NTNU – Trondheim
Norwegian University of
Science and Technology

Signal Processing for Communicating Gravity Wave Images from the NTNU Test Satellite

Marianne Bakken

Master of Science in Electronics
Submission date: July 2012
Supervisor: Tor Audun Ramstad, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Problem Description

The payload defined for the NTNU Test Satellite to be launched in 2014 is an infrared camera for observation of atmospheric gravity waves. Pictures taken by the infrared camera should have sufficient resolution and quality, and cover appropriate areas to be able to derive interesting properties of the waves.

The task in this thesis is to consider the different issues related to signal processing for achieving good quality image rendition while being able to transmit as many pictures as possible taking the transmission channel capacity into consideration.

One problem to be considered is blur resulting from the satellite motion and the necessary long exposure times. Methods based on deblurring on one hand, and the combination of multiple pictures with motion compensation on the other hand, should be developed.

The image transfer rate depends on the image resolution and the number of bits per pixel used. The bit number can be substantially reduced by image/video compression. Appropriate compression techniques should be developed taking the image characteristics and quality requirements into consideration.

The algorithms developed should be of moderate complexity to fit into the available processing capability in the satellite. Simulations should be made to indicate the potential of the suggested methods.

As the satellite construction is highly multidisciplinary where many parts depend on each other, it is not expected that this work will result in final algorithms, but rather point to avenues for final algorithm design.

Preface

This report is one of the eight master's theses that has been carried out as a part of the NTNU Test Satellite (NUTS) project during the spring 2012. The NUTS project is a student project aiming to design, develop, test, launch and operate a double CubeSat by 2014. It is highly multidisciplinary, with final year students from six departments at NTNU contributing in all stages.

To be a part of the NUTS project has been challenging, but also very interesting and rewarding. In addition to the work regarding my thesis, a lot of time has been spent on preparing and holding presentations, recruiting new students, making flyers and information for the webpage, all in order to spread the word about the project. I have also had the opportunity to participate in interesting workshops and conferences, both in Norway and abroad.

Designing a payload for a satellite is a multidisciplinary task, and this project evolved to cover a much wider field than first intended. This report is not only a master's thesis, it also serves as a documentation for the work that has been done regarding the NUTS payload, and it has therefore grown to become quite extensive.

Quite a few people have helped me through this project. First of all, I would like to thank my supervisor Tor Ramstad, for meeting me on a regular basis and giving helpful advice both regarding signal processing and report writing. Secondly, I should thank the members of the NUTS team for an amazing year with social gatherings, fruitful discussions, unforgettable trips to Brussels and Andøya and a lot of support. I should especially thank the project manager Roger Birkeland for providing such an interesting master's project. I would also like to thank Patrick Espy for sharing his knowledge about atmospheric physics and sensor technology, and for carrying out and analysing a sensor experiment for me. Lise Randeberg has also given me useful input regarding camera technology.

Finally, I would like to thank all the people that have read and given feedback on different parts of my thesis; Tor Ramstad, Roger Birkeland, Snorre Rønning, Mehmet Altan and Patrick Espy, and especially Sigvald Marholm and Irene Bakken for having the patience to read the complete thesis and giving me very valuable feedback and support the last week.

Abstract

The NTNU Test Satellite (NUTS) is planned to have a payload for observation of atmospheric gravity waves. The gravity waves will be observed by means of an infrared camera imaging the perturbations in the OH airglow layer. So far, no suitable camera has been found that complies with the restrictions that follows when building a small satellite. Uncooled InGaAs has however been concluded to be the most suitable detector type in terms of wavelength response and weight.

InGaAs sensors are known to have a high dark current when not cooled, and processing must therefore be applied to remove the background offset and noise. The combination of the high speed of the satellite and the long exposure time that is required for the camera will create motion blur. Simulations with synthetic test images in MATLAB showed that the integration time should at least be kept under 1 second in order not to destroy the wave patterns. Longer integration times may however be required in order to get a sufficient SNR.

Two signal processing solutions to this problem was investigated: motion blur removal by deconvolution and image averaging with motion compensation. The former strategy is to apply a long exposure time to get a strong signal, and then remove the blur with deconvolution techniques using knowledge of the blur filter. Simulations applying the Lucy-Richardson (LR) algorithm showed that it was not able to remove strong blur, and was very sensitive to errors in the blur filter and noise in the image. The other approach is to obtain a sequence of images with short exposure time in order to avoid motion blur, and provide the necessary SNR by shifting the images according to the known motion and combine them into one image. This concept is simpler and more reliable than the deconvolution approach, and simulations showed that it is less sensitive to errors in the speed estimate than the deconvolution algorithm. It was concluded that this is the most suitable approach for the NUTS application, and it should be implemented on-board the satellite in order to provide a good SNR for the compression to function optimally.

The downlink datarate of NUTS is of only 9600 bit/s, and it has been estimated that 2.45 Mb of payload data can be downloaded on average per day. This corresponds to less than 5 uncompressed images of 256×256 pixels with 8 bit per pixel.

A sequence of overlapping combined images should be obtained to provide a scan of a desired area, and it was suggested that it should be encoded as video to enable efficient compression and transmission of as many images as possible to the ground station. A three-dimensional DPCM algorithm combined with a dead-zone quantizer and stack-run coding was implemented in MATLAB. Simulations demonstrated that this simple compression scheme can provide a bit rate of less than 1 bit/px for a sequence of gravity wave images. One of the quantizers that was tried gave 0.83 bits per pixel with reasonable quality. If this number can be achieved in practice, the image transfer rate would be increased to 45 images per day, which is a significant improvement.

Sammendrag (Abstract in Norwegian)

NTNU Test Satellitt (NUTS) er planlagt å ha en nyttelast for observasjon av atmosfæriske tyngdebølger. Tyngdebølgene vil bli observert ved hjelp av et infrarødt kamera som tar bilder av forstyrrelser i natthimmellyset (airglow). Så langt har det ikke blitt funnet noen kameraer som passer til restriksjonene som følger når man bygger en liten satellitt. Det har imidlertid blitt konkludert med at InGaAs uten kjøling vil være den mest passende sensortypen når det gjelder bølgelengderespons og vekt.

InGaAs sensorer har en høy mørkestrøm når de ikke er kjølt, og prosessering må derfor til for å fjerne bakgrunnssignalet. Kombinasjonen av den høye farten til satellitten og at kameraet krever lang eksponeringstid vil forårsake bevegelsesuskarphet i bildet. Simuleringer med syntetiske testbilder i MATLAB viste at integrasjonstiden må holdes godt under 1 sekund for å ikke ødelegge bølgemønstrene. Lengre integrasjonstid kan imidlertid være nødvendig for å få et tilstrekkelig signal-støy forhold.

To signalbehandlingsmetoder ble vurdert som mulige løsninger på dette problemet: fjerning av bevegelsesuskarphet ved hjelp av dekonvolusjon, og midling av bilder kombinert med bevegelseskompensasjon. Den første av de to strategiene går ut på å bruke en lang eksponeringstid for å få et godt signal, for deretter å fjerne uskarphetene med dekonvolusjonsteknikker. Simuleringer med Lucy-Richardson algoritmen viste at denne algoritmen ikke var i stand til å fjerne kraftige uskarpheter. Den var også veldig sensitiv for støy i bildet og feil i uskarphetsfilteret. Den andre strategien går ut på å ta en bildesekvens med kort eksponeringstid for å unngå bevegelsesuskarphet, og sørge for å få det nødvendige signal-støy forholdet ved å forskyve bildene i henhold til den kjente bevegelsen og kombinere dem til ett bilde. Dette er en enklere og mer pålitelig strategi enn dekonvolusjon, og simuleringer viste at den også er mindre sensitiv for feil i fartsestimatet. Det ble konludert med at dette er den mest passende strategien for denne applikasjonen, og at den skal implementeres ombord på satellitten for å gi tilstrekkelig signal-støy-forhold slik at

kompresjonsalgoritmen kan fungere skikkelig.

Nedlinken til satellitten har en datarate på bare 9600 bit/s, og det har blitt estimert at bare 2,45 Mb med data fra nyttelasten kan bli lastet ned per dag i gjennomsnitt. Dette tilsvarer mindre enn 5 ukomprimerte bilder med 256×256 piksler og 8 bit per piksel.

For å skanne et ønsket område kan man ta en sekvens med kombinerte bilder som overlapper. Det ble foreslått at denne sekvensen bør kodes som video for å gjøre effektiv kompresjon mulig og få overført så mange bilder som mulig til bakkestasjonen. En tredimensjonell DPCM algoritme kombinert med en kvantiserer med dødsone og stack-run koding ble implementert i MATLAB. Simuleringer demonstrerte at dette enkle kompresjonssystemet kan oppnå en bitrate på under 1 bit per piksel for en sekvens med bilder av tyngdebølger. En av kvantisererne som ble testet ut ga 0.83 bit per piksel, med akseptabel kvalitet. Hvis dette kan oppnås i praksis, vil antall bilder overført per dag øke til 45, som er en vesentlig forbedring.

Contents

List of Figures

List of Tables

List of Acronyms

ADCS Attitude Determination and Control System

A/D Analog-to-digital

ANSAT Norwegian Student Satellite Program

dB decibels

DPCM Differential Pulse Code Modulation

DSNR Detector Signal-to-Noise Ratio

EPS Electrical Power System

ESA European Space Agency

FMC Forward Motion Compensation

FOV Field of View

GOP Group of Pictures

GSD Ground Sample Distance

GW Gravity Waves

HAWI Hydroxyl Airglow Wave Imager

HDR High Dynamic Range imaging

InGaAs Indium Gallium Arsenide

ISS International Space Station

LEO Low Earth Orbit

LR Lucy-Richardson

MAD Mean Absolute Difference

MC Motion Compensation

MSE Mean Squared Error

NAROM Norwegian Centre for Space-related Education

NASA National Aeronautics and Space Administration

NEI Noise Equivalent Irradiance

NUTS NTNU Test Satellite

OBC On-Board Computer

OH Hydroxyl

P-POD Poly Picosatellite Orbital Deployer

PSF Point Spread Function

PSNR Peak-to-peak Signal to Noise Ratio

QE Quantum Efficiency

SAD Sum of Absolute Difference

Si Silicon

SNR Signal-to-Noise Ratio

SQNR Signal-to-quantization noise ratio

SR Stack-run

STK Analytical Graphics, Inc. Satellite Toolkit

SWIR Short-wave infrared

TDI Time-Delayed Integration

VNIR Visible and near-infrared

Chapter 1

Introduction

1.1 The NUTS Payload

The *NTNU Test Satellite* (NUTS) payload is planned to be an infrared camera observing atmospheric *gravity waves* in the *Hydroxyl* (OH) airglow layer in the mesosphere. Gravity waves, not to be confused with gravitational waves from General Relativity, are fluid-dynamical large-scale waves propagating vertically and horizontally through the Earth's atmosphere. They are mostly generated in the lower atmosphere by air blowing over mountains and other weather phenomena. The waves are believed to play a major role in the global north-south/south-north (meridional) atmospheric circulation, which is a vital component in global climate and weather models. Despite this, their properties are poorly understood, mainly due to a lack of observational data.

