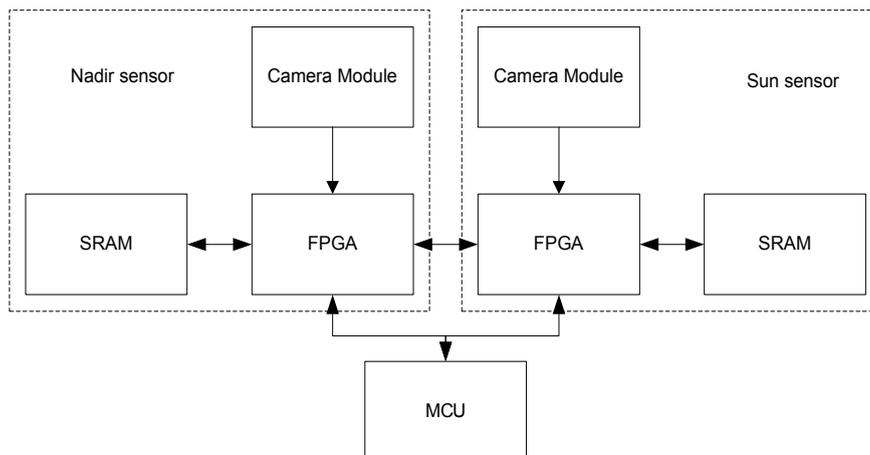
- The main voltage bus is 3.3 V voltage bus that powers most of the components (such as the microcontroller) in the hardware design.
- The camera module needs a 5 V and a 3.3 V voltage bus to power its analog components (such as the pixel array) and to make the camera module outputs compatible with the rest of the hardware.
- The FPGAs uses a 1.5 V and a 3.3 V voltage bus to power its internal transistors and IO banks to make them compatible with the rest of the hardware.

The hardware design incorporates the 5 V from the CubeSat's power supply and uses a 3.3 V and 1.5 V linear voltage regulator to power the hardware.

The on-chip ADC of the microcontroller is used to measure the currents from the 3.3 V voltage bus and the two memory units. An LM2902 operational amplifier is used to buffer between analog measurements from the current sensors and the microcontroller. This ensures that there are no compatibility issues between the current sensors and the microcontroller.

## 3.3 Layout

It was decided to use a dual FPGA layout. Figure 3.9 illustrates how the different components are connected to one another to form the layout of the sun and nadir sensor. One FPGA, one SRAM and one camera module forms one sensor. The two FPGAs are also connected to each other to minimize the effect of hardware failures.



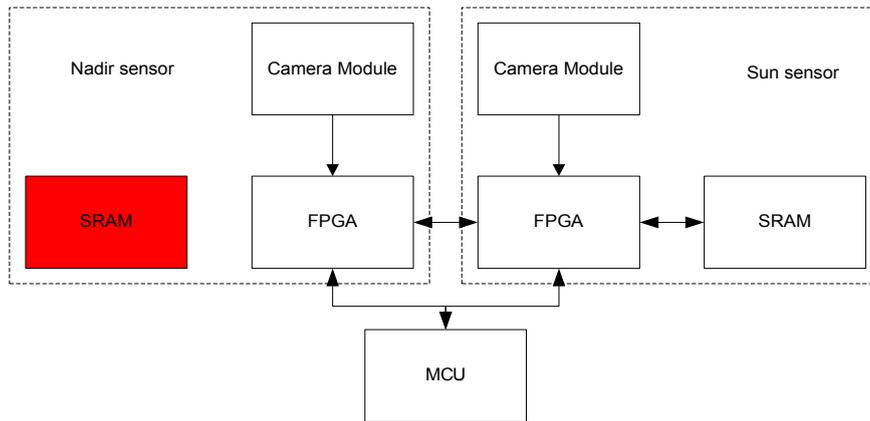
**Figure 3.9** – Pseudo layout of the hardware

### 3.3.1 Hardware failure

There are four types of hardware failure that may occur and will have an effect on the sun and nadir sensors' functionality:

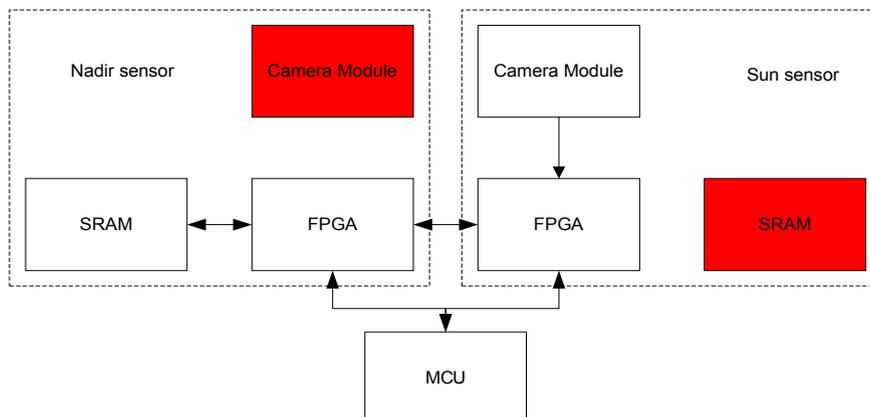
- FPGA failure
- memory failure
- camera failure
- simultaneous camera and memory failure

When a FPGA or camera module fails, a sensor fails, however figures 3.10 and 3.11 illustrate the two hardware failures' effects that can be minimized.



**Figure 3.10** – Hardware failure: memory failure

Figure 3.10 illustrates the hardware failure if one of the two SRAMs should fail. A latch up, for example, occurred and power cycling of the memory did not correct the latch up. It is possible for both camera modules to use the same memory, however, the ability for both sun and nadir images to be stored at once, will be lost. Memory space of 600 kB is required to store both images, but only 512 kB is available with one memory unit. One sensor's image should occupy the memory, the calculations should be completed and then the second sensor will be able to occupy the memory with its image.



**Figure 3.11** – Hardware failure: memory and camera module failure

Figure 3.11 illustrates a hardware failure if a camera module from one sensor and a memory chip from the other sensor should fail. This means that the sensor with the failed camera module can no longer function. The sensor with the functioning camera module and failed memory unit can however use the non-functioning sensor's memory unit. This will ensure that there will be at least one working sensor when only one camera module is still functioning.

### 3.3.2 PCB layout

Figure 3.12 shows the first prototype PCB layout for the sun and nadir sensor.



**Figure 3.12** – Sun and nadir sensor prototype

Figure 3.13 shows the final PCB layout with the PC104 connector. The PC104 standard is used in CubeSats to connect the separate modules of the CubeSats to one another. The 5 V and I<sup>2</sup>C bus are connected to the other components of the CubeSat through the PC104 connector.

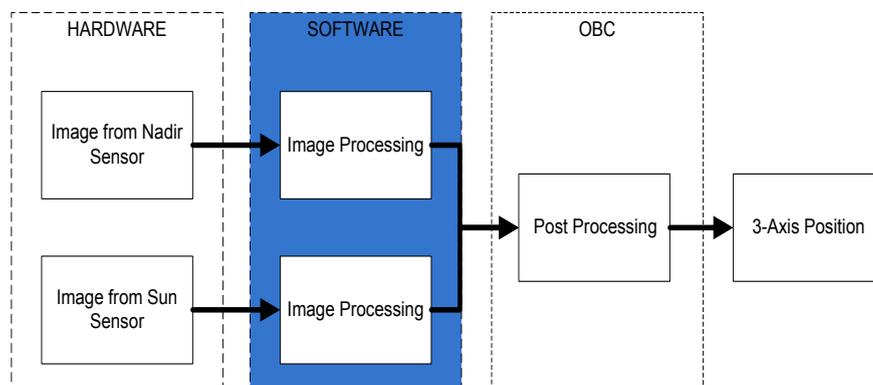


**Figure 3.13** – PCB layout

## Chapter 4

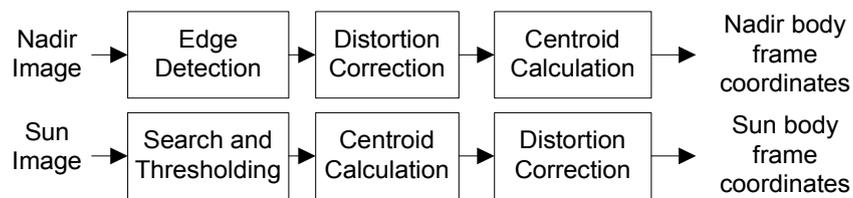
# Methodology: Algorithms for sun and nadir sensors

With the hardware design completed, the algorithms for the sun and nadir sensors are examined next.



**Figure 4.1** – Algorithm development for the sun and nadir sensors

The algorithms that were examined for the sun and nadir sensors are image processing algorithms. The goal is to determine the centroid of the Earth and the Sun from the images taken by the nadir and sun sensors, respectively, with algorithms that are not too complex, do not require too much time to complete and can be implemented on the 8-bit microcontroller discussed in section 3.2.2.



**Figure 4.2** – Image processing for the sun and nadir sensor

The main processes of determining the Earth and Sun's centroid are shown in figure 4.2. The image processing of both sensors consist of the following:

- Thresholding - to determine whether the image pixels are part of the background or foreground.
- Edge detection - to determine where the Earth and the Sun is.
- Distortion correction - to correct the effect of the fisheye lens on the image.
- Centroid calculation - to calculate the body frame coordinates of the nadir and Sun directions.

## 4.1 Thresholding of images

From the images of their respective sensors, a distinction between the object or foreground (the Earth and the Sun) and the background (space) is required to be identified. The method selected is called thresholding. Thresholding is the simplest method of image segmentation, because the process of thresholding is to label the pixels of an image as "object" or "background" pixels, depending on whether they are above or below a set threshold [27].

### 4.1.1 Fixed vs dynamic threshold

There are two types of thresholds namely a fixed threshold and a dynamic threshold. A dynamic threshold is a threshold that varies between segments of an image [28] or, in the case of the sun and nadir sensors, varies over sample images. The threshold is in this case a function of the average luminance of the image. Luminance is an indicator of how intense light is reflected [29]. The average luminance is calculated by:

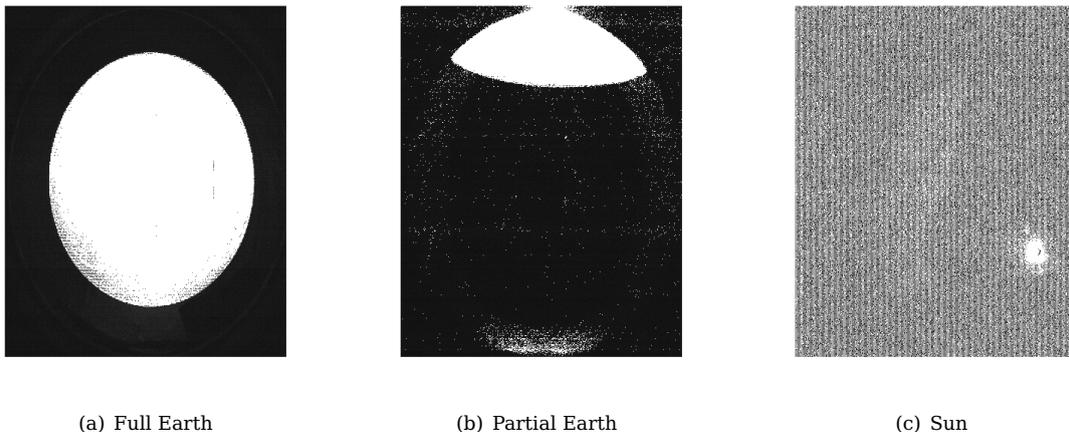
$$\text{average luminance} = \frac{\sum_{i=0}^{N-1} \text{pixel}_{lum}(i)}{N} \quad (4.1.1)$$

where N is total amount of pixels and i is the current pixel being compared to the threshold.

Two problems occur with this method of thresholding:

- Dynamic thresholding requires more time to complete.
- The average luminance of an image may be too low to be able to distinguish accurately between object and background pixels.

The average luminance requires more time to calculate, because the entire image's pixels are used. The average luminance has a direct link to the balance between object and background pixels. Figure 4.3 shows how errors may occur with dynamic thresholding. Figure 4.3(a) and 4.3(b) show the difference in object determination for a full profile and a partial profile of the Earth, while figure 4.3(c) shows the object determination for the Sun when using dynamic thresholding.

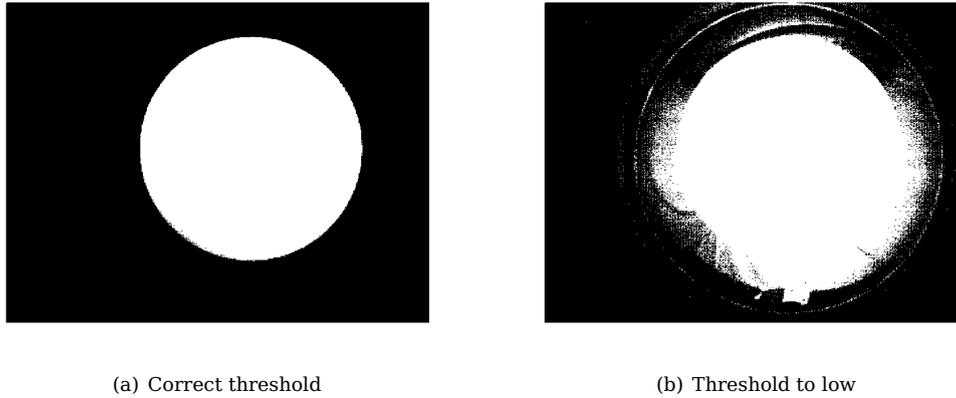


**Figure 4.3** – Errors occurring with dynamic thresholding

In the case of the nadir sensor, when the full profile of the Earth is in view, the average luminance will be higher, because the Earth will represent a large percentage of the image. The average luminance will be a good estimate for the threshold. However, as the Earth moves toward the edge of the FOV, the average luminance will decrease, because the space pixels will represent more of the image and therefore play a bigger role in the average luminance. This will in turn decrease the threshold and increase the chance for "space" pixels being mistaken as "Earth" pixels. The same errors will occur with the sun sensor, as the

Sun will always represent a very small percentage of the whole image.

Using a fixed threshold will not increase the time required for differentiating the object from the background. The only error that may occur will be when the threshold is set too low or too high, creating an error where background pixels are mistaken for object pixels or vice versa. Figure 4.4 shows the error that occurs when the threshold is set too low.

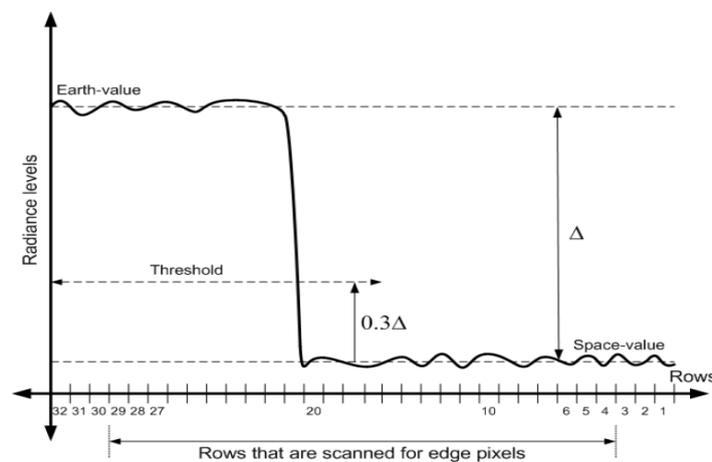


**Figure 4.4** – Errors occurring with fixed thresholding.

However, this error can be corrected easily with software on the microcontroller and does not require any extra time to compute. The only time required will be to rewrite a register in the microcontroller that holds the threshold value. Fixed thresholding is selected above dynamic thresholding to distinguish the object from the background. Therefore fixed thresholding is implemented in this project.

#### 4.1.2 Choosing a fixed threshold

Figure 4.5 shows how the fixed threshold is calculated, [30]. The threshold is calculated by the difference in luminance between the object (the Earth or the Sun) and the background (space). The difference between the object and the background is  $\Delta$ . By calculating  $\Delta$  between the luminance level of the object and of the background and adding 30% of that  $\Delta$  to the luminance level of the background, a threshold is calculated that can distinguish between the object and the background of an image.



**Figure 4.5** – Calculating a fixed threshold [30]

## 4.2 Edge detection

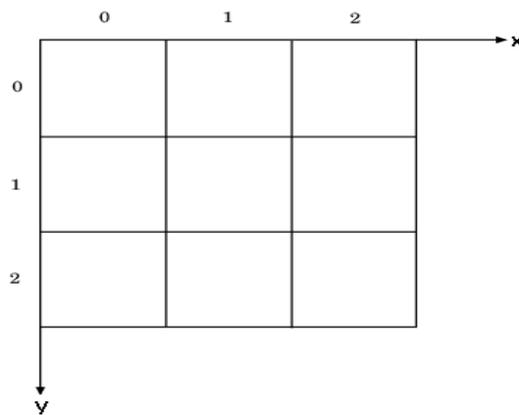
Thresholding is the first step of edge detection. As discussed in 4.1, thresholding only determines whether pixels are part of the object or part of the background. Edge detection finds the first pixels to distinguish the object from the background. These pixels are referred to as the edge profile of the object (either the Earth or the Sun).

The edge detection of the sun and nadir sensors consist of two processes:

- a search algorithm to search for edge pixels
- the actual edge detection method

### 4.2.1 Searching for edge pixels

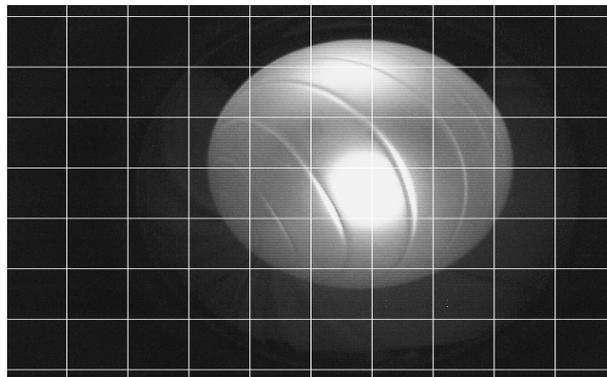
Search algorithms are implemented on the sun and nadir sensors to shorten the time required to find the necessary edge pixels required for the centroid calculations. Before a search algorithm can be implemented, a coordinate system must be defined on which search algorithms are implemented. Figure 4.6 illustrates how the pixel coordinate system has its origin in the top left corner with the x-axis from left to right and the y-axis from top to bottom.



**Figure 4.6** – The pixel coordinate axes

#### 4.2.1.1 Searching algorithm for the nadir sensor

To search through the entire image for the Earth is a time consuming and unnecessary process. A sequential search is used to "sample" the image at distinctive points to form a "grid search", as shown in figure 4.7.



**Figure 4.7** – The "grid search" algorithm for the nadir sensor

The "grid search" consist of two sequential searches. One search is performed on the horizontal axis of the image and the other on the vertical axis. The horizontal search finds edge pixels specifically determined to calculate the x-coordinate of the Earth's centroid. The vertical search has the same purpose, but for the y-coordinate of the Earth's centroid. The difference is that the horizontal search keeps the image's rows constant when searching pixels in specific columns and vice versa for the vertical search.

Both searches follow the algorithm:

$$Edge\ searching = \sum_{i=0}^{\frac{N}{K}-1} \sum_{j=0}^{\frac{M}{K}-1} pixel(i \times K, j \times K) \quad (4.2.1)$$

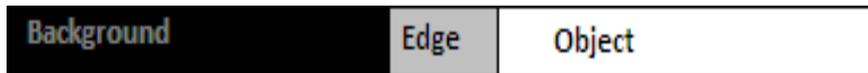
where N and M are the width and height of the image, K is the sampling factor and  $i \times K$  and  $j \times K$  are the row and column coordinates of the pixel that will be tested for an edge. K is the factor used to determine the speed at which the algorithm is completed. For example: if K is 10, it will mean that the search will be 100 times faster than searching the entire image. Each K'th pixel will be referred to as a sample pixel.

#### 4.2.1.2 Searching algorithm for the sun sensor

When in the FOV, the Sun will always occupy a very small percentage of the image. The search algorithm for the Sun consists of finding the first pixel that is above a threshold set for the sun sensor. The time to complete this process depends on where on the image the Sun is located. The search starts at the top left corner of the image. If the Sun is close to this point, the search will take less time than when the Sun is further away. A solution to this problem is discussed in section 6.6.1.

#### 4.2.2 Edge detection for the nadir sensor

The sampled pixels consist of background, object and edge pixels as shown in figure 4.8. The edge detection is used to distinguish and extract the edge pixels from the background and object pixels by using the background and object pixels.

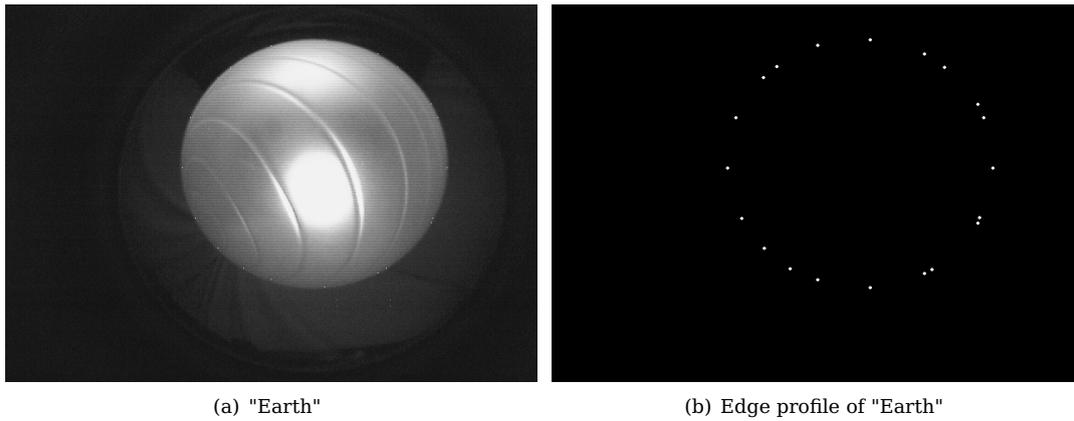


**Figure 4.8** – Difference between background, edge and object pixels

Edge detection uses two rules to determine if a pixel is a background, object or edge pixel:

- If the current sample pixel is above the threshold set for the nadir sensor (an Earth object pixel) and the previous sample pixel is below the same threshold (a space background pixel), then the edge pixel that lies between these two sample pixels is an edge pixel on the left hand side of the Earth's profile.
- If the current sample pixel is below the threshold set for the nadir sensor and the previous sample pixel is above the same threshold, then the edge pixel that lies between these two sample pixels is an edge pixel on the right hand side of the Earth's profile.

The same rules apply for the vertical search to find edge pixels at the top and at the bottom of the Earth's profile. The actual edge pixels are then searched for between the points that follow the two rules. An extra smaller sequential search is done between sampled pixels that comply to one of these rules, to find the specific edge pixel. Figure 4.9 shows the final edge detection.



(a) "Earth"

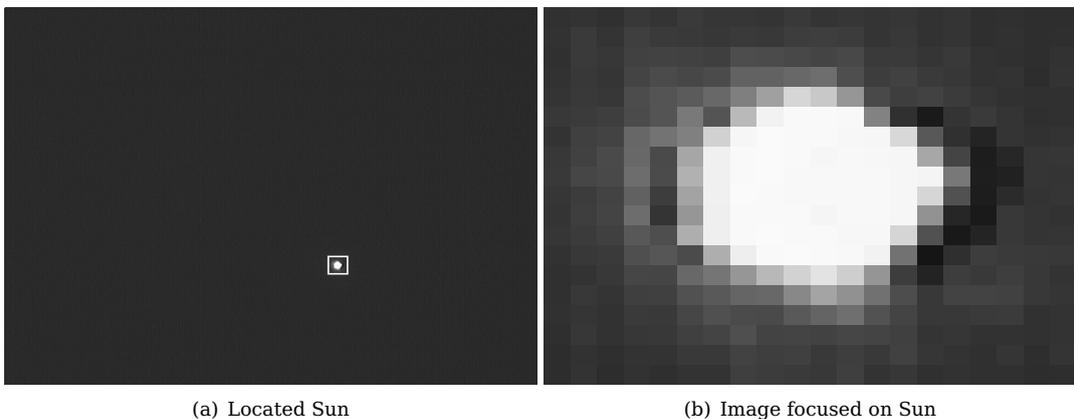
(b) Edge profile of "Earth"

**Figure 4.9** – The edge pixels of the Earth's profile

The edge pixels more or less form pairs of two. One pixel will give accuracy in the horizontal direction and the other in the vertical direction. It is also clear that the selected edge pixels form a good profile of the Earth.

### 4.2.3 Edge detection for the sun sensor

As mentioned in section 4.2.1.2 the first Sun pixel is searched for to determine where the Sun is located on the image.



(a) Located Sun

(b) Image focused on Sun

**Figure 4.10** – The area searched for Sun pixels

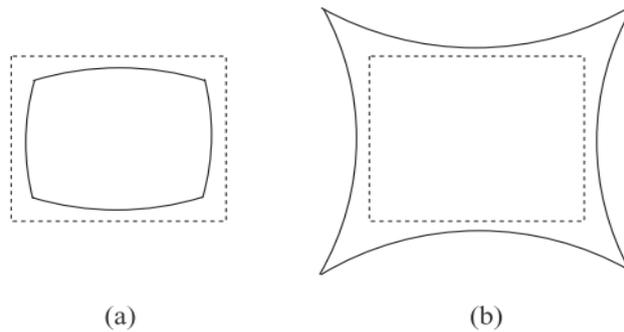
Figure 4.10 shows the small area that is selected to search for Sun pixels (pixels above the sun sensor's threshold). The edge detection consists of finding all the pixels above the threshold within that small area. Where the area is located, is determined by the first Sun pixel. It is a small area with respect to the whole image to search and is therefore completed faster.

## 4.3 Distortion

All optical lenses have distortion, either because of imperfections in the material, or the shape of the lens. Telephoto lenses have distortion near the edge of the field of view. Wide angle lenses, such as the fisheye lens, have a distortion that is more visible throughout the entire field of view. Most distortions are radial, which means that the distortion increases with distance from the distortion centre point. The distortion centre point is where the distortion is at a minimum. The two most common radial distortions are barrel

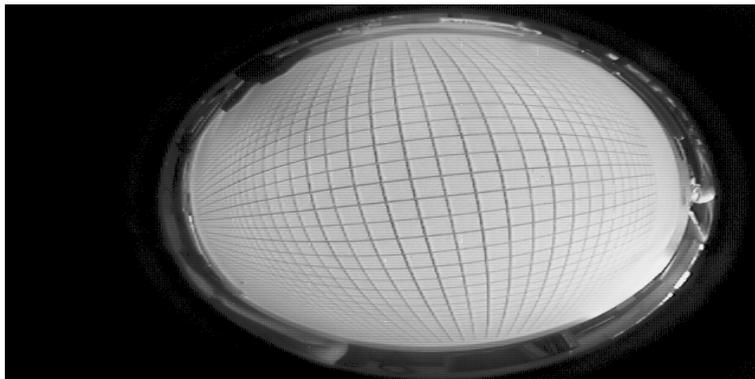
and pincushion distortion, where pincushion is the opposite of barrel distortion.

To understand the mechanism of distortion, the effect of a lens on the radii in the original image is examined. Barrel distortion shortens radii of the image closer to the edge of the field of view. The opposite is true for pincushion distortion. Figure 4.11 shows the two distortions.



**Figure 4.11** – Radial distortions. The dashed rectangle indicates the original image. (a) Barrel distortion and (b) Pincushion distortion

Figure 4.12 shows the barrel distortion of the fisheye lens used.



**Figure 4.12** – Fisheye lens distortion

The image taken in figure 4.12 is of a sheet of paper with horizontal and vertical lines printed on it. The fisheye lens used for the sensors is convex and therefore barrel distortion is expected. It is clear that the distortion of the fisheye lens is barrel distortion, because the lines get shorter as they move closer to the edge of the fisheye lens. To correct the distortion of the lens, a distortion model is made. This model is then used to create a distortion correction model to correct optical errors.

### 4.3.1 Distortion model

From [31] barrel distortion can be modelled by using a polynomial distortion model:

$$r_d = r_u + k_1 r_u^3 + k_2 r_u^5 + \dots \quad (4.3.1)$$

For smaller FOV lenses, the model can be simplified to its first and third terms. The model only uses odd order terms that might not model the distortion correctly. However, in this application, fisheye lenses are used for the sun and nadir sensors. The same polynomial distortion model can be used, but then a fifth or seventh order polynomial is required. Devernay and Faugeras [32] propose a different model.

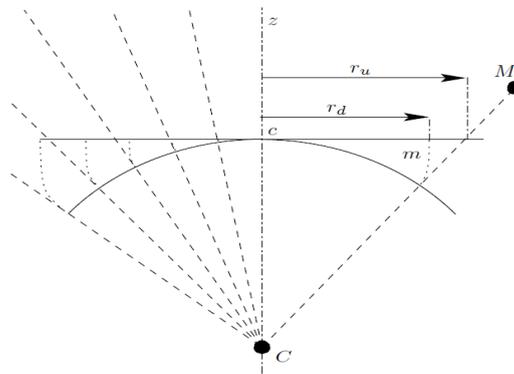
Basu and Licardie [33] propose the following Fish-eye transform or FET:

$$r_d = s \cdot \log(1 + \lambda r_u) \quad (4.3.2)$$

where  $r_d$  and  $r_u$  are the distorted and undistorted radii respectively,  $s$  is a simple scaling factor and  $\lambda$  is a value that corresponds to the amount of distortion. Basu and Licardie [33] mentions that the the FET is based on a simplification of the complex logarithmic mapping. The foveal region is projected at very high resolution, while resolution decreases continuously in the periphery source. The FET takes into consideration that the distortion can be nonlinear and increases dramatically towards the edge of the field of view. Basu and Licardie [33] also propose a polynomial FET, or PFET, that will better model the distortions of a fisheye lens:

$$r_d = a_0 + a_1 r_u + a_2 r_u^2 + \dots + a_n r_u^n = \sum_{i=0}^n a_i r_u^i \quad (4.3.3)$$

Devernavy and Faugeras [32] propose another distortion model, the FOV distortion model, which is based on the design of fisheye lenses. They propose that there is a relationship between the distorted radius and the angle between the optical axis and the distance between the optical point and the undistorted point.



**Figure 4.13** – FOV distortion model [32]

Figure 4.13 shows the relationship between the distorted radius  $r_d$  or line  $cm$  and the proposed angle between line  $CM$  and the optical axis. The equation that represents this relationship is:

$$r_d = \frac{1}{\omega} \arctan\left(2r_u \tan \frac{\omega}{2}\right) \quad (4.3.4)$$

where  $\omega$  represents the field of view that corresponds with an ideal fisheye lens. If  $\omega$  does not correspond to the real fisheye lens' field of view, the real fisheye lens may not be able to follow the model accurately.

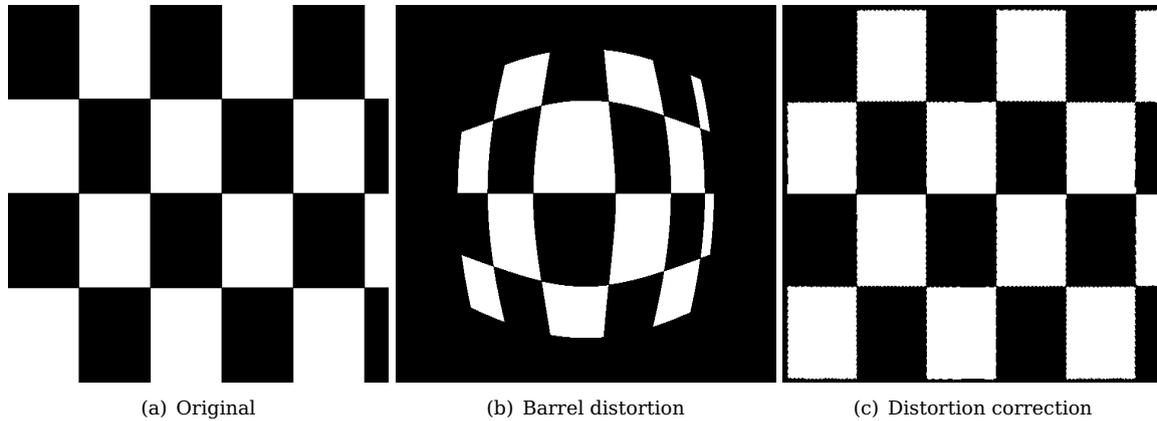
Amongst all the models available, the PFET was selected, because of the following reasons:

- The FOV distortion model may not be able to follow the correct model with only one variable to determine the form of the model and singularities that may occur with the tan and arctan terms.
- The polynomial distortion model is more accurate for smaller angle lenses, as the amount of terms in the equation is limited to a minimum, but it may not model the distortion correctly for wider angled lenses.
- The FET is a suitable model to use, but the PFET model performs better than the FET.

The tests in the next chapter will confirm that PFET was the most suitable choice for modelling the distortion.

After the distortion model has been calculated, the distortion correction model is derived by using the inverse form of the distortion model.

For example: in the case of barrel distortion, pincushion distortion is implemented to correct the errors introduced by the barrel distortion.



**Figure 4.14** – Distortion correction

Figure 4.14 shows an example of barrel distortion that is applied to a checkerboard pattern and then corrected with pincushion distortion. The FET distortion model is used for this example.

## 4.4 Centroid calculation

### 4.4.1 Centroid calculation for the nadir sensor

After the Earth's edge pixels have been corrected through the distortion correction model, the corrected pixels will be used to calculate the centroid of the Earth. Three algorithms were investigated to calculate the centroid of the Earth:

- a least squares estimation procedure that is used for a moon sensor application
- a geometry method of calculating a circle's centroid
- a circle least squares method

#### 4.4.1.1 Least squares estimation

Belezan, Mortari and Perfetti [34] used a least squares estimation on the moon to determine the attitude of a satellite. This method is suitable for a nadir sensor, as this method uses the circular profile of the moon to calculate the centroid. For the nadir sensor, the Earth will have a circular profile when in full view of the sensor. The method starts with calculating the error of each circle radius from the edge pixels:

$$e_i = \sqrt{(x_i - x_c)^2 + (y_i - y_c)^2} - R \quad (4.4.1)$$

where  $R$  is the radius of the moon and  $(x_c, y_c)$  and  $(x_i, y_i)$  are the unknown coordinates of the centre of the moon and the  $i$ 'th edge pixel respectively. Since the moon is far away from the sensor, it is assumed that the moon's radius is constant. If not, the equation would have to take into consideration the flux in the moon's radius as the satellite moves towards and away from the moon.