The gravity waves can for instance be observed as perturbation patterns in the airglow layers of the upper mesosphere. Ground-based observations have been made by taking pictures of the airglow at night, which have provided some knowledge about the properties of the waves. The NUTS payload camera is supposed to take pictures of the OH airglow to provide data from other locations than the ground based observations, and in this way contribute to a better understanding of the global properties of the waves. A similar payload was planned by NASA in the late 90's, but the mission was discontinued [1]. More than a decade later, observation of gravity waves by means of infrared camera has not yet been done from a satellite.

Many different aspects spanning several disciplines must be considered regarding the NUTS payload. First of all, an overview of the requirements for the camera must be established, according to the properties of the satellite and the phenomenon in question. Optical remote sensing from satellites is a well established field, and the same holds for observation of gravity waves from the ground. But combining the two technologies and fit it in a CubeSat is not a trivial task. When the requirements for the camera have been found, a camera with suitable detector, optics and readout electronics must be either purchased or built. How the images should be communicated to the ground station must also be considered. The downlink data rate is quite limited, and the satellite can only communicate with the ground station when it is visible for a short period of time a few times a day. If a suitable compression algorithm is applied, it will be possible to download more images, which is preferable due to the short life-time of the satellite. Depending on the expected quality of the output of the camera, it might be necessary to perform some simple image processing on-board the satellite to decrease the noise and enhance the quality.

1.2 Previous Work

A pre-study considering many of the aspects mentioned above was carried out by the author and Snorre Stavik Rønning during autumn 2011. The task of finding the requirements for the camera turned out to be more complex than first assumed, requiring knowledge of remote sensing, orbital mechanics, optics, infrared detector technology and atmospheric physics all together. Due to many unknown parameters, no definite specification for the camera was found, but a few basic requirements such as detector type and resolution was established. This work therefore had to be continued in spring 2012, together with the task of finding a suitable camera. For completeness, most of the work done during the pre-study is also described in this thesis.

One of the major topics of the pre-study was a study of the motion blur problem in images. MATLAB simulations showed that this could be a problem for the NUTS payload camera for long exposure times, which might be necessary to get the sufficient *Signal-to-Noise Ratio* (SNR) from the infrared detector. To mitigate this problem, it was investigated how motion blur can be removed by post-processing. An algorithm applying non-blind deconvolution to invert the effect of the blur filter was successfully applied to test images in MATLAB. This algorithm may however cause artefacts in practice if the speed is not known exactly, which can result in irreversible damage of the image if the algorithm is applied on-board the satellite. Another strategy is to perform the post-processing on ground, but this also have some issues, since the compression process may introduce errors crucial to the performance of the deconvolution algorithm.

Observations of gravity waves by means of cameras has been done from a few ground stations for instance in Antarctica [2] and on Hawaii [3]. These observations are also based on obtaining images of the airglow, but in the visible range of the electromagnetic spectrum.The wavelength of the gravity wave patterns have been found to be in the range of 15-40 km with a mean of 26 km, and their wave phase speeds to be around 25 m/s. But it is not possible to do global measurements from the ground, and therefore many large-scale gravity wave properties still remains a mystery.

More details about the NASA project and a similar CubeSat project are presented in Section 2.4.5.

1.3 The Aim of This Thesis

The aim of this thesis is to give an overview of the whole payload system from photons to bits, and to suggest suitable operation modes and algorithms for the different stages. Most of the work have been carried out from a system point-of-view, with preparation of images for transmission as the main focus. In order to choose a suitable compression algorithm, and make sure that the images are of sufficient quality to be further interpreted and processed on ground, many aspects have to be taken into account. First of all, the downlink capacity of the satellite is an important factor, which has to be discussed. Secondly, information about the camera and the gravity wave phenomenon is also necessary, to provide a guess of what the images will look like and what SNR that can be expected.

The design of the payload module is still in an early stage, and many assumptions have been made to be able to reach any conclusions at all. The focus of this thesis is to provide an overview of all the aspects that must be regarded when designing a signal processing system for such a payload, and suggest possible solutions rather than presenting a perfect and finished implementation.

1.4 Outline

First, Chapter 2 gives an introduction to the satellite, the gravity wave phenomenon and optical remote sensing, in order to provide the necessary background information and set the context for the following chapters.

Chapter 3 is the first of the three main chapters of this thesis, with focus on the infrared camera. A theoretical background on detector technology and noise will be given, as well as experimental results showing what kind of noise that can be expected in the images. A discussion of the camera parameters and possible camera candidates is also given, and the chapter is brought to a close with a discussion of what the images will look like.

Chapter 4 follows with focus on image enhancement. An introduction to the motion blur problem is given, as well as simulations and a discussion of its possible impact on the images. Two different strategies for motion blur removal by post processing are presented theoretically, simulated and compared. Strategies for removal of detector noise is also discussed.

A strategy for compression of the image sequences is presented in Chapter 5, based on assumptions of the quality and content of the images. A three-dimensional *Differential Pulse Code Modulation* (DPCM) system with motion compensation for compression of low-rate video is proposed. A simplified version of the complete compression algorithm is implemented in MATLAB to provide a demonstration.

A proposal for a complete signal processing system including both image enhancement and compression is then presented in Chapter 6.1, and a simple simulation is performed to serve as a proof of concept. Finally, the results are concluded in Chapter 6.2, and a suggestion for further work is presented.

Chapter 2

Background

This chapter is meant to provide the reader with the necessary background for the discussion in the following chapters. First, an overview of the NUTS satellite is provided in Section 2.1. The downlink capacity for the satellite is discussed in Section 2.2, and some simulations are made to provide a reasonable estimate of this. Then, an introduction of atmospheric gravity waves and how they can be observed is given in Section 2.3. Some terminology and concepts of optical remote sensing is then presented in Section 2.4, together with examples of related projects.

2.1 The NTNU Test Satellite (NUTS)

This section is meant to give an overview of several aspects of the NTNU Test Satellite, in order to provide essential information for the following chapters of the report, as well as setting the context for this thesis. For further reading, it is referred to [4] for a general overview of the mission. More detailed specifications may be found in [5], and the NUTS website[1] provides a publication list as well as updated information on the subsystems. An illustration of the satellite is shown in Figure 2.1.

2.1.1 The NUTS Project

As already introduced, NUTS is a small satellite that is being developed and built by students at NTNU. It all started in 2006, when a pre-study of a new satellite project at NTNU was done by three students [5] from the Department of Electronics and Telecommunications, resulting in a specification and a design proposal for the *Norwegian Student Satellite Program* (ANSAT) run by *Norwegian Centre for Space-related Education* (NAROM). The goal of the ANSAT is to launch three student satellites by the end of 2014, and involves the University of Oslo, Narvik

[1] `http://nuts.cubesat.no`

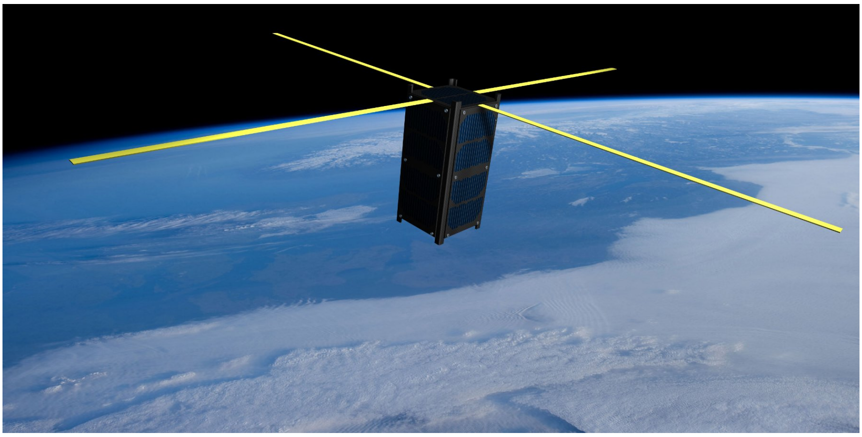


Figure 2.1: Illustration of the NTNU Test Satellite (NUTS) (Satellite image: Courtesy of Kai Inge Midtgård Rokstad. Background image: Astronaut photograph STS131-E-11693 obtained from the ISS, courtesy NASA JSC Image Science & Analysis Laboratory)

University College and NTNU. It is intended to stimulate cooperation between different educational institutions and the industry, and to give the students experience with team work and hands-on training. The NUTS project was officially started in September 2010, and after that, final year students from several departments have contributed.

One of the main challenges of such a long-term student project is administration. Most of the master students are only involved in the project for one year, before a new group of students takes over, which makes it hard to keep an overview. Fortunately, the project has a full-time employed manager. In order to involve students for a longer time period it was decided in spring 2012 to involve undergraduate students on a voluntary basis.

2.1.2 Small Satellites

NUTS will follow the CubeSat standard [6], which is a picosatellite standard developed to make it easier to launch small payloads into space. A single CubeSat is a 10 cm cube with a mass of up to 1.33 kg, and a double is thus $20 \times 10 \times 10$ cm with a mass of up to 2.66 kg. The CubeSat standard specifies requirements for design and testing, such that the satellite can be qualified for launch with a *Poly Picosatellite Orbital Deployer* (P-POD) [6], which provides a safe interface between the CubeSat and the main payload.

Small satellites offer a fast and affordable access to space; the development time and financial costs are usually just a small fraction of what can be expected for conventional missions carried out by *National Aeronautics and Space Administration* (NASA) or *European Space Agency* (ESA). Since the development of the CubeSat standard, it has become more and more common for universities to develop their own satellties, resulting in a diverse and innovative international community. Small satellites are especially useful for technology demonstrations, resulting in a wide range of payloads. Networks of picosatellites flying in formation have also been suggested. Such a network would achieve a larger range and gather information in a totally different way than one single satellite of the same total weight. The first planned satellite network of this kind, called QB50, will consist of 50 double and triple CubeSats and is a collaboration between several universities from all over the world [7].

There are several restrictions to take into account when building a small and inexpensive satellite. Commercial off-the-shelf electronic components are often used in contrary to expensive space qualified components developed for the space industry. The size and mass constraints limit the available area for solar panels and batteries, and thus also the available power for communication and operation of the payload.

2.1.3 Orbit and Launch

The orbit of the NUTS CubeSat will be a polar *Low Earth Orbit* (LEO). The altitude of a LEO is generally between 500 and 2000 km [8], but the NUTS orbit will most likely lie between 450 and 650 km in order to limit the orbital lifetime. Figure 2.2 shows an illustration of a polar orbit.

During launch, NUTS will comprise two of three CubeSat units in a P-POD that is carried as a piggyback in a launch of a commercial payload. Until a specific launch is scheduled, the exact orbit will remain unknown. For now it is assumed to be sun-synchronous and circular, between 350 and 650 km. It will then orbit the Earth approximately 14 times in 24 hours [9], passing near the north and south pole for each period. In this way, the whole globe is covered as it rotates inside the orbit.

2.1.4 The NUTS Subsystems

NUTS consists of several subsystems, as illustrated in Figure 2.3. The function of each of these subsystems will be influenced by the others, and the aspects most vital to the payload and the general function of the satellite is therefore presented in this section. An overview of the main function of each subsystem can be found in the document "The NUTS Subsystems", written by the master students participating in the project spring 2012. A draft version of this document is enclosed in Appendix A.

The structure of the satellite will be as shown in Figure 2.4(a). An outer frame of carbon fiber will be the main structure holding everything together. The different modules are connected by the backplane, which provides communication and power interfaces. The power will be supplied from a battery that is charged by means of solar panels mounted on the exterior walls of the satellite. The *Electrical Power System* (EPS) module makes sure that the batteries are charged efficiently and safely, and provides two regulated 3.3 V and two regulated 5.0 V power rails

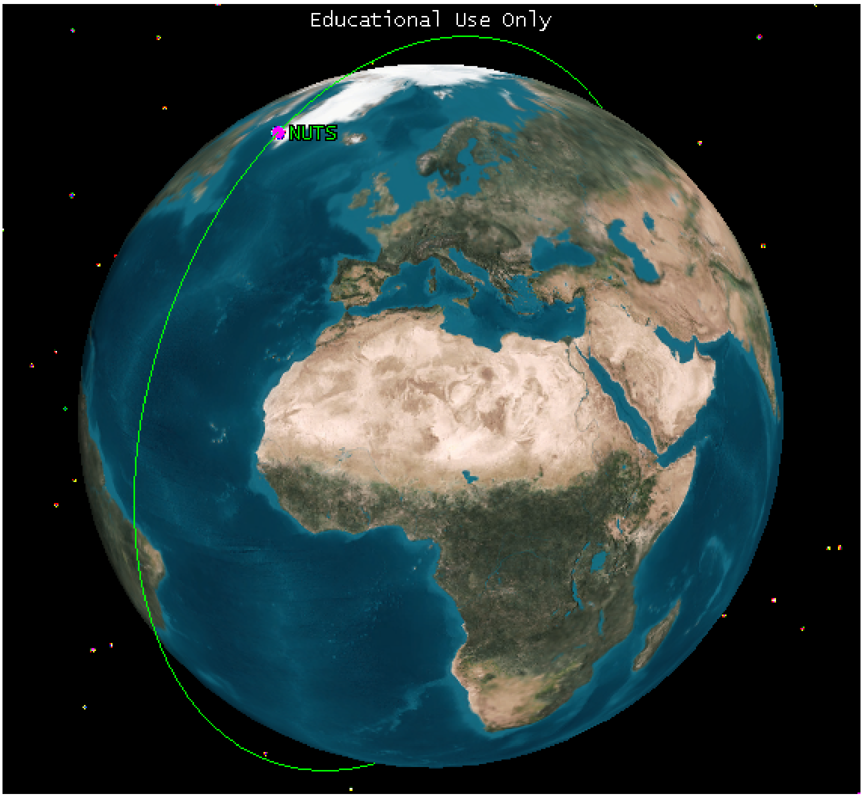


Figure 2.2: Illustration a polar LEO, made by Snorre Stavik Rønning through Analytical Graphics, Inc. Satellite Toolkit (STK).

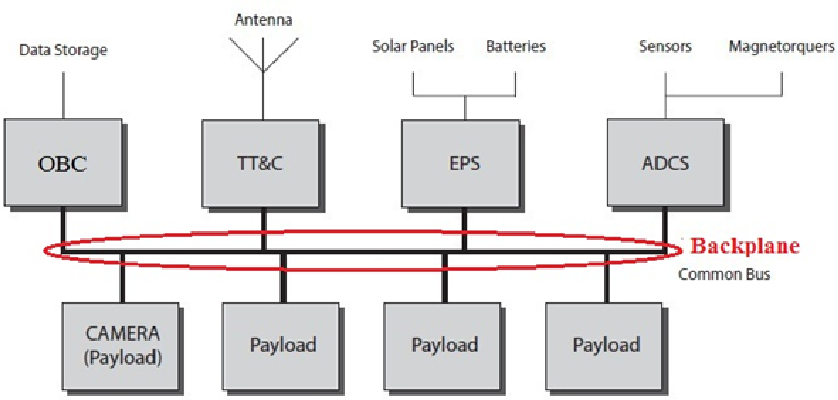
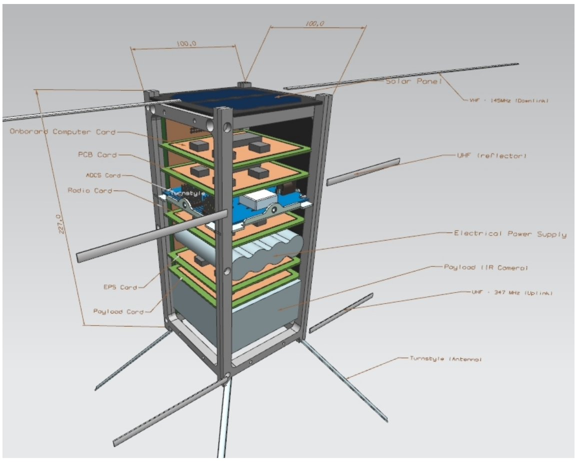
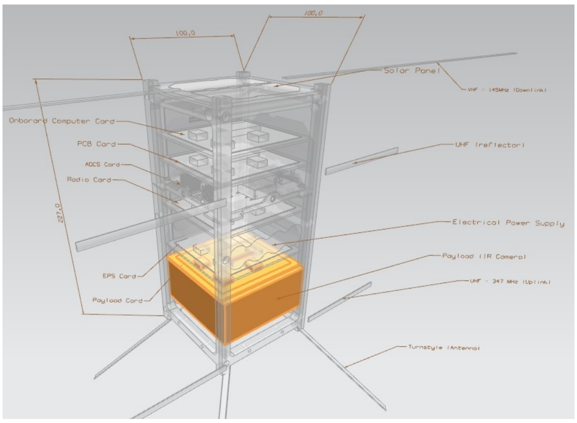


Figure 2.3: NUTS system overview. By courtesy of Dan Erik Holmstrøm.



(a) Internal structure



(b) Position of the payload

Figure 2.4: 3D model of NUTS. By courtesy of Kai Inge Midtgård Rokstad

to the backplane connector.

The main payload of NUTS will, as mentioned, be an infrared camera for observation of gravity waves, an atmospheric phenomenon which is further explained in Section 2.3. It will be situated at the bottom of the satellite with its lens pointing in the nadir[2] direction, as roughly illustrated in Figure 2.4(b). The camera will take pictures of the OH airglow layer in the atmosphere at various locations, which will be transmitted to a ground station for further analysis. These pictures can only be obtained during night, which means that the camera depends on the batteries to be sufficiently charged. It must also be made sure that the satellite has an orbit that involves both day and night, which is not always the case.

No absolute constraints have been put on the size and weight of the camera yet, but some indications have been given. Figure 2.5 shows a drawing of the layout of the bottom of the satellite. The camera module must at least stay within the grey area, which is approximately 80×80 mm. The height restriction is probably around 5-6 cm. Due to circuitry and antenna fastening, there are only two possible positions of the camera lens as indicated by the two circles in Figure 2.5, which results in maximum lens diameters of either 50 or 38 mm [10]. When it comes to weight, it is beneficial that the camera is as lightweight as possible. But one should keep in mind that in addition to the total weight constraint of 2.66 kg, the CubeSat specification also requires that the mass center of the whole satellite stays within a certain radius from the geometrical center [6]. Since the camera is situated at the bottom, it might be necessary to move the batteries or other heavy parts to compensate.

The *Attitude Determination and Control System* (ADCS) of the satellite will control the angular orientation (attitude) of the satellite such that the camera points stably towards the Earth. An estimation algorithm uses inputs from various sensors (sun sensor, gyroscope and magnetometer) in order to estimate the attitude. If the estimated attitude deviates from the wanted reference, the orientation of the satellite needs to be changed. This is done by means of magnetourqers, which affect the magnetic field of the satellite that will align with the magnetic field of the Earth. There will possibly be rotation around the nadir-zenith axis, but this can be measured and is hopefully very slow.

The *On-Board Computer* (OBC) will provide computing power and storage for the payload, issue commands to the other modules in the satellite and monitor the state of the whole system. It has a powerful micro controller, making it suitable for on-board processor-intensive tasks.

NUTS will have two transceivers and antennas for the VHF and UHF bands to be able to communicate with ground stations, in addition to a transmitter for UHF that will only send a beacon signal. Each of the two transceivers will have a bandwidth of 25 kHz and a data rate of 9600 bps. The AX.25 communication protocol will be used to enable communication with radio amateurs. The main ground station is situated on a roof at NTNU and will be operated by students participating in the project. Ground stations at Narvik, Andøya, Svalbard and Oslo may also be used if available.

[2]Downward, toward the Earth. Opposite of zenith.