The mean value of the error is defined as:

$$\bar{e}(x_c, y_c) = \frac{1}{N} \sum_i^N e_i \quad (4.4.2)$$

where  $N$  is number of edge pixels. The standard deviation of the error is:

$$\sigma(x_c, y_c) = \sqrt{\frac{1}{N} \sum_i^N (e_i - \bar{e})^2} \quad (4.4.3)$$

The centroid is calculated by minimizing the standard deviation. The method used is the Nelder-Mead simplex method which approximates a local optimum of a problem with  $N$  variables when the objective function varies smoothly and is unimodal.

The least square estimation procedure is a heuristic method, which means that it is an iterative method and it can converge to non-stationary points. The computational power required for this method will be too much to comply with the selected microcontroller.

#### 4.4.1.2 Equation of a circle from three points

Bourke [35] proposes a method where the centroid of a circle can be calculated by using only three points on the edge of the circle.

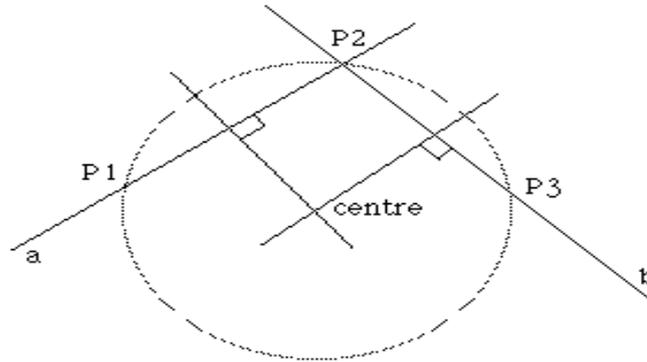


Figure 4.15 – Equation of a circle with three points [35]

Figure 4.15 shows how, using only three edge pixels, its possible to calculated a circle's centroid, where  $a$  and  $b$  are the two chords and  $P1$ ,  $P2$  and  $P3$  are the three edge pixels. This method uses the rule that when taking any chord of a circle and drawing a line from the center of the chord that is perpendicular to that chord, it will go through the center of the circle. Using a second chord and applying the same rule, the effect will be that the point where the two perpendicular lines cross, will be the circle's centroid.

The line equations for lines  $a$  and  $b$  are:

$$y_a = m_a(x - x_1) + y_1 \quad (4.4.4)$$

$$y_b = m_b(x - x_2) + y_2 \quad (4.4.5)$$

where  $m_a$  and  $m_b$ , the gradients of there respective lines, are:

$$m_a = \frac{y_2 - y_1}{x_2 - x_1} \quad (4.4.6)$$

$$m_b = \frac{y_3 - y_2}{x_3 - x_2} \quad (4.4.7)$$

For a perpendicular line, the gradient is the inverse and negative of the original line. The two perpendicular lines are then:

$$y'_a = -\frac{1}{m_a} \left( x - \frac{x_1 + x_2}{2} \right) + \frac{y_1 + y_2}{2} \quad (4.4.8)$$

$$y'_b = -\frac{1}{m_b} \left( x - \frac{x_2 + x_3}{2} \right) + \frac{y_2 + y_3}{2} \quad (4.4.9)$$

The centroid will be where these two lines cross; therefore where  $y'_a$  is equal to  $y'_b$ . Combining equation 4.4.8 with equation 4.4.9 to calculate the x coordinate of the circle's centroid:

$$x = \frac{m_a m_b (y_1 - y_3) + m_b (x_1 + x_2) - m_a (x_2 + x_3)}{2(m_b - m_a)} \quad (4.4.10)$$

The y coordinate can then be calculated by substituting the x coordinate in either equation 4.4.8 or 4.4.9. The result from the centroid calculation can be made more accurate when using multiple edge pixels, repeating the process a few times and then averaging them to obtain an averaged centroid for the circle.

This is a better method to use than the least squares estimation in the previous section. This method's computing speed is much faster, because it is not an iterative method. There are however flaws in this method. If the wrong three edge pixels are selected to calculate a centroid, singularities can form in equations 4.4.6 through to 4.4.10. These errors can however be corrected with simple search algorithms to find three matching edge pixels.

For example, if  $x_1$  and  $x_2$  were equal,  $m_a$  will be  $\infty$ . This will make x in equation 4.4.10 strive to  $\infty$  as well, but  $y'_a$  will become the average between  $y_1$  and  $y_2$  and x can then be calculated out of equation 4.4.9. A search algorithm can be implemented to take these singularities into consideration when choosing three edge pixels, but this will require extra time to compute.

Although this method will work in calculating the Earth's centroid, it is better used for a horizon sensor, where only a horizon of the Earth is visible in the field of view. This is because this method will work better with longer chords. This method does not use any optimization, unlike the least square estimation. If the edge pixels used in this method were determined incorrectly, the error will not be minimized in some way.

#### 4.4.1.3 Least squares circle

The third method which can be used for centroid calculation is the least squares circle (LSC) [36]. This method is almost a combination of the previous two methods mentioned. The LSC starts with the standard equation for a circle:

$$(x + A)^2 + (y + B)^2 = r^2 \quad (4.4.11)$$

where (-A,-B) is the circle's centroid and r is the radius. Equation 4.4.11 is expanded to the following:

$$\begin{aligned} x^2 + 2Ax + A^2 + y^2 + 2By + B^2 &= r^2 \\ x^2 + y^2 + 2Ax + 2By + C &= 0 \end{aligned} \quad (4.4.12)$$

the radius can be described as  $\sqrt{A^2 + B^2 - C}$ . According to the LSC, the total error for N points is:

$$\phi = \sum_{i=1}^N (x_i^2 + y_i^2 + 2Ax_i + 2By_i + C) \quad (4.4.13)$$

where  $x_i$  and  $y_i$  are the assumed edge pixels. To find the centroid (-A,-B), the partial derivatives of the total error  $\phi$  is minimized:

$$\frac{\partial \phi}{\partial A} = 2 \sum x_i^2 A + 2 \sum x_i y_i B + \sum x_i C + \sum (x_i^2 + y_i^2) x_i = 0 \quad (4.4.14)$$

$$\frac{\partial \phi}{\partial B} = 2 \sum x_i y_i A + 2 \sum y_i^2 B + \sum y_i C + \sum (x_i^2 + y_i^2) y_i = 0 \quad (4.4.15)$$

$$\frac{\partial \phi}{\partial C} = 2 \sum x_i A + 2 \sum y_i B + NC + \sum (x_i^2 + y_i^2) = 0 \quad (4.4.16)$$

The minimized partial derivatives of the total error is written in matrix form to obtain A,B and C as the object of the equation:

$$\underbrace{\begin{bmatrix} 2\sum x_i^2 & 2\sum x_i y_i & \sum x_i \\ 2\sum x_i y_i & 2\sum y_i^2 & \sum y_i \\ 2\sum x_i & 2\sum y_i & N \end{bmatrix}}_{\mathbf{M}} \begin{bmatrix} A \\ B \\ C \end{bmatrix} = - \underbrace{\begin{bmatrix} \sum (x_i^2 + y_i^2) x_i \\ \sum (x_i^2 + y_i^2) y_i \\ \sum (x_i^2 + y_i^2) \end{bmatrix}}_{\mathbf{K}} \quad (4.4.17)$$

$$\begin{bmatrix} A \\ B \\ C \end{bmatrix} = \mathbf{M}^{-1} \mathbf{K} \quad (4.4.18)$$

This method is more suited to a nadir sensor, where the Earth's full profile is visible in the field of view. Because this method uses error minimization, it is more suited to determine the Earth's centroid when only a partial profile is visible in the field of view.

#### 4.4.2 Centroid calculation for the sun sensor

The Sun's centroid is easier to calculate than the Earth's centroid. Since the Sun occupies a small part of the image, as previously mentioned, and its position was located by the search algorithms, the centroid can be calculated by taking the average of all the Sun's pixels within the closed area of the edge detection.

$$\text{Sun}_x = \frac{\sum_{i=1}^N \text{pixel}_{\text{value}}(i) \times \text{Sun pixel}_x(i)}{\sum_{i=1}^N \text{pixel}_{\text{value}}(i)} \quad (4.4.19)$$

$$\text{Sun}_y = \frac{\sum_{i=1}^N \text{pixel}_{\text{value}}(i) \times \text{Sun pixel}_y(i)}{\sum_{i=1}^N \text{pixel}_{\text{value}}(i)} \quad (4.4.20)$$

where  $\text{Sun}_x$  and  $\text{Sun}_y$  are the x and y coordinates of the Sun's distorted centroid,  $\text{Sun pixel}_x(i)$  and  $\text{Sun pixel}_y(i)$  are the x and y coordinates of a Sun pixel and  $\text{pixel}_{\text{value}}(i)$  is the value of each Sun pixel. All the Sun pixels are weighed against each other to calculate a more accurate centroid.

This centroid is the distorted centroid of the Sun. Distortion correction is done on this centroid to correct the error caused by the distortion of the fisheye lens.

## Chapter 5

# Experimental setups and calibrations

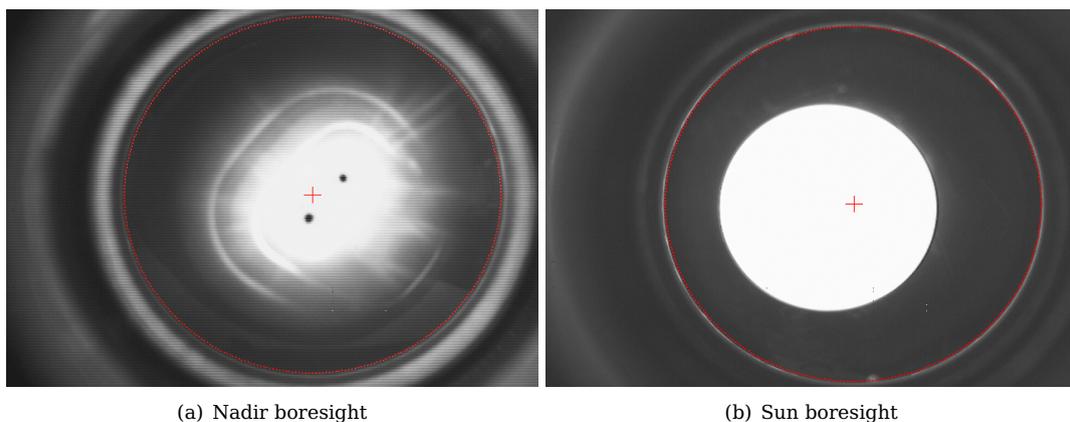
### 5.1 Camera calibration

The cameras used for this project must be calibrated before they can be used as sun and nadir sensors. Significant points in the image, such as the boresight of the lens and the distortion centre point, must be identified, as well as the distortion and distortion correction models. Setup parameters such as the focus and exposure time must also be determined.

The dominant colours for these sensors are black and white and the greyscale between them, where space is represented by black and all other celestial bodies are shades of grey and white. Therefore, the camera modules are implemented as monochrome cameras. The monochrome mode of the camera modules improve the resolution of the images. Only the luminance of each pixel is of importance for these sensors and therefore only the luminance from the YUV output from the cameras are used.

#### 5.1.1 Boresight

The first point of interest is where the boresight of the fisheye lens is situated on the image. The boresight is the optical centre of the lens. This point is important for centroid calculations and distortion corrections.



**Figure 5.1** - Determining the boresight of the nadir and sun sensor

The tests for determining the boresight of the nadir and sun sensors consist of shining a light directly onto the the fisheye lens, making the pixels saturate. A halo effect occurs when the light is reflected from the edge of the fisheye lens, as shown in figure 5.1. Using MATLAB, a circle with a specific radius, is placed on the images as shown in figure 5.1. The centre of the correctly positioned circle determines the position of the boresight of the lens.

The centre of the sensor image could not be assumed to be the position of the boresight, because the lens holder on the camera modules were positioned in such a way that a portion of the field of view did not fall on the pixel array of the image sensor. An example of this can be seen in figure 5.7(a). The lens holders were moved to the correct position. Epoxy is used to keep the lens holders in the correct position.

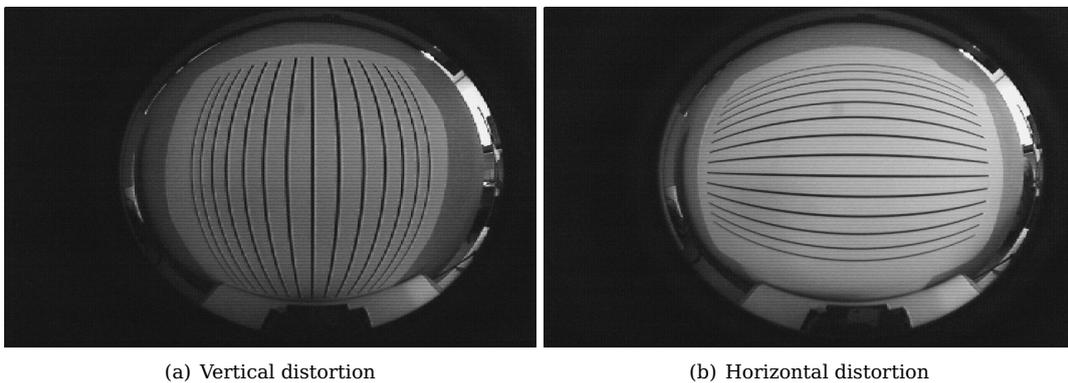
The results of the boresight tests are shown in Table 5.1.

	X centre pixel coordinate	Y centre pixel coordinate	Radius
Nadir sensor	371	236	226 pixels
Sun sensor	369	250	226 pixels

**Table 5.1** – Results from the boresight test

### 5.1.2 Distortion centre point

The distortion centre point is not only important for the calculation of the distortion model, but also for the centroid calculation. If the wrong distortion centre point is selected, the distortion correction will introduce an error in the calculation of the centroid. Figure 5.2 shows the test performed to find the distortion centre point.



**Figure 5.2** – Test for the distortion centre point

The tests consist of capturing images of horizontal and vertical lines printed on sheets of paper. The lines will bend as a result of the distortion from the fisheye lens, as seen in figure 5.2. The distortion centre point test seeks the horizontal and vertical lines closest to the distortion centre point, as these lines will be presented as straight, or nearly straight, lines.



**Figure 5.3** – Distortion centre point at different points on an image

Figure 5.3 shows how an image would distort at different distortion centre points. The checkered box is centred in the image for all three images. If the distortion centre point is at a different point than the

boresight of the lens and this difference is not taken into consideration, an error will be introduced in the centroid calculation, as the distortion correction will not be implemented correctly.

The result of the distortion centre point test indicates that the distortion centre point and boresight of the fisheye lens are situated at the same point. The test is only performed on the nadir sensor, but since a similar camera and lens are utilised for both nadir and sun sensor, it is assumed that the same result for the distortion centre point test would be observed for the sun sensor.

### 5.1.3 Focus

#### 5.1.3.1 Nadir sensor: focal length

To see if the camera is in focus, the focal length of the lens is calculated. The focal length is a measurement of the distance where light rays will be converged or focussed between the lens and the image plane [37]. Appendix A.3 shows that the fisheye lens has a focal length of 1.24 mm. Figure 5.4 shows the setup for calculating the focal length.

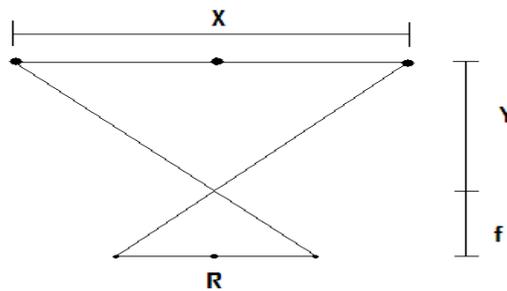


Figure 5.4 – Calculating the focal length

The three points in figure 5.4 represent three points on a white board. There is a distance of 10 cm between each point. X represents the total distance of 20 cm, Y is the distance between the white board and the camera lens, f is the focal length, and R is the distance between the three points displayed on the image. R is calculated by counting the pixels between the points and then multiplied with the length of a pixel,  $7.6 \mu\text{m}$ .

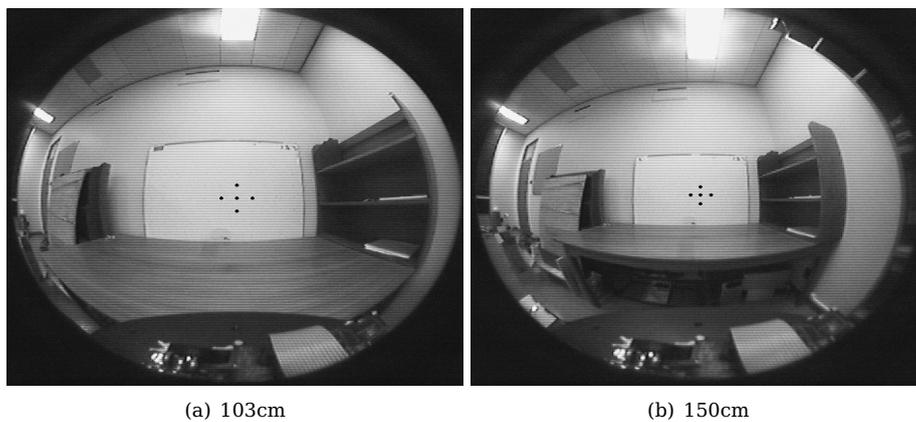


Figure 5.5 – Calculating the focal length at 103 cm and 150 cm respectively

The test is performed near the distortion centre point of the lens and at various distances from the whiteboard, as shown in figure 5.5, to prevent distortion of affecting test results and to see if the focal length would stay constant over longer distances.

The relationship between the distance between the points on the board and the distance between the board and the camera, should be equal to the relationship between the distance between the points on the image plane and the focal length. The focal length is therefore calculated by the following ratio:

$$\frac{X}{Y} = \frac{R}{f} \quad (5.1.1)$$

Table 5.2 shows the results of the focal length tests at different distances between the camera and the whiteboard. The focal length stays constant over all the distances that were measured. The focal length is found not to be 1.24 mm, but 1.25 mm. This difference is minimal and the camera is therefore considered infocus.

Distance from centre point	Undistorted distance between points	Focal length
44.5 cm	0.2812 mm	1.25 mm
103 cm	0.1216 mm	1.25 mm
150 cm	0.0836 mm	1.25 mm
206.5 cm	0.0608 mm	1.25 mm

**Table 5.2** – Results from the focal length test

### 5.1.3.2 Sun sensor: the Sun's radius

A different test is performed to see if the sun sensor is in focus. The sun sensor is pointed towards the Sun and images are produced. The test is performed to measure the diameter of the Sun. In section 5.5 a method is described to determine the Earth's radius seen from a specific distance in space. The same method can be implemented to determine the Sun's radius. The Sun's diameter is approximately  $0.53^\circ$  as seen from Earth. Table 5.3 shows the Sun's diameter from the images in figure 5.6. The decrease in the diameter due to the distortion of the fisheye lens, can be expected.

Position	Radius of Sun
Boresight	8 pixels
Near FOV edge	6 pixels

**Table 5.3** – Sun's radius at different positions

However,  $0.53^\circ$  should be about 2 to 4 pixels, depending on the distortion. This indicates that the sun sensor is not focused.



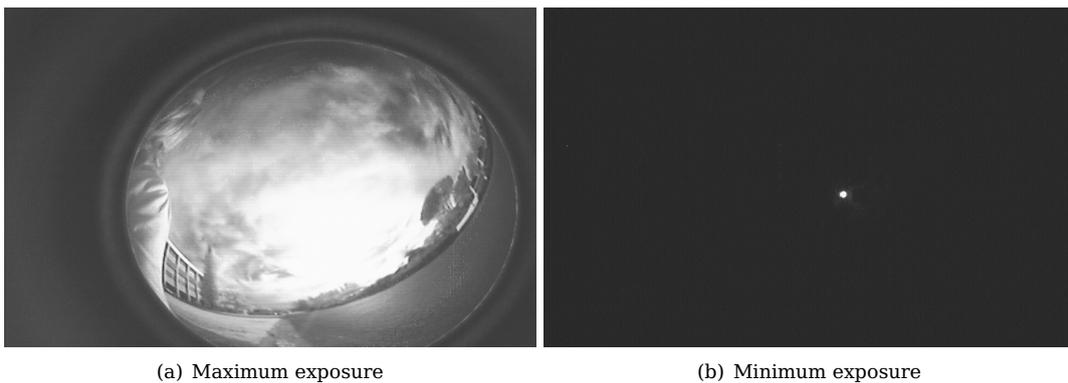
**Figure 5.6** – Change in the Sun's radius between the boresight and the FOV edge

But since both sun and nadir sensors have the same FOV radius of 226 pixels, an assumption is made that the sun sensor has the same focal length as the nadir sensor. The error in the increased diameter of the Sun relative to the camera, may have to do with the pixels saturating and creating an overflow to neighbouring pixels, which results in the Sun being enlarged on the image.

### 5.1.4 Exposure time

Exposure time is the time allowed for the photons to be accumulated on the photographic medium [38], in this case the pixel array of the image sensor. The longer the exposure time, the better contrast will be visible on the image, but unwanted effects, for example blurring, may be visible if an object is moving at a high speed relative to the camera. The shorter the exposure time, the less contrast is visible and the less blurring will be visible as well.

Figure 5.7(a) shows an image captured with the sun sensor (and therefore through the neutral density filter) with the exposure time set to automatic. The sensor is pointed towards the Sun. The neutral density filter filters out the light intensity of the Sun enough to display different objects on the image. There is however, still too much light falling on the pixel array of the image sensor. Only the Sun must be visible on the image. Experimenting with the exposure time, images were "filtered" down to the image in figure 5.7(b), where only the Sun is visible. The exposure time for figure 5.7(b) is 127  $\mu$ s.



**Figure 5.7** – Maximum and minimum exposure time

## 5.2 Distortion

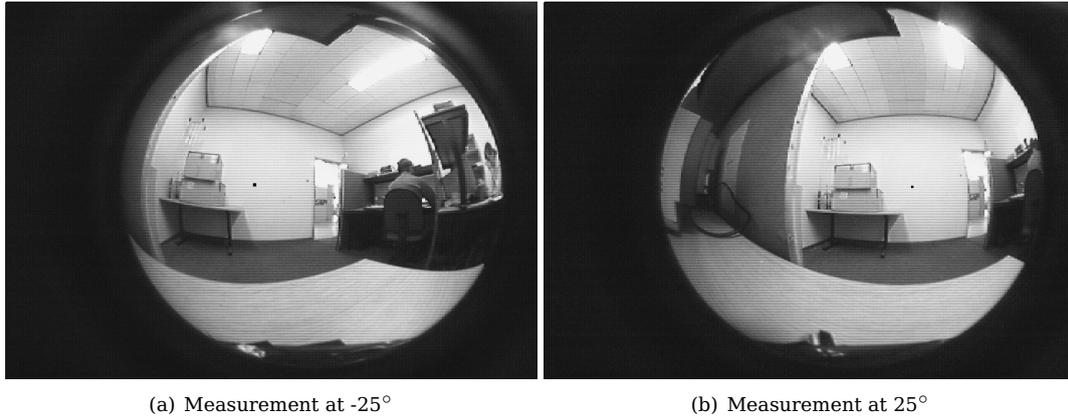
### 5.2.1 Distortion model

To determine the distortion model of the fisheye lens accurately, a metric rotary stage is utilised. The metric rotary stage provides accurate  $1^\circ$  rotations. The tests performed for the distortion model use these rotations to measure the rotation relative to the image, and compare them to the rotation of the metric rotary stage. The camera is mounted on the metric rotary stage. Figure 5.8(a) shows the metric rotary stage utilised and figure 5.8(b) the setup with the camera.



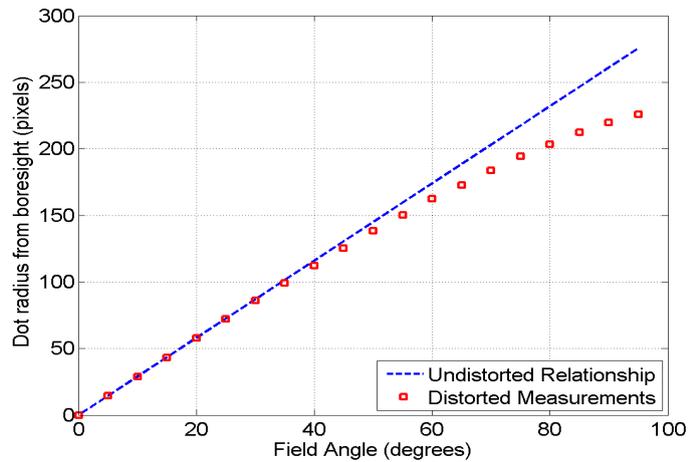
**Figure 5.8** – Metric rotary stage used in tests

During the test the camera follows a black dot on a wall far away from it. The reason for placing the camera far away from the black dot is to ensure the same approximate distance to the dot when the camera rotates. The camera is positioned initially to ensure the black dot to be centred on the boresight/distortion centre point of the lens. The metric rotary stage is then rotated with increments of  $5^\circ$  and the pixel coordinate of the black dot is documented. The increments have a range from  $-95^\circ$  to  $95^\circ$  (a  $190^\circ$  FOV). Figure 5.9 shows images captured at  $-25^\circ$  and  $25^\circ$ .



**Figure 5.9** – Images of horizontal distortion measurements

The results of the distortion model test is shown in figure 5.10. The dashed line represents the relationship between the radial distance and field angle if no distortion is visible. The points represents the measured radial distance at the specific  $5^\circ$  field angle intervals. The ideal undistorted result would be when the points follow the dashed line. It is evident from figure 5.10 that there is distortion and the distortion increases when closer to the edge of the field of view.



**Figure 5.10** – Radial distortion of fisheye lens

The dashed line is a straight line and is therefore represented by the straight line equation:

$$y = mx + c$$

$$c = 0$$

$$m = \frac{[\text{radius at } 5^\circ] - 0}{5^\circ - 0^\circ}$$

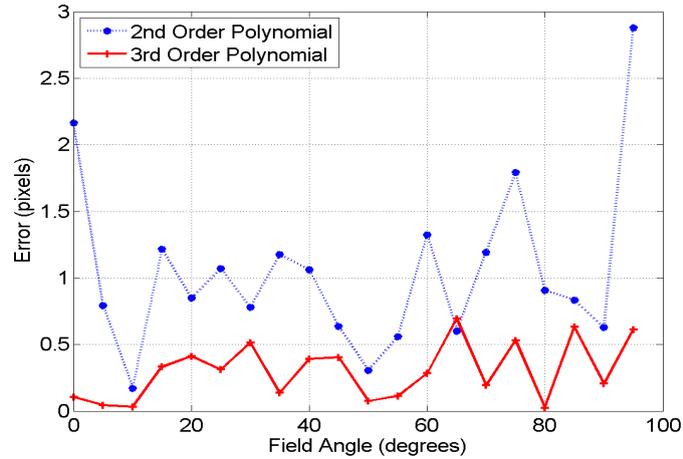
$$y = 2.9x$$

(5.2.1)

with  $m$  the gradient and  $c$  the intercept. Since the line begins at the origin,  $c$  is zero. The gradient is calculated by using the distorted radius measured at  $5^\circ$ , as it is assumed that the distortion at this point

is negligible and represents the linear relationship between the radius from the boresight and the field angle.

MATLAB's `polyfit()` function is implemented to determine the PFET model for the distortion model. The function has a parameter that determines the order of the polynomial.



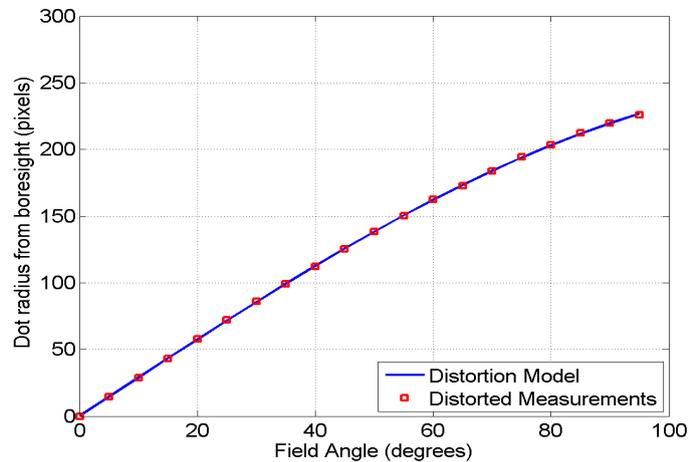
**Figure 5.11** – Error between simulated and measured Distortion Models

Figure 5.11 indicates the error between the measured distortion model and the 2nd and 3rd order polynomial. A 3rd order polynomial was selected since the error between it and the measured distortion is significantly smaller than that of a 2nd order polynomial. Taking the values given by MATLAB, the distortion model is selected as:

$$r_d = ar_u^3 + br_u^2 + cr_u + d \quad (5.2.2)$$

$$r_d = (-2.5572 \times 10^{-6})r_u^3 + (7.6076 \times 10^{-5})r_u^2 + 0.9953r_u + 0.1036 \quad (5.2.3)$$

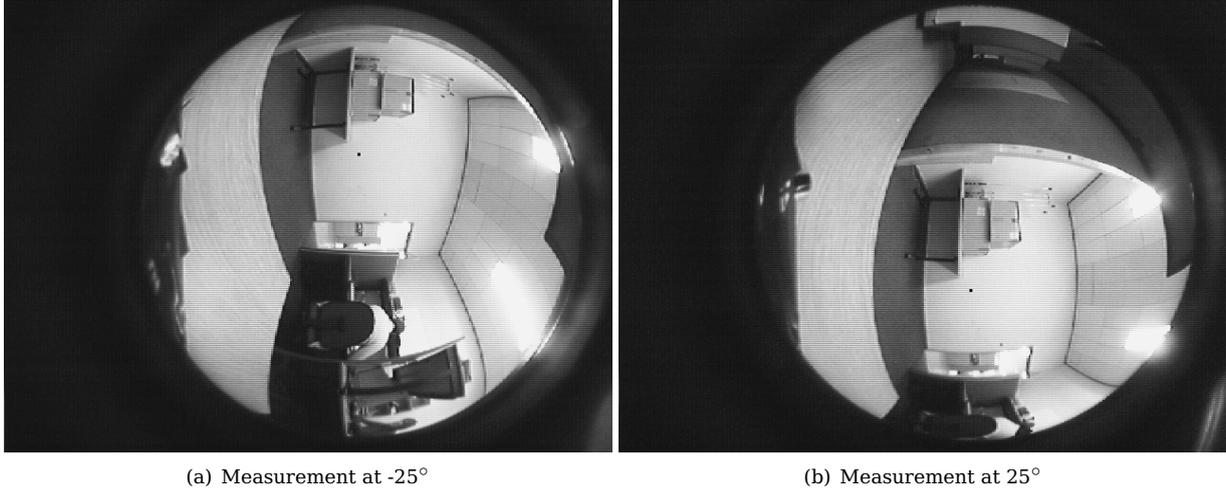
Figure 5.12 shows the PFET distortion model. The distortion model follows the measured distortion accurately.



**Figure 5.12** – PFET distortion model

Figure 5.12 shows the result for the horizontal radial distortion. For the vertical radial distortion, the same test was done, with the same equipment and setup, but the camera is rotated by  $90^\circ$ . This results in the effect that the black dot is moving in a vertical direction, shown in figure 5.13. The results, however,

are identical to that of the horizontal radial distortion.



**Figure 5.13** – Images of vertical distortion measurements

## 5.2.2 Distortion correction model

With the distortion modeled, the distortion correction model can be calculated. The distortion correction model is calculated from the roots of the distortion model. The distortion model is rewritten in the following form from which the roots can be calculated:

$$\begin{aligned}
 r_d &= ar_u^3 + br_u^2 + cr_u + d \\
 e &= d - r_d \\
 0 &= ar_u^3 + br_u^2 + cr_u + e
 \end{aligned} \tag{5.2.4}$$

where  $e$  has now become the 4th element of the 3rd order polynomial. To find  $r_u$ , the roots of the 3rd order polynomial must be determined. Since the distortion model is a 3rd order polynomial, there must be three roots. The general form of these three roots are [39]:

$$\begin{aligned}
 r_{u1} &= -\frac{b}{3a} - \frac{1}{3a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2e + \sqrt{(2b^3 - 9abc + 27a^2e)^2 - 4(b^2 - 3ac)^3} \right]} \\
 &\quad - \frac{1}{3a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2e - \sqrt{(2b^3 - 9abc + 27a^2e)^2 - 4(b^2 - 3ac)^3} \right]}
 \end{aligned} \tag{5.2.5}$$

$$\begin{aligned}
 r_{u2} &= -\frac{b}{3a} + \frac{1 + j\sqrt{3}}{6a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2e + \sqrt{(2b^3 - 9abc + 27a^2e)^2 - 4(b^2 - 3ac)^3} \right]} \\
 &\quad + \frac{1 - j\sqrt{3}}{6a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2e - \sqrt{(2b^3 - 9abc + 27a^2e)^2 - 4(b^2 - 3ac)^3} \right]}
 \end{aligned} \tag{5.2.6}$$

$$\begin{aligned}
 r_{u3} &= -\frac{b}{3a} + \frac{1 - j\sqrt{3}}{6a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2e + \sqrt{(2b^3 - 9abc + 27a^2e)^2 - 4(b^2 - 3ac)^3} \right]} \\
 &\quad + \frac{1 + j\sqrt{3}}{6a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2e - \sqrt{(2b^3 - 9abc + 27a^2e)^2 - 4(b^2 - 3ac)^3} \right]}
 \end{aligned} \tag{5.2.7}$$

The nature of the roots are described by the discriminant of a 3rd order polynomial:

$$\begin{aligned}
 \Delta &= 18abcd - 4b^3d + b^2c^2 - 4ac^3 - 27a^2d^2 \\
 &= 1.01 \times 10^{-5}
 \end{aligned} \tag{5.2.8}$$

The discriminant of the distortion model is positive, which means that the distortion model has three distinct roots.

By simulating each root in MATLAB, it was found that only one of the three roots gave a result where distortion is being corrected. The distortion correction model is therefore:

$$r_u = -\frac{b}{3a} + \frac{1 + j\sqrt{3}}{6a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2(d - r_d) + \sqrt{(2b^3 - 9abc + 27a^2(d - r_d))^2 - 4(b^2 - 3ac)^3} \right]} + \frac{1 - j\sqrt{3}}{6a} \sqrt[3]{\frac{1}{2} \left[ 2b^3 - 9abc + 27a^2(d - r_d) - \sqrt{(2b^3 - 9abc + 27a^2(d - r_d))^2 - 4(b^2 - 3ac)^3} \right]} \quad (5.2.9)$$

The imaginary part of the equation is not a problem, as the discriminant shows that it is cancelled out. The results from MATLAB also show that the results from the distortion correction model is always real.