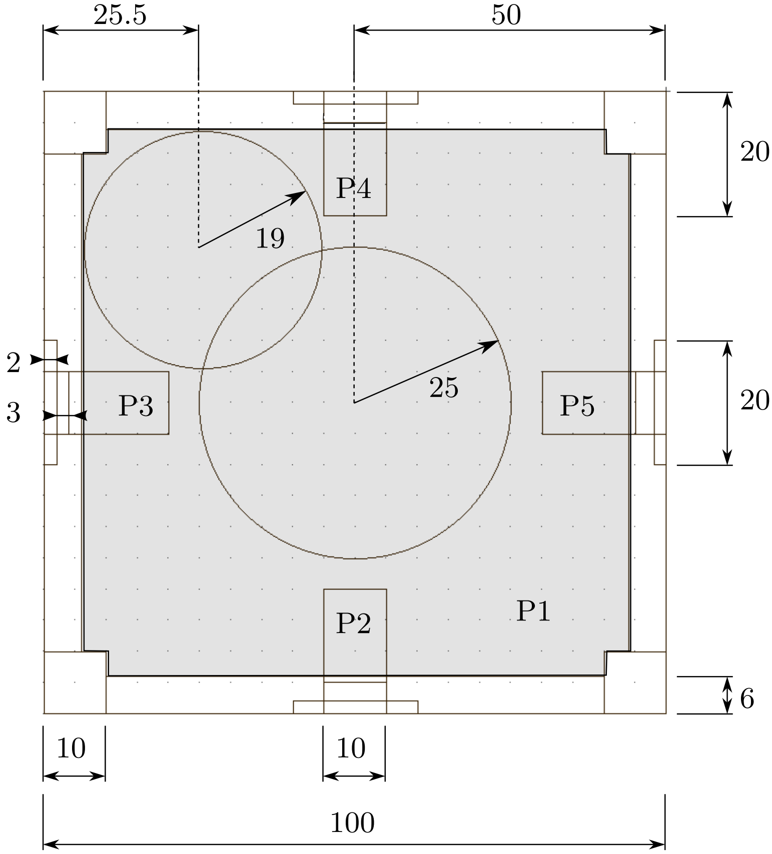


Figure 2.5: The bottom of the satellite indicating possible positions for the camera lens. By courtesy of Sigvald Marholm

2.1.5 Environmental Factors

Space is known to be an extreme environment in many ways. The temperature differences between shadow and direct sunlight is expected to be quite large. Different assumptions have been made when it comes to which temperature the different parts of the satellite will experience. Within the NUTS team, operating temperatures between $-40°C$ and $+85°C$ has been assumed, but no explicit study of this has been done yet. However, [11] presents temperatures measured on-board the CP3 CubeSat, which has orbital parameters quite similar to those assumed for NUTS. As shown in Figure 2.6, CP3 experienced exterior temperatures from $-30°C$ to $+20°C$ under normal operation. The time between the temperature minima is about the same as the orbital time, and reasonable to believe that these minima occurred the instant before the satellite came out of the shadow and the temperature started to rise again. It can be expected that the interior temperature of the satellite will vary even less.

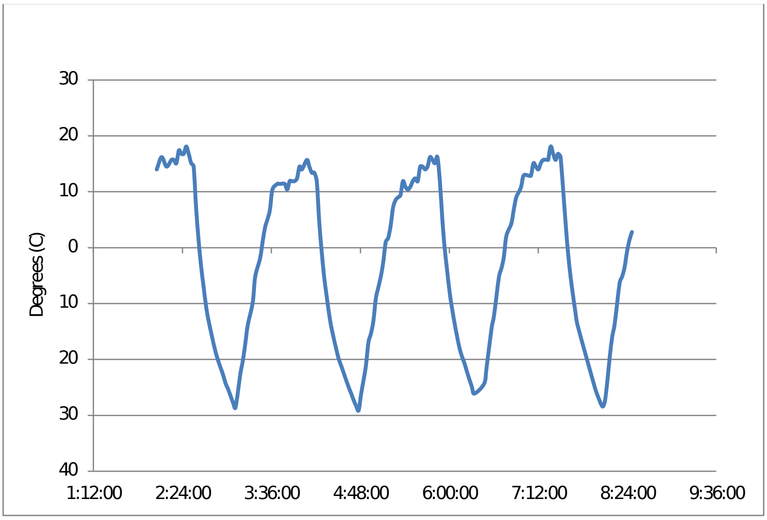


Figure 2.6: Measured exterior temperatures for the CubeSat CP3, from [11]

2.1.6 The European CubeSat Symposium

As a part of the Norwegian and international space technology community, the students in the NUTS project have participated at several national and international conferences and workshops. This gives the master student an unique experience in presenting their work for an international audience, and get more insight in international space technology. NUTS was represented with three presentations and one poster at the European CubeSat Symposium 2012 in Brussels in January, among them a presentation of the pre-study for the infrared camera payload held by Snorre Stavik Rønning and the author. The submitted abstract is given on the following page, and the slides for the presentation can be found in Appendix B.

Observation of Gravity Waves from a Small Satellite by Means of an Infrared Camera

S.S. Rønning, M. Bakken, R. Birkeland, P. Espy, R. Hibbins and T.A. Ramstad

Department of Electronics and Telecommunications, Norwegian University of Science and Technology (NTNU), Trondheim, Norway

The NUTS (NTNU Test Satellite) is a satellite being built in a student CubeSat project at the Norwegian University of Science and Technology. The project was started in September 2010 and is a part of the Norwegian student satellite program run by NAROM (Norwegian Centre for Space-related Education). The NUTS project goals are to design, manufacture and launch a double CubeSat by 2014. The satellite will fly two transceivers in the amateur radio bands. Final year master students from several departments are the main contributors in the project.

As a main payload, an infrared camera designed to observe gravity waves in the middle atmosphere is planned. Gravity waves, created by air blowing over mountains and weather phenomena, propagate throughout the atmosphere and drive the large scale flows in the middle atmosphere. Despite this their properties are poorly understood, mainly due to a lack of observational data. At an altitude of about 90 km in the atmosphere we find a layer of OH molecules that emit short-wave infrared radiation. When gravity waves propagate through this layer wave patterns in the radiation intensity are observed. Ground observations have found the wavelength of these patterns to be around 20 km and wave phase speeds to be around 25 m/s. But such observations have been limited to a few ground stations, and the possibility for global coverage that observation from a satellite offers would be a useful contribution to further research.

We discuss the design of a camera system and observation schedule to derive global data on the wave parameters of wavelength, intensity, phase speed and direction within the CubeSat constraints of available power, weight, size and download data rate. The choice of an off-the-shelf infrared camera is also considered, as well as signal processing algorithms for image restoration and compression.

2.2 Downlink Capacity for NUTS

The *downlink capacity* at the ground station, i.e. how much data that can be downloaded from the satellite, is an important parameter when deciding which compression scheme to choose for the payload data. This issue is closely related to the antennas and the communication module, and therefore a joint effort was made by Sigvald Marholm and the author to estimate the download capacity for the communication link between the satellite and the ground station in Trondheim. The results in this section can also be found in the thesis *Antenna Systems for NUTS* [10] written by Sigvald Marholm.

2.2.1 Orbit and Visibility Time

A satellite in a polar LEO will (almost) pass the north and south pole for every period. But since the Earth is rotating around its own axis, the satellite's ground track will vary. The satellite can only communicate with the ground station when it has line of sight, and it is therefore advantageous to have a ground station at high latitude to get as many passes per day as possible.

When the satellite passes over the ground station, a minimum *elevation angle* (see Figure 2.7) is required to obtain a stable radio link. For higher orbital altitudes, a larger elevation angle is required to get a sufficient received SNR at the ground station, as shown in [10]. The maximum length of a pass, or the *visibility time*, will depend on the orbital altitude and the required minimum elevation angle, as shown in 2.8. As an example, an altitude of 600 km and a minimum elevation angle of 20° can be assumed, which gives a visibility time of around 6 minutes. But for most passes it will unfortunately be smaller, since the satellite seldom passes straight above the ground station.

2.2.2 Simulations with STK and MATLAB

In order to estimate the average downlink capacity between a ground station and the satellite, knowledge about the total visibility time of several passes through a day or a week is necessary. Figure 2.8 gives an indication of the maximum visibility time, but it will vary a lot from pass to pass. To compute the total visibility time is much more complex. Several parameters have to be taken into account, among them the location of the ground station and the orbital parameters of the satellite.

Analytical Graphics, Inc. Satellite Toolkit (STK) was used to simulate the elevation angle as a function of time for passes over a ground station in Trondheim[3] during one week. A perfectly circular, sun-synchronous orbit[4] was assumed, and the simulations were performed for orbital altitudes of 350, 500 and 650 km. The obtained data was exported and read into MATLAB with the function `read_stk_elev()`, and then processed by using `threshold_stk_elev()` to compute the average downlink capacity in bits per day for different minimum elevation

[3]Coordinates: 63°25'47"N, 10°23'36"E

[4]Orbital elements: eccentricity = 0, inclination = 98°, RAAN = 0°, J4 perturbations included

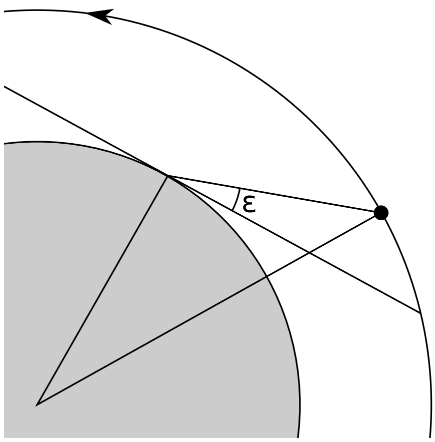


Figure 2.7: An illustration of the elevation angle (ε) of a satellite in orbit. Courtesy of Sigvald Marholm.