Figure 5.14 shows the distortion correction. The distortion model and measurements are sent through the distortion correction model and the result for both instances is the same. The correct distortion correction model has been calculated.

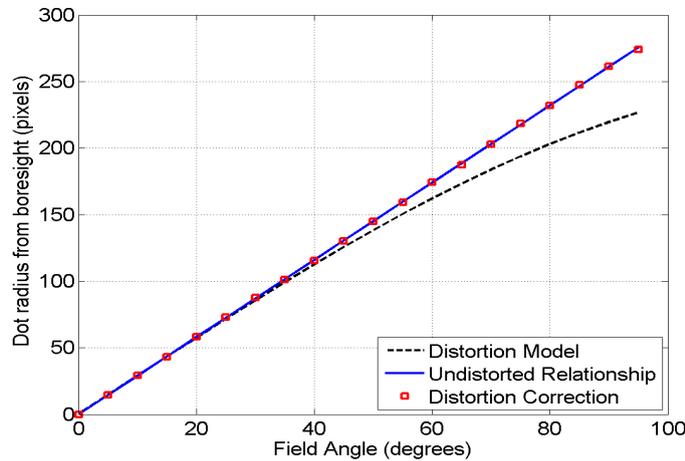


Figure 5.14 – Distortion correction

### 5.3 Resolution

The resolution for the sun and nadir sensors is defined as the amount of pixels required to represent a degree in body frame angles. The radius of the fisheye lens’ FOV is 226 pixels, table 5.1, with the FOV being 190°. This gives a linear effective resolution of 0.42° per pixel. This, however, is not the true resolution. Figure 5.15 shows the change in resolution.

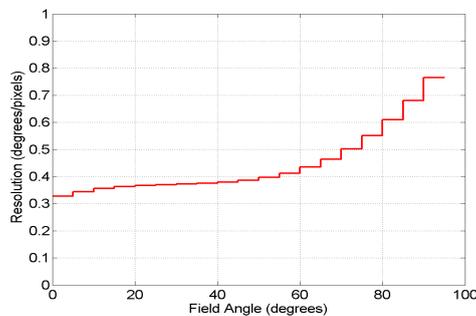
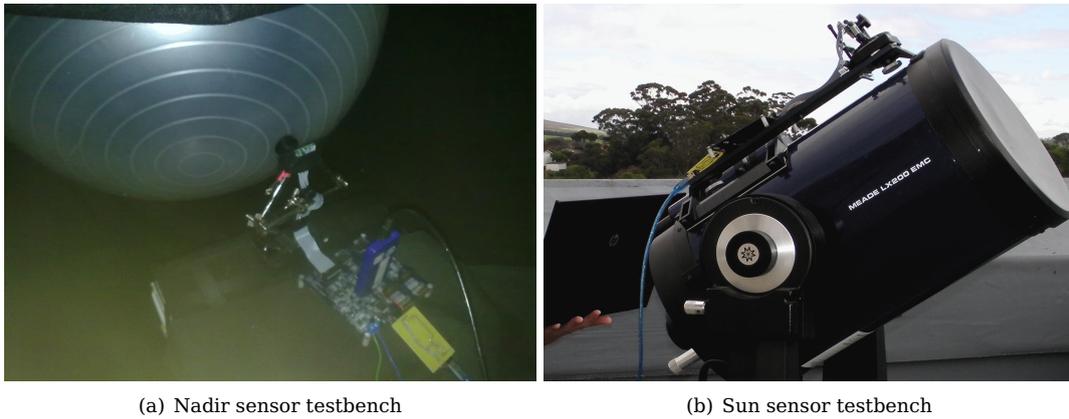


Figure 5.15 – Resolution over FOV

The resolution is calculated from the measurements taken to model the distortion. Since the distortion was measured every  $5^\circ$ , the resolution is represented in  $5^\circ$  steps. As expected, the resolution is worse at the edge of the FOV, as the radius that represents  $5^\circ$  at the edge of the FOV is shorter due to the distortion from the lens.

## 5.4 Testbenches

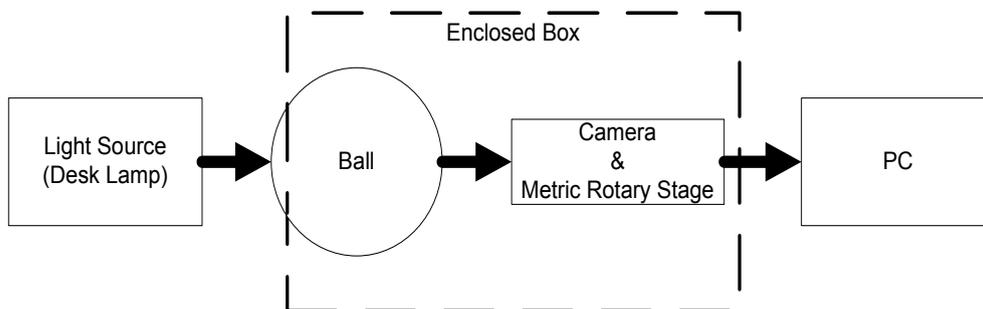
It will be difficult to calibrate and evaluate the sun and nadir sensors in their orbit environment. It is expensive to launch satellites and once the sensors are in space, it will be impossible to retrieve the sensors for adjustments. Testbenches are used to simulate the environment the sensors will be used in. Testbenches are much less expensive and it is easier to make adjustments and repeat tests. Figure 5.16 shows images of the sun and nadir sensors' testbenches.



**Figure 5.16** – Images of testbenches for the sun and nadir sensors

### 5.4.1 Nadir sensor testbench

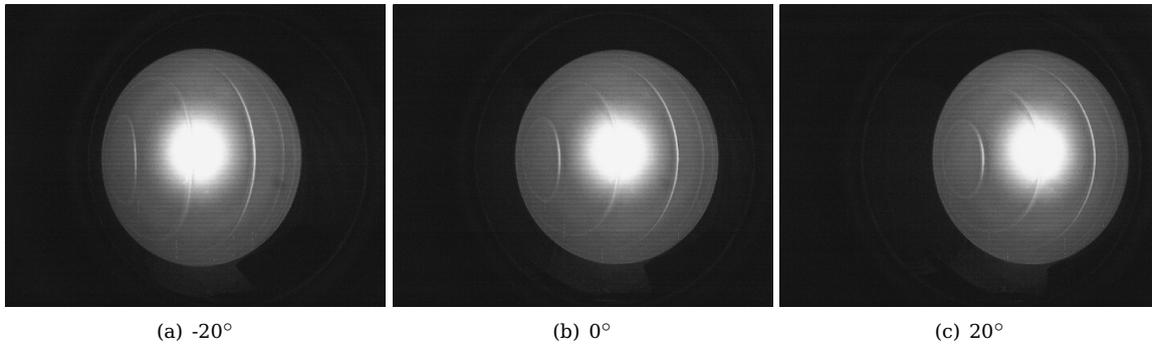
The nadir sensor's testbench is the most difficult to implement, because calibration errors can easily occur. The error occurrence will be explained in sections 5.5 and 5.6. Figure 5.17 shows the testbench implemented for the calibration of the nadir sensor.



**Figure 5.17** – Nadir Sensor testbench

The purpose of this testbench is to simulate an illuminated Earth and a dark space background. The simulation is implemented by using a clear aerobics ball, enclosed in a box lined with black cloth, and shining light from a desk lamp through it. The ball is illuminated from behind. Illuminating the ball from the front will reflect the light from the ball's surface onto the background, making the background more visible in the images. A hole is cut in the back of the box where a partial area of the ball fits through. The light is focused on this partial area of the ball. A metric rotary stage is rotated in the horizontal plane in

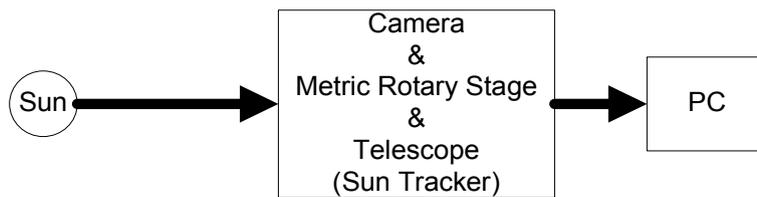
increments of  $1^\circ$  and measurements are then taken at each degree. The vertical axis is measured in the same way as the horizontal axis, but the nadir sensor is first rotated by  $90^\circ$ .



**Figure 5.18** – Images of the ball during rotation measurements

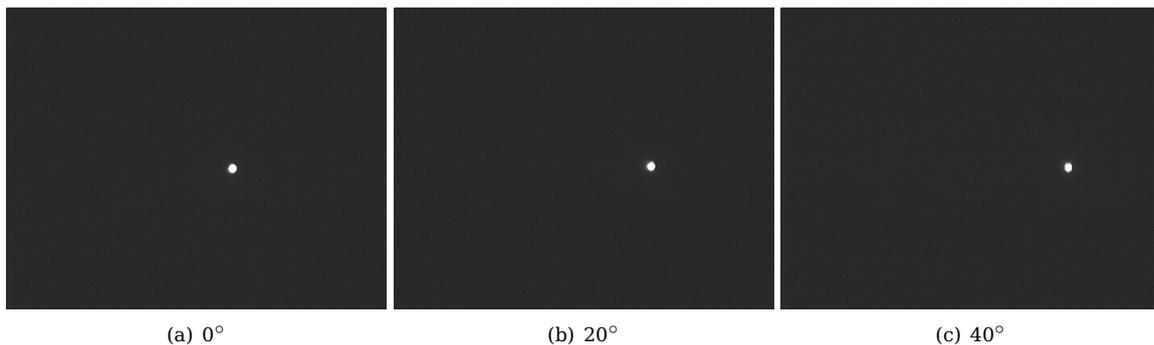
### 5.4.2 Sun sensor testbench

The testbench for the sun sensor is easier to set up than that of the nadir sensor, because there are less potential calibration errors, as shown in section 5.6, to take into consideration. Figure 5.19 illustrates the testbench implemented for the sun sensor.



**Figure 5.19** – Sun sensor testbench

Because the neutral density filter filters out most of the light, only displaying the Sun, the measurements made by the sun sensor will be very close to that observed by the sensor in space. The sun sensor is mounted on the metric rotary stage (for accurate degree rotations required for the measurements). The sensor and metric rotary stage is in turn mounted on a MEADE LX200 EMC telescope. The telescope is then used to track the Sun accurately.



**Figure 5.20** – Images of the Sun during rotation measurements

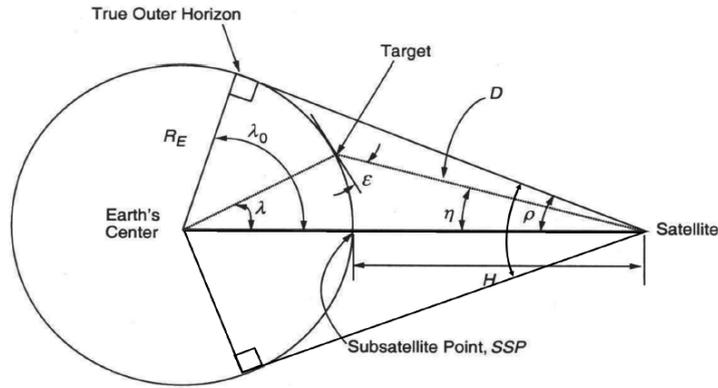
With the nadir sensor, the test object is fixed relative to the Earth. The Sun, however, is moving relative to the Earth. Therefore, the Sun moves in the images captured by the sensor and creates unwanted errors in

the calibration of the sun sensor. The telescope has the ability to track the Sun and will therefore cancel out the movement of the Sun, making the Sun stationary relative to the telescope and the sun sensor.

The sensor is positioned with the Sun at the boresight of the fisheye lens. The metric rotary stage is incremented by  $5^\circ$  in both directions and measurements are taken at each increment. The vertical axis is measured in the same way as the horizontal axis, but the sun sensor is rotated by  $90^\circ$ .

## 5.5 Angular relationship

The Earth and Sun each has a certain angular diameter as viewed from the satellite at a given altitude. For example, if the orbital height of the satellite is assumed to be 750 km, where the aerodynamical force is low enough to use a solar sail, using the angular relationship between the satellite and the Earth or the satellite and the Sun, the angular diameter of the Earth and the Sun can be calculated. Figure 5.21 shows the angular relationship between the satellite and the Earth [40].



**Figure 5.21** – Angular relationship between the satellite, target and Earth [40]

The best case scenario is when the Earth is centred in the boresight of the nadir sensor. The angle of interest in figure 5.21 is  $\rho$ , the angular radius of the Earth as seen from the satellite, and therefore also the angular radius of the Earth in the nadir sensor's FOV. The angular radius can be calculated by:

$$\sin(\rho) = \frac{R_E}{R_E + H} \quad (5.5.1)$$

where  $R_E$  is the Earth's radius, 6378.136 km, and  $H$  is the altitude of the satellite. The result for  $\rho$  is  $63.48^\circ$  and therefore the Earth's angular diameter from the satellite's perspective, is  $127.0^\circ$  at 750 km.

With the same equation, the Sun's angular diameter from the satellite's perspective, can be calculated. Equation (5.5.1) will change to:

$$\sin(\rho) = \frac{R_S}{R_S + H_S} \quad (5.5.2)$$

where  $R_S$  is the radius of the Sun, 695 500 km [41], and  $H_S$  the altitude of the satellite with respect to the Sun. The mean distance between the Sun and the Earth is  $149.6 \times 10^6$  km. Assuming that the mean distance between the Sun and the Earth is estimated from the centre points,  $H_S$  can be calculated as:

$$R_S + H_S = D - R_E - H \quad (5.5.3)$$

where  $D$  is the distance between the Earth and the Sun. By substituting (5.5.3) into (5.5.2),  $\rho$  is calculated as  $0.27^\circ$  and consequently the Sun's angular diameter from the satellite's perspective is  $0.53^\circ$ .

The results are shown in table 5.4.

Sensor	Angular diameter of the object
Nadir	127.0°
Sun	0.53°

**Table 5.4** – Angular diameter of sun and nadir sensor at an altitude of 750 km

To see if the nadir sensor’s testbench follows the angular relationship, the relationship between the aerobics ball and the Earth is required to be calculated. The relationship will follow (5.1.1), where the ratio between the radius of the ball and the distance from the camera to the ball, should be equal to the ratio between the radius of the Earth and the distance from the satellite to the Earth:

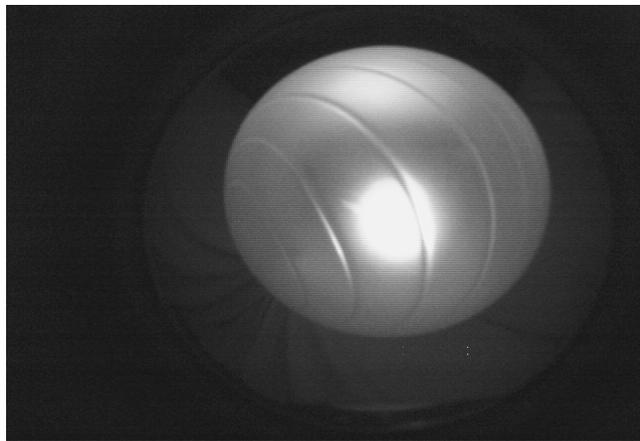
$$\frac{H}{R_E} = \frac{D_{ball}}{R_{ball}} \tag{5.5.4}$$

$$\frac{750km}{6378.136km} = \frac{D_{ball}}{3 \times 10^{-4}km}$$

where  $R_{ball}$  is the radius of the ball and  $D_{ball}$  is the distance the sensor must be placed from the ball.  $D_{ball}$  is calculated as 3.52 cm.

At 750 km, the Earth’s angular diameter from the satellite’s perspective is 127°, as shown in table 5.2, and the fisheye lens has a FOV of 190°. This means when the sensor is placed 3.52 cm from the ball and the ball is centred at the fisheye lens’ boresight, it should span 66.8% of the undistorted image diameter.

Figure 5.22 shows how the ball is represented when the camera is at a distance of 3.52 cm from the ball.



**Figure 5.22** – The ball at 3.52cm

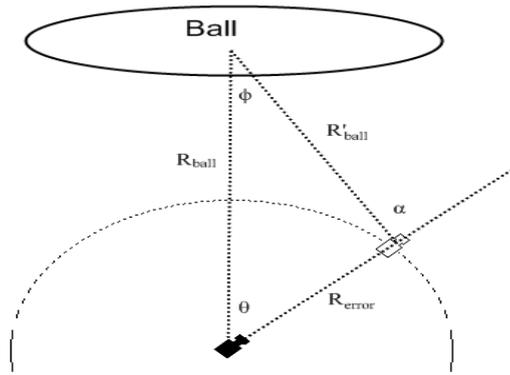
The distorted radius of the ball at 3.52 cm is 168 pixels (see table 5.5). The undistorted radius of the fisheye lens is 275.5 pixels. Therefore, the ball, spans 66.06% of the image diameter.

	Distorted	Undistorted
$R_{ball}$	168 pixels	182 pixels

**Table 5.5** – The radius of the ball at 3.52 cm

## 5.6 Rotation point (optic point calculation)

When the body frame angles are measured with the nadir sensor, the distance from the ball, when rotating the metric rotary stage, is required to stay constant to make the measurement valid. The nadir sensor's optic point is therefore required to be positioned at the centre of the metric rotary stage. Figure 5.23 shows the error that occurs when the optic point of the fisheye lens is not centred on the metric rotary stage.



**Figure 5.23** – Error in body frame measurements

The angles of interest in figure 5.23 are:

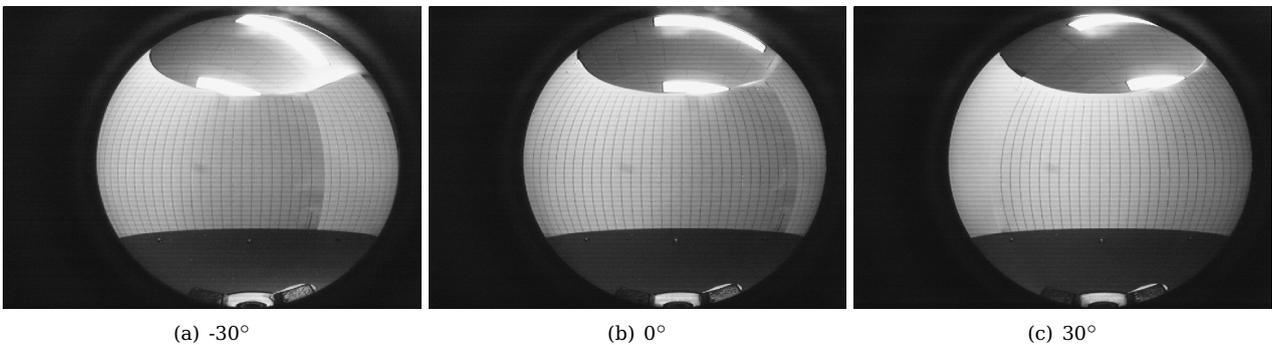
- $\theta$  - the correct body frame angle
- $\alpha$  - the incorrect body frame angle
- $\phi$  - the error between  $\theta$  and  $\alpha$

As an example, if the optic point has an 1 mm offset from the metric rotary stage's centre and the body frame angle expected is  $30^\circ$ , the error angle can be calculated from the law of sines and cosines [42]:

$$R_{ball}'^2 = R_{error}^2 + R_{ball}^2 - 2R_{error}R_{ball}\cos(\theta) \quad (5.6.1)$$

$$\frac{\sin(\phi)}{R_{error}} = \frac{\sin(\theta)}{R_{ball}'} \quad (5.6.2)$$

The error in the body frame angle,  $\phi$ , will be  $0.8343^\circ$ . If the optic point had a 1 mm offset from the rotation centre of the satellite in space, the error will be insignificant, because  $R_{ball}$  will be the altitude of 750 km. The equivalent error in space, for an error of 1 mm on the testbench, is an error distance of 21.84 km. Therefore rotations of the nadir sensor around the centre of mass of the satellite will cause insignificant errors.



(a)  $-30^\circ$

(b)  $0^\circ$

(c)  $30^\circ$

**Figure 5.24** – Images captured during rotation measurements

The test setup to find the optic point is placing paper with vertical lines printed onto them in a circular pattern around the metric rotary stage. Figure 5.24 shows images of the test for the optic point. The lines on the paper have a constant distance of 1 cm between each of them. The aim of the test is to see if the distance between the lines, near the distortion centre point, will stay constant. If the distances stay constant, the optic point is found.

It was found that the distances between the lines stayed constant at 16 pixels when the point shown in figure 5.25 is used as the optic rotation point.

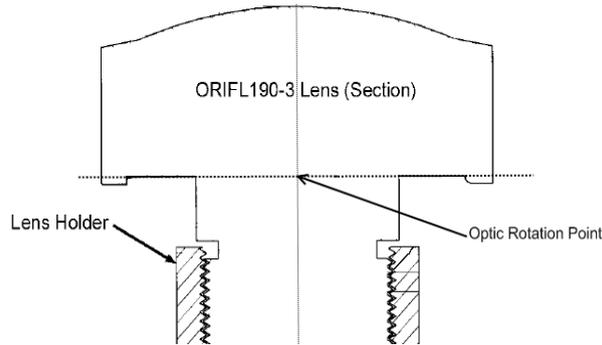


Figure 5.25 – Optic point (Appendix A.3)

### 5.7 Threshold determination

The threshold for the sensors are determined as described in section 4.1.2. As already mentioned, the pixels from the camera are sampled as 8-bit values. The values of the pixel will range from 0 to 255, the range of an unsigned 8-bit value. Figure 5.26 shows the grayscale from black to white. The value 0 is black and 255 is white for a pixel.



Figure 5.26 – Grayscale

Table 5.6 shows the threshold level calculated in section 4.1.2:

Sensor	Background average	Object average	Threshold
Nadir	34	153	70
Sun	34	248	99

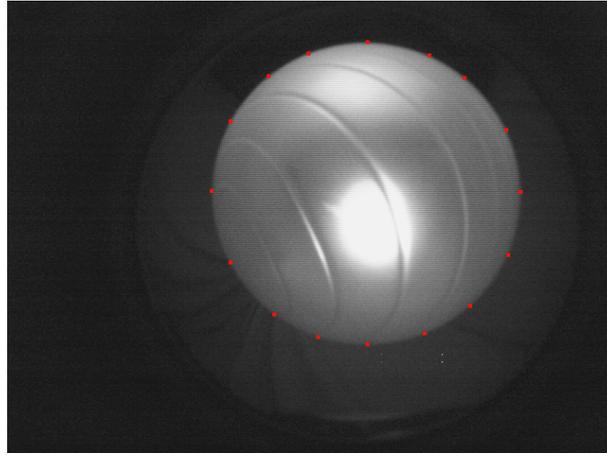
Table 5.6 – Average luminance values for sensor images

The threshold level for the nadir sensor is a good estimate and works well for the nadir sensor, as shown in figure 4.9(b). Note that this value is a good estimate for the nadir testbench and not necessarily for the true space environment.

Although the threshold calculated for the sun sensor is a good estimate, it is too low to prevent a reflection from an object closer to the satellite than the Sun, for example the moon, to show a false Sun on the image. The threshold for the sun sensor is taken at 90% of the Sun’s average luminance, a value rounded off to 225.

## 5.8 Sampling factor for the nadir sensor search algorithm

Figure 5.27 shows that 16 edge pixels give a good profile of the ball utilised in the nadir sensor's test-bench. By choosing the correct sampling factor  $K$  from equation 4.2.1, the 16 pixels required for the edge detection can be identified.

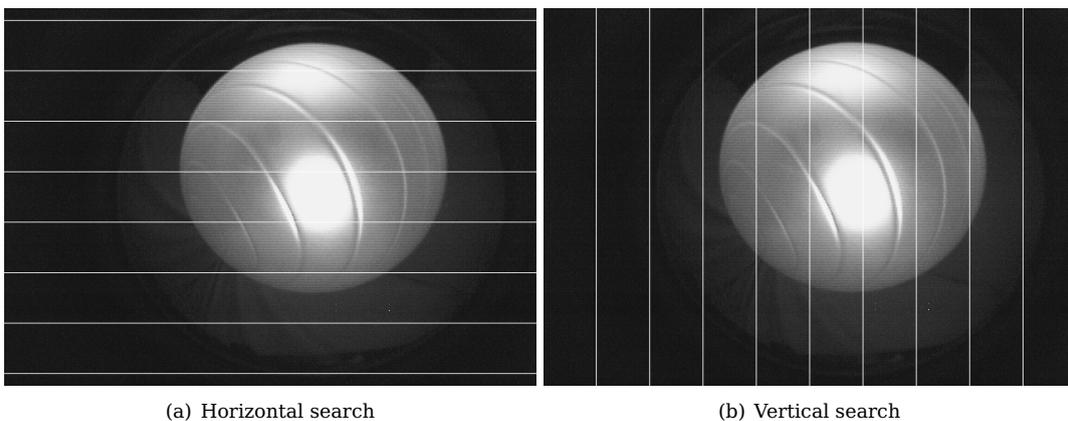


**Figure 5.27** – Edge profile of Earth

The amount of edge pixels that will be identified, can be calculated as follows:

$$\begin{aligned} \text{edge pixels} &= 4 \times \text{total intersected lines} \\ &= 4 \times \frac{\text{diameter of the Earth}}{K} \end{aligned} \quad (5.8.1)$$

where  $K$  is the sampling factor from equation 4.2.1. Figure 5.28 shows the specific lines used by the horizontal and vertical search algorithms to search for edge pixels. For each line that intersects the Earth, two edge pixels can be determined. Since there are two search algorithms, the amount of edge pixels that can be determined are doubled. The total lines that intersect the Earth are dependant on the diameter of the Earth and the sampling factor.



(a) Horizontal search

(b) Vertical search

**Figure 5.28** – Lines searched for edge pixels

If the radius of the Earth is 161 pixels at an altitude of 750 km, the sampling factor is calculated as 80.5 from equation 5.8.1 if 16 edge pixels are required. However, the binary search in section 6.5.2.1, requires the  $\log_2(K)$  to be an integer to compute correctly. Therefore, the sampling factor is set to 64, as this is the closest number to 80.5 to comply with the specification for the binary search. With the sampling factor set to 64, the total edge pixels that will be identified at an altitude of 750 km, is 20 for a full profile of the earth. This is verified in figure 4.9(b).

# Chapter 6

## Software implementation

### 6.1 Overview

This chapter describes the software coding implemented for the sun and nadir sensors. The microcontroller and FPGAs use different programming languages: the microcontroller is programmed in C code and the FPGAs in VHDL code. The aim of the software is to process the data from the camera modules in the minimum time as required.

#### 6.1.1 Data type definitions and algorithm complexity

The microcontroller uses MICROCHIP's C18 compiler, with its MPLAB IDE, to convert the source code to object code that can be programmed onto the microcontroller. Because the microcontroller is a PIC18F4520, the C18 compiler for the PIC18 family of microcontrollers from MICROCHIP is used. In the C18 compiler's user guide, a list of data types are given [43] (see table 6.1).

DATA TYPE	BITS	SIGNED RANGE	UNSIGNED RANGE
char	8	-128 to 127	0 to 255
short	16	-32 768 to 32 767	0 to 65 535
int	16	-32 768 to 32 767	0 to 65 535
short long	24	-8 388 608 to -8 388 607	0 to 16 777 215
long	32	-2 147 483 648 to 2 147 483 647	0 to 4 294 967 295

**Table 6.1** – Data types for the C18 compiler

The compiler also defines the floating point data types, *double* and *float*, with 32 bits of storage each. It is important to use the correct data type, because if a value is too large for the selected data type, overflow errors will occur. However, if larger than necessary data types are used, the microcontroller will require more time to process the data. The microcontroller is only a 8-bit processor. If, for example, an *int* is required, the microcontroller will take double the time to process the data.

To explain the complexity of algorithms, the big O notation is applied. The big O notation is implemented to described the behaviour and complexity of a function [44]. Table 6.2 shows the most common big O

Operation	Description	Complexity
Addition	Adding two n-bit integers	O(n)
Subtraction	Subtracting two n-bit integers	O(n)
Multiplication	Multiplying two n-bit integers (long multiplication)	O(n <sup>2</sup> )
Division	Dividing two n-bit integers (long division)	O(n <sup>2</sup> )
Lookup Table	Selecting an element of a lookup table	O(1)
Binary Search	Binary search algorithm	O(log n)

**Table 6.2** – Computational complexity of mathematical operations

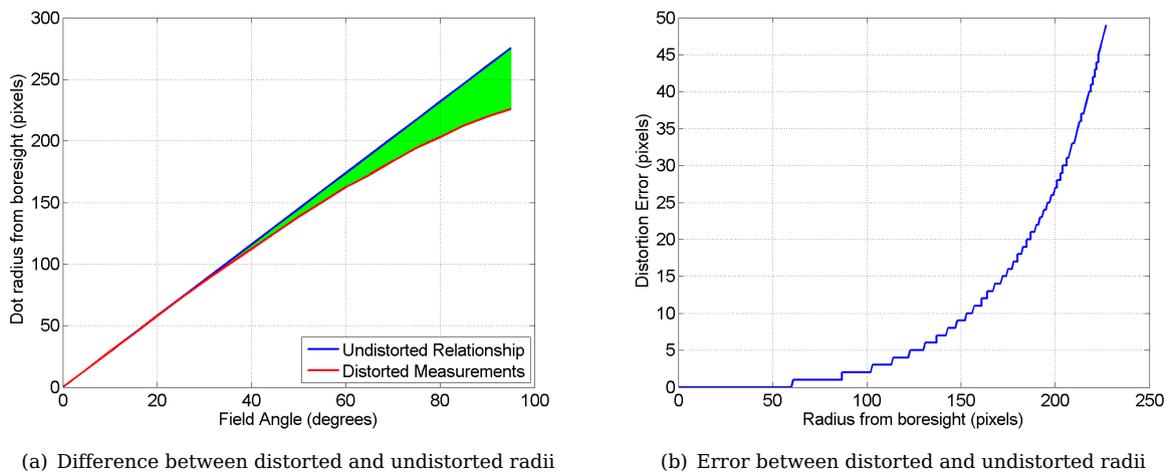
operations that are found in this project [45]. The less complex an algorithm is, the less time it will take to complete. For the big O notation this means the lower the order of  $n$  is, the less complex an algorithm will be and therefore the less time an algorithm will require to complete.

## 6.2 Distortion correction lookup table

The distortion correction model in section 5.2.2 is a complex equation. There are multiple square and 3rd power roots (where a square root already has a complexity of  $O(n^2)$ ), divisions and multiplications. Instead of wasting valuable time and computation power, a distortion correction lookup table is implemented:

$$\text{Lookup}[\text{distorted radius}] = \text{undistorted radius} \quad (6.2.1)$$

The difficulty when using a lookup table, is to select the values required to be stored. From the centre to the edge of the FOV, 226 undistorted radii are required to be stored in the lookup table, with the last few radii values being more than 255. This means that the data type *int* or *short* are required instead of the smallest data type *char*. For the compiler and microprocessor implemented, this is not possible, because the memory would have been required to store the lookup table into 452 bytes, which is larger than the maximum value of 256 bytes in which the memory pages of the microcontroller are divided into. Therefore, the lookup table must use the data type *char* to ensure that there is enough memory space for the entire lookup table.



**Figure 6.1** – Data used for the distortion correction lookup table

The problem is solved by, instead of storing the value of the undistorted radius at each element of the lookup table, the error between the distorted and undistorted radius is stored. The error at the last element of the lookup table is 49 and is well within the range of the data type *char*.

The advantages of using a lookup table are the following:

- The computation complexity of a lookup table is  $O(1)$ , because if the distorted radius is known, it can be used as an index to find the undistorted radius.
- Since the distortion radius is fixed in both  $x$  and  $y$  directions, the same lookup table can be used to determine the undistorted radius in the  $x$  and  $y$  positions.

### 6.3 Interpolation

As mentioned, the distortion lookup table is used to correct the error made by the distortion of the fisheye lens on the measurements of the nadir and sun sensors.

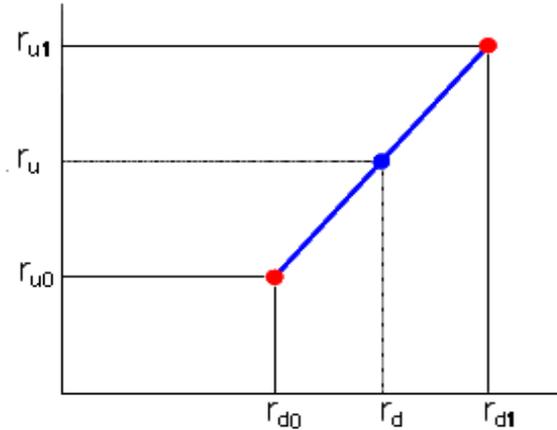


Figure 6.2 – Using interpolation for sub-pixel accuracy

To minimise storage on the microprocessor, the values in the distortion lookup table are stored as 8-bit correction values. To keep sub-pixel accuracy, linear interpolation [46] is used to undistort the sub-pixel resolution of the measurement. The interpolation is done as shown in figure 6.2:

$$\frac{r_u - r_{u0}}{r_d - r_{d0}} = \frac{r_{u1} - r_{u0}}{r_{d1} - r_{d0}} \quad (6.3.1)$$

where the interpolation is used to calculate  $r_u$ , the undistorted radius. In this form, the interpolation uses a large quantity of computational speed, since it uses multiplication and division. The equation is simplified with the following assumptions

- $r_d$  is given, as it is the measured value.
- $r_{d0}$  and  $r_{d1}$  are always the floor and ceiling values of  $r_d$  respectively. For example, if  $r_d$  is 5.5, then  $r_{d0}$  and  $r_{d1}$  will be 5 and 6 respectively. This also means that the  $r_{d1} - r_{d0}$  part of the equation will always be equal to 1.
- Furthermore, because of the distortion lookup table, the following definitions are made:

$$r_u = r_d + r_e \quad (6.3.2)$$

$$r_{u0} = r_{d0} + r_{e0} \quad (6.3.3)$$

$$r_{u1} = r_{d1} + r_{e1} \quad (6.3.4)$$

where the undistorted radius is defined as the distorted radius plus the distortion error. However, because  $r_{d0} - r_{d1}$  is always 1, (6.3.4) can be rewritten as

$$r_{u1} = r_{d0} + 1 + r_{e1} \quad (6.3.5)$$

Therefore (6.3.1) can be simplified to:

$$\begin{aligned} \frac{r_d + r_e - r_{d0} - r_{e0}}{r_d - r_{d0}} &= \frac{r_{d0} + 1 + r_{e1} - r_{d0} - r_{e0}}{1} \\ \frac{r_d - r_{d0}}{r_d - r_{d0}} + \frac{r_e - r_{e0}}{r_d - r_{d0}} &= 1 + r_{e1} - r_{e0} \\ r_e &= r_{e0} + (r_{e1} - r_{e0})(r_d - r_{d0}) \end{aligned} \quad (6.3.6)$$

In this form, the interpolation calculates only the error required to correct the distortion. This form ensures faster interpolation, because there is only one multiplication required.

The correction errors as well as  $r_{d0}$  is derived from  $r_d$  only. Therefore, the distortion correction with interpolation is calculated as follows:

$$r_u = r_{d0} + r_{e0} + (r_{e1} - r_{e0})(r_d - r_{d0}) \quad (6.3.7)$$

The undistorted x and y coordinates can then be calculated as follows:

$$x_u = \frac{r_u}{r_d} x_d \quad (6.3.8)$$

$$y_u = \frac{r_u}{r_d} y_d \quad (6.3.9)$$

The code for the linear interpolation is shown in Appendix C.2.

## 6.4 Controlling the camera module

Before the measurement algorithms for the sun and nadir sensors are implemented, the control of the two camera modules need to be implemented in software. In section 3.2.2, several control pins were identified as necessary for the layout of the sun and nadir sensors. These control pins are used to send commands from the microcontroller to the FPGAs when specific data is required. The control pins are defined in table 6.3.

Control pin	Definition	Discription
SUNEN/NADEN	Enable	Controls the specific FPGA or put it in sleep mode.
RNW	Read/Write	Controls whether reading from or writing to memory.
CLK	Clock	Controls speed of data from memory to microcontroller.
EDGE	Edge	Reconfigures FPGA to edge mode.
SHIFT	Shift	Extra functions.
SUNACK/NADACK	Acknowledge	Acknowledge sent to microcontroller when image is in memory.
HREF	Horizontal Reference	Indicates when a line of pixels are being transmitted to memory.
PCLK	Pixel Clock	Controls speed of data from camera to memory.

**Table 6.3** – Definitions of the control pins

All control pins are GPIO from the microcontroller, except for pins HREF and PCLK. These two control pins are from the camera modules and are used by the FPGAs to detect when an image is ready to be written to memory (see Appendix A.2). The commands to the FPGAs through the other control pins are defined in the truth table (table 6.4).

SHIFT	EDGE	RNW	SUNEN	NADEN	Commands	
					Sun sensor	Nadir sensor
X	X	X	1	1	Active Standby	Active Standby
0	0	0	0	1	Write image to memory	Active Standby
0	0	0	1	0	Active Standby	Write image to memory
0	0	1	0	1	Send image to MCU	Active Standby
0	0	1	1	0	Active Standby	Send image to MCU
0	1	1	0	1	Send edge data to MCU	Active Standby
0	1	1	1	0	Active Standby	Send edge data to MCU
1	0	0	0	0	Send image to nadir sensor's memory	Write image from sun sensor to memory
1	0	1	0	0	Write image from nadir sensor to memory	Send image to sun sensor's memory
1	1	0	0	0	Send edge data to MCU	Send image from memory to sun sensor
1	1	1	0	0	Send image from memory to nadir sensor	Send edge data to MCU

**Table 6.4** – Truth table of commands

### 6.4.1 Microcontroller

The microcontroller has four control actions that involve the camera modules (see figure 6.3). The microcontroller uses the control pins, of table 6.3, to implement the commands in the truth table (table 6.4) to command the FPGAs and camera modules.

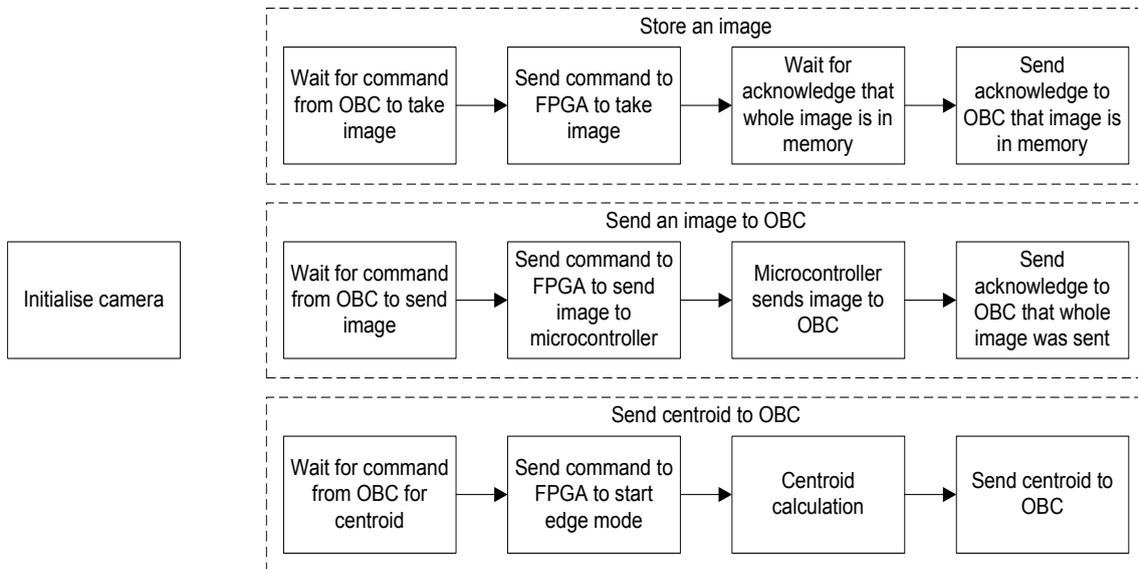


Figure 6.3 – Microcontroller control actions to the FPGAs and camera modules

The microcontroller initializes the default settings for the camera modules. The setup consists of setting the camera modules’ exposure time, frame size, data format, et cetera, as required. The microcontroller communicates through an I<sup>2</sup>C bus with the camera modules. Since the MSSP module of the microcontroller is kept for communication with the OBC, software I<sup>2</sup>C modules are coded to communicate with the camera modules.

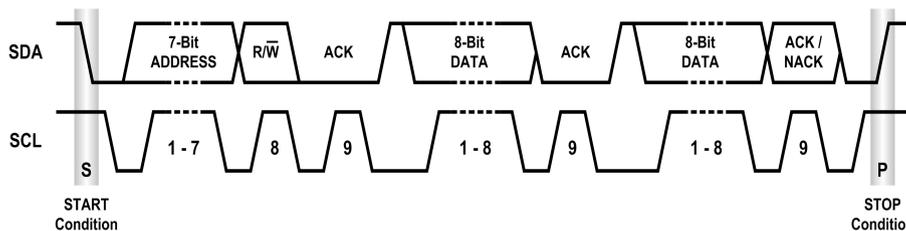


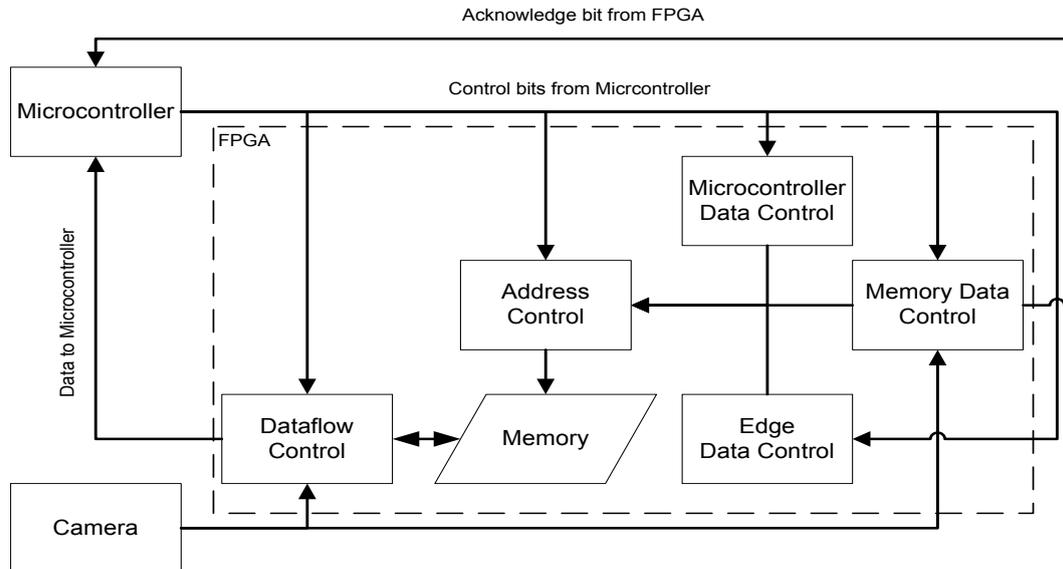
Figure 6.4 – I<sup>2</sup>C time diagram

The microcontroller only sends commands to the camera modules. Therefore only three conditions are necessary for implementation of the software I<sup>2</sup>C. The three conditions are the stop, start and transmit conditions (figure 6.4). An I<sup>2</sup>C transmission always follows the same procedure: a start condition, the required data-transmit conditions and lastly a stop condition. An acknowledge from the camera modules are read after each byte of transmitted data.

The code for the software I<sup>2</sup>C [47] stop, start and transmit conditions are shown in appendix C.1.

### 6.4.2 FPGAs

Although the microcontroller sends out the commands when to capture or retrieve an image, it is the FPGAs that process those commands. Figure 6.5 shows all the modules programmed into the FPGAs that allow them to store and retrieve images as well as manipulate the data.



**Figure 6.5** - Layout of FPGA modules

Note that the memory in figure 6.5 is in fact the external memory, but it forms a unit with the FPGA. The FPGA modules are defined as:

- Dataflow Control - controls the flow of data to and from the memory as well as to the microcontroller.
- Microcontroller Data Control - sends the image stored in the memory to the microcontroller
- Memory Data Control - stores the image from the camera module in the memory
- Edge Data Control - sends the pixels required for edge detection to the microcontroller
- Address Control - generates the addresses for the data to be stored in and read from memory

### 6.4.3 I<sup>2</sup>C protocol for the OBC

As previously mentioned, the MSSP on-chip module of the microcontroller's purpose is for the communication between the microcontroller and the OBC. The protocol for the OBC to send and receive data from the microcontroller is the following:

- The OBC sends the correct slave address to the microcontroller. The least significant bit of the address determines whether the OBC is writing data to or reading data from the microcontroller.
- After sending the slave address, the OBC sends the address of the register it wants to read from or write to, for example, when the OBC wants to write a new threshold value for the nadir sensor in the threshold register. The registers each has a size of a byte.
- After the OBC has sent the address of the register it wants to write to, the next byte sent is the data for that register. If the OBC indicated that it wants to read a centroid coordinate for example, a 2 byte value is sent from the microcontroller. The register for sending measurements is an 8-bit register that is a reference to 16-bit registers that store the signed 16-bit measurements. However, if an image is required to be sent to the OBC, 300 kB of data is sent over the I<sup>2</sup>C bus.

The same protocol is followed when using the backup UART bus.

### 6.4.4 Memory allocation

As mentioned in section 3.2.1, the pixel clock of the camera module runs at 13.5 MHz and the memory was selected to comply to this restriction. However, delays may occur in the address control of the FPGA. Figure 6.6(a) shows how memory can be seen as a sequential list of addresses, which are written to and read from addresses 0x00000 to 0x7FFFF (the range of the specific memory).

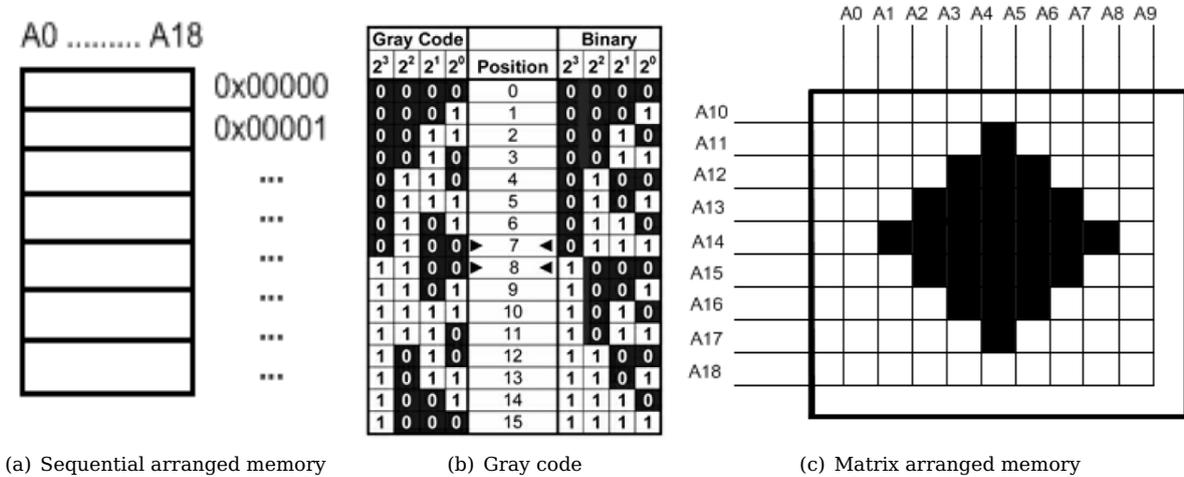


Figure 6.6 – Different arrangements of images in memory

One option is to implement gray code. Gray code is a different method of representing binary code, where only one bit flip occurs when gray code is implemented [48]. Figure 6.6(b) shows the difference between incrementing with gray and binary code. However, to implement gray code on a FPGA is complex (see appendix D.1) [49], and requires more tiles from the FPGA to implement. It is also more difficult to jump an address. Jumping addresses is necessary for the edge detection of the sun and nadir sensors.

The method of memory arrangement rather selected, is a type of matrix arrangement, because it simplifies calculations. The matrix arrangement corresponds with the pixel coordinates of an image. Since the image size is 640 x 480, the column and row addresses are defined as two different entities. The column addresses are 10-bit wide and the row addresses are 9-bit wide, as 640 is a 10-bit and 480 is a 9-bit value. This ensures that the minimum delays may occur and it is easier, if necessary, to jump column and row addresses when reading from the memory. With the memory size being 512 kB and an image is 300 kB, 41.4% of the memory is available for other uses. However, with the matrix arrangement in memory, only 6.12% of the memory is easy to access (the last memory address used for the matrix arrangement is 0x78280). The matrix arrangement corresponds to data being stored sequentially in a block of memory (one row), but these blocks are stored in memory with unused memory between them. The unused memory is accessible, but an algorithm for addressing it is required. Therefore, only the memory after the last address implemented for the matrix arrangement is easy to access, because no algorithm for addressing is required.

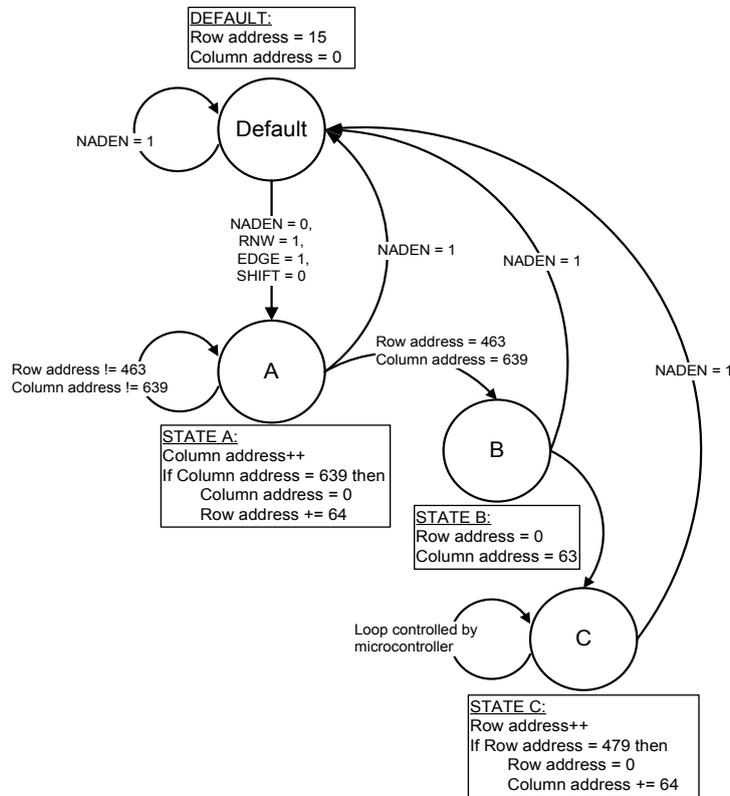
The matrix arrangement is implemented when both reading from or writing to the memory. The matrix addressing is shown in appendix D.2. The code is a simple counter that increments on the rising edges of a clock and will send either the addresses to write to or read from the memory, depending on the control pins that are active.

### 6.5 Nadir sensor

In this section the coding of the algorithms to determine the centroid of the Earth is described. The coding of the algorithms are divided between the microcontroller and FPGA. The FPGA can access the data faster from the memory and the microcontroller has dedicated arithmetic modules to process the data.

### 6.5.1 Search algorithm

The search algorithm for the nadir sensor is implemented on the FPGA, as the FPGA has direct access to the memory. The algorithm is implemented as a state machine on the FPGA [50]. Figure 6.7 shows the flow diagram of the state machine implemented on the FPGA for the search algorithm.



**Figure 6.7** – Search algorithm state machine

The state machine for the search algorithm is implemented as four states. When the state machine is not in use, it goes into a default state. The default state sets the column and row addresses at the starting position required by the horizontal line search.

The default column and row addresses are 0 and 15 - the 1<sup>st</sup> column and the 16<sup>th</sup> row of the image (see figure 4.7). As discussed in the previous chapter, the image has 480 rows and the search algorithms have a sampling factor of 64. To see how many lines will be searched by the horizontal search algorithm, the image row count is divided by the sampling factor. The result is 7.5 or 7 with a remainder of 32 lines. When searching the horizontal lines, the row position of the line is not important, but the column position is. Therefore, the starting position for the horizontal search is set 16 rows lower. This will have the effect that the first and last 16 rows are not searched. The horizontal search algorithm is however simplified, because the rows will only have to be incremented by the sampling factor 64 and no provisions have to be made for the remainder of 32 lines.

State A is where the addresses are generated for the horizontal search. The column addresses are incremented on the rising edge of the clock generated by the microcontroller. When the column addresses reach the end of a line, the address is reset to the first column address and the row address is incremented with a factor of 64, the sampling factor. The state machine will stay in this state until the end of the 464<sup>th</sup> row is read.

State B resets the column and row addresses to the starting address required to start with the vertical line search.

State C is identical to state A except that the row addresses are incremented and when the end of a vertical line is reached, the row address is reset to the beginning of the row addresses and the column addresses

are incremented with a factor of 64. The loop of state C is controlled by the microcontroller. The loop will only end when the FPGA is disabled and the state returns to the default state.

The search algorithm state machine's code is shown in appendix D.3.

### 6.5.2 Edge detection

The edge detection for the nadir sensor is executed while the search algorithm is running. Because the clock that controls the search algorithm is generated on the microcontroller, it is easy to collaborate between the search algorithm and the edge detection. Therefore, the edge detection is divided into a horizontal and a vertical edge detection, to execute in sync with the search algorithm.

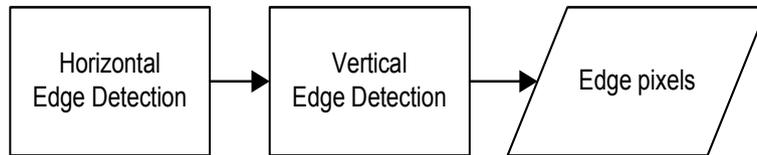


Figure 6.8 – Edge detection for nadir sensor

When searching through the horizontal lines, for example, the pixels of a specific horizontal line is sent to the microcontroller, but not all pixels are examined for edge pixels. Every 64<sup>th</sup> pixel in a line is examined for edge pixels to comply with the sampling factor selected for the search algorithm. These pixels are referred to as the sample pixels.

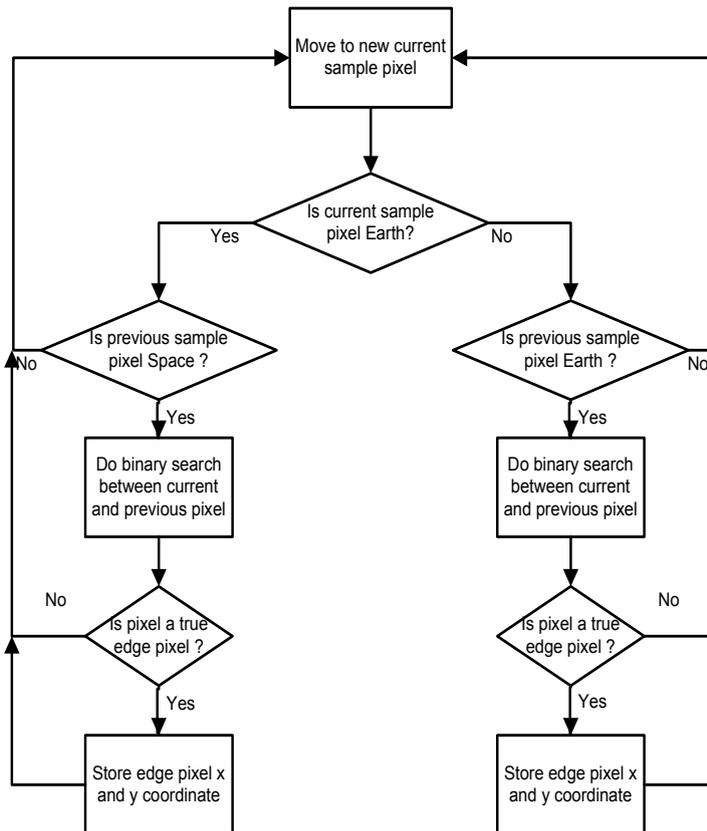


Figure 6.9 – Flow diagram of edge detection for the nadir sensor

The flow diagram in figure 6.9 implement the rules stated in section 4.2.2 for the edge detection algorithm. The same flow diagram in figure 6.9 is implemented for the horizontal and vertical edge detections. The flow diagram indicates that the edge detection code implements a number of if-statements to comply

with the rules stated for the edge detection. The rules for edge detection can be implemented with one if-statement:

```
if (current sample pixel above and previous sample pixel below threshold or vice versa)
    search edge pixel between current and previous pixel
```

However, this form requires more time to compute, because each statement within the if-statement is required to be compared with the threshold before the if-statement can be computed as true or false. Instead, this form is used:

```
if (current sample pixel above threshold)
    if (previous sample pixel below threshold)
        search edge pixel between current and previous pixel
else
    if (previous sample pixel above threshold)
        search edge pixel between current and previous pixel
```

This form of the edge detection is faster, because a pixel is either above or below the threshold. Therefore, if the current sample pixel is below the threshold, it is not necessary to compare whether the previous sample pixel is below the threshold. The disadvantage of this form of edge detection is that it requires more program memory from the microcontroller.

The same error that occurs in section 6.5.1 with the 480 rows of the image and the sampling factor, occurs when executing the vertical edge detection. However, the remaining 32 pixels in each row cannot be ignored, as the row position is of importance for the vertical search algorithm. This problem is corrected by adjusting the vertical edge detection to take into consideration that only 32 pixels are searched at the end of each row instead of 64 pixels (appendix C.3).

After the edge detection has determined if an edge pixel can be found between the current and previous sample pixel, a binary search is executed to find the real edge pixel and then this pixel is examined to determine if it is a true or a false edge pixel.

### 6.5.2.1 Binary search

Figure 6.10 shows an example of pixel values between two sampled pixels to find an edge pixel. The sampled pixels are always multiples of 64, except for the last sample in the vertical lines, which is 32. By searching the pixels between the sampled pixels, the edge pixels can be determined to an accuracy of a pixel.



**Figure 6.10** – Pixels between two sample pixels

The complexity of searching through the in-between-sample pixels is  $O(n)$ , or in this case  $O(63)$ , for the worst case scenario where every pixel between the sample pixels is required to be checked. This is very time consuming, since the search is required to be repeated between all sample pixels that comply to the rules for edge detection. A binary search is implemented instead.

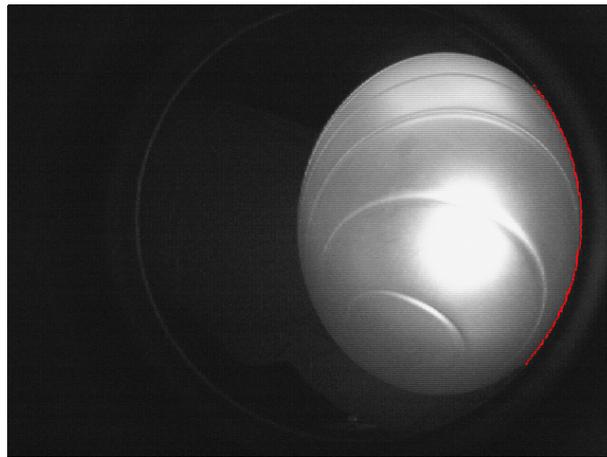
A binary search is an algorithm that is used to locate the position of an element in a sorted array. The problem with a binary search is that a sorted array is required for the algorithm to work correctly. For example, it is possible for a pixel to have a value of 24, the next pixel a value of 27, but the pixel after that a value of 26, due to signal noise and analog to digital conversions from the image sensor, as indicated in figure 6.10.

However, this problem is avoided in most cases, since the value of the pixel is only significant to determine if the pixel is above or below the threshold. An error will occur only if the pixels near the threshold is not sorted.

The computational complexity of a binary search is  $O(\log(n))$ . Since the space between two sampled pixels is a multiple of 2, the complexity changes to  $O(\log_2(n))$  and in this case  $O(\log_2(64))$  or  $O(6)$ , because to find the actual edge pixel between two sampled pixels, the worst case scenario is that the binary search algorithm is required to be repeated six times. For the case of the last 32 pixels in the vertical edge detection, the worst case scenario is a repetition of five times.

### 6.5.2.2 Partial profile edge detection

When the Earth moves further than the edge of the fisheye lens' FOV, a false edge is created, as shown in figure 6.11. If this edge is computed by the edge detection algorithm as is, the false edge will be considered a true edge of the Earth.



**Figure 6.11** – False edge

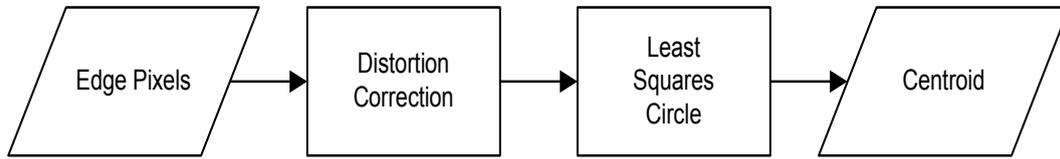
To prevent mistaking a false edge for a true edge, a boundary is set in the edge detection. The radius of the fisheye lens' FOV is known and the radius of the edge pixels from the boresight can be calculated by:

$$r_{\text{edge}} = \sqrt{(x_{\text{edge}} - x_{\text{boresight}})^2 + (y_{\text{edge}} - y_{\text{boresight}})^2} \quad (6.5.1)$$

If the radius of the edge pixel from the boresight is larger than or equal to the edge of the FOV, the edge pixel will be considered a false edge and will not be used for centroid calculation. With the optimization of the least squares circle method and this boundary, the centroid of the Earth can be calculated from the partial edge profile as well.

### 6.5.3 Centroid calculation

Figure 6.12 illustrates the process followed after the edge pixels have been identified.



**Figure 6.12** – Calculating the Earth’s centroid

The edge pixels are sent through the distortion correction lookup table before they are implemented in the least squares circle method.

#### 6.5.3.1 Least squares circle

The least squares circle method consists of matrix calculations. The microcontroller has no matrix functions, therefore arrays are implemented as matrices. From equation 4.4.18 it is necessary to calculate the inverse of the matrix in equation 4.4.17. The matrix in equation 4.4.17 is a 3 x 3 matrix. Figure 6.13 shows the inverse of a 3 x 3 matrix [51].

$$A^{-1} = \frac{1}{|A|} \begin{bmatrix} \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{13} & a_{12} \\ a_{33} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} \\ \begin{vmatrix} a_{23} & a_{21} \\ a_{33} & a_{31} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & \begin{vmatrix} a_{13} & a_{11} \\ a_{23} & a_{21} \end{vmatrix} \\ \begin{vmatrix} a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{12} & a_{11} \\ a_{32} & a_{31} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} \end{bmatrix}$$

**Figure 6.13** – Inverse of a 3 x 3 matrix

The determinant of the matrix can become a large number when working with pixel coordinates. Therefore, the radii of the edge pixels are used instead of the edge pixels’ coordinates to ensure that the determinant of the matrix does not exceed the range of the data type assigned to it.

From equation 4.4.18, only the first two rows of the inverse matrix is required to be calculated, as these two rows are required to calculate the centroid of the Earth (see Appendix C.5).

The centroid of the Earth is calculated using 32-bit double data types. This is to ensure that there is at least a two decimal accuracy in the calculation of the centroid. The problem that may occur when sending the centroid as a *double* to the OBC, is that the OBC may not handle *doubles* the same way as the microcontroller. Therefore, the centroid is multiplied with a factor of 100, to ensure that the two decimal accuracy is kept, and then the centroid is parsed to a 16-bit signed integer (see appendix C.5). The OBC will be able to handle this data type correctly.

## 6.6 Sun sensor

In this section the coding of the algorithms used to determine the centroid of the Sun, is described. Similar to the nadir sensor, the coding of the algorithms are divided between the microcontroller and FPGA.

### 6.6.1 Search algorithm

The search algorithm for the sun sensor adds another module to the FPGA layout as shown in figure 6.5. This module is implemented to determine the first Sun pixel. Figure 6.14 shows how this extra module connects to rest of the FPGA layout.

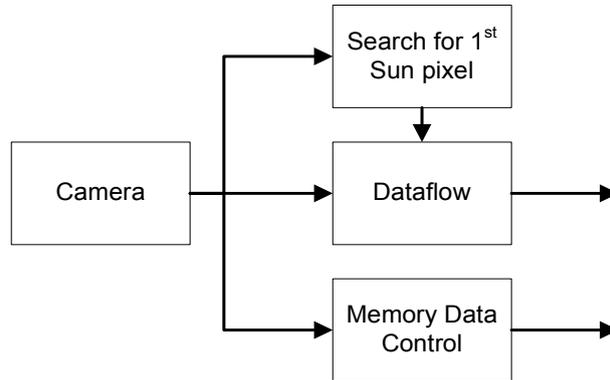


Figure 6.14 – Extra module in the FPGA of the sun sensor

In section 4.2.1.2 it is stated that the first Sun pixel is determined without using extra time. The first Sun pixel is determined with parallel processing, an ability that is available when using a FPGA. The first Sun pixel is determined in parallel with the image being written to the memory. The coordinates of the first Sun pixel is stored in a register on the FPGA when reading state C in figure 6.15.

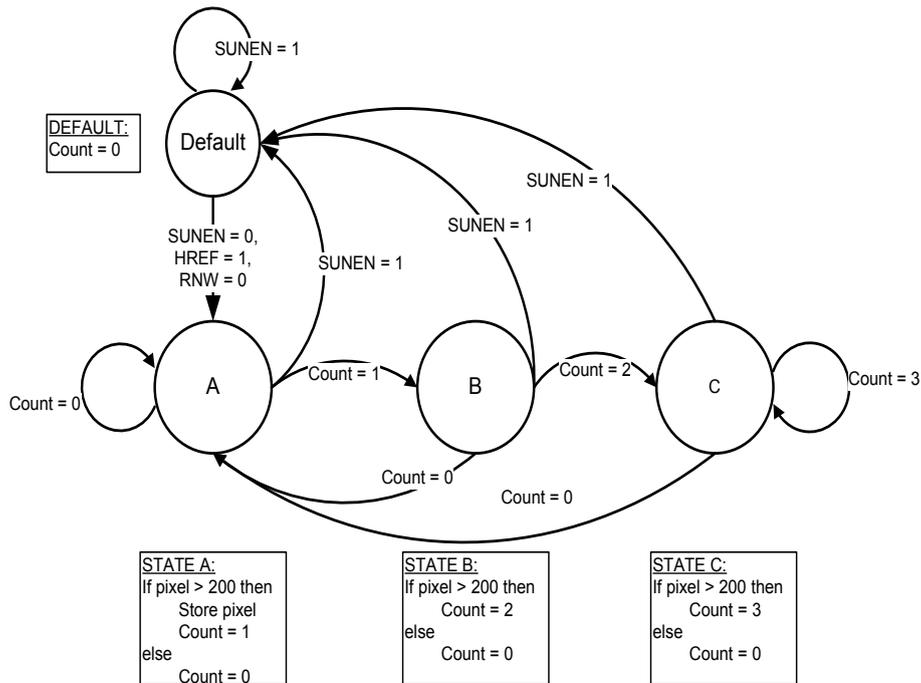


Figure 6.15 – Flow diagram for searching the first Sun pixel

Figure 6.15 shows the flow diagram of the implemented state machine for searching the first Sun pixel. The algorithm is implemented with one rule. If three consecutive pixels are above the threshold, then the first pixel is the first Sun pixel. Three consecutive pixels are examined to ensure that dead pixels that latch to a value above the threshold, are not mistaken as the first Sun pixel. Therefore, the state machine has four states.

The default state resets the counter to 0 and stays in this state until activated. When activated, the state machine will begin at state A and will only move to state B if the pixel examined at state A, is above the

threshold. The same will happen at state B when moving to state C, but if the pixel examined at state B is below the threshold, the state machine will move back to state A. When the state machine reaches state C, it will stay in that state until the module is reset through the microcontroller. The state machine compares pixels and moves to the next state on the rising edge of the pixel clock from the camera module.

### 6.6.2 Area search

Figure 6.16 is an area search flow diagram of the state machine implemented for the sun sensor.