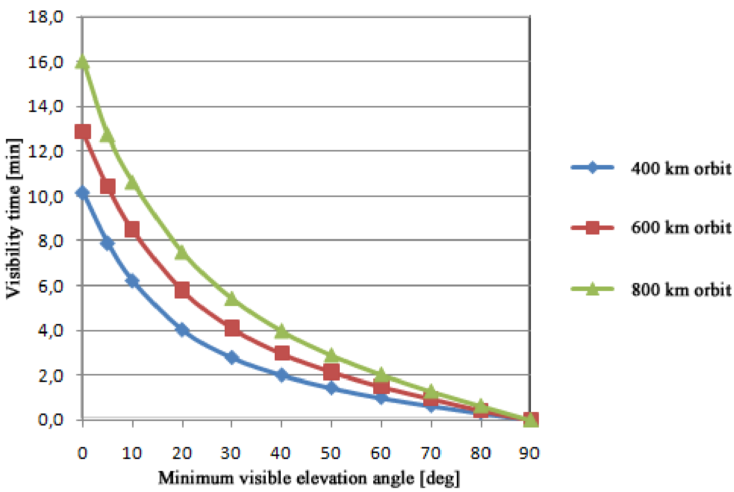


Figure 2.8: The maximum visibility time as a function of minimum visible elevation angle for different orbit altitudes. Reproduced by courtesy of Asbjørn Dahl

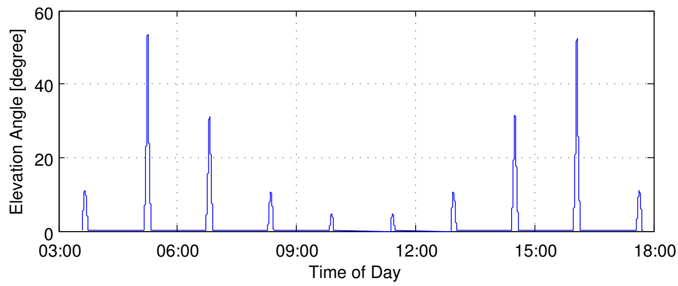


Figure 2.9: An example of elevation versus time.

Table 2.1: Required minimum elevation angle for different altitudes and the corresponding downlink capacities.

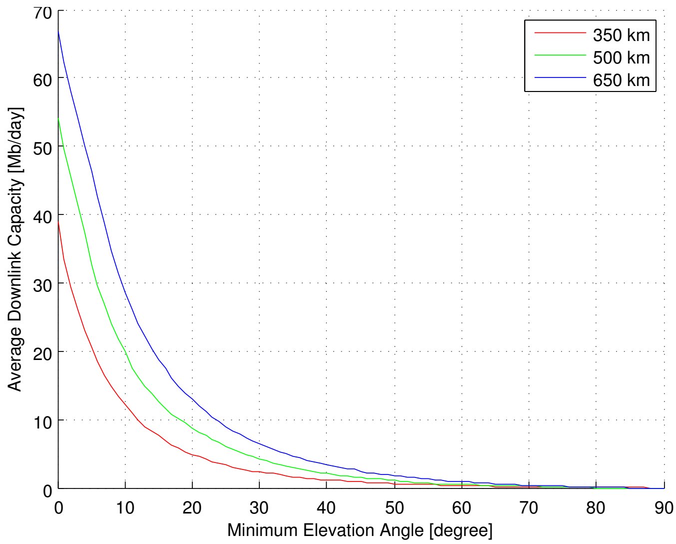
Altitude	Min. elev.	Average capacity
350 km	21°	4.54 Mb/day
500 km	28°	4.90 Mb/day
650 km	34°	4.97 Mb/day

angles, assuming a downlink data rate of 9600 bit/s. The MATLAB functions can be found in Appendix E.1.

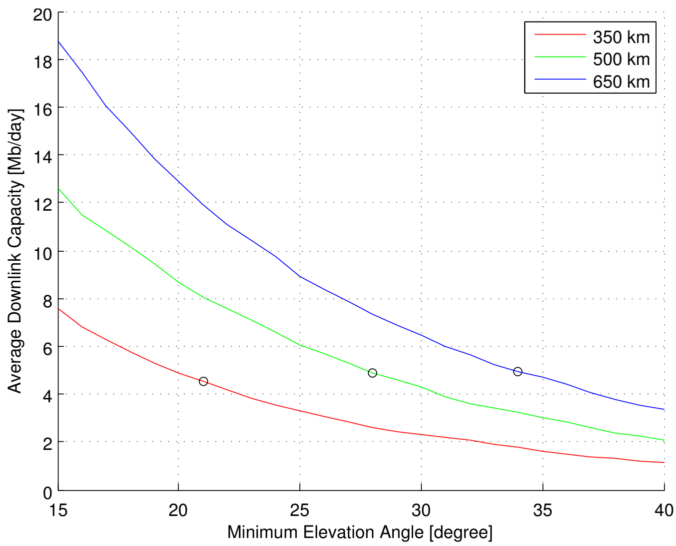
2.2.3 Results

An example of how the elevation varies is given in Figure 2.9, which shows the elevation angle as a function of time for the passes during the first day for an altitude of 500 km. It is seen that many of the passes have an elevation angle below 20 degrees, and only two of them are above 40 degrees. The resulting average downlink capacity is shown in Figure 2.10(a). It decreases rapidly if the minimum elevation angle is increased, since many of the short passes will be discarded when a high elevation is required. It is also seen that for a fixed minimum elevation angle, a higher orbit gives a larger downlink capacity. This is because a higher altitude results in higher elevation and longer visibility time. On the other hand, a larger elevation angle is required to get sufficient SNR for higher altitudes. It turns out that the resulting downlink capacities are relatively independent on the orbital altitude, as indicated in Figure 2.10(b). The resulting minimum elevation angles and corresponding downlink capacities are listed in Table 2.1.

The results in Table 2.1 gives the average *total* download capacity. Some of it has to be used for housekeeping, and how much of the capacity that will be left for the payload data will probably vary, but approximately half has previously been assumed. This would for instance mean that only 2.45 Mb of payload data can be downloaded per day on average (for an orbital altitude of 500 km), which is not much.



(a) Elevations from 0° to 90°.



(b) Elevations from 15° to 40°. The circles indicate the computed minimum elevation angle for the different altitudes.

Figure 2.10: Downlink capacity in Mb per average day for NUTS.

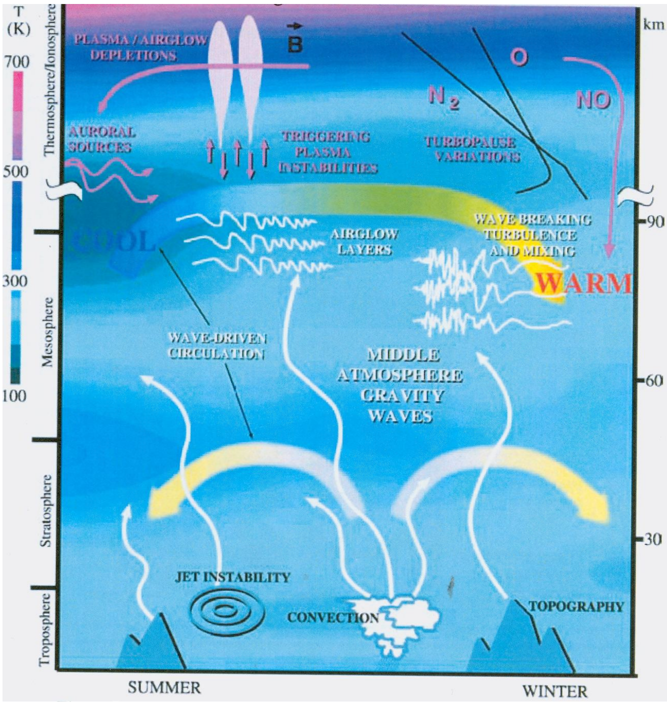


Figure 2.11: Illustration of gravity waves in the atmosphere, from [1]

2.3 Atmospheric Gravity Waves

Gravity waves have never been observed by means of an infrared camera from a satellite before. If our mission succeeds in delivering images of acceptable quality it will be a useful resource for research within atmospheric physics. This section gives a brief introduction to what this phenomenon is, and how it can be observed. It is summarized with a list of key features that will impact the following discussion regarding choice of camera and signal processing.

2.3.1 What are Gravity Waves?

Gravity waves, not to be confused with gravitational waves from General Relativity, are fluid-dynamical large-scale waves propagating vertically and horizontally through the Earth's atmosphere. They are mostly generated in the lower atmosphere by air blowing over mountains and weather phenomena. Due to the decreasing atmospheric density, they increase in amplitude as they propagate upward, as described in [1] and shown in Figure 2.11. Gravity waves are analogous to water waves; they are both generated in a fluid medium with gravity and buoyancy as restoring forces. The most dramatic effects are seen in the mesosphere, lower termosphere and ionosphere, and the waves are understood to play a major role in the global north-south/south-north (meridional) atmospheric circulation. A more detailed description of gravity waves in the context of atmospheric physics may be found in [12].

2.3.2 Observation of Gravity Waves

Gravity waves can be observed by means of various remote sensing methods. Most of the existing airborne observations has been done by measurements of temperature and wind by means of radar [1], but this method only provides vertical one-dimensional profiles. Because of this, ground based camera observations have also been used to provide two-dimensional images and information about transversal movement [2]. This method uses cameras to take pictures of the radiation from airglow layers of the upper mesosphere to observe gravity wave perturbation patterns [1]. The atmospheric airglow originates from atoms and molecules in the upper atmosphere that are excited by sunlight, and release this energy by night in form of visible green light as shown in Figure 2.12, but also infrared radiation. The strongest airglow emissions come from a layer of OH at an altitude of around 90 km in the atmosphere that emit infrared radiation with two intensity peaks in wavelengths at 1434 and 1381 nm [1]. Airglow emissions amplify the perturbation caused by gravity waves propagating through them, which makes them a very suitable observation medium.

Ground-based observations have shown that the gravity waves can be observed as transversal sine patterns in the airglow, with amplitudes around 5-10% of the average radiation intensity level. These observations have also found the wavelength of the gravity wave patterns to be in the range of 15-40 km with a mean of 26 km, and their wave phase speeds to be around 25 m/s. An image taken from ground is shown in Figure 2.13.

2.3.3 Key Features

Some key features of *Gravity Waves* (GW)s that are important for the following discussions are listed below.