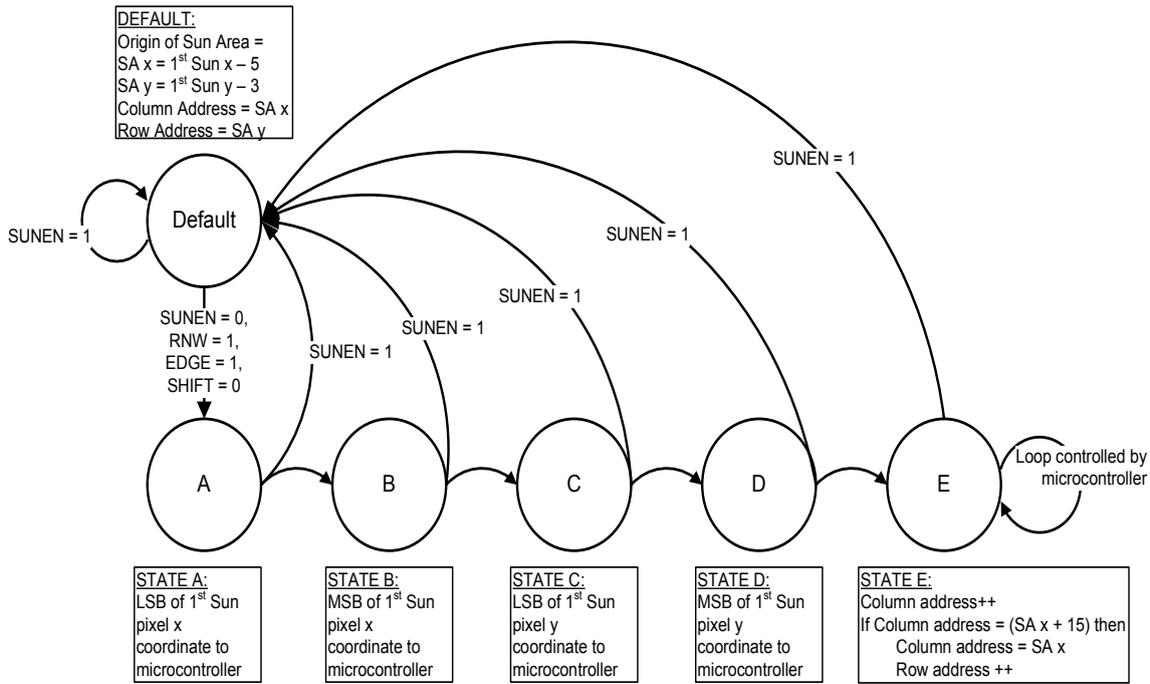


Figure 6.16 – Area search flow diagram for sun sensor

The default state sets the origin of the area. These coordinates correspond to the specific row and column address in the memory. Before the area search can begin, the first Sun pixel’s coordinates are sent to the microcontroller, as these coordinates are required for the centroid calculation. These coordinates are two 16-bit integers. Therefore, the first four states send the coordinates to the microcontroller, eight bits at a time. The last state is identical to the horizontal and vertical line searches implemented in the nadir sensor’s FPGA. The loop at state E is controlled by the microcontroller, as the clock that changes the states is generated by the microcontroller. State E is the state that sends the pixels enclosed in the area search, to the microcontroller for centroid calculations for the sun sensor. The state changes at the rising of this clock.

### 6.6.3 Centroid calculation

Figure 6.17 shows the process followed to calculate the centroid of the Sun. The possible Sun pixels are the

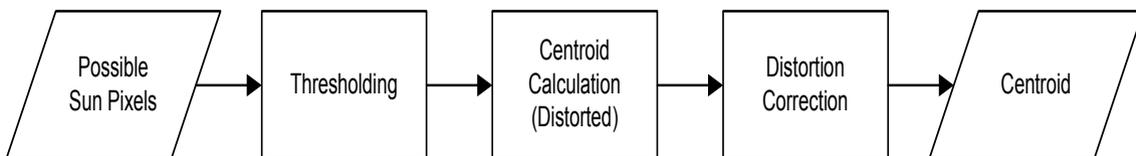


Figure 6.17 – Calculating the Sun’s centroid

pixels sent from the FPGA, as the threshold implemented on the FPGA is lower than the actual threshold implemented to determine the Sun pixels. The threshold on the FPGA is 200. As the average of the Sun

pixels' coordinates are used in the calculation of the Sun's centroid, the centroid is calculated with the following:

$$\text{Sun centroid}_x = \text{First Sun pixel}_x + \frac{\sum_{i=1}^N \text{pixel}_{\text{value}}(i) \times \text{pixel}_x(i)}{\sum_{i=1}^N \text{pixel}_{\text{value}}(i)} \quad (6.6.1)$$

$$\text{Sun centroid}_y = \text{First Sun pixel}_y + \frac{\sum_{i=1}^N \text{pixel}_{\text{value}}(i) \times \text{pixel}_y(i)}{\sum_{i=1}^N \text{pixel}_{\text{value}}(i)} \quad (6.6.2)$$

where N is the total Sun pixels, pixel(i) is the current pixel x and y coordinates and pixel<sub>value</sub>(i) is the value representing the luminance of the pixel. The luminance level of the Sun pixels are used to weigh each pixel against the other pixels to calculate a more accurate centroid. The first Sun pixel's coordinates are also used in the calculations, as the weighted pixels calculate the centroid with reference to the search area's coordinates. By adding the first Sun pixel's coordinates, the centroid is then referenced to the image's coordinates. The calculated centroid is still distorted. Distortion correction is implemented after the centroid is calculated, because the Sun pixels are situated close to one another and therefore the difference in the distortion between these pixels are minimal.

# Chapter 7

## Results

### 7.1 Nadir sensor measurement results

In this section the measurement results of the nadir sensor are discussed. The goal given for the nadir sensor is an accuracy of less than  $0.2^\circ$  for a full profile of the Earth.

Figure 7.1 shows the body frame angles measured in all directions with the nadir sensor. These measurements are documented using the testbench described in section 5.4.1 for the nadir sensor. The result of a 100 samples of each angle that was measured, are documented.

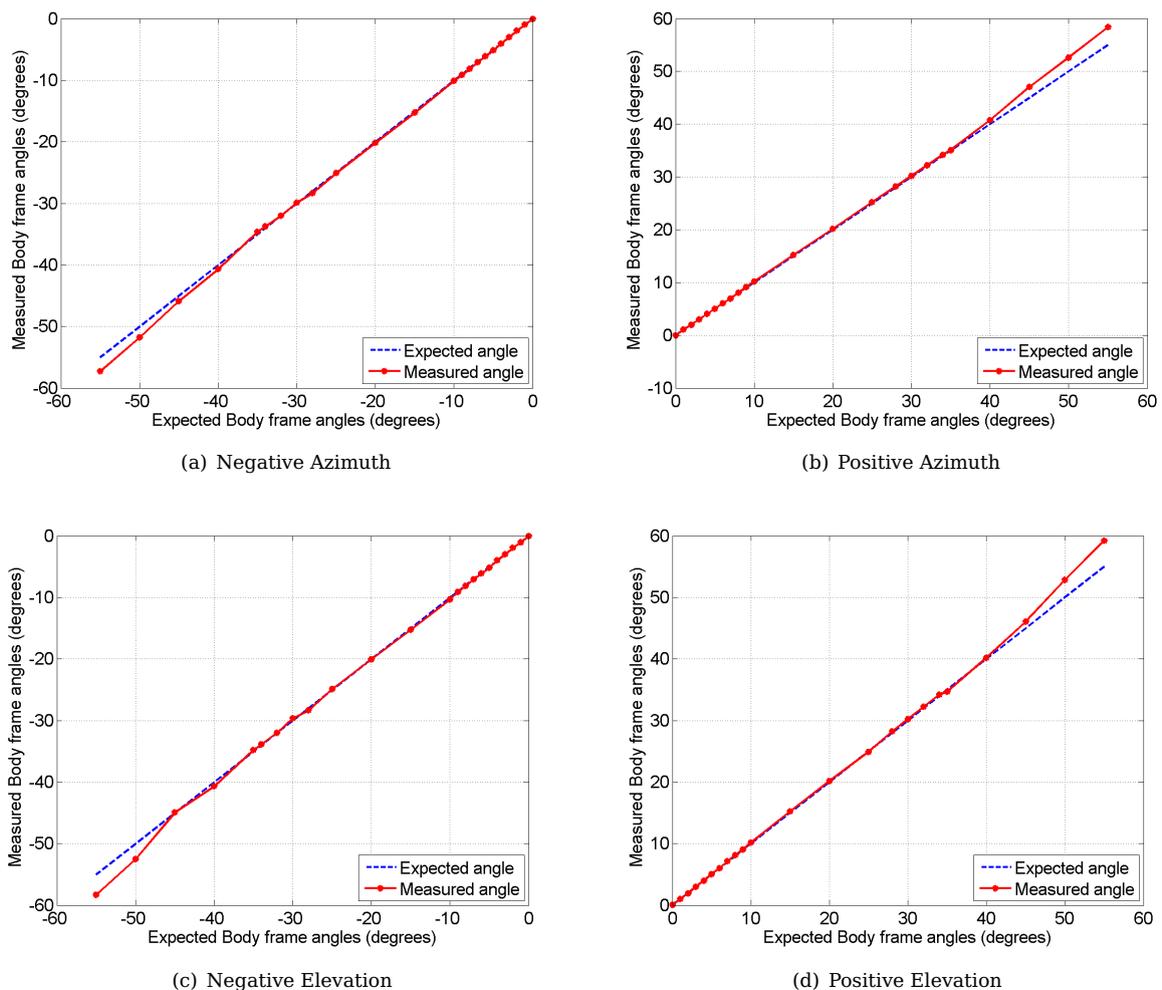
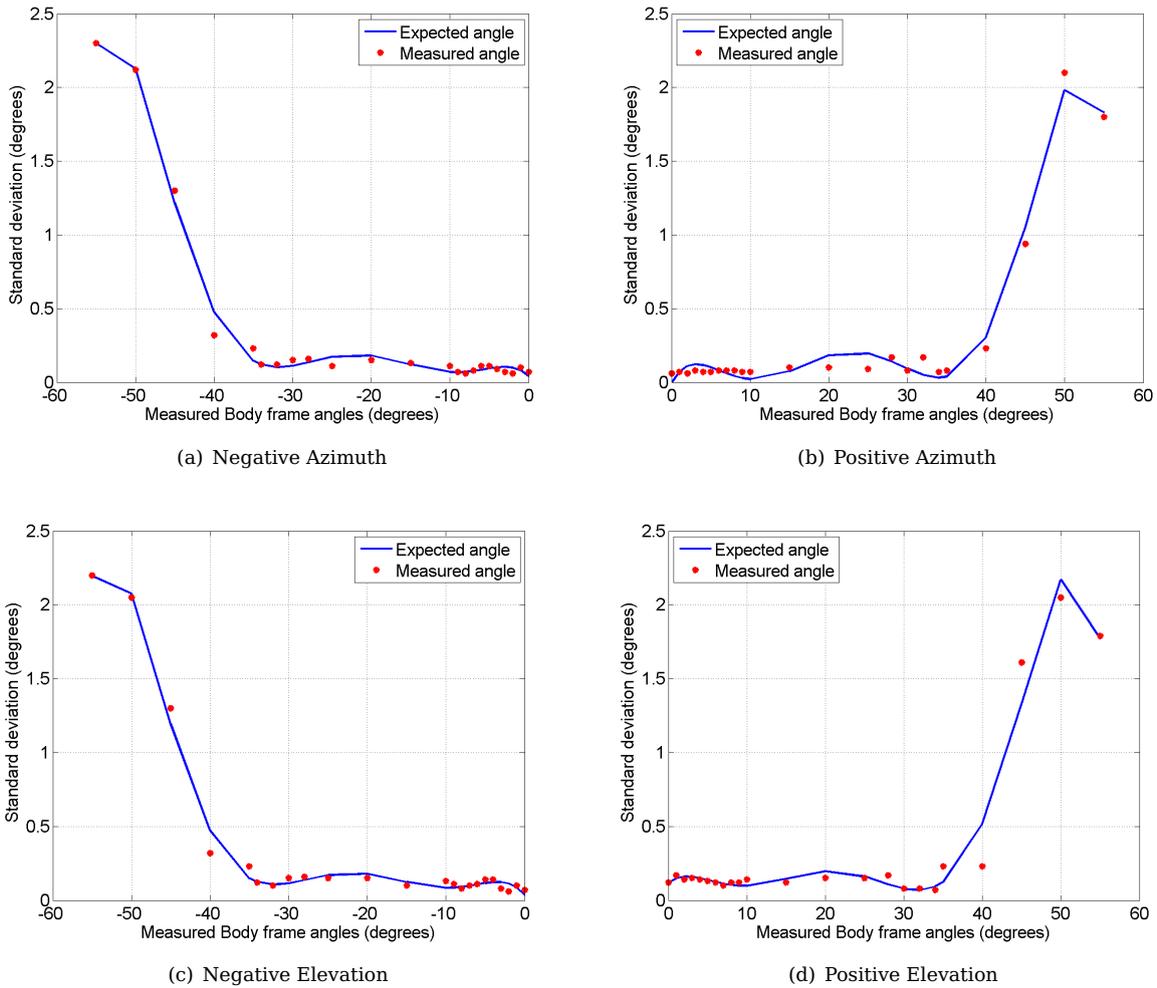


Figure 7.1 – Nadir sensor body frame angles

The dashed line in figure 7.1(a), 7.1(b), 7.1(c) and 7.1(d) represents the expected body frame angle to be measured at each angle. The measured points are the average from the 100 samples at the specific measured body frame angles.

For these measurements, 30° indicates where the edge of the ball utilised for the testbench starts touching the edge of the fisheye lens' FOV. As figure 7.1 shows the measured angles follow the expected angles very well, with a average offset error between the measured and expected angles of 0.12° for figure 7.1(a), 0.16° for figure 7.1(b), 0.13° for figure 7.1(c) and 0.15° for figure 7.1(d).

When the ball moves outside the edge of the FOV of the fisheye lens, the offset error increases as the portion of the full profile of the ball becomes smaller. The remainder of the ball profile moves into the area of the FOV with the most distortion. The less the ball profile is in view, the more distortion errors occur in the calculation of the ball's centroid.



**Figure 7.2** – Nadir sensor RMS error

Figure 7.2 shows the RMS error of the nadir sensor measurement over the FOV. For better accuracy on the nadir sensor, the standard deviation [52] of the 100 samples for each angle is calculated to examine how the RMS error varies over the FOV:

$$E[X^n] = \frac{1}{N} \sum_{i=1}^N x_i^n \tag{7.1.1}$$

$$\sigma_X = \sqrt{E[X^2] - E[X]^2} \tag{7.1.2}$$

where  $\sigma_X$  is the standard deviation. The points in figures 7.2(a) to 7.2(d) represent the calculated standard deviations over the FOV. The line represents the polynomial model that best fits the standard deviations. The model is created using MATLAB's `polyfit()` function.

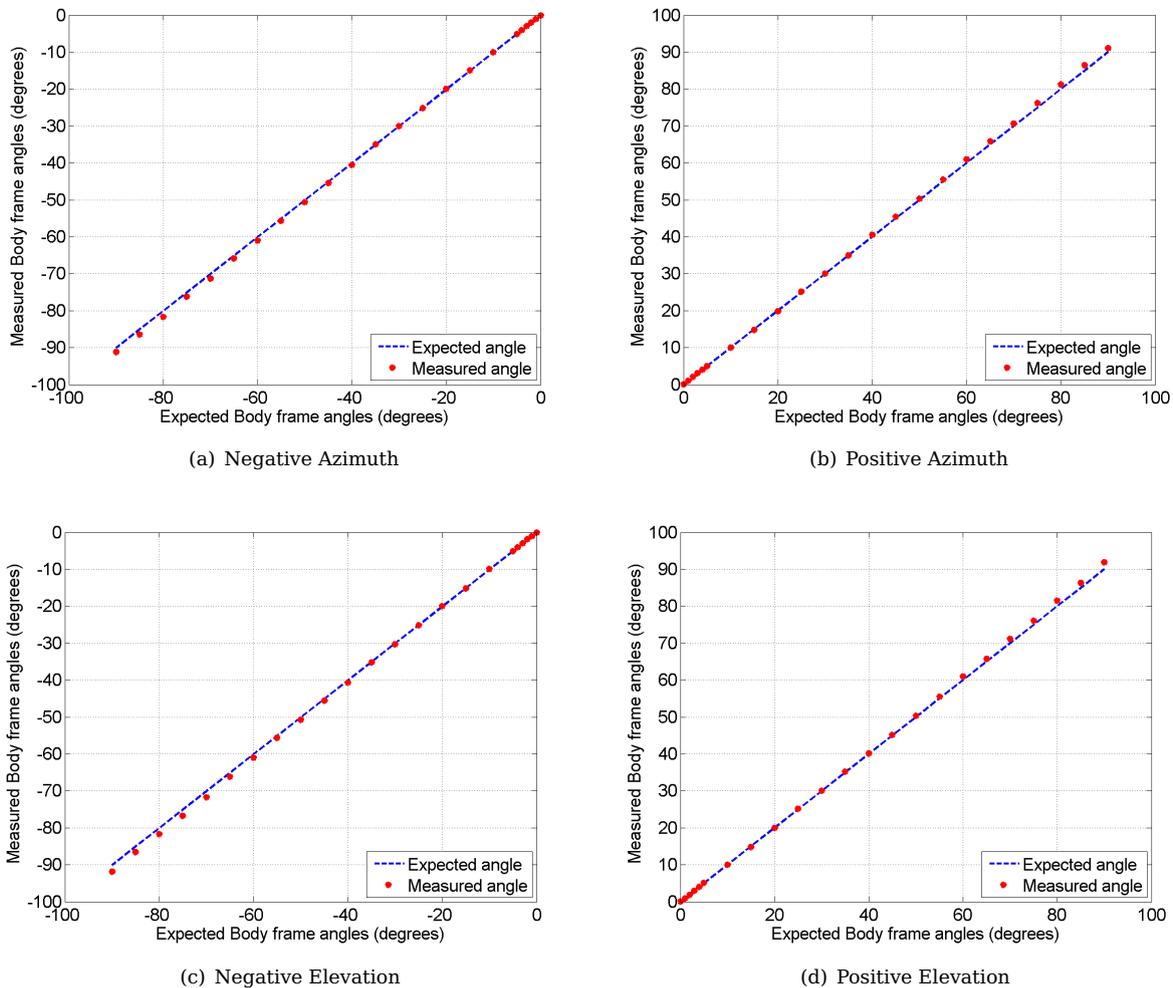
Except for minor differences due to practical limitations from the test setup, the model in each figure of figure 7.2 seems identical. The standard deviation stays below  $0.18^\circ$  for angles below  $30^\circ$ .

For the same reason the offset errors increase, the standard deviation also increases when the ball profile moves beyond the edge of the FOV. Therefore more of the ball's visible profile is situated in the image area with the most distortion.

## 7.2 Sun sensor measurement results

The measurement results for the sun sensor is discussed in this section. The goal given for the sun sensor is an accuracy of below  $0.2^\circ$  for the FOV  $\pm 45^\circ$  and below  $2^\circ$  for the FOV between  $\pm 45^\circ$  to  $\pm 90^\circ$ .

Figure 7.3 shows the body frame angles measured in all directions with the sun sensor.

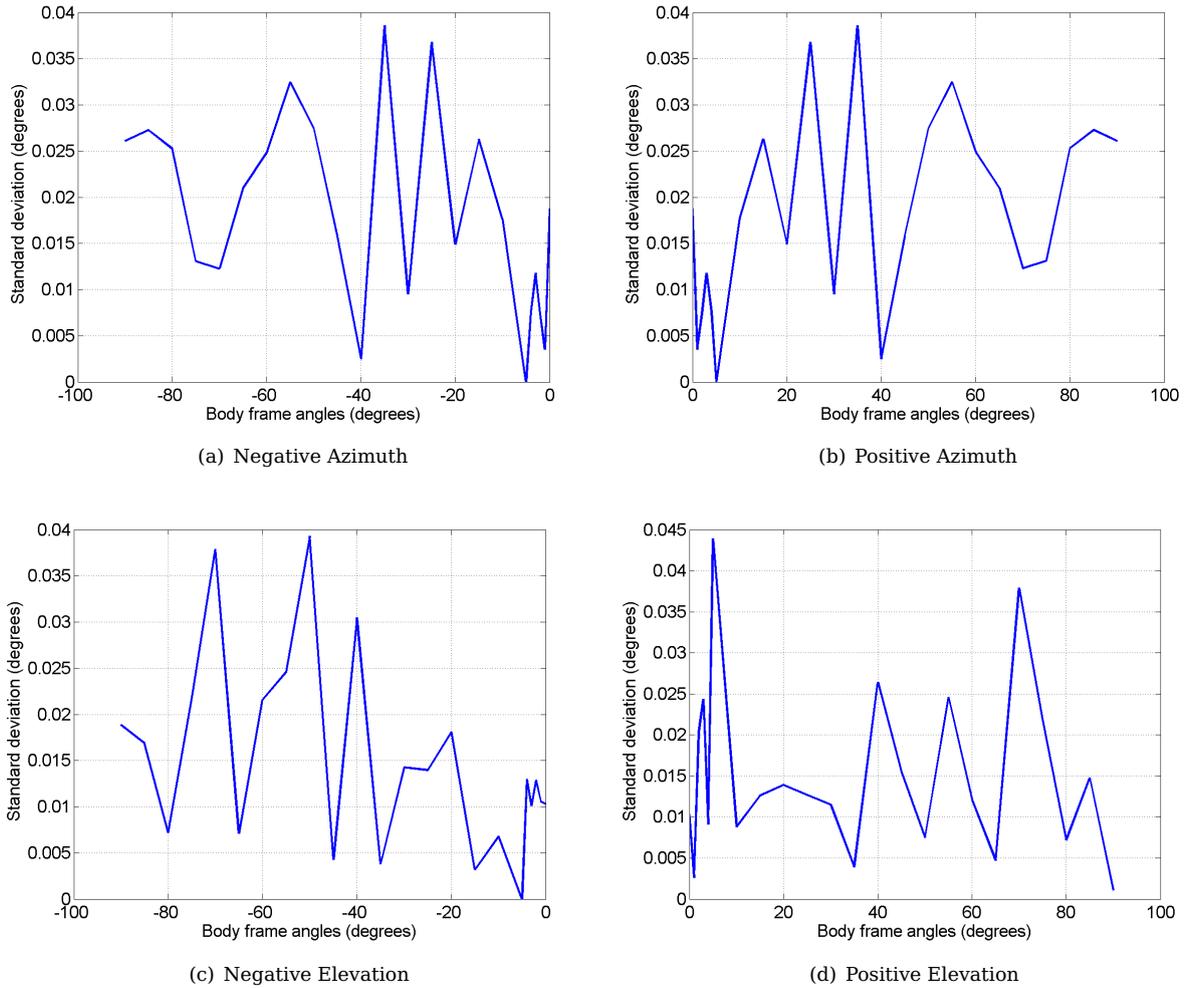


**Figure 7.3** – Sun sensor body frame angles

Similar to the body frame angles in the nadir sensor, the dashed line in figures 7.3(a) to 7.3(d) represent the expected body frame angles from the sun sensor. The measured points are the averages from 100 samples documented for each measured angle. For all directions the offset error stays below  $0.3^\circ$  for the FOV below  $40^\circ$ . Between the  $40^\circ$  and  $60^\circ$  the error increases to a maximum of  $1^\circ$ . For the rest of the

FOV the offset error increases to a maximum of  $1.85^\circ$ , but converges back when reaching the edge of the FOV. It appears as if the distortion correction model over corrects the measurements from the sun sensor, as the difference between the offset error and the expected measurements are always positive, except for the measurements at  $\pm 90^\circ$ .

Figure 7.4 shows the RMS error of the sun sensor in all directions. Equation 7.1.2 is used to calculate the standard deviation for the 100 samples at each measured angle.



**Figure 7.4** – Sun sensor RMS error

Figures 7.4(a) to 7.4(b) show the model created from the standard deviations in each direction. There is no repeating model in any of the figures, as the standard deviations appear to look like random noise. However, the maximum standard deviation is only  $0.045^\circ$ .

### 7.3 Power consumption

Power consumption is determined by measuring the current through a series resistor. Previous current measurements indicated that the maximum current required will be under 100 mA. The series resistor is selected as 10 Ω, giving the measurements a resolution of 10 mV per 1 mA. However, a 10 Ω resistor was not available. Instead two parallel 22 Ω resistors are used. The parallel resistor are measured with a multimeter and measured as 11.1 Ω.

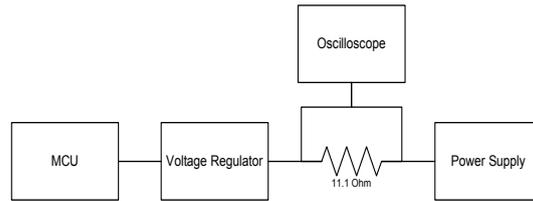


Figure 7.5 – Power measurement

Figure 7.5 shows the setup for measuring the power consumption of the sun and nadir sensors. An oscilloscope is utilised to measure the voltage over the shunt resistor. Table 7.1 shows the results of the power consumption measured from the prototype board. The board is powered by a regulated 5 V voltage rail.

MCU	SRAM	Cameras	FPGA	Voltage measured	Equivalent Current	Power
Reset	Off	Off	2xOn	93 mV	8.4 mA	41.9 mW
Active Standby - MCU & FPGA On				138 mV	12.4 mA	62.2 mW
On	1xOn 1xOff	Off	2xOn	187 mV	16.8 mA	84.2 mW
On	2xOn	Off	2xOn	240 mV	21.6 mA	108.1 mW
On	2xOn	1xOn 1xOff	2xOn	548 mV	49.4 mA	246.8 mW
On	2xOn	2xOn	2xOn	798 mV	71.9 mA	359.5 mW
Active	1xActive 1xOn	1xImaging 1xOn	2xActive	880 mV	79.3 mA	396.4 mW
Active	2xActive	2xImaging	2xActive	960 mV	86.5 mA	432.4 mW

Table 7.1 – Power consumption measured

The power consumption in table 7.1 is the average power measured when the sensors are in different states. The power consumption when an image is being produced, is shown in Figure 7.6. The power required to produce an image increases for 48 ms, where the maximum power during imaging is noted in table 7.1.

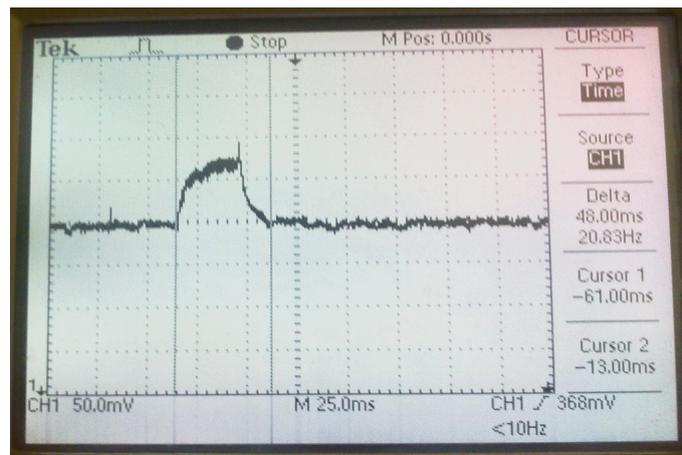
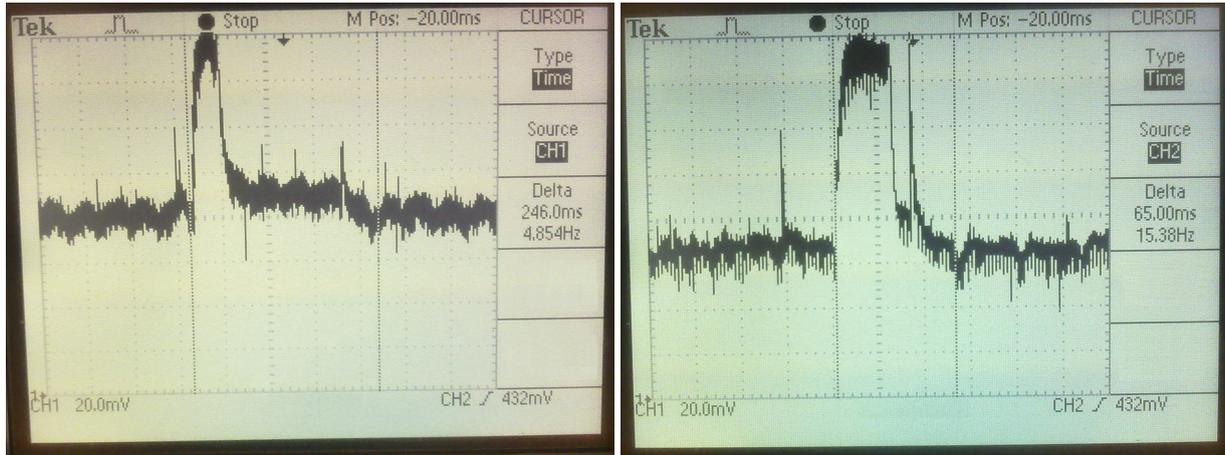


Figure 7.6 – Time required for the current to stabilise after an image is produced

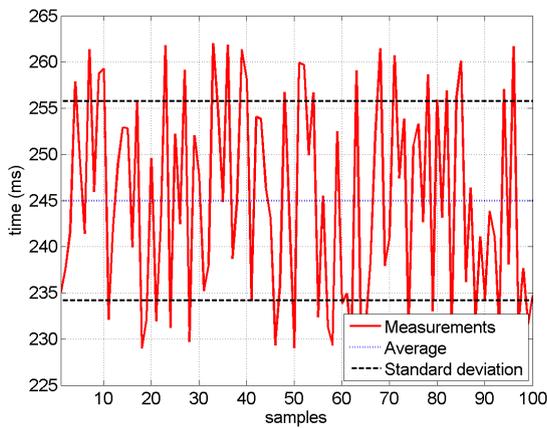
## 7.4 Time requirements

The goal for the sun and nadir sensors are to be able to calculate a centroid at least once every second. Figure 7.7 shows the results of the time measured from the prototype board when calculating the Sun and the Earth's centroids.

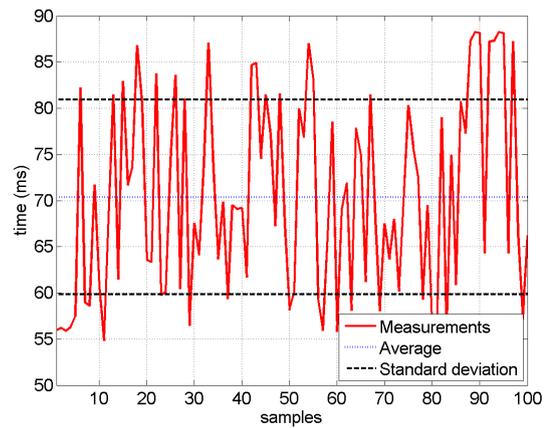


(a) Nadir sensor: length of imaging process

(b) Sun sensor: length of imaging process



(c) Nadir sensor time measurement deviations



(d) Sun sensor time measurement deviations

**Figure 7.7** – Processing time measurements for sun and nadir sensors

Figures 7.7(a) and 7.7(b) show the processing time required for the nadir and sun sensor respectively to complete the process from producing an image to calculating the centroid. The time required to calculate the centroid from the nadir and sun sensor, respectively, is 246 ms and 65 ms, but it is not accurately measured from the oscilloscope.

To verify the time seen on the oscilloscope, a timer on the microcontroller is utilised to measure a more accurate time interval. The on-chip timer is set up to calculate the time with the following equation:

$$\begin{aligned} \text{Time} &= [\text{Timer register}] \times \frac{\text{Prescaler}}{F_{\text{Osc}}} \\ &= [\text{Timer register}] \times \frac{256}{5 \times 10^6} \end{aligned} \quad (7.4.1)$$

Figures 7.7(c) and 7.7(d) show the time measured with the on-chip timer for the nadir and sun sensor, respectively, over a 100 samples. The average time for the nadir and sun sensors to complete the process from producing an image to calculating the centroid is 245.0 ms and 70.4 ms, respectively. Both sensors' time measurements have a standard deviation of 11 ms.

## 7.5 Mass measurements

Since the mass of the satellite is also taken into consideration, because of the limit on the it for CubeSats, the prototype sensors' mass are documented as well. Table 7.2 shows the results of the mass measurements.

<b>Component</b>	<b>Mass</b>	<b>Quantity</b>
Sensor PCB	45 g	1
Fisheye lens	17.4 g	2
Camera	15.1 g	2
<b>Total</b>	110 g	

**Table 7.2** – Total mass of sun and nadir sensor

If the maximum mass for a 1U (1 unit) CubeSat is 1.33 kg, the sun and nadir sensor will be 8.27% of the total mass. Note that the mass measured does not include the components required to hold the hardware in place, for example screws, stand-offs, etc. and the mass was measured using the prototype PCB and not the final PCB. However, the differences between the prototype and the final pcb is minimal.

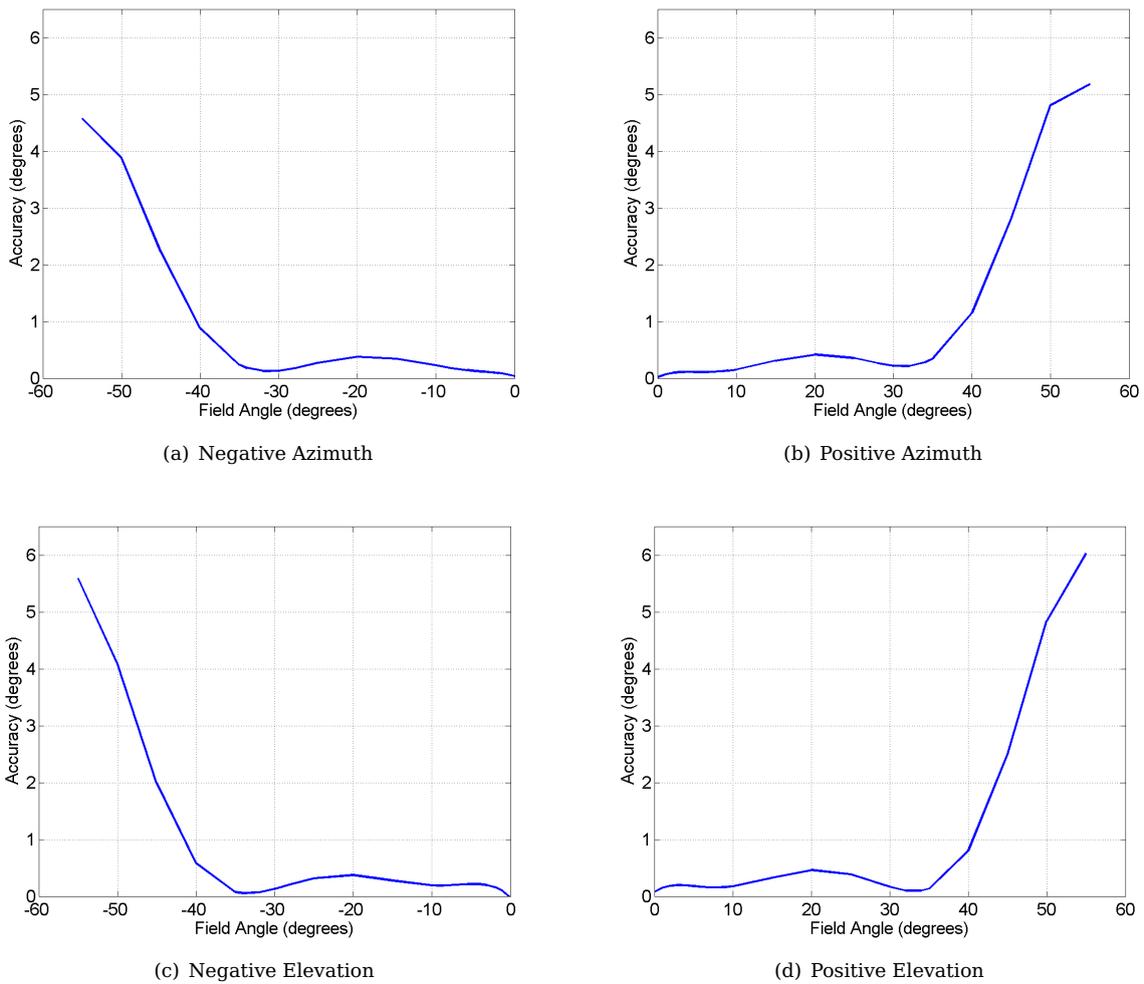
# Chapter 8

## Conclusion

The performance goal given for the nadir sensor is to produce accuracy below  $0.2^\circ$ . The accuracy is calculated as follows:

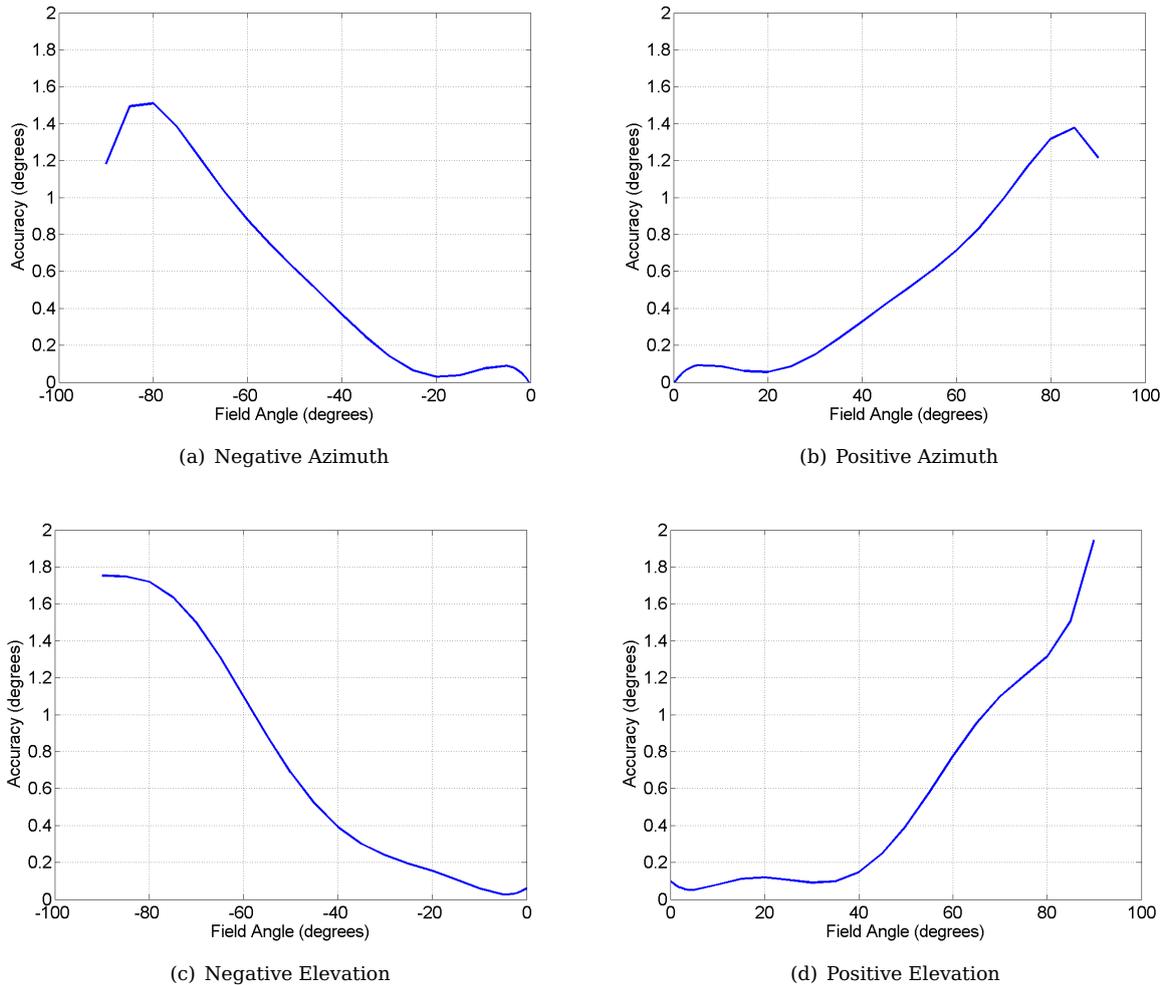
$$\text{Accuracy} = \text{Offset error} + \text{RMS error}$$

Figure 8.1 shows the accuracy for the nadir sensor over the measured FOV. The accuracy for the nadir sensor for a full profile of the Earth varies between  $0.1^\circ$  and  $0.46^\circ$ . The low accuracies are because of offset and RMS errors that increase as the ball moves towards the edge of the FOV.



**Figure 8.1** – Accuracy for nadir sensor

The goal for the sun sensor is to perform with an accuracy of below  $0.2^\circ$  for the FOV of  $\pm 45^\circ$  and below  $2^\circ$  for the FOV between  $\pm 45^\circ$  and  $\pm 90^\circ$ . Figure 8.2 indicates the accuracy for the sun sensor. The accuracy for the sun sensor is calculated similarly to the accuracy of the nadir Sensor. The accuracy stays below  $0.2^\circ$  for a FOV of  $\pm 30^\circ$ . However, the accuracy never increases above  $2^\circ$  for the rest of the measured FOV.



**Figure 8.2** – Accuracy for sun sensor

The power consumption for the sun and nadir sensors appear to be very efficient, as the maximum power required is below 450 mW and this is only required for 48 ms every second. The active standby power is also very efficient at 62.2 mW.

It is clear from the time measurements, that both the sun and nadir sensors' processing times are below the 1 s sampling time which was set as a goal for the project. The average time to complete the process from producing an image to calculating the centroid, is 245.0 ms and 70.4 ms for the nadir and sun sensor, respectively.

Overall the goals set for this project have been met, however recommendations can be made to possibly improve the performance results (see section 9.2).

## Chapter 9

# Summary and Recommendations

### 9.1 Summary

In this thesis the design and calibration of a sun and nadir sensor for a CubeSat are discussed.

CMOS camera modules were given to represent the sun and nadir sensors. Fisheye lenses and a neutral density filter were selected to enable the nadir sensor to see the entire profile of the Earth and to attenuate the sunlight for the sun sensor up to the point where the Sun is represented by a small illuminated area on the image. The hardware surrounding these camera modules were selected to enable the project to store and retrieve images and process these images to the necessary pixels required for the centroid calculations. The choice of memory resulted in SRAM units, as this type of memory has the necessary write speeds and low power to store the images. The memory units are powered by power transistors. These are used for power cycling in the event of a SEE in the memory units. A dual FPGA and microcontroller layout is implemented to control the two camera modules, to store and retrieve images and to calculate the centroid of the Earth and the Sun.

The image processing required to find the centroids of the Earth and the Sun is broken down into thresholding, pixel search algorithms, edge detection, distortion correction and centroid calculations. The method for thresholding presented by Van Rensburg [30] was implemented, where a fixed threshold is determined by the luminance of the Earth or Sun and space. The search algorithms are designed to shorten the time required for edge detection. The edge detection procedure applies the threshold to determine whether pixels are part of the Earth or the Sun and follows set rules to filter out the edge pixels from the Earth and the Sun. The distortion model implemented is the PFET proposed by Basu and Licardie [33]. The least square circle [36] algorithm was selected to calculate the Earth's centroid, as it minimizes the error from the edge pixels. Because the Sun is always a small percentage of the image, all pixels' coordinates above the threshold are averaged to calculate the Sun's centroid.

The positions of the boresight and distortion centre points are determined, as they are required for the centroid calculations and the calibration of the nadir and sun sensors. The focal length of the nadir sensor and the radius of the Sun are determined to find the optimal focus of the sensors. The distortion correction model is determined by measuring the distortion over the FOV of the fisheye lens. The optic point is determined to minimise the errors in the calibration testbenches.

Instead of calculating the distortion correction for each distorted edge pixel, a lookup table is implemented in software to shorten the time required to undistort the edge pixels before calculating the centroids. The search algorithms are implemented as state machines on the FPGA, as the FPGA has direct memory access. The first Sun pixel is determined in parallel with the image being written to memory. A binary search is implemented to search the nadir edge pixels between the sampled pixels for a pixel accuracy. Interpolation is implemented to achieve sub-pixel accuracy during distortion correction. The radii of the nadir edge pixels are calculated to determine whether the pixels are indeed a valid edge or a false edge created by the Earth moving beyond the fisheye lens' FOV.

The results of the nadir body frame angles show that the accuracy of the nadir sensor does not stay below the set goal of  $0.2^\circ$  over the FOV where a full profile of the Earth is visible. There are offset and RMS errors when a full profile of the Earth is visible and these errors increase as the partial Earth profile viewed decreases. The standard deviation of the sun sensor stays below  $0.045^\circ$  throughout the entire FOV. There are, however, large offset errors at the higher end of the FOV. The power and time measurements show that the sun and nadir sensors work well within the goals set for this project.

## 9.2 Recommendations and improvements

### 9.2.1 I<sup>2</sup>C on FPGA

Between the microcontroller and the two FPGAs there are eight control pins (as shown in table 6.3) and eight data pins. These pins are necessary to implement all the functions required for the sun and nadir sensors. If less pins were required, the space required for the layout of the connections between the microcontroller and the two FPGAs would be less, and less power will be required from the microcontroller.

It is possible to scale down the eight control pins and eight data pins between the microcontroller and two FPGAs to just two pins, by implementing an I<sup>2</sup>C module on the two FPGAs. The I<sup>2</sup>C bus only requires a data and clock line and the same I<sup>2</sup>C bus can be used for both FPGAs. Figure 9.1 shows the I<sup>2</sup>C bus implemented on the FPGAs.

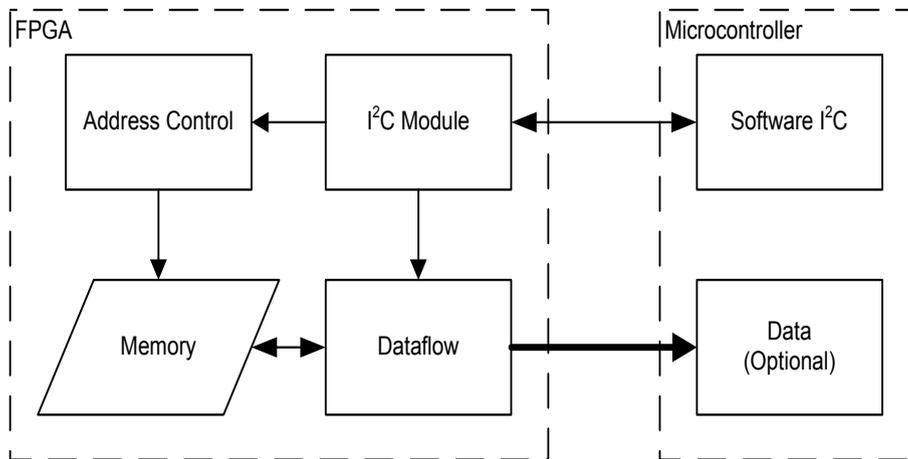


Figure 9.1 - I<sup>2</sup>C module for FPGA

The advantages of implementing an I<sup>2</sup>C module are the following:

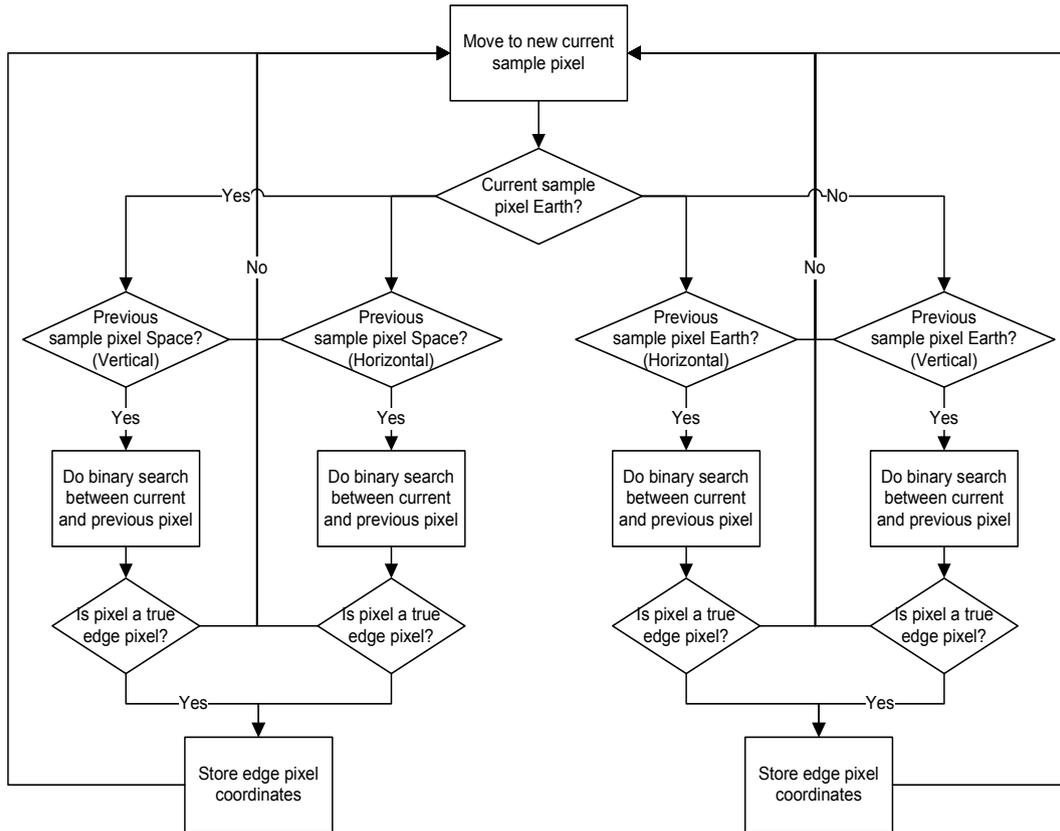
- Less connections are required between components. Therefore the entire layout will be more compact.
- Less power will be required from the microcontroller.
- Address control of the memory will be possible from the microcontroller, as the specific memory addresses that are required for calculations can be generated on the microcontroller and sent to the FPGA and the memory unit via the I<sup>2</sup>C bus.

The disadvantage of implementing an I<sup>2</sup>C module are the following:

- The AGLN030 FPGA might not have enough tiles and gates to accompany an I<sup>2</sup>C module along with the rest of the modules already implemented on the FPGA.
- More time may be required to compute the function of the sun and nadir sensors, as the I<sup>2</sup>C bus is a serial process and the direct control pins are a parallel process.

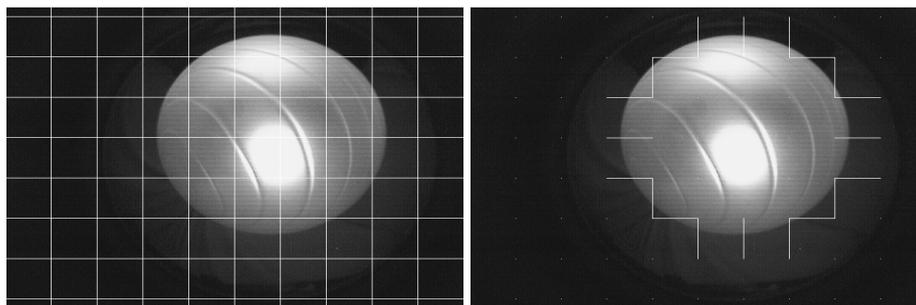
**9.2.2 Address control from microcontroller**

Figure 6.8 shows the flow diagram of the edge detection that is repeated during the execution of both horizontal and vertical search algorithms of the nadir sensor. However, if the microcontroller could control the addressing of the memory unit directly, the horizontal and vertical search algorithms can be simplified to a single search algorithm and therefore the edge detection can be simplified to a single flow diagram as well. The simplified edge detection is shown in figure 9.2.



**Figure 9.2** – Edge detection for nadir sensor with address control on microcontroller

The edge detection of the nadir sensor will still follow the same edge detection rules, as stated in section 4.2.2. However, the current sample pixel will be tested in both vertical and horizontal directions before moving to the next current sample pixel.



(a) Implemented edge detection. Address (b) Edge detection with address control performed on microcontroller

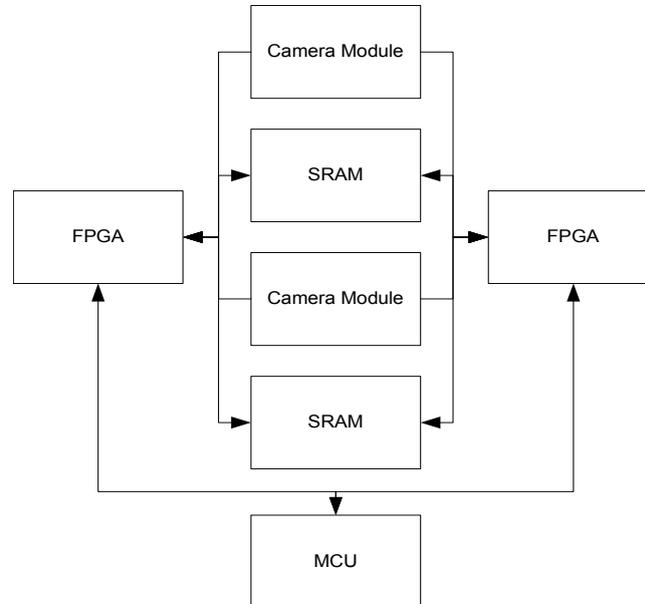
**Figure 9.3** – Difference in edge detection procedures

Figure 9.3 shows the difference between the current search algorithms implemented and the single search

algorithm that could be implemented if the microcontroller had control over the memory unit's addressing. The lines in figures 9.3(a) and 9.3(b) show the pixels that may possibly be searched for edge pixels. It is clear that the search algorithm in figure 9.3(b) searches through much less pixels than the search algorithms in figure 9.3(a) and therefore the search algorithm uses less time to complete.

### 9.2.3 Alternative hardware layout

Figure 9.4 shows a different layout to the one implement in this project. The difference between this alternative layout and the one implemented is that only one FPGA is required to be active when processing the images. The second FPGA would be a backup FPGA.



**Figure 9.4** - Alternative hardware layout

The advantages of implementing this layout are the following:

- Fewer control pins are necessary, as the same control pins are implemented for both camera modules and memory units.
- The images can be processed faster, because only one command from the microcontroller and the parallel processing ability of the FPGA are required to activate both camera modules and store both images.

The disadvantages of implementing this layout are the following:

- Although the footprint of most IGLOO Nano FPGAs are constant (therefore requiring no extra space on the PCB) the current AGLN030 does not have enough gates and tiles to implement both sun and nadir sensors' search algorithms.
- With more gates and tiles active in an FPGA, more power is required when using the FPGA.
- The system would become more complex, because buffers would be incorporated to ensure that if the primary FPGA would fail, it would not prohibit the backup FPGA of taking control of the system.

This option for alternative layout should only be considered when less processing time becomes a higher priority than a lower power consumption.

# Appendix A

## Datasheets

### A.1 C3188A camera module

### C3188A 1/3" Color Camera Module With Digital Output

#### General Description

The C3188A is a 1/3" color camera module with digital output. It uses OmniVision's CMOS image sensor OV7620. Combining CMOS technology together with an easy to use digital interface makes C3188A a low cost solution for higher quality video image application.



The digital video port supplies a continuous 8/16 bit-wide image data stream. All camera functions, such as exposure, gamma, gain, white balance, color matrix, windowing, are programmable through I<sup>2</sup>C interface.

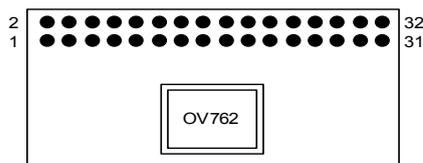
If combined with an OV511+ (USB controller chip) it can easily form a USB camera for PC applications.

#### Features:

- 326,688 pixels, VGA / CIF format
- Small size: 40 x 28 mm
- Lens: f=6mm (Optional)
- 8/16 bit video data: CCIR601, CCIR656, ZV port
- Read out - progressive / interlace
- Data format -YCrCb 4:2:2, GRB 4:2:2, RGB
- I<sup>2</sup>C interface
- Built in 10bit 2 ch A/D converter
- Electronic exposure / Gain / White balance control
- Image enhancement - brightness, contrast, gamma, saturation, sharpness, window, etc
- Internal / external synchronization scheme
- Frame exposure / line exposure option
- Wide dynamic range, anti blooming, zero smearing
- Single 5V operation
- Low power consumption (<120mW)
- Monochrome composite video signal output (60Hz)

#### Specification

Imager	OV7620, CMOS image sensor
Array Size	664x492 pixels
Pixel size	7.6 x 7.6 $\mu$ m
Scanning	Progressive / interlace
Effective image area	4.86mm x 3.64mm
Electronic Exposure	500:1
Gamma Correction	128 curve settings
S/N Ratio	>48dB
Min Illumination	2.5lux @F1.4
Operation Voltage	5VDC
Operation Current	120mW Active 10 $\mu$ W Standby
Lens	f6mm, F1.6



PCB Layout (Top view)

#### Pin Description

1~8	Y0~Y7	Digital output Y Bus.
9	PWDN	Power down mode
10	RST	Reset
11	SDA	I <sup>2</sup> C Serial data
12	FODD	Odd Field flag
13	SCL	I <sup>2</sup> C Serial clock input
14	HREF	Horizontal window reference output
15	AGND	Analog Ground
16	VSYN	Vertical Sync output
17	AGND	Analog Ground
18	PCLK	Pixel clock output
19	EXCLK	External clock input (need to remove crystal)
20	VCC	Power Supply 5VDC
21	AGND	Analog Ground
22	VCC	Power Supply 5VDC
23~30	UV0-UV7	Digital output UV bus.
31	GND	Common ground
32	VTO	Video Analog Output (75 $\Omega$ monochrome)

#### Application Example

- | Video Conferencing
- | PC Multimedia
- | Video Phone
- | Video Mail
- | Still Image
- | Machine Vision
- | Process control

Note: Evaluation Board is available for C3188A

## A.2 OV7620 colour image sensor



Preliminary Company Confidential  
 OV7620 Product Specifications -Rev. 1.3 (5/13/00)

### OV7620 SINGLE-CHIP CMOS VGA COLOR DIGITAL CAMERA

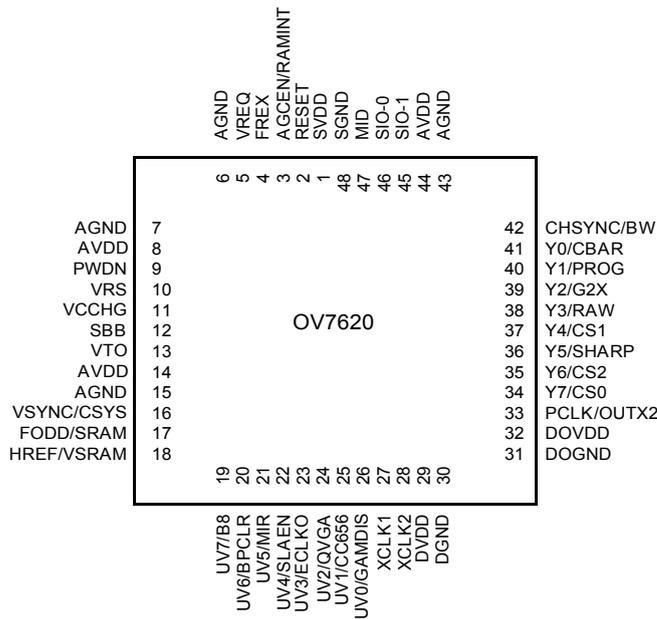
#### Features

- 326,688 pixels, 1/3" lens, VGA / QVGA format
- Read out - progressive / Interlace
- Data format - YCrCb 4:2:2, GRB 4:2:2, RGB Raw Data
- 8/16 bit video data: CCIR601, CCIR656, ZV port
- Video Timing - 525 line, 30 fps
- Wide dynamic range, anti-blooming, zero smearing
- SCCB (Serial Camera Control Bus) interface
- Electronic exposure / Gain / white balance control
- Image enhancement - brightness, contrast, gamma, saturation, sharpness, window, etc.
- Internal / external synchronization scheme
- Frame exposure / line exposure option
- 5 Volt operation, low power dissipation.

#### General Description

OV7620 is a highly integrated high resolution (640x480) Interlaced / Progressive Scan CMOS digital color / black&white video camera chip. The digital video port supports 60Hz YCrCb 4:2:2 16Bit / 8 Bit format, ZV Port output format, RGB raw data 16Bit/8Bit output format and CCIR601/CCIR656 format. The built-in SCCB interface provides an easy way of controlling the built-in camera functions.

- Video Conferencing
- Video Phone
- Video Mail
- Still Image
- PC Multimedia



OV7001 48-Pin Out Diagram

Array Elements	664 x 492
Pixel Size	7.6 x 7.6 um
Image Area	4.86 x 3.64mm
Electronic Exposure	500 : 1
Scan Mode	progressive interlace
Gamma Correction	128 Curve Settings See specifics
Minimum Illumination	2.5 lux @ f1.4 0.5 lux @ f1.4 (3000K)
S/N Ratio	> 48dB
Power Supply	5VDC, ±5%
Power Requirements	<120mW Active <10uW Standby
Package	48-pin LCC

Preliminary Company Confidential  
 OV7620 Product Specifications - Rev. 1.2 (5/13/00) OMNIVISION TECHNOLOGIES INC.

---

To decrease data transfer rate while high resolution image unnecessary, OV7620 provide a solution, that is it can output QVGA resolution image. This mode decrease pixel rate one half. The resolution default value is 320x240 and can be programmable. Every line only output one half data. For Interlaced Mode, all field line output (320), for Progressive Scan Mode, only one half line data output.

The digital video port also offer RGB Raw Data 16 Bit/8 Bit format. The output sequence is matched to OV7620 Color Filter Pattern, that is UV channel output sequence is G R G R ..., Y Channel output sequence is B G B G,....To 8 Bit RGB Raw data output format, just use Y channel and disable UV channel, output sequence is B G R G ....

OV7620 support CCIR656 YCrCb 4:2:2 digital output format. The SAV(Start of Active Video) and EAV(End of Active Video) is just at the beginning and the end of HREF window. So the position of SAV and EAV is changing with active pixel window. Also you can get 8 bit RGB raw data with SAV and EAV information.

OV7620 support some flexible YUV output format. One is standard YUV 4:2:2. Another is U V sequence swap format, that means UV channel output V U V U ...(16 Bit) and V Y U Y ...(8 Bit). The 3rd format is Y/UV sequence swap in 8 Bit output, that is Y U Y V ....

OV7620 can be use as black&white camera. At this mode, it's vertical resolution will be higher than color mode. All data will be output from Y port and UV port will be tri-state. Data (Y/RGB) output rate is same as 16 Bit mode.

OV7620 can be programmable to swap Y/UV or RGB output byte MSB and LSB. Y7 - Y0 default sequence is Y7 is MSB and Y0 is LSB. When swap, Y7 is LSB and Y0 is MSB, relative middle bits are swapped.

An important factor about digital camera is how convenient the interface is, OV7620 has made the frame rate programmable and the A/D synchronous to the actual pixel rate. Essentially, it is a whole image capture system in a single chip. Since the internal AEC has a range of 1:260, and AGC have 24dB, for the most of applications, the camera can adjust itself to meet the lighting condition without user intervention.

OV7620 support hardware/software RESET function: when RESET pin tie to high, whole chip will be reset including all register. Hardware sleeping mode: when PWDN tie to high, chip clock will be stop and internal circuit reset except all SCCB register. Also there is a SCCB control software reset control register 12 bit 7, which is same as hardware RESET pin function.

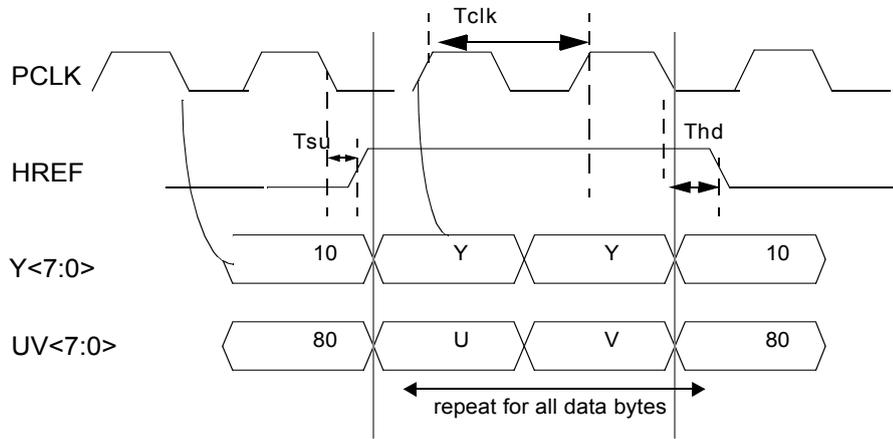
OV7620 hardware reset time minimum is 1 ms.

OV7620 support hardware/software power saving mode. When the PWDN pin tie to high, whole chip will be set to power down status without any current consumption. For software power down control, all current set to zero except crystal circuit. In power down mode, all SCCB register value will be kept.

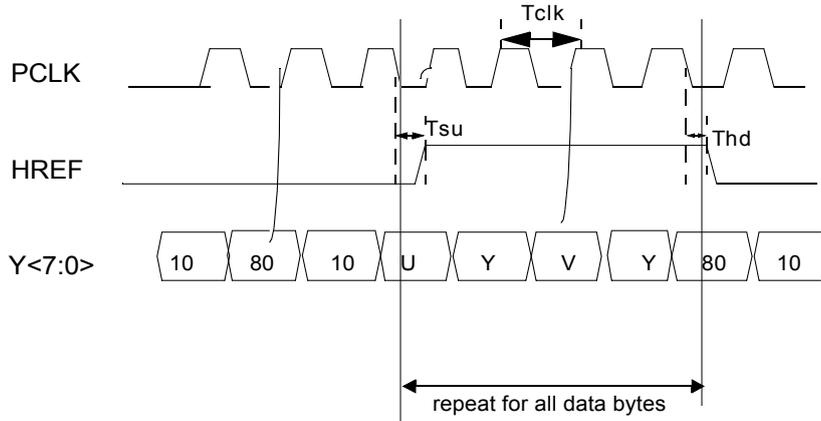
Two control mechanism have been built into OV7620: A. one time read-in of pin states at power up or RESET status, including hardware and software reset; B. SCCB interface. Two methods are mutually exclusive, only one is used at a time, selected by pin SBB. Method A has limited access to full chip features.

The power up reset method is a one time setting, the setting can not be altered later. The pins

---



Pixel Data 16 bit Timing  
 Use PCLK rising edge latch data bus



Pixel Data 8 bit Timing  
 Use PCLK rising edge latch data bus

FIG 1.2 Pixel Data Bus (YUV Output)

Note: Tclk is pixel clock period. When OV7620 system clock is 27MHz, Tclk=74ns for 16 Bit output; Tclk=37ns for 8 Bit output. Tsu is HREF set-up time, maximum is 15 ns; Thd is HREF hold time, maximum is 15 ns.

**A.3 ORIFL190-3 fisheye lens**



ORIFL190-3

190 Degree Field Of View Fisheye Lens  
For 1/3" Format Cameras

**Specifications:**

- Field of View: 190 degrees
- Focal Plane Field Diameter: 3.4 mm
- Focal length: 1.24 mm
- F/number: 2.8
- Focus range: 0.5 inches to infinity
- MTF @ 70 cycles/mm (with 640x480 sensors):
  - 76% on symmetrical axis
  - >72% throughout 190 degree field
- MTF @ 150 cycles/mm (with 1280x1024 sensors):
  - 42% on symmetrical axis
  - >42% throughout field
- Maximum Image Height Distortion measured from F-theta condition, at edge of field: 17.3%
- Lens Housing Outer Diameter: 0.943" (23.95mm)
- Lens Length: 1.016": (25.82mm)
- Lens Mount: Micro 12mm x 0.5mm thread
- Back Focal Length 2.77 mm.
- Lens body to Image Sensor Dimension 2.25 mm.



F1: The ORIFL190-3 Lens

**Introduction:**

The ORIFL190-3 is a high quality fisheye lens that provides a 190 degree field of view. The circular image produced is 3.4 mm in diameter allowing 1/3 inch format cameras/sensors to capture a symmetrical hemispherical image. It is also compatible with 1/2" and 1/4" format cameras/sensors although the viewable field will vary. Designed and built exclusively by and for Omnitech Robotics, the ORIFL190-3 is optimized for small size and high image quality. The anodized aluminum lens body is only 24 mm in diameter, yet the optics have excellent sharpness, contrast, field compression linearity, and field luminance and color correctness throughout the field of view. The glass lens construction and coated optics provide a 1.24 mm focal length and F/2.8 speed for good low light capability. The large primary lens uses an O-ring seal to provide water and humidity resistance. Compatibility with most web-cam, circuit board and bullet style cameras is provided with the standard "micro mount" 12mm x 0.5mm pitch mounting thread.