GW wavelength Mean: 26 km, Minimum of interest: 15 km

GW phase speed Mean: 25 m/s

GW waveform Sine wave, amplitudes 5-10 % of average radiation intensity

OH spectrum Intensity peaks at 1434 and 1381 nm

OH height approx. 89 km

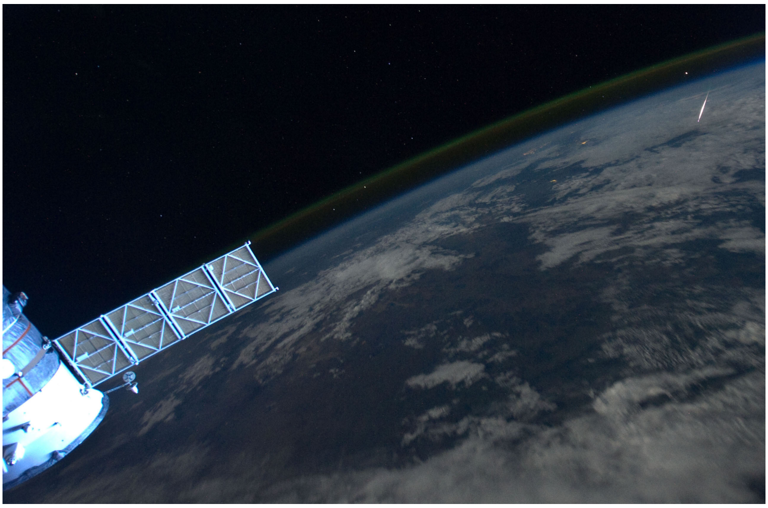


Figure 2.12: Image taken from the ISS. The airglow can be seen as a glowing green layer in the atmosphere. (Courtesy NASA)

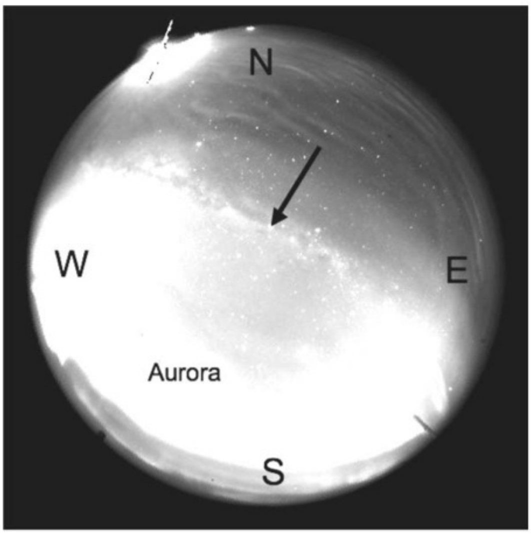


Figure 2.13: All-sky image showing gravity waves over Halley, Antartica, from [2]. The wave patterns in the upper right is far weaker than the Aurora at the lower left.

2.4 Optical Remote Sensing

Optical remote sensing in the visible and infrared region has various applications; meteorological imaging, surveillance purposes, detection of hazards like earthquakes or forest fires, vegetation mapping and many more. When designing a complete optical remote sensing system, each link in the chain of modules, from optics to image processing, will interact with each other. Atmospheric absorption, orbital mechanics and properties of the satellite must also be taken into account. This section will give a brief introduction to some terminology and concepts in optical remote sensing, and show some examples of projects similar to the NUTS payload.

2.4.1 Imaging Operation Modes

There are several ways to build a 2D image of a scene with a detector on a moving platform. The most obvious mode of operation resembles the way we would normally use a camera: use the whole 2D detector at once and "stare" at the scene long enough to aquire enough photons, and move on before the next image is taken. But it is also common to use a one-dimensional detector (a linear array) perpendicular to the direction of motion, and utilize the motion of the platform to do a scan of the scene. This mode of operation is called *push-broom imaging*. Timing is very important in this case; the exact speed of the platform must be known in order to obtain a continuous image of the scene. It is also possible to do some sort of mechanical scan from side to side while the platform is moving forward, which only requires a single detector. This operation mode is often called *whisk-broom imaging*. All three operational modes are discussed in [8].

Since satellites are moving with a high speed, the image will experience a shift during the exposure. This will typically introduce a blur in the direction of the movement, but the extent of this blur depends on the exposure time and the speed compared to the coverage area of the image. This effect is called *motion blur* and will be discussed in detail in Chapter 4. The simplest solution is to use a short integration time, but this can lead to low SNR, as further discussed in Section 3.2. Other methods commonly used in remote sensing systems are *Time-Delayed Integration* (TDI) and *Forward Motion Compensation* (FMC), as described in [13]. TDI uses several rows of pixels in the along-track direction to obtain multiple images of the same area, which is combined into one image to increase the effective integration time. FMC on the other hand, is based on controlling the pointing of the detector mechanically, such that the speed is reduced. This can be achieved for instance with moving mirrors, mechanical steering of the camera or through attitude control of the satellite. It is also possible to remove motion blur by post-processing, which will be discussed further in Chapter 4.

2.4.2 Spatial Resolution and Image Coverage

The spatial resolution is a measure of how fine details an optical system can resolve. In digital imaging it will be limited due to sampling since the detector has a limited number of pixels. In remote sensing the "footprint" of a pixel on the imaged scene is sometimes referred to as a *rezel*, and the spatial resolution can thus be given by *rezel size*[5]. Due to diffraction, the spatial resolution will also be limited by the optics of the imaging system according to the Rayleigh criterion as discussed in [8]. The actual spatial resolution will be given by the maximum of the two terms.

The instantaneous area covered by the imaging system is also an important parameter. It is sometimes referred to as the *field of view* or *ground coverage*, but will in the following be denoted as *image coverage* to avoid confusion with other parameters.

2.4.3 Camera Parameters

When choosing a camera, it is important to consider the impact of the parameters given in the specifications. The following list gives an overview of some common parameters that often appear in camera datasheets, both for the visible and the infrared region.

Pixel resolution Or *detector array size*, given in pixels × pixels. In the following a quadratic detector array is assumed, and the number of pixels in one of the dimensions is denoted as N_{px}.

***Field of View* (FOV)** The angle describing the area the camera can "see", determined by the focal length and detector size.

Focal length The distance between detector and the lens.

Pixel pitch The physical size of a pixel in the detector array.

Integration time Also called *exposure time* and *shutter speed*. Determines for how long the shutter is open and thus how many photons that are detected.

***Quantum Efficiency* (QE)** A measure of how many electrons that are generated per incoming photon, indicating sensitivity to radiation. Often wavelength dependent.

Spectral response The range of wavelengths the camera can detect.

***Noise Equivalent Irradiance* (NEI)** $\left[\frac{\text{photons}}{cm^2\ s}\right]$ The incident irradiance[6] that gives SNR equal to one.

2.4.4 Atmospheric Absorption

The atmospheric absorption of electromagnetic radiation is strongly dependent on wavelength, as seen in Figure 2.14. Especially in the infrared region there are several narrow peaks that must be taken into account, as discussed in [8]. Some of these peaks, or absorption lines are given in Table 2.2. The absorption may be a

[5]The spatial resolution is often referred to as *Ground Sample Distance* (GSD) in remote sensing literature, but the term *rezel size* is more general since one is not necessarily imaging the ground.

[6]Radiant flux density $\left[\frac{W}{m^2}\right]$

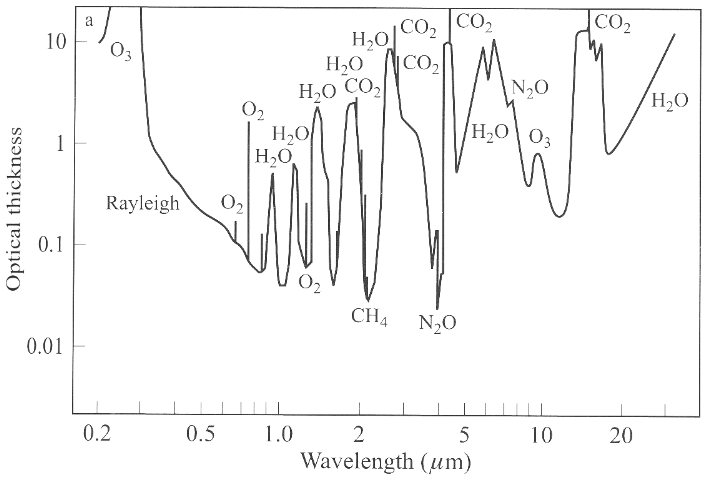


Figure 2.14: The absorption spectrum for the ultraviolet, visible and infrared region, from [8]. *Optical thickness* is a measure of absorption.

problem for applications that aim at imaging the ground, but luckily most of the water vapour is situated in the troposphere [12], well below the OH airglow layer. Additionally, the peak in the OH radiation spectrum at 1.38 μm coincides quite well with the water vapour absorption peak at 1.37 μm, which can be utilized to reduce interfering background radiation from Earth.

Table 2.2: Some atmospheric absorption lines in the infrared region, as given in [8]

Wavelength	Molecule
1.12	H_2O
1.25	O_2
1.37	H_2O
1.85	H_2O
1.95	CO_2

2.4.5 Examples of Similar Projects

In order to further illustrate the purpose of the NUTS payload, and to discuss its feasibility, it is at interest to consider a few similar projects.

The Waves Explorer

At the end of the 90's NASA planned a satellite mission, *The Waves Explorer*, for atmospheric research purposes [1]. It involved several payloads with different cameras and other scientific equipment which was supposed to investigate various properties of gravity waves on a global scale. One of the payload cameras, the *Hydroxyl Airglow Wave Imager* (HAWI) was planned to image infrared radiation from the OH layer mentioned in Section 2.3 in order to observe gravity waves. An overview of the most interesting specifications are reproduced in Table 2.3, and more detailed information can be found in [1]. The launch was planned to be in 2007, but the project was discontinued due to lack of financial support.

Table 2.3: Specifications for The Waves Explorer.

Orbit	650 km circular, 40° inclination
Payload weight budget	173 kg
HAWI specification:	
Spectral cutoff	1650 nm
Detector type	HgCdTe
Operation temperature	160 K (cooled by radiator)
Detector array size	256 × 256 pixels
Spatial resolution	$\leq$ 4 km
Operation mode	Push-broom

The SwissCube

Another interesting mission, the SwissCube, is a single cubesat developed by students at École Polytechnique Fédérale de Lausanne [14]. It was launched in 2009 and carries a telescope and a CMOS detector that captures images of airglow radiation at 767 nm wavelength. The SwissCube is still in operation, and has sent several successful images to the ground station. It is an interesting case because it has proved it possible to do atmospheric imaging with such a small satellite, but it is only measuring the strength of the airglow and has not done any attempt to identify any gravity waves. An overview of a few specifications is given in Table 2.4, and more details about the payload can be found in [15].

Table 2.4: Specifications for the SwissCube.

Payload weight budget	100 g
Detector type	CMOS
Spectral response	767 nm, bandwidth of 20 nm
Spatial resolution	$\leq$ 5 km
Detector array size	188 × 120 pixels

Chapter 3

The Infrared Camera

The properties of the satellite orbit, the infrared camera and the remote sensing system as a whole, will all affect the resulting image. This poses some requirements on the camera to ensure that the obtained images contain useful information fit for further analysis.

First, a brief introduction to infrared camera technology is given, and it is concluded that uncooled InGaAs is the most suitable sensor type for our application. Secondly, noise and SNR of the detector is treated, with emphasis on the different noise types in InGaAs detectors, and how the integration time influences the SNR. Then the specification of camera parameters are discussed, which involves several parameters regarding the satellite, operation mode, optics and detector. Since many of these parameters remain unknown, no definite specification for the camera has been obtained yet, but an attempt has been made to give an overview of how these parameters interact, and what kind of requirements that can be expected regarding optics and detector. Based on this discussion, some possible camera candidates are presented. Finally, synthetic test images were made based on assumptions on the phenomenon and the camera, which will be very useful when developing image processing and compression algorithms.

3.1 Infrared Radiation and Camera Technology

The infrared part of the electromagnetic spectrum has wavelengths that ranges from 0.7 to 300 μm and can be divided into several spectral regions, as illustrated in Figure 3.1. These names and boundaries of these regions in the literature vary a bit, but for now the classification in Figure 3.1 will be used. This means that the peaks in the OH airglow spectrum mentioned in Section 2.3 are situated in the *Short-wave infrared* (SWIR) region.

Due to the large variation in wavelengths within the infrared spectrum, there are also several different sensor types and applications for the different regions of the spectrum. *Silicon* (Si) sensors are commonly used in the *Visible and near-*

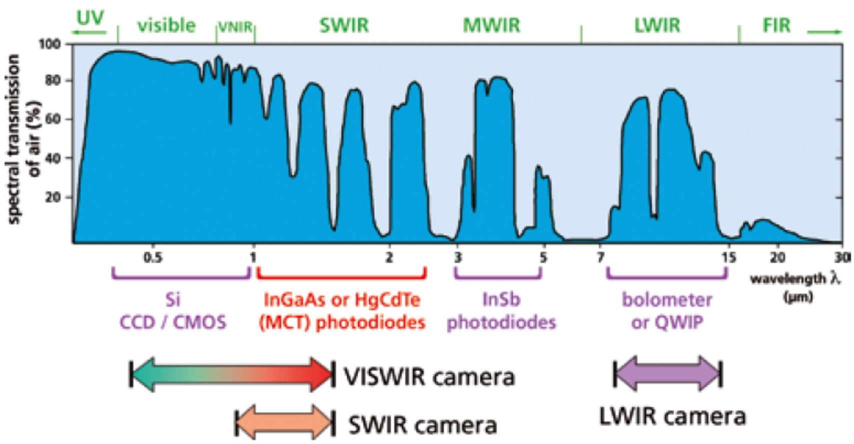


Figure 3.1: The visible and infrared region. From [16]

.

infrared (VNIR) region, but does unfortunately not go beyond 1,1 μm[17]. For longer wavelengths than this, other technologies must be used, as indicated in Figure 3.1. For the SWIR region, *Indium Gallium Arsenide* (InGaAs) is the most suitable sensor type [17]. InGaAs has a lower bandgap energy than Si, and is therefore sensitive to longer wavelengths, as illustrated in Figure 3.2. The wavelength response of InGaAs typically covers a range from 0.9 to 1.7 μm.

Additionally, there are two main classes of infrared cameras; cooled and uncooled. As discussed further in Section 3.2.3, thermal noise can be a big problem for infrared sensors. It is therefore common to apply external cooling to reduce the thermal noise in the detector, especially for scientific applications that require very high SNR. These cameras are usually heavy and power demanding, and therefore not suited on-board a small satellite. Recently, commercial light-weight infrared cameras without cooling have become more common, usually designed for applications such as night vision and thermal inspection, as discussed for instance in [16].

3.2 Noise and SNR

3.2.1 Signal-to-Noise Ratio Metrics

As discussed in [18], many different definitions of the SNR are being used as metrics for the image quality in remote sensing systems. The basic definition of SNR is simply

$$SNR \equiv \frac{\text{signal}}{\text{noise}} , \tag{3.1}$$

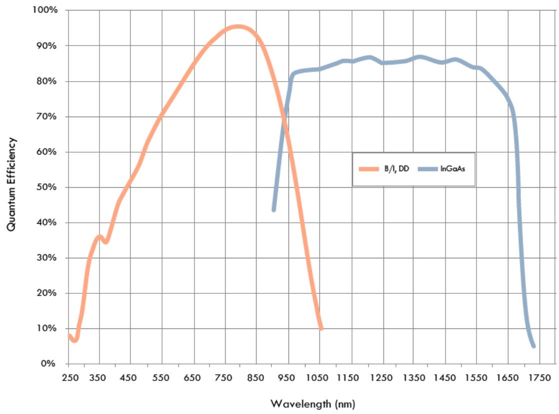


Figure 3.2: Quantum efficiency for Si and InGaAs. From [17]

but the signal and the noise can be measured either as an amplitude or a power, and the result can be given in either linear scale or logarithmically with *decibels* (dB)[1]. The SNR metrics from the detector point of view is often defined differently than from the signal processing point of view. This can lead to some confusion when looking at the complete remote sensing system as a whole, as attempted in this report. For clarification, some different SNR metrics are therefore discussed below.

From the detector point of view, the signal and noise are counts of photons and electrons, and it is therefore common to define the SNR as an amplitude or signal level ratio. When comparing the incoming signal to the detector noise, the *Detector Signal-to-Noise Ratio* (DSNR) can be defined as given in [18]:

$$\text{DSNR} = \frac{\text{mean target signal}}{\text{noise standard deviation}} = \frac{\bar{\text{s}}_{\text{target}}}{\sigma_{\text{noise}}} \tag{3.2}$$

this can be seen as a measure of uncertainty in the detector output when imaging a target with constant intensity level. It is also common to look at the difference between two signal intensities, and compare this to the detector noise:

$$\text{DSNR}_{\Delta\text{s}} = \frac{\text{signal difference}}{\text{noise standard deviation}} = \frac{\Delta\text{s}_{\text{target}}}{\sigma_{\text{noise}}} \tag{3.3}$$

In signal and image processing on the other hand, SNR is usually given as a power ratio, commonly defined as the variance of the wanted signal versus the variance of a distortion, as further discussed in Section 4.1.

[1]Since the conversion to dB is defined differently for amplitude and power, the resulting SNR in dB will be the same in both cases. (Power definition: $(x^2)_{dB} = 10\log x^2 = 20\log(x)$, amplitude definition: $x_{dB} = 20 \cdot \log(x)$

3.2.2 Signal

The signal in a remote sensing system has many possible units of measure. One of them is the number of photoelectrons generated in the detector. The relationship between the number of photons hitting the detector, n_p, and the number of photoelectrons generated in the detector during exposure, n_pe, depends on the QE of the detector. The average amount of photoelectrons, $\bar{n}_\text{pe}$, will also increase with the integration time:

$$\bar{n}_\text{pe} = QE \cdot \bar{n}_\text{p} \tag{3.4}$$

$$= QE \cdot \bar{n}'_p \cdot t_\text{int} \tag{3.5}$$

where $\bar{n}'_p$ is the number of photons hitting the detector per second, which depends on the target radiation intensity and atmospheric effects, as well as optics and detector size. The details of these effects are discussed further in [19] and [20], but for now the value $\bar{n}'_p$ is regarded as unknown.

After *Analog-to-digital* (A/D) conversion, the signal is measured in counts of the A/D converter.

3.2.3 Noise Sources in InGaAs Sensors

The different types of noise discussed in this section appear in other optical detector types as well, but for now the focus will be on their influence in InGaAs sensors.

Photon Noise

Photon noise is the shot noise associated with the number of incoming photons in the detector [19]. Shot noise appears due to the discrete nature of electrons and photons, and arises both in electronic and photonic devices. The signal plus photon noise is Poisson-distributed [19] with expectation value equal to the signal level ($\bar{n}_\text{pe}$). Since the standard deviation of a Poisson distribution is equal to the square root of its expectation [21], it is clear that the noise increases as the signal increases. The photon noise can often be modelled as Gaussian, since the Poisson distribution approaches the normal distribution for large numbers.

Since photon noise is inevitable, the performance of an ideal detector is said to be photon noise limited. Using the SNR definition in (3.2), the DSNR of an ideal detector (i.e. with photon noise as the only noise source) can be given as

$$\text{DSNR}_\text{photon} = \frac{\text{mean signal value}}{\sigma_\text{photon}} = \frac{\bar{n}_\text{pe}}{\sqrt{\bar{n}_\text{pe}}} = \sqrt{\bar{n}_\text{pe}} = \sqrt{QE \cdot \bar{n}'_p \cdot t_\text{int}}, \tag{3.6}$$

The SNR increases with the square of the number of photoelectrons, and the photon noise is therefore more dominating for lower radiation intensity and short integration time.

Dark current

Dark current is thermally generated charges that occurs even though there is no light incident on the detector. This leads to a constant background offset in addition to *dark noise*, which is the random shot noise associated with the dark current. The *dark charge* is the number of charges counted during the integration time, and is proportional to the dark current: $n_{\text{dark}} = i_{\text{dark}} \cdot t_{\text{int}}$.

In the same way as for photon noise, the dark noise is also a kind of shot noise, and therefore Poisson distributed. Its standard deviation is given by the square root of average dark charge, $\bar{n}_{dark}$, and can be expressed as

$$\sigma_{\text{dark}} = \sqrt{\bar{n}_{\text{dark}}} = \sqrt{\bar{i}_{\text{dark}} \cdot t_{\text{int}}} \tag{3.7}$$

where $\bar{i}_{\text{dark}}$ is the average dark current given in electrons per second.

The offset level of the dark current is not considered as noise, but rather as a constant background signal. It may however change with temperature, as shown in the experiment in Section 3.2.5. Since the dark current is integrated along with the photoelectrons, it will also increase for longer integration times.

Dark current may also cause *fixed pattern noise* due to permanent differences in dark current for the different pixels. The variations from pixel to pixel arise from random manufacturing differences of the diodes, but the pattern does not change over time. It may however change with temperature or integration time, as shown in Section 3.2.5.