F2: Example Image

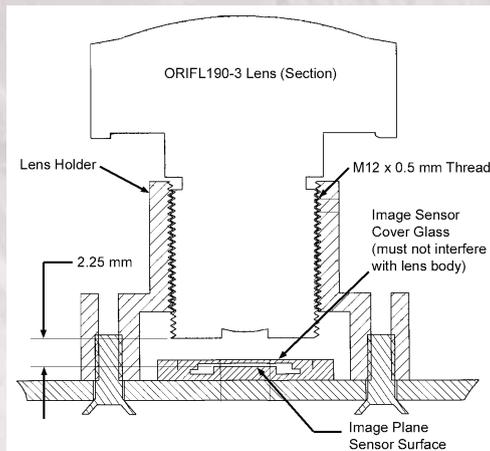


F3: The ORIFL190-3 is compatible with a range of cameras

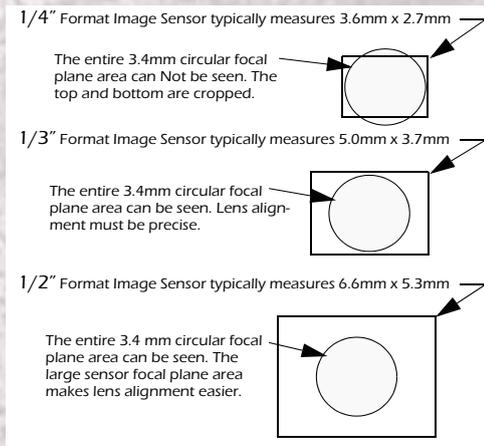
length. The gap between the final glass lens element and the image plane (Back Focal Length) is 2.77 mm when the lens is properly installed and in focus. The gap between the end of the lens housing and the image plane is 2.25 mm when the lens is properly installed and in focus. If the image sensor uses a cover glass or filter, it must be less than this dimension. Figure F5 illustrates nominal mounting dimensions for reference. If needed, ORI also sells a very thin (0.5 mm) visible bandpass filter that cuts IR and UV transmission to less than 3%, while passing the visible spectrum from 380 nm to 650 nm (94% typical), to provide improved color correctness of CMOS or CCD image sensors.

A third factor to consider is the electronic shutter capability of the camera, since there is not a mechanical aperture control or shutter on the lens. A fast electronic shutter (exposure time) may be necessary to obtain a proper exposure in bright light conditions. This is possible with most CMOS cameras, but may difficult be with older or high sensitivity CCD cameras and bright lighting conditions.

Figures F7 through F12 show sample images for the ORIFL190-3. Figures F13 through F17 illustrate design curves for the lens performance.



F5: Nominal mounting dimensions for the ORIFL190-3 fisheye lens with a PCB camera image sensor



F6: Comparison of the ORIFL190-3 focal plane area to different image sensor sizes

**Potential Applications**

- Wide Field Of View Robotic Vehicle Image Sensing
- Wide Field Of View Security Cameras
- Wide Field Of View Astronomy Applications: All sky imaging, cloud cover measurement, light pollution measurement.

**A.4 AS7C34096A SRAM**

August 2004
AS7C34096A



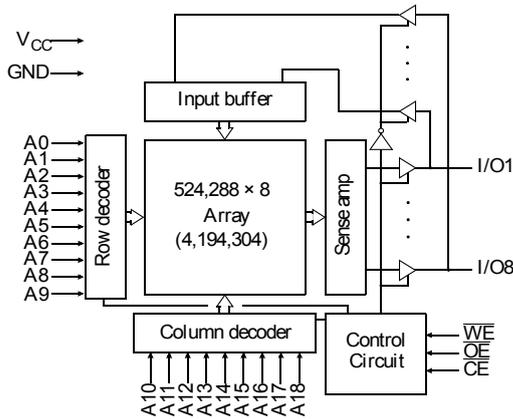
3.3V 512K × 8 CMOS SRAM

**Features**

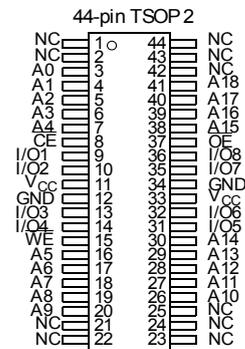
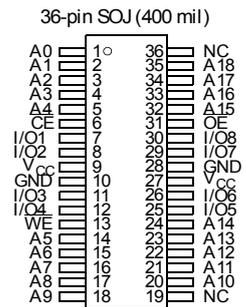
- Pin compatible to AS7C34096
- Industrial and commercial temperature
- Organization: 524,288 words × 8 bits
- Center power and ground pins
- High speed
  - 10/12/15/20 ns address access time
  - 4/5/6/7 ns output enable access time
- Low power consumption: ACTIVE
  - 650 mW / max @ 10 ns
- Low power consumption: STANDBY
  - 28.8 mW / max CMOS

- Equal access and cycle times
- Easy memory expansion with  $\overline{CE}$ ,  $\overline{OE}$  inputs
- TTL-compatible, three-state I/O
- JEDEC standard packages
  - 400 mil 36-pin SOJ
  - 44-pin TSOP 2
- ESD protection ≥ 2000 volts
- Latch-up current ≥ 200 mA

**Logic block diagram**



**Pin arrangements**



**Selection guide**

	-10	-12	-15	-20	Unit	
Maximum address access time	10	12	15	20	ns	
Maximum outputenable access time	4	5	6	7	ns	
Maximum operating current	Industrial	180	160	140	110	mA
	Commercial	170	150	130	100	mA
Maximum CMOS standby current	8	8	8	8	mA	

## A.5 IRF7210PbF power MOSFET

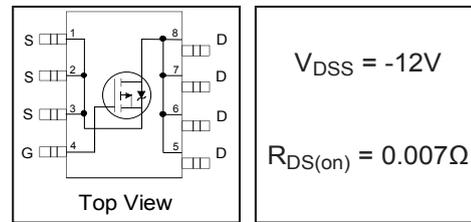


PD - 97040

# IRF7210PbF

HEXFET® Power MOSFET

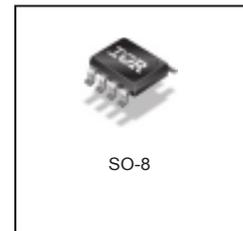
- | Ultra Low On-Resistance
- | P-Channel MOSFET
- | Surface Mount
- | Available in Tape & Reel
- | Lead-Free



### Description

These P-Channel MOSFETs from International Rectifier utilize advanced processing techniques to achieve the extremely low on-resistance per silicon area. This benefit provides the designer with an extremely efficient device for use in battery and load management applications.

The SO-8 has been modified through a customized leadframe for enhanced thermal characteristics and multiple-die capability making it ideal in a variety of power applications. With these improvements, multiple devices can be used in an application with dramatically reduced board space. The package is designed for vapor phase, infra red, or wave soldering techniques.



### Absolute Maximum Ratings

	Parameter	Max.	Units
V <sub>DS</sub>	Drain- Source Voltage	-12	V
I <sub>D</sub> @ T <sub>A</sub> = 25°C	Continuous Drain Current, V <sub>GS</sub> @ -4.5V	±16	A
I <sub>D</sub> @ T <sub>A</sub> = 70°C	Continuous Drain Current, V <sub>GS</sub> @ -4.5V	±12	
I <sub>DM</sub>	Pulsed Drain Current <sup>Ⓜ</sup>	±100	
P <sub>D</sub> @ T <sub>A</sub> = 25°C	Power Dissipation	2.5	W
P <sub>D</sub> @ T <sub>A</sub> = 70°C	Power Dissipation	1.6	
	Linear Derating Factor	0.02	W/°C
V <sub>GS</sub>	Gate-to-Source Voltage	± 12	V
V <sub>GSM</sub>	Gate-to-Source Voltage Single Pulse tp<10μs	16	V
T <sub>J</sub> , T <sub>STG</sub>	Junction and Storage Temperature Range	-55 to + 150	°C

### Thermal Resistance

	Parameter	Max.	Units
R <sub>θJA</sub>	Maximum Junction-to-Ambient <sup>Ⓜ</sup>	50	°C/W

www.irf.com

1

08/19/05

## A.6 INA139 high-side measurement shunt monitor



Burr-Brown Products  
from Texas Instruments



INA139  
INA169

SBOS181D – DECEMBER 2000 – REVISED NOVEMBER 2005

# High-Side Measurement CURRENT SHUNT MONITOR

## FEATURES

- q COMPLETE UNIPOLAR HIGH-SIDE CURRENT MEASUREMENT CIRCUIT
- q WIDE SUPPLY AND COMMON-MODE RANGE
- q INA139: 2.7V to 40V
- q INA169: 2.7V to 60V
- q INDEPENDENT SUPPLY AND INPUT COMMON-MODE VOLTAGES
- q SINGLE RESISTOR GAIN SET
- q LOW QUIESCENT CURRENT (60µA typ)
- q SOT23-5 PACKAGE

## APPLICATIONS

- q CURRENT SHUNT MEASUREMENT: Automotive, Telephone, Computers
- q PORTABLE AND BATTERY-BACKUP SYSTEMS
- q BATTERY CHARGERS
- q POWER MANAGEMENT
- q CELL PHONES
- q PRECISION CURRENT SOURCE

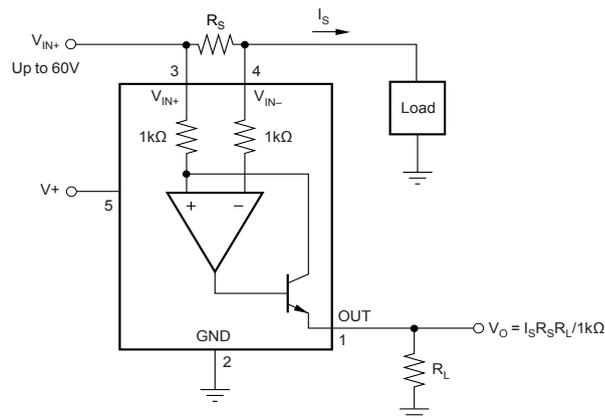
## DESCRIPTION

The INA139 and INA169 are high-side, unipolar, current shunt monitors. Wide input common-mode voltage range, high-speed, low quiescent current, and tiny SOT23 packaging enable use in a variety of applications.

Input common-mode and power-supply voltages are independent and can range from 2.7V to 40V for the INA139 and 2.7V to 60V for the INA169. Quiescent current is only 60µA, which permits connecting the power supply to either side of the current measurement shunt with minimal error.

The device converts a differential input voltage to a current output. This current is converted back to a voltage with an external load resistor that sets any gain from 1 to over 100. Although designed for current shunt measurement, the circuit invites creative applications in measurement and level shifting.

Both the INA139 and INA169 are available in SOT23-5 packages and are specified for the -40°C to +85°C industrial temperature range.



Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

All trademarks are the property of their respective owners.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS  
INSTRUMENTS  
www.ti.com

Copyright © 2000-2005, Texas Instruments Incorporated

## A.7 PIC18F45K20 microcontroller



# PIC18F2XK20/4XK20

## 28/40/44-Pin Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology

### Power-Managed Modes:

- Run: CPU on, Peripherals on
- Idle: CPU off, Peripherals on
- Sleep: CPU off, Peripherals off
- Idle Mode Currents Down to 1.0  $\mu$ A, typical
- Sleep Mode Current Down to 0.1  $\mu$ A, typical
- Timer1 Oscillator: 1.0  $\mu$ A, 32 kHz, 1.8V, typical
- Watchdog Timer: 2.0  $\mu$ A, 1.8V, typical
- Two-Speed Oscillator Start-up

### Peripheral Highlights:

- High-Current Sink/Source 25 mA/25 mA
- Three Programmable External Interrupts
- Four Independent Input-Change Interrupts
- 8 Independent Weak Pull-ups
- Programmable Slew Rate
- Capture/Compare/PWM (CCP) module
- Enhanced Capture/Compare/PWM (ECCP) module:
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-Shutdown and Auto-Restart
- Master Synchronous Serial Port (MSSP) module supporting 3-wire SPI (all 4 modes) and I<sup>2</sup>C™ Master and Slave modes with address mask
- Enhanced Addressable USART module:
  - Supports RS-485, RS-232 and LIN/J2602
  - RS-232 operation using internal oscillator block (no external crystal required)
  - Auto-Wake-up on Break
  - Auto-Baud Detect
- 10-bit, up to 14-Channel Analog-to-Digital Converter module (ADC):
  - Auto-acquisition capability
  - Conversion available during Sleep
  - Internal 1.2V Fixed Voltage Reference (FVR) channel
  - Independent input multiplexing
- Dual Analog Comparators:
  - Rail-to-rail operation
  - Independent input multiplexing
- Programmable On-Chip Voltage Reference (CVREF) module (% of VDD)

### Flexible Oscillator Structure:

- Four Crystal modes, up to 64 MHz
- 4X Phase Lock Loop (available for crystal and internal oscillators)
- Two External RC modes, up to 4 MHz
- Two External Clock modes, up to 64 MHz
- Internal Oscillator Block:
  - 8 user selectable frequencies, from 31 kHz to 16 MHz
  - Provides a complete range of clock speeds from 31 kHz to 64 MHz when used with PLL
  - User tunable to compensate for frequency drift
- Secondary Oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if primary or secondary oscillator stops

### Special Microcontroller Features:

- C Compiler Optimized Architecture:
  - Optional extended instruction set designed to optimize re-entrant code
- Self-Programmable under Software Control
- Priority Levels for Interrupts
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- Single-Supply 3V In-Circuit Serial Programming™ (ICSP™) via two pins
- In-Circuit Debug (ICD) via Two Pins
- Operating Voltage Range: 1.8V to 3.6V
- Programmable 16-Level High/Low-Voltage Detection (HLVD) module:
  - Supports interrupt on High/Low-Voltage Detection
- Programmable Brown-out Reset (BOR):
  - With software enable option

## PIC18F2XK20/4XK20

Device	Program Memory		Data Memory		I/O <sup>(1)</sup>	10-bit A/D (ch) <sup>(2)</sup>	CCP/ ECCP (PWM)	MSSP		EUSART	Comp.	Timers 8/16-bit
	Flash (bytes)	# Single-Word Instructions	SRAM (bytes)	EEPROM (bytes)				SPI	Master I <sup>2</sup> C™			
PIC18F23K20	8K	4096	512	256	25	11	1/1	Y	Y	1	2	1/3
PIC18F24K20	16K	8192	768	256	25	11	1/1	Y	Y	1	2	1/3
PIC18F25K20	32K	16384	1536	256	25	11	1/1	Y	Y	1	2	1/3
PIC18F26K20	64k	32768	3936	1024	25	11	1/1	Y	Y	1	2	1/3
PIC18F43K20	8K	4096	512	256	36	14	1/1	Y	Y	1	2	1/3
PIC18F44K20	16K	8192	768	256	36	14	1/1	Y	Y	1	2	1/3
PIC18F45K20	32K	16384	1536	256	36	14	1/1	Y	Y	1	2	1/3
PIC18F46K20	64k	32768	3936	1024	36	14	1/1	Y	Y	1	2	1/3

Note 1: One pin is input only.

2: Channel count includes internal fixed voltage reference channel.

## PIC18F2XK20/4XK20

### 26.0 ELECTRICAL CHARACTERISTICS

#### Absolute Maximum Ratings †)

Ambient temperature under bias .....	-40°C to +125°C
Storage temperature .....	-65°C to +150°C
Voltage on any pin with respect to V <sub>SS</sub> (except V <sub>DD</sub> , and $\overline{\text{MCLR}}$ ) .....	-0.3V to (V <sub>DD</sub> + 0.3V)
Voltage on V <sub>DD</sub> with respect to V <sub>SS</sub> .....	-0.3V to +5.0V
Voltage on $\overline{\text{MCLR}}$ with respect to V <sub>SS</sub> (Note 2) .....	0V to +12.5V
Total power dissipation (Note 1) .....	1.0W
Maximum current out of V <sub>SS</sub> pin .....	300 mA
Maximum current into V <sub>DD</sub> pin .....	250 mA
Input clamp current, I <sub>IK</sub> (V <sub>I</sub> < 0 or V <sub>I</sub> > V <sub>DD</sub> ) .....	±20 mA
Output clamp current, I <sub>OK</sub> (V <sub>O</sub> < 0 or V <sub>O</sub> > V <sub>DD</sub> ) .....	±20 mA
Maximum output current sunk by any I/O pin .....	25 mA
Maximum output current sourced by any I/O pin .....	25 mA
Maximum current sunk by all ports .....	200 mA
Maximum current sourced by all ports .....	200 mA

Note 1: Power dissipation is calculated as follows:

$$P_{dis} = V_{DD} \times \{I_{DD} - \sum I_{OH}\} + \sum \{(V_{DD} - V_{OH}) \times I_{OH}\} + \sum (V_{OL} \times I_{OL})$$

- 2: Voltage spikes below V<sub>SS</sub> at the  $\overline{\text{MCLR}}$ /V<sub>PP</sub>/RE3 pin, inducing currents greater than 80 mA, may cause latch-up. Thus, a series resistor of 50-100Ω should be used when applying a "low" level to the  $\overline{\text{MCLR}}$ /V<sub>PP</sub>/RE3 pin, rather than pulling this pin directly to V<sub>SS</sub>.

† NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at those or any other conditions above those indicated in the operation listings of this specification is not implied. Exposure to maximum rating conditions for extended periods may affect device reliability.

## A.8 IGLOO Nano AGLN030 FPGA



Advance v0.7

# IGLOO nano Low-Power Flash FPGAs with Flash\* Freeze Technology



## Features and Benefits

### Low Power

- nanoPower Consumption—Industry's Lowest Power
- 1.2 V to 1.5 V Core Voltage Support for Low Power
- Supports Single-Voltage System Operation
- Low-Power Active FPGA Operation
- Flash\* Freeze Technology Enables Ultra-Low Power Consumption while Maintaining FPGA Content
- Easy Entry to / Exit from Ultra-Low-Power Flash\* Freeze Mode

### Small Footprint Packages

- As Small as 3x3 mm in Size

### Wide Range of Features

- 10 k to 250 k System Gates
- Up to 36 kbits of True Dual-Port SRAM
- Up to 71 User I/Os

### Reprogrammable Flash Technology

- 130-nm, 7-Layer Metal, Flash-Based CMOS Process
- Live-at-Power-Up (LAPU) Level 0 Support
- Single-Chip Solution
- Retains Programmed Design When Powered Off

### In-System Programming (ISP) and Security

- Secure ISP Using On-Chip 128-Bit Advanced Encryption Standard (AES) Decryption via JTAG (IEEE 1532-compliant)
- FlashLock® to Secure FPGA Contents

### High-Performance Routing Hierarchy

- Segmented, Hierarchical Routing and Clock Structure

### Advanced I/Os

- 1.2 V, 1.5 V, 1.8 V, 2.5 V, and 3.3 V Mixed-Voltage Operation
- Bank-Selectable I/O Voltages—up to 4 Banks per Chip
- Single-Ended I/O Standards: LVTTTL, LVCMOS 3.3 V / 2.5 V / 1.8 V / 1.5 V / 1.2 V
- Wide Range Power Supply Voltage Support per JESD8-B, Allowing I/Os to Operate from 2.7 V to 3.6 V
- Wide Range Power Supply Voltage Support per JESD8-12, Allowing I/Os to Operate from 1.14 V to 1.575 V
- I/O Registers on Input, Output, and Enable Paths
- Selectable Schmitt Trigger Inputs
- Hot-Swappable and Cold-Sparing I/Os
- Programmable Output Sew Rate and Drive Strength
- Weak Pull-Up/-Down
- IEEE 1149.1 (JTAG) Boundary Scan Test
- Pin-Compatible Packages across the IGLOO Family

### Clock Conditioning Circuit (CCC) and PLL†

- Up to 6x CCC Blocks, One with an Integrated PLL
- Configurable Phase Shift, Multiply/Divide, Delay Capabilities, and External Feedback
- Wide Input Frequency Range (1.5 MHz up to 250 MHz)

### Embedded Memory

- 1 kbit of FlashROM User Nonvolatile Memory
- SRAMs and FIFOs with Variable-Aspect-Ratio 4,608-Bit RAM Blocks (x1, x2, x4, x9, and x18 organizations)†
- True Dual-Port SRAM (except x 18 organization)†

### Enhanced Commercial Temperature Range

- -20°C to +70°C

Table 1 • IGLOO nano Devices

IGLOO nano Devices	AGLN010	AGLN015	AGLN020	AGLN030 <sup>†</sup>	AGLN060	AGLN125	AGLN250
System Gates	10 k	15 k	20 k	30 k	60 k	125 k	250 k
Typical Equivalent Macrocells	86	128	172	256	512	1,024	2,048
VersaTiles (D-flip-flops)	260	384	520	768	1,536	3,072	6,144
Flash* Freeze Mode (typical, $\mu$ W)	2	4	4	5	10	16	24
RAM kbits (1,024 bits) <sup>2</sup>	—	—	—	—	18	36	36
4,608-Bit Blocks <sup>2</sup>	—	—	—	—	4	8	8
FlashROM Bits	1 k	1 k	1 k	1 k	1 k	1 k	1 k
Secure (AES) ISP <sup>2</sup>	—	—	—	—	Yes	Yes	Yes
Integrated PLL in CCCs <sup>2,3</sup>	—	—	—	—	1	1	1
VersaNet Globals	4	4	4	6	18	18	18
I/O Banks	2	3	3	2	2	2	4
Maximum User I/Os (packaged device)	34	49	52	77	71	71	68
Maximum User I/Os (Known Good Die)	34	—	52	83	71	71	68
Package Pins							
UC/CS	UC36		UC81, CS81	UC81, CS81	CS81	CS81	CS81
QFN	QN48	QN68	QN68	QN48, QN68			
VQFP				VQ100	VQ100	VQ100	VQ100

#### Notes:

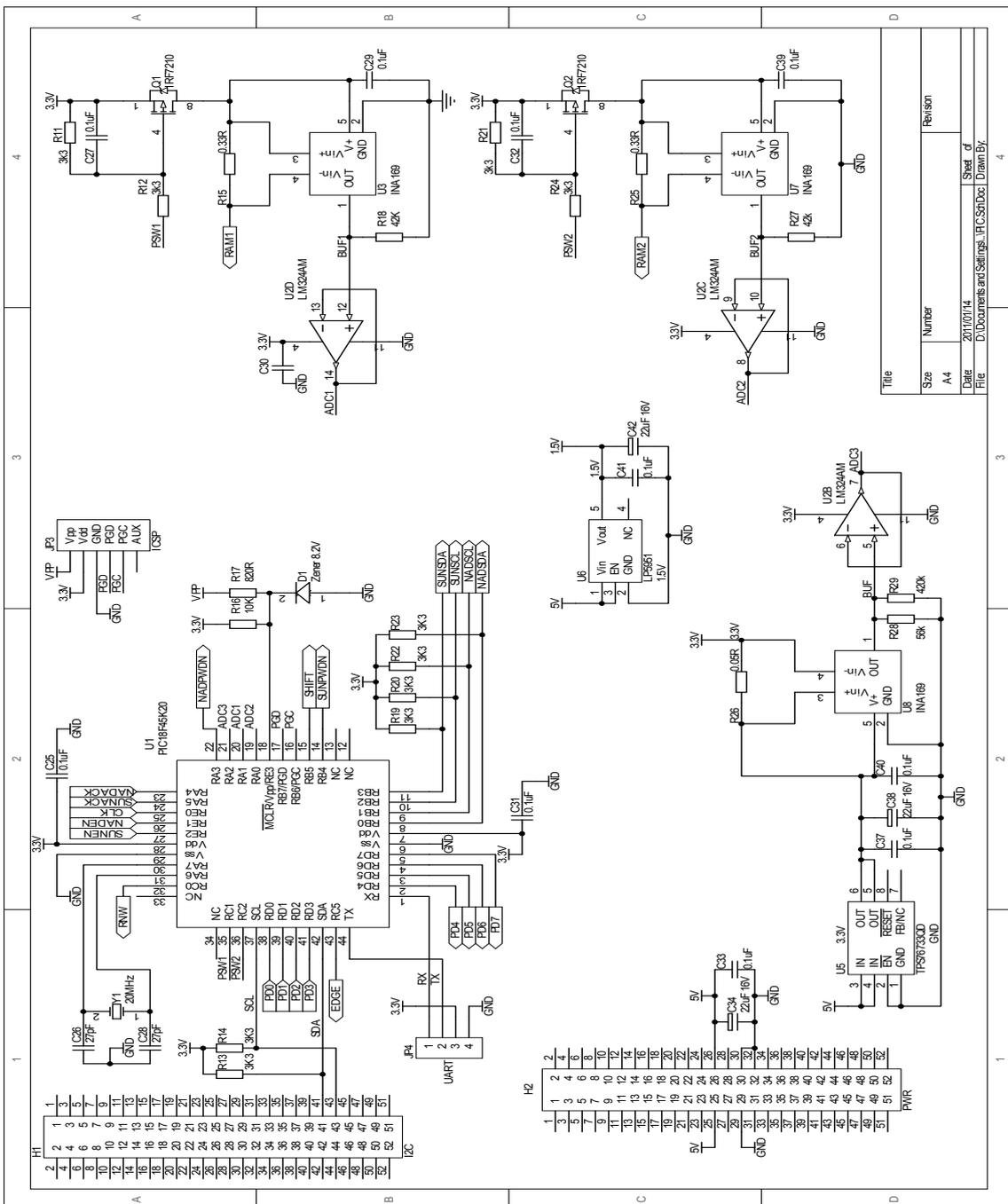
1. AGLN030 is available in the Z feature grade only and offers package compatibility with the lower density nano devices. Refer to "IGLOO nano Ordering Information" on page III.
2. AGLN030 and smaller devices do not support this feature.
3. AGLN060, AGLN125, and AGLN250 in the CS81 package do not support PLLs.
4. For higher densities and support of additional features, refer to the IGLOO and IGLOOe handbooks.

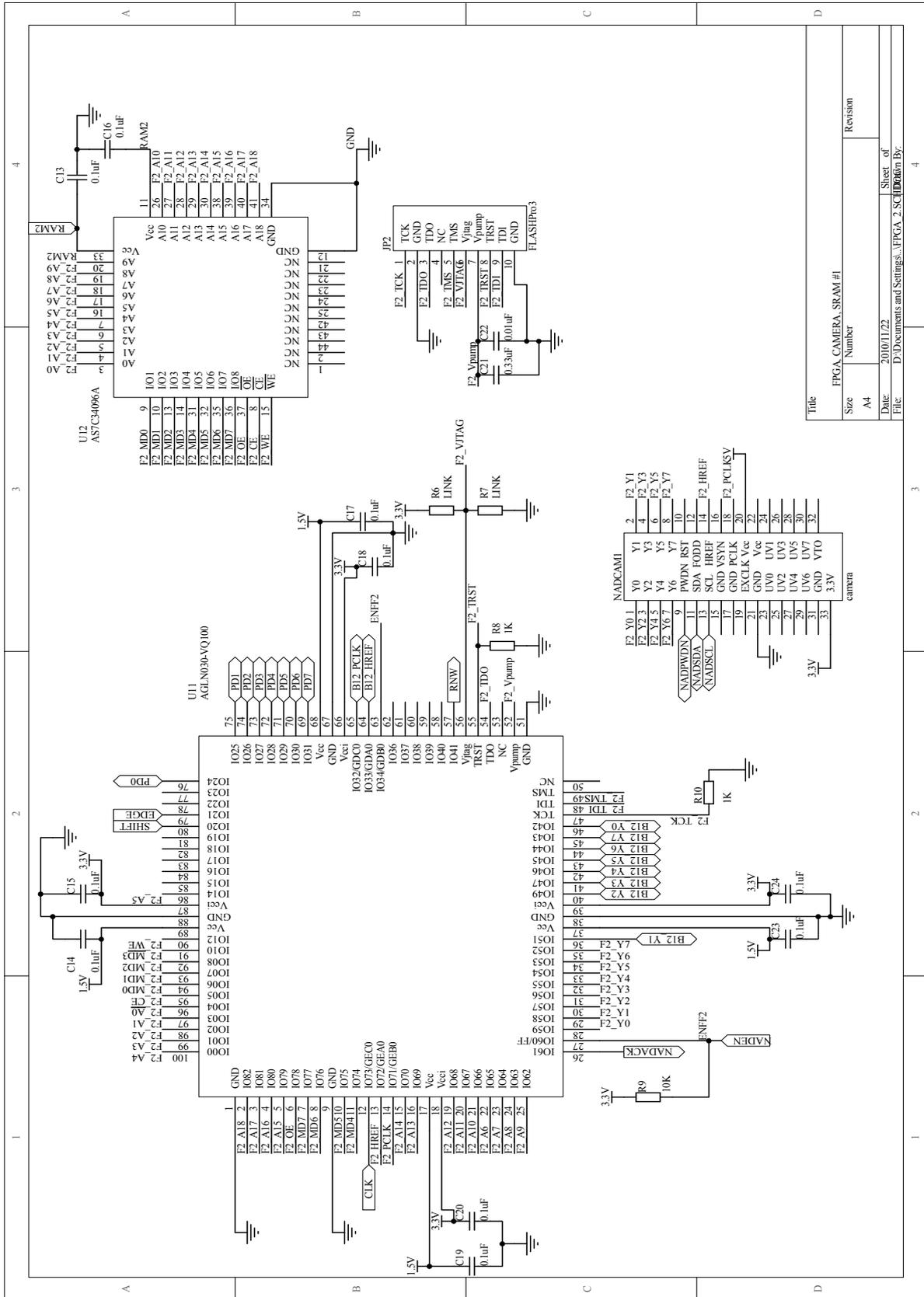
† AGLN030 and smaller devices do not support this feature.

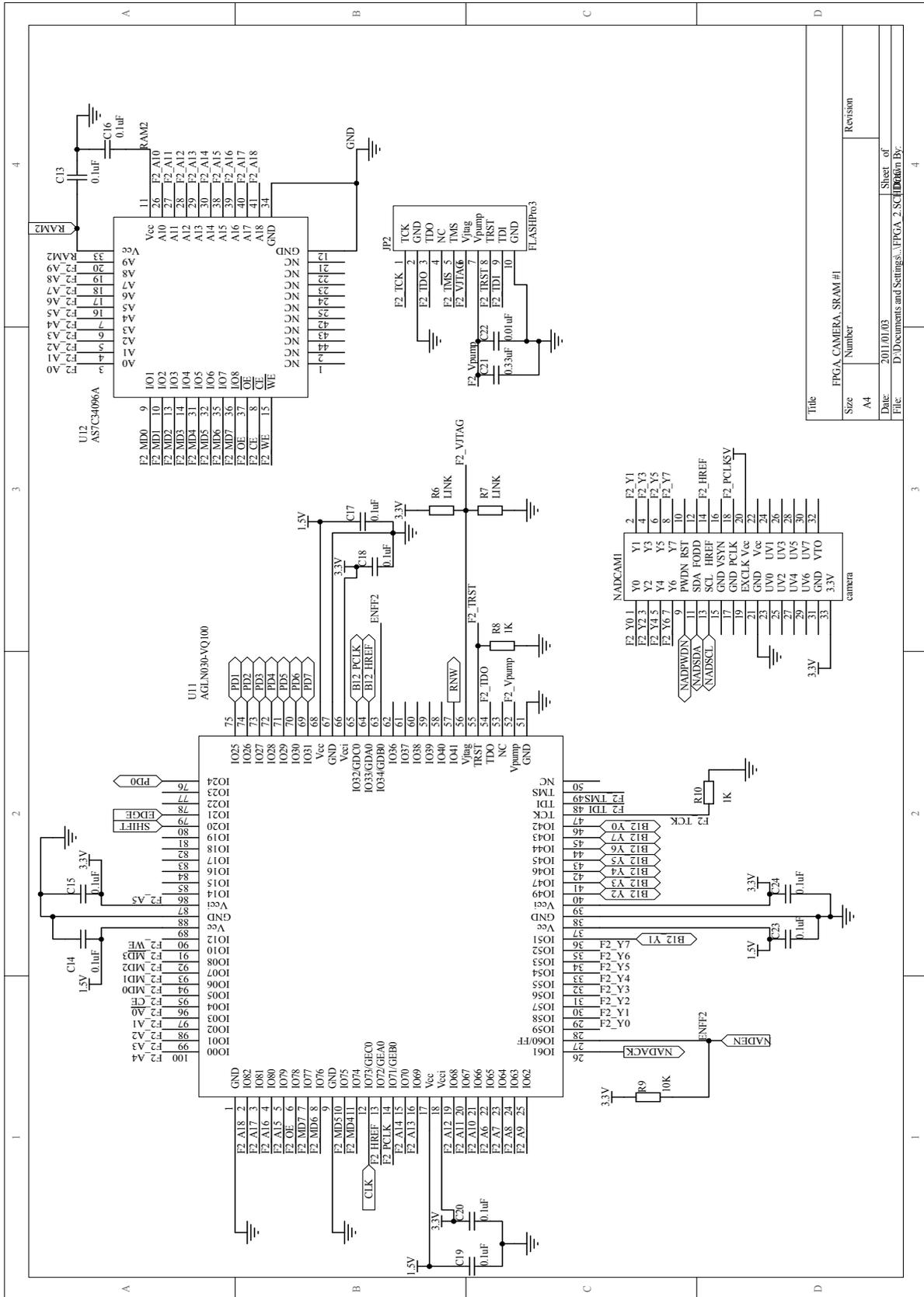
# Appendix B

## Hardware Layout

### B.1 Schematics







## Appendix C

# Source code for microcontroller

### C.1 Software I2C

```
1 // NSDA_D - I2C data pin
// NSCL_D - I2C clock pin
// NSDA - GPIO tristate for data pin
// NSCL - GPIO tristate for clock pin
void i2c_dly() {} // delay
6
void i2c_start_Nadir() { // Start condition
    NSDA_D = 1; i2c_dly();
    NSCL_D = 1; i2c_dly();
    NSDA_D = 0; i2c_dly();
11    NSCL_D = 0; i2c_dly();
}

void i2c_stop_Nadir() { // Stop condition
16    NSCL_D = 0; i2c_dly();
    NSDA_D = 0; i2c_dly();
    NSCL_D = 1; i2c_dly();
    NSDA_D = 1; i2c_dly();
}

21 // Parameters: d - Data to be transmitted over I2C bus
// Returns: b - Acknowledge
char i2c_tx_Nadir(unsigned char d) {
    char x;
    static char b;
26    for (x=0;x<8;x++) {
        if(d&0x80)
            NSDA =1;
        else
            NSDA = 0;
31        i2c_dly();
        NSCL = 1;
        i2c_dly();
        d <<= 1;
        NSCL = 0;
36    }
    NSDA = 1; NSCL = 1; i2c_dly();
    b = NSDA_D; // Acknowledge
    NSCL = 0; NSDA = 0;
    return b;
41 }
```

## C.2 Interpolation

```

1 // Linear Interpolation
// Parameters: xd - Distorted x coordinate
//             yd - Distorted y coordinate
4 //             dcc[] - Distortion Corrected Coordinates
void interpolation(double xd, double yd, int dcc[]) {
    double rd = 0; // distorted radius
    double ru = 0; // undistorted radius
    int rd0 = 0; // floor(rd)
9
    rd = sqrt(xd*xd + yd*yd);
    rd0 = (int)(rd);
    ru = (double)(rd0) // ru = ru0 + (re1 - re0)(rd - rd0)
        + (double)(distortCorrectLookup[rd0])
14        + ((double)(distortCorrectLookup[rd0+1])
        - (double)(distortCorrectLookup[rd0]))
        * (rd - (double)(rd0));

    dcc[0] = (int)(ru*xd/rd*100);
19    dcc[1] = (int)(ru*yd/rd*100); // Undistorted y coordinate
}

```

## C.3 Edge detection for Nadir sensor

```

1 // Edge detection for Nadir sensor
// Parameters: edgeXPos[] - x coordinates of all edge pixels
//             edgeYPos[] - y coordinates of all edge pixels
// Returns:    edgePixels - Total edge pixels
5 unsigned char edgeDetect (int edgeXPos[], int edgeYPos[]) {
    unsigned char search[64] = {0}; // Sample pixels
    unsigned char prev_test = 0; // previous searched pixels value
    char edgePixels = 0; // Total edge pixels
    int y = 0; // y coordinate
10    int x = 0; // x coordinate
    int k = 0;
    double radius = 0; // distorted radius

    // Edge mode
15    RNW = 1;
    SHIFT = 0;
    EDGE = 1;
    NADEN = 0;

    // DEFAULT STATE
20    CLK = 1;
    CLK = 0;

    // STATE A: Horizontal search
25    for (y = 1; y <= 8; y++) {
        for (x = 1; x <= 10; x++) {
            for (k = 0; k < 64; k++) {
                CLK = 1;
                search[k] = DATA;
30                CLK = 0;
            }
        }
    }
}

```

```

    if (search[63] >= threshold) {
        if (prev_test < threshold) {
35             edgeXPos[edgePixels] = (x<<6) - 63 + binary_search(search,63,0,6);
                edgeYPos[edgePixels] = (y<<6) + 15 - 64;
                radius = sqrt(((double)(edgeXPos[edgePixels]-367)
                    * (double)(edgeXPos[edgePixels]-367))
                    + ((double)(edgeYPos[edgePixels]-237)
                    * (double)(edgeYPos[edgePixels]-237)));
40             if (radius < 226) {
                    edgePixels++;
                }
            }
65         }
        else {
            if (prev_test >= threshold) {
                edgeXPos[edgePixels] = (x<<6) - 63 + binary_search(search,0,63,6);
                edgeYPos[edgePixels] = (y<<6) + 15 - 64;
50             radius = sqrt(((double)(edgeXPos[edgePixels]-367)
                    * (double)(edgeXPos[edgePixels]-367))
                    + ((double)(edgeYPos[edgePixels]-237)
                    * (double)(edgeYPos[edgePixels]-237)));
                if (radius < 226) {
55                 edgePixels++;
                }
            }
        }
        prev_test = search[63];
60    }
    prev_test = 0;
}

// STATE B
65 CLK = 1;
    CLK = 0;

// STATE C: Vertical search
70 for (x = 1; x <= 10; x++) {
    for (y = 1; y <= 7; y++) {
        for (k = 0; k < 64; k++) {
            CLK = 1;
            search[k] = DATA;
            CLK = 0;
75        }

        if (search[63] >= threshold) {
            if (prev_test < threshold) {
                edgeXPos[edgePixels] = (x<<6) - 1;
                edgeYPos[edgePixels] = (y<<6) - 63 + (binary_search(search,63,0,6));
80             radius = sqrt(((double)(edgeXPos[edgePixels]-367)
                    * (double)(edgeXPos[edgePixels]-367))
                    + ((double)(edgeYPos[edgePixels]-237)
                    * (double)(edgeYPos[edgePixels]-237)));
                if (radius < 226) {
85                 edgePixels++;
                }
            }
        }
    }
90    else {
        if (prev_test >= threshold) {

```

```

    edgeXPos[edgePixels] = (x<<6) - 1;
    edgeYPos[edgePixels] = (y<<6) - 63 + binary_search(search,0,63,6);
    radius = sqrt(((double)(edgeXPos[edgePixels]-367)
    95         * (double)(edgeXPos[edgePixels]-367))
        + ((double)(edgeYPos[edgePixels]-237)
        * (double)(edgeYPos[edgePixels]-237)));
    if (radius < 226) {
        edgePixels++;
    100    }
    }
    prev_test = search[63];
}
105
// Still STATE C
for (k = 0; k < 32; k++) {
    CLK = 1;
    search[k] = DATA;
    CLK = 0;
110 }

if (search[31] >= threshold) {
    if (prev_test < threshold) {
115         edgeXPos[edgePixels] = (x<<6) - 1;
        edgeYPos[edgePixels] = (y<<5) - 31 + binary_search(search,31,0,5);
        radius = sqrt(((double)(edgeXPos[edgePixels]-367)
            * (double)(edgeXPos[edgePixels]-367))
            + ((double)(edgeYPos[edgePixels]-237)
            * (double)(edgeYPos[edgePixels]-237)));
120         if (radius < 226) {
            edgePixels++;
        }
    }
}
125
else {
    if (prev_test >= threshold) {
        edgeXPos[edgePixels] = (x<<6) - 1;
        edgeYPos[edgePixels] = (y<<5) - 31 + binary_search(search,0,31,5);
130         radius = sqrt(((double)(edgeXPos[edgePixels]-367)
            * (double)(edgeXPos[edgePixels]-367))
            + ((double)(edgeYPos[edgePixels]-237)
            * (double)(edgeYPos[edgePixels]-237)));
        if (radius < 226) {
135             edgePixels++;
        }
    }
}
prev_test = 0;
140 }

// Active Standby
NADEN = 1;
SHIFT = 0;
145 EDGE = 0;

return edgePixels;
}

```

## C.4 Binary search

```

1 // Binary search
2 // Parameters: search[] - Pixels for binary search
//           light - Position of pixel above threshold
//           dark - Position of pixel below threshold
//           runtime - how many times loop is repeated
// Returns: light_pixel - The true edge pixel
7 int binary_search (unsigned char search[],
//           unsigned char light,
//           unsigned char dark,
//           unsigned char runtime) {
//
12     char i = 0;
//     unsigned char temp = 0;
//     unsigned char light_pixel = light;
//     unsigned char dark_pixel = dark;
//
//     for (i = 0; i < runtime; i++) {
17         temp = ((light_pixel+dark_pixel)>>1);
//         if(search[temp] >= threshold)
//             light_pixel = temp;
//         else
//             dark_pixel = temp;
22     }
//
//     return light_pixel;
// }

```

## C.5 Centroid calculation for Nadir sensor

```

1 // Centroid calculation for Nadir sensor
// Parameters: edgePixels - Total edge pixels
//           edgeXPos[] - x coordinates of edge pixels
//           edgeYPos[] - y coordinates of edge pixels
5 //           centerPositionNadir[] - centroid coordinates
void calc_Nadir (unsigned char edgePixels,
//           int edgeXPos[],
//           int edgeYPos[],
//           int centerPositionNadir[]) {
10     unsigned char cur_edge = 0; // Current edge pixel
//     int ru[2] = {0}; // Distortion corrected coordinates
//     double dx = 0; // Distorted x coordinate
//     double dy = 0; // Distorted y coordinate
//     double xp = 0; // Distortion corrected x pixel coordinate
15 //     double yp = 0; // Distortion corrected y pixel coordinate
//     double term = 0;
//     double A[3][3] = {0}; // Matrix A
//     double invA[2][3] = {0}; // Inverse of Matrix A
//     double B[3] = {0}; // Matrix B
20 //     double detA = 0; // Determinant of Matrix A
//
//     // Filling matrices
//     for (cur_edge = 0; cur_edge < edgePixels; cur_edge++) {
25         dx = ((edgeXPos[cur_edge]) - 371);
//         dy = ((edgeYPos[cur_edge]) - 236);
//     }
// }

```

```

// Distortion Correction
interpolation(dx,dy,ru);
30 xp = ((double)(ru[0]))/100;
yp = ((double)(ru[1]))/100;

term = xp*xp + yp*yp;

35 A[0][0] = A[0][0] + 2*xp*xp;
A[0][1] = A[0][1] + 2*xp*yp;
A[0][2] = A[0][2] + xp;

A[1][0] = A[1][0] + 2*yp*yp;
40 A[1][1] = A[1][1] + 2*yp*yp;
A[1][2] = A[1][2] + yp;

A[2][0] = A[2][0] + 2*xp;
45 A[2][1] = A[2][1] + 2*yp;
A[2][2] = A[2][2] + 1;

B[0] = B[0] - term*xp;
B[1] = B[1] - term*yp;
50 B[2] = B[2] - term;
}

// Calculating determinant of Matrix A
detA = A[0][0]*(A[1][1]*A[2][2]-A[1][2]*A[2][1])
      -A[0][1]*(A[1][0]*A[2][2]-A[1][2]*A[2][0])
55      +A[0][2]*(A[1][0]*A[2][1]-A[1][1]*A[2][0]);

// Calculating the invers of Matrix A
invA[0][0] = (A[1][1]*A[2][2]-A[1][2]*A[2][1]);
invA[0][1] = -(A[0][1]*A[2][2]-A[0][2]*A[2][1]);
60 invA[0][2] = (A[0][1]*A[1][2]-A[0][2]*A[1][1]);

invA[1][0] = -(A[1][0]*A[2][2]-A[1][2]*A[2][0]);
invA[1][1] = (A[0][0]*A[2][2]-A[0][2]*A[2][0]);
invA[1][2] = -(A[0][0]*A[1][2]-A[0][2]*A[1][0]);
65

// Centroid coordinates calculations
centerPositionNadir[0] = ((int)((-(invA[0][0]*B[0]
      + invA[0][1]*B[1]
      + invA[0][2]*B[2])/detA*100)));
70

centerPositionNadir[1] = ((int)((-(invA[1][0]*B[0]
      + invA[1][1]*B[1]
      + invA[1][2]*B[2])/detA*100)));
}

```

## C.6 Centroid calculation for Sun sensor

```

1 // Centroid calculation for Sun sensor
// Parameters: centerPositionSun[] - Centroid coordinates for Sun sensor
void calc_Sun (int centerPositionSun[]) {
    unsigned int total_pixel_values = 0;
    unsigned int x = 0;
6    unsigned int y = 0;
    unsigned char pixelData = ' ';

```

```

11  double xd = 0;
    double yd = 0;
    int start_x = 0;
    int start_y = 0;

    // Edge mode
    EDGE = 1;
    SHIFT = 0;
16  RNW = 1;
    SUNEN = 0;

    // STATE A
    CLK = 1;
21  start_x = (int)(DATA);
    CLK = 0;

    // STATE B
    CLK = 1;
26  start_x = start_x + (((int)DATA)<<8);
    CLK = 0;

    // STATE C
    CLK = 1;
31  start_y = (int)(DATA);
    CLK = 0;

    // STATE D
    CLK = 1;
36  start_y = start_y + (((int)DATA)<<8);
    CLK = 0;

    // STATE E
    for (y = 0; y < 15; y++) {
41     for (x = 0; x < 15; x++) {
        CLK = 1;
        pixelData = DATA;
        CLK = 0;
        if (pixelData >= thresholdSun) {
46             xd = xd + (double)(x)*(double)(pixelData);
                yd = yd + (double)(y)*(double)(pixelData);
                total_pixel_values = total_pixel_values + (int)(pixelData);
        }
    }
51 }

    // Active Standby
    SUNEN = 1;
    EDGE = 0;
56  SHIFT = 0;
    RNW = 0;

    //Distorted centroid calculation
    xd = (double)(start_x - 369 + (xd)/(double)(total_pixel_values) - 5);
61  yd = (double)(start_y - 250 + (yd)/(double)(total_pixel_values) - 3);

    // Distortion correction
    interpolation(xd,yd,centerPositionSun);
}

```

## Appendix D

# Source code for FPGA

### D.1 Gray code

```
1  -- File: gray_1.vhd
   -- One bit block for the Gray counter gray_n.vhd
   -- 2/2000 IVOVI
   -- qout: One bit output of the counter
5  -- zout: 1, if all the less significant bits are zero

   LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;

10  PACKAGE pkggray_1 IS
       COMPONENT gray_1
           PORT( arst, clk, qin, zin : IN STD_LOGIC;
                qout : INOUT STD_LOGIC;
                zout : OUT STD_LOGIC);
15  END COMPONENT;
   END pkggray_1;

   LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;

20  ENTITY gray_1 IS
       PORT( arst, clk, qin, zin : IN STD_LOGIC;
            qout : INOUT STD_LOGIC;
            zout : OUT STD_LOGIC);
25  END gray_1;

   ARCHITECTURE archgray_1 OF gray_1 IS
   BEGIN
       PROCESS(arst, clk)
30         BEGIN
           IF arst='1' THEN
               qout <= '0';
           ELSIF clk'EVENT AND clk='1' THEN
35             qout <= qout XOR (qin AND zin);
           END IF;
       END PROCESS;
       zout <= zin AND NOT qin;
   END archgray_1;
```

```

1  -- File: gray_n.vhd
2  -- Gray counter with variable width (generic width)
  -- 2/2000 IV0VI

  LIBRARY IEEE;
  USE IEEE.STD_LOGIC_1164.ALL;

7  ENTITY gray_n IS GENERIC(width: INTEGER:=3);
    PORT( async_rst, clock : IN STD_LOGIC;
          q : INOUT STD_LOGIC_VECTOR(width DOWNT0 0));
  END gray_n;

12 ARCHITECTURE archgray_n OF gray_n IS
    COMPONENT gray_1 PORT( arst, clk, qin, zin : IN STD_LOGIC;
                          qout : INOUT STD_LOGIC;
                          zout : OUT STD_LOGIC);

17  END COMPONENT;
  -- inner interconnection of 1-bit sections
  SIGNAL z : STD_LOGIC_VECTOR(width DOWNT0 0);
  -- auxiliary signal for MSB
  SIGNAL qx : STD_LOGIC;

22 BEGIN
  -- less significant bits
  create_lsb: FOR i IN 1 TO width-1 GENERATE
    createbit: gray_1 PORT MAP( async_rst, clock,
                              q(i-1), z(i-1),
27  q(i), z(i));

  END GENERATE;
  -- most significant bit
  create_msb: gray_1 PORT MAP( async_rst, clock,
                              qx, z(width-1),
32  q(width), z(width));

  -- auxiliary signal for MSB
  qx <= q(width-1) OR q(width);
  -- parity bit generation
  PROCESS(async_rst, clock)
37  BEGIN
    IF async_rst='1' THEN
      q(0) <= '1';
    ELSIF clock'EVENT AND clock='1' THEN
      q(0) <= NOT q(0);
42  END IF;
  END PROCESS;
  z(0) <= '1';
  END archgray_n;

```

## D.2 Memory read and write

### D.2.1 Write

```

1  architecture behavior of image2ram is
    signal adr_row_pix : std_logic_vector ( 8 downto 0) := "000000000";
    signal adr_col_pix : std_logic_vector ( 9 downto 0) := "0000000000";
begin
5  process (href,rnw,enable,clk_pix)
    begin
        -- Default starting point in memory
        if enable = '1' then
            adr_row_pix <= "000000000";
            adr_col_pix <= "0000000000";
10         elsif rising_edge(clk_pix) and href = '1' and enable = '0' and rnw = '0' then
            adr_col_pix <= adr_col_pix + 1;           -- Address change for column addresses
            if adr_col_pix = 639 then                -- One horizontal line is done
                adr_col_pix <= "0000000000";       -- Column address reset
15                 adr_row_pix <= adr_row_pix + 1;   -- Increment row address
                                                    -- for new horizontal line
            end if;
        end if;
    end if;
    -- Send address to memory
20    image_col <= adr_col_pix;
    image_row <= adr_row_pix;
end process;
end behavior;

```

### D.2.2 Read

```

1  -- Procedure same as for writing to memory, except for the control lines that are set
2  -- for reading from the memory
architecture behavior of image2pic is
    signal adr_row_pic : std_logic_vector ( 8 downto 0) := "000000000";
    signal adr_col_pic : std_logic_vector ( 9 downto 0) := "0000000000";
begin
7  process (rnw,enable,clk_pic,edge)
    begin
        if enable = '1' then
            adr_row_pic <= "000000000";
            adr_col_pic <= "0000000000";
12         elsif rising_edge(clk_pic) and enable = '0' and rnw = '1' and edge = '0' then
            adr_col_pic <= adr_col_pic + 1;
            if adr_col_pic = 639 then
                adr_col_pic <= "0000000000";
                adr_row_pic <= adr_row_pic + 1;
17                 end if;
            end if;
            pic_col <= adr_col_pic;
            pic_row <= adr_row_pic;
        end process;
22 end behavior;

```

### D.3 Search algorithm for nadir sensor

```

1  architecture behavior of edge2pic is
   signal adr_edge_row : std_logic_vector ( 8 downto 0) := "000001111";
3  signal adr_edge_col : std_logic_vector ( 9 downto 0) := "0000000001";

   type state_type is (A,B,C,D,E);
   signal current_state : state_type := A;

8  begin
   process (rnw,enable,clk_pic,edge,shift)
   begin
       -- default state
       if enable = '1' then
13      adr_edge_row <= "000001111";
          adr_edge_col <= "0000000001";
          current_state <= A;
       elsif (enable = '0') and (rnw = '1') and (edge = '1') and (shift = '0') then
18      case current_state is
          -- STATE A: Read image horizontally
          when A =>
               if (rising_edge(clk_pic)) then
                   adr_edge_col <= adr_edge_col + 1;
                   if (adr_edge_col = 639) then
23      adr_edge_col <= "0000000000";
                       adr_edge_row <= adr_edge_row + 64;
                   end if;
                   pic_col <= adr_edge_col;
                   pic_row <= adr_edge_row;
28      if (adr_edge_row = 463) and (adr_edge_col = 639) then
                       current_state <= B;
                   end if;
               end if;
          -- STATE B: Reset memory address for vertical read
33      when B =>
               if (rising_edge(clk_pic)) then
                   current_state <= C;
               end if;
               adr_edge_row <= "0000000000";
38      adr_edge_col <= "0000111111";
          -- STATE C: Read image vertically
          when C =>
               if (rising_edge(clk_pic)) then
                   adr_edge_row <= adr_edge_row + 1;
43      if (adr_edge_row = 479) then
                       adr_edge_row <= "0000000000";
                       adr_edge_col <= adr_edge_col + 64;
                   end if;
                   pic_col <= adr_edge_col;
48      pic_row <= adr_edge_row;
               end if;
          when others =>
               end case;
           end if;
53      end process;
   end behavior;

```

**D.4 First Sun pixel and area search for sun sensor**

```

1  architecture behavior of memory_edge is

    signal adr_row_pix : std_logic_vector ( 8 downto 0) := "000000000";
    signal adr_col_pix : std_logic_vector ( 9 downto 0) := "0000000000";

6  signal first_pix_y : std_logic_vector ( 8 downto 0) := "000000000";
    signal first_pix_x : std_logic_vector ( 9 downto 0) := "0000000000";

    type state_type is (A,B,C,D,E,F,G);
11  signal current_state : state_type := A;

    begin

        read : process (clk_pix,enable,rnw,edge,shift)

16         variable count : std_logic_vector (1 downto 0) := "00";
            begin
                if (enable = '1') then
                    adr_col_pix <= "0000000000";
                    adr_row_pix <= "0000000000";
21         count := "00";
                    -- Stores image in memory
                    elsif rising_edge(clk_pix) and href = '1' and enable = '0' and rnw = '0' then
                        adr_col_pix <= adr_col_pix + 1;
                        if adr_col_pix = 639 then
26         adr_col_pix <= "0000000000";
                            adr_row_pix <= adr_row_pix + 1;
                        end if;

                            -- Find First Sun Pixel
31         if (pix_data >= 200) and count = 0 then
                                first_pix_x <= adr_col_pix;
                                first_pix_y <= adr_row_pix;
                                count := count + 1;
                            elsif (pix_data >= 200) and count = 1 then
36         count := count + 1;
                            elsif (pix_data >= 200) and count = 2 then
                                count := count + 1;
                            elsif (pix_data < 200) and count < 3 then
41         count := "00";
                                end if;
                            end if;
                        end process read;

                        -- Area search
46  write: process (clk_pic,enable,rnw,edge,shift)
                            variable out_row_adr : std_logic_vector ( 8 downto 0) := "000000000";
                            variable out_col_adr : std_logic_vector ( 9 downto 0) := "0000000000";
                            begin
                                if enable = '1' then
31         current_state <= G;
                                    out_col_adr := first_pix_x -5 ;
                                    out_row_adr := first_pix_y -3 ;
                                elsif enable = '0' and edge = '1' and shift = '0' and rnw = '1' then
                                    case current_state is
56         -- STATE A: LSB first Sun pixel x coordinate

```

```

        when A =>
            if rising_edge(clk_pic) then
                current_state <= B;
            end if;
61         edge_data <= first_pix_x(7 downto 0);
        -- STATE B: MSB first Sun pixel x coordinate
        when B =>
            if rising_edge(clk_pic) then
                current_state <= C;
66         end if;
            edge_data <= "000000"&first_pix_x(9 downto 8);
        -- STATE C: LSB first Sun pixel y coordinate
        when C =>
            if rising_edge(clk_pic) then
                current_state <= D;
71         end if;
            edge_data <= first_pix_y(7 downto 0);
        -- STATE D: MSB first Sun pixel y coordinate
        when D =>
            if rising_edge(clk_pic) then
                current_state <= E;
76         end if;
            edge_data <= "0000000"&first_pix_y(8);
        -- STATE E: Area search
81         when E =>
            if rising_edge(clk_pic) then
                out_col_adr := out_col_adr + 1;
                if out_col_adr = (first_pix_x+10) then
                    out_col_adr := first_pix_x-5;
                    out_row_adr := out_row_adr + 1;
86                 end if;
                image_col <= out_col_adr;
                image_row <= out_row_adr;
            end if;
91         edge_data <= mem_dat;
        when others =>
            edge_data <= first_pix_x(7 downto 0);
            if rising_edge(clk_pic) then
                current_state <= B;
96         end if;
    end case;
end if;
end process write;
end behavior;

```

## Appendix E

# Python source code

### E.1 Source code to download image from microcontroller

```
1 import serial
import Image
import ImageFile
import ImageFilter
5
try:
    ser = serial.Serial(           # Open COM port
        port=3,                   # number of device, numbering starts at
                                # zero. if everything fails, the user
10                                # can specify a device string, note
                                # that this isn't portable anymore
                                # if no port is specified an unconfigured
                                # an closed serial port object is created
        baudrate=115200,          # baud rate
15        bytesize=8,            # number of databits
        parity='N',              # enable parity checking
        stopbits=serial.STOPBITS_ONE, # number of stopbits
        timeout=None,            # set a timeout value, None for waiting forever
20        xonxoff=0,              # enable software flow control
        rtscts=0,                # enable RTS/CTS flow control
        interCharTimeout=None    # Inter-character timeout, None to disable
    )
    value = 1;
except Exception:
25     value = 0;
    print 'Error'

if value == 1:
    x = ' ';
30     ser.write('a'); # Control to take image with Nadir sensor. 'q' for Sun sensor
    while x != 'Y': # Wait for acknowledge
        x = ser.read();

    if x == 'Y':
35         ser.write('s') # Contro to download image from Nadir sensor. 'w' for Sun sensor
        im = Image.fromstring("L", (640,480), ser.read(640*480)); # Downloading image
        im.save("NadirImage.bmp", "BMP"); # Save image on computer
        im.show(); # Display image
ser.close(); # Close COM Port
```

# Bibliography

- [1] T. Irvine. (2010, December 15). *Sputnik* [Online]. Available: <http://www.vibrationdata.com/Sputnik.htm>
- [2] NASA. (2010, December 15). *Sputnik 1 - NSSDC ID: 1957-001B* [Online]. Available: <http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=1957-001B>
- [3] Wikipedia. (2010, December 15). *Satellite Classification* [Online]. Available: [http://centaur.sstl.co.uk/SSHP/sshp\\_classify.html](http://centaur.sstl.co.uk/SSHP/sshp_classify.html)
- [4] S. Lee and A. Hutputanasin and A. Toorian and W. Lan and R. Munakata. (2010, December 16). *CubeSat Design Specification* [pdf]. Available: [http://www.cubesat.org/images/developers/cds\\_rev12.pdf](http://www.cubesat.org/images/developers/cds_rev12.pdf)
- [5] Wikipedia. (2010, April). *CubeSat* [Online]. Available: <http://en.wikipedia.org/wiki/CubeSat>
- [6] K. Sarda and S. Eagleson and E. Caillibot and C. Grant and D. Kekez and F. Pranajaya and R.E. Zee. (2006, April). *Canadian advanced nanospace experiment 2: Scientific and technological innovation on a three-kilogram satellite* [pdf]. Available: <http://www.utias-sfl.net/docs/canx2-acta-2006.pdf>
- [7] Wikipedia. (2010, December 16). *List of CubeSats* [Online]. Available: [http://en.wikipedia.org/wiki/List\\_of\\_CubeSats](http://en.wikipedia.org/wiki/List_of_CubeSats)
- [8] EPFL. (2010, December 16). *SwissCube* [Online]. Available: <http://swisscube.epfl.ch/>
- [9] G. Bozovic and O. Scaglione and C. Koechile and M. Noca and Y. Perriard. *SwissCube: development of an ultra-light and efficient Inertia Wheel for the attitude control and stabilization of CubeSat class satellites* [pdf].
- [10] H.J. Kramer. (2010, December 17). *DTUSat-2 (Danish Technical University Satellite-2)* [Online]. Available: <http://events.eoportal.org/presentations/10001991/10001992.html>
- [11] J.B. Bjarnø and R.W. Fléron. (2008, August). *DTUSAT-2: THE NEXT GENERATION ANIMAL MIGRATION RESEARCH PLATFORM*[pdf]
- [12] A.F.J. Moffat and W.W. Weiss and S.M. Rucinski and R.E. Zee and M.H. Kerkwijk and S.W. Mochnacki and J.M. Matthews and J.R. Percy and P. Ceravolo and C.C. Grant. (2010, December 17). *The Canadian BRITE NanoSatellite Mission* [pdf]. Available: <http://www.utias-sfl.net/docs/brite-astro-2006.pdf>
- [13] S. Eagleson and S. Mauthe and K. Sarda and H. Spencer and R.E. Zee. (2010, December 17). *The MOMENT Magnetic Mapping Mission Martian Science on a Nanosatellite Platform* [pdf]. Available: <http://www.utias-sfl.net/docs/moment-ssc-2007.pdf>
- [14] University of Louisiana. (2010, December 17). *Cajun Advanced Picosatellite Experiment* [Online]. Available: <http://cape.louisiana.edu/about/mission.html>
- [15] Wikipedia. (2010, December 17). *CAPE-1* [Online]. Available: <http://en.wikipedia.org/wiki/CAPE-1>
- [16] D. Litwiller. (2010, Novmeber 1). *CCD vs. CMOS: Facts and Fiction* [pdf]. Available: [http://www.dalsa.com/public/corp/Photonics\\_Spectra\\_CCDvsCMOS\\_Litwiller.pdf](http://www.dalsa.com/public/corp/Photonics_Spectra_CCDvsCMOS_Litwiller.pdf)

- [17] Wikipedia. (2010, November 2). *Lens (optics)* [Online]. Available: [http://en.wikipedia.org/wiki/Lens\\_\(optics\)](http://en.wikipedia.org/wiki/Lens_(optics))
- [18] Wikipedia. (2010, September 25). *Fisheye lens* [Online]. Available: [http://en.wikipedia.org/wiki/Fisheye\\_lens](http://en.wikipedia.org/wiki/Fisheye_lens)
- [19] Wikipedia. (2010, December 14). *F-number* [Online]. Available: <http://en.wikipedia.org/wiki/F-number>
- [20] Wikipedia. (2010, October 19). *Neutral density filter* [Online]. Available: [http://en.wikipedia.org/wiki/Neutral\\_density\\_filter](http://en.wikipedia.org/wiki/Neutral_density_filter)
- [21] Edmund Optics. (2010, October 19). *Kodak Wratten Neutral Density (ND) Filters* [Online]. Available: <http://www.edmundoptics.com/onlinecatalog/displayproduct.cfm?productID=2928>
- [22] M. O'Bryan. (2010, November 30). *Radiation Effects & Analysis* [Online]. Available: <http://radhome.gsfc.nasa.gov/radhome/see.htm>
- [23] Single Event Effects Symposium. (2010, November 30). *Summary of Single Event Effects* [Online]. Available: <http://radhome.gsfc.nasa.gov/radhome/see.htm>
- [24] Omnivision. (2000, May). *OV7620 SINGLE-CHIP CMOS VGA COLOR DIGITAL CAMERA* [pdf]. Available: <http://mxhaard.free.fr/spca50x/Doc/Omnivision/OV7620.pdf>
- [25] M. Barr. (2001, May). *Embedded Systems Memory Types* [Online]. Available: <http://www.netrino.com/Embedded-Systems/How-To/Memory-Types-RAM-ROM-Flash>
- [26] R. Koga. (2010, November 15). *EEPROMs for Space Applications* [pdf]. Available: [http://klabs.org/richcontent/MAPLDCon00/Abstracts/koga\\_a.pdf](http://klabs.org/richcontent/MAPLDCon00/Abstracts/koga_a.pdf)
- [27] Wikipedia. (2010, October 14). *Thresholding (image processing)* [Online]. Available: [http://en.wikipedia.org/wiki/Thresholding\\_\(image\\_processing\)](http://en.wikipedia.org/wiki/Thresholding_(image_processing))
- [28] R.C. Gonzalez and R.E. Woods, *Digital Image Processing*. Pearson Education, 2008
- [29] Wikipedia. (2010, August). <http://en.wikipedia.org/wiki/Luminance> [Online]. Available: <http://en.wikipedia.org/wiki/Luminance>
- [30] H.M. van Rensburg, "An Infrared Earth Horizon Sensor for a LEO Satellite," M.S. thesis, Dept. E&E. Eng., Stellenbosch University, Stellenbosch, Western Cape, 2008.
- [31] K.J. Friedrich, Nadir Sensor for a CubeSat, 2009.
- [32] F. Devernay and O. Faugeras. (2010, October). *Straight lines have to be straight* [pdf]. Available: <http://www.springerlink.com/content/m9cx2b2au3eyj8gp/fulltext.pdf>
- [33] A. Basu and S. Licardie. (2010, October). *Modeling Fish-Eye Lenses* [pdf]. Available: [http://ieeexplore.ieee.org/xpls/abs\\_all.jsp?arnumber=583883&tag=1](http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=583883&tag=1)
- [34] M. Belezan and D. Mortari and R. Perfetti, "Moon Image Processing for Spacecraft Attitude Estimation", 1997.
- [35] P. Bourke. (2010, October). *Equation of a circle from 3 points (2 dimensions)* [Online]. Available: <http://local.wasp.uwa.edu.au/~pbourke/geometry/circlefrom3/>
- [36] (2010, October). *Least Squares Circle* [Online]. Available: [http://www.infogoaround.org/JBook/LSQ\\_Circle.html](http://www.infogoaround.org/JBook/LSQ_Circle.html)
- [37] Wikipedia. (2010, October). *Focal length* [Online]. Available: [http://en.wikipedia.org/wiki/Focal\\_length](http://en.wikipedia.org/wiki/Focal_length)
- [38] Wikipedia. (2010, November). *Exposure (photography)* [Online]. Available: [http://en.wikipedia.org/wiki/Exposure\\_\(photography\)](http://en.wikipedia.org/wiki/Exposure_(photography))
- [39] Wikipedia. (2010, December 27). *Cubic function* [Online]. Available: [http://en.wikipedia.org/wiki/Cubic\\_function](http://en.wikipedia.org/wiki/Cubic_function)

- [40] J.R. Wertz and W.J. Larson, *Space Mission Analysis and Design*, *Space Technology Library*, Microcosm Press & Kluwer Academic Publishers, 1999.
- [41] B. Dunbar. (2007, November). *Sun* [Online].  
Available: [http://www.nasa.gov/worldbook/sun\\_worldbook.html](http://www.nasa.gov/worldbook/sun_worldbook.html)
- [42] *Calculus*, 5th ed., Brooks/Cole, 2003
- [43] MICROCHIP. (2007, August). *MPLAB® C18 C COMPILER USER'S GUIDE* [pdf].  
Available: [http://ww1.microchip.com/downloads/en/DeviceDoc/C18\\_User\\_Guide\\_51288j.pdf](http://ww1.microchip.com/downloads/en/DeviceDoc/C18_User_Guide_51288j.pdf)
- [44] Wikipedia. (2010, November). *Big O notation* [Online].  
Available: [http://en.wikipedia.org/wiki/Big\\_O\\_notation#Family\\_of\\_Bachmann.E2.80.93Landau\\_notations](http://en.wikipedia.org/wiki/Big_O_notation#Family_of_Bachmann.E2.80.93Landau_notations)
- [45] Wikipedia. (2010, August). *Computational complexity of mathematical operations* [Online].  
Available: [http://en.wikipedia.org/wiki/Computational\\_complexity\\_of\\_mathematical\\_operations](http://en.wikipedia.org/wiki/Computational_complexity_of_mathematical_operations)
- [46] Wikipedia. (2010, December). *Linear interpolation* [Online].  
Available: [http://en.wikipedia.org/wiki/Linear\\_interpolation](http://en.wikipedia.org/wiki/Linear_interpolation)
- [47] ROBOT ELECTRONICS. (2010, November). *Using the I2C Bus* [Online].  
Available: [http://www.robot-electronics.co.uk/acatalog/I2C\\_Tutorial.html](http://www.robot-electronics.co.uk/acatalog/I2C_Tutorial.html)
- [48] Encoder Products Company. (2010, November). *Gray Codes, Natural Binary Codes, and Conversions* [pdf].  
Available: <http://www.encoder.com/techbulletins/TB-120.pdf>
- [49] I. Višcor. (2010, November). *Gray counter in VHDL* [pdf].  
Available: [http://www.isibrno.cz/ivovi/gray\\_counter.pdf](http://www.isibrno.cz/ivovi/gray_counter.pdf)
- [50] *Digital System Design with VHDL*, 2nd ed., Pearson Education Limited, 2004.
- [51] *Advanced Engineering Mathematics*, 2nd ed., Jones and Bartlett Publishers, 2000.
- [52] *Probability, Random Variables, and Random Signal Principles*, McGraw-Hill, 2001.