In contrast to Si detectors, which are usually photon noise limited, InGaAs detectors are dark noise limited devices [17]. This is due to the lower bandgap energy of InGaAs, which makes it sensitive for longer wavelengths but also results in higher dark current. A high dark current level leads to more dark noise, but it also causes the detector to saturate for long integration times, and therefore puts a limit to the maximum integration time that can be allowed [19]. As discussed in [22], the dark current can be efficiently reduced by cooling. However, as long as the integration time is kept short, the dark count level and the corresponding fixed pattern noise can be removed by a simple background subtraction, as further discussed in Section 4.2.3.

Readout Noise

The readout circuitry consisting of amplifiers and an A/D converter will also generate noise. The standard deviation of the readout noise is often measured in counts and specified in the datasheet of the camera. The A/D converter would also introduce some quantization noise, but for now it is assumed to have many levels, such that the quantization noise is negligible compared to the noise from the detector.

3.2.4 SNR for an InGaAs Detector

The following discussion focus on the SNR from the detector itself, before the A/D converter and readout. It is assumed that the fixed pattern noise can be removed,

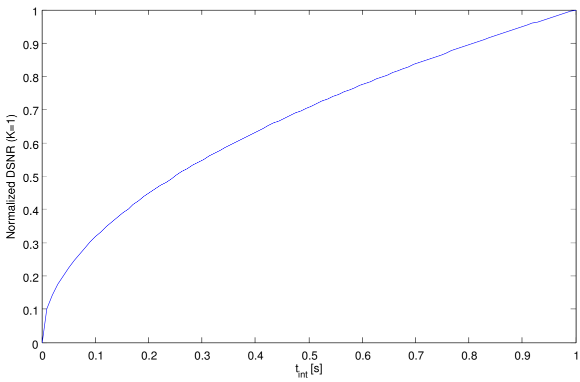


Figure 3.3: Normalised DSNR as a function of integration time.

and that the dark noise and the photon noise are the dominating noise contributors from the camera before readout.

Assuming that the total detector noise can be found by adding the noise sources, and that the sources are independent Gaussian distributed variables, the variance of the total noise will be equal to the sum of the variances. This leads to the following standard deviation for the detector noise:

$$\sigma_{\text{noise}} = \sqrt{\sigma^2_{\text{photon}} + \sigma^2_{\text{dark}} + \sigma^2_{\text{readout}}} \approx \sqrt{\bar{n}_{\text{pe}} + \bar{n}_{\text{dark}}}\ , \tag{3.8}$$

assuming that the readout noise in most cases is negligible compared to the two shot noises. Using the metric from (3.2), the SNR of the detector can be expressed as

$$\text{DSNR} = \frac{\bar{n}_{\text{pe}}}{\sqrt{\bar{n}_{\text{pe}} + \bar{n}_{\text{dark}}}} \tag{3.9}$$

which gives

$$\text{DSNR} = \frac{QE \cdot \bar{n}'_p \cdot t_{\text{int}}}{\sqrt{QE \cdot \bar{n}'_p \cdot t_{\text{int}} + \bar{i}_{\text{dark}} \cdot t_{\text{int}}}} = \frac{QE \cdot \bar{n}'_p}{\sqrt{QE \cdot \bar{n}'_p + \bar{i}_{\text{dark}}}} \cdot \sqrt{t_{\text{int}}} \tag{3.10}$$

when inserting the expressions for $\bar{n}_{\text{pe}}$ and $\bar{n}_{\text{dark}}$. $\bar{i}_{\text{dark}} = 0$ gives the same expression as for the ideal detector in (3.6).

The non-linear relationship between the incoming photons, the dark noise and the quantum efficiency can be summarized in a factor K:

$$\text{DSNR} = \text{K} \cdot \sqrt{\text{t}_{\text{int}}}, \tag{3.11}$$

which can be convenient to use for simulations when the signal strength is unknown. The *normalised DSNR* is obtained by setting K=1, and is plotted in Figure 3.3 to illustrate the dependance on integration time.

3.2.5 Noise Measurements for an InGaAs Sensor

The high dark current level of InGaAs sensors will create a background signal that is added on top of the signal from the target. In order to investigate the impact of this background signal for different integration times and temperatures, experiments were performed with a one-dimensional InGaAs sensor by Patrick Espy at the Department of Physics at NTNU.

Experiment

The sensor that was used is an Andor IDus InGaAs detector with 1024 pixels and 25 μm pitch, with cooling to regulate the temperature. The complete specifications for the sensor are enclosed in Appendix D.

Several images were taken at different temperatures and integration times with closed shutter in order to detect the background signals.

By simple averaging and subtraction, three components of the background signal with different statistical properties were found: the background offset level, the fixed pattern noise and the thermal noise. The thermal noise component of the background signal was found by taking the difference between two of the images. Several images were then averaged to reduce the influence of the thermal noise, and get the fixed part of the background (fixed pattern noise plus offset). The background offset was then found by averaging the fixed background over the elements. By subtracting the offset from the fixed background, the fixed pattern noise was found. The results are measured in counts of the A/D converter.

Results

Figure 3.4 shows the results for the background offset level and the thermal noise for increasing integration time. It is seen from Figure 3.4(a) that the background offset increases linearly with integration time, but the slope is strongly dependent on temperature. For 20°C, it has a slope of more than 800 counts per second, for 0°C it is reduced to less than 100 counts per second, and for lower temperatures it has a constant value around 1000 counts. This corresponds well with the theoretical behaviour of average dark charge mentioned in Section 3.2.3. There seems to be a fixed offset of 1000 counts in the detector.

The thermal noise in Figure 3.4(b) on the other hand, shows a less regular behaviour. First of all, it is about 200 times smaller than the offset. The curves for the negative temperatures have a constant level of about 5 counts, and the same holds for 0°C up to 2 seconds. There seem to be some kind of threshold that the background offset in Figure 3.4(a) must exceed before the thermal noise changes its behaviour and starts to increase. This is probably due to a random readout noise which dominates for low temperatures and integration times.

The fixed pattern noise in Figure 3.4(c) is about the same order of magnitude as the background offset for high temperatures, but has a floor of about 150 counts for low temperatures and short integration times. The results are summarized for an integration time of 1 second and three different temperatures in Table 3.1.

The most important thing to note from this experiment is that the thermal component is almost negligible compared to the background offset and fixed pattern noise, especially for temperatures below 0°C. But the thermal noise is the only part of the background that is random in time, and the two other components can therefore be easily removed by background subtraction, as further discussed in Section 4.2.3. Some camera manufacturers claim, for instance in [17], that the photon signal will drown in the high level of background signal, and that deep cooling (down to -90°C degrees) therefore is required. From this experiment it seems that temperatures between -20°C and 0°C will be sufficient to keep the thermal noise relatively low. As discussed in 2.1, temperatures of -30°C to +20°C can be expected for the NUTS satellite. Since the camera only will operate during night, cooling of the detector does not seem to be necessary.

Table 3.1: Measured background signal for an InGaAs sensor, at 1 second integration time

Temperature $[°C]$	**-20**	**0**	**20**
Offset (avg)	1054	1143	1826
FPN[a] **(std)**	139	228	1338
Thermal noise (std)	4.54	5.54	10.96

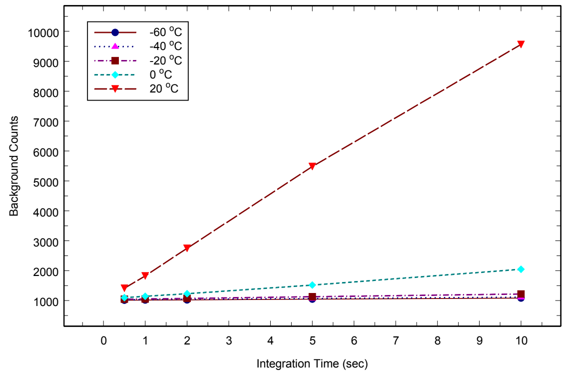
[a] Fixed Pattern Noise

3.2.6 Other Possible Disturbances

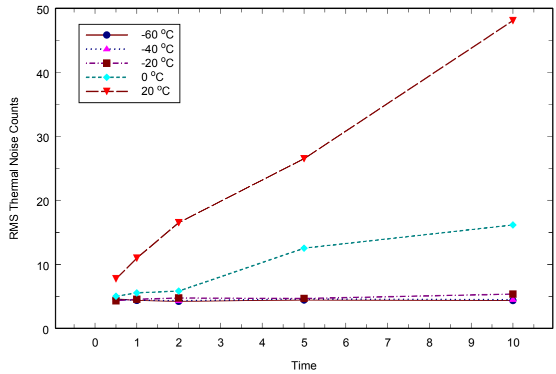
When aiming to observe the gravity wave patterns in the airglow, it would be beneficial if there are as little disturbances from other radiation sources as possible.

Out-of-band background radiation from the Earth can be efficiently reduced by applying an optical bandpass filter. As already discussed, there is a peak in the water vapour absorption spectrum that coincides quite well with one of the peaks of the OH spectrum. The water vapour is situated well below the OH airglow, and is almost always present except for a few spots above dry places on Earth, for instance the Sahara and Antarctica [23]. The water vapour will therefore block the radiation from the Earth, while leaving the radiation from the airglow unaffected. One should however beware that by applying an optical bandpass filter, the energy of the incoming signal is reduced. The passband of the filter should therefore not be made too narrow.

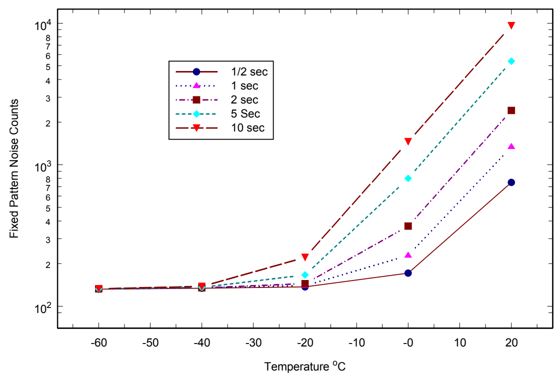
Gravity wave images taken from ground are often disturbed by auroral activity. However, the aurora is mainly present in the visible part of the spectrum, and will not appear in the SWIR region [24].



(a)



(b)



(c)

Figure 3.4: (a) Measured background offset and (b) thermal noise vs. integration time for different temperatures. (c) Fixed pattern noise vs. temperature for different integration times. Reproduced by courtesy of Patrick Espy.

3.3 Camera Operation and Specification

3.3.1 Mode of Operation

Perturbations of the satellite velocity can be a big problem for the push-broom and whisk-broom modes mentioned in Section 2.4. Additionally, most off-the-shelf cameras have two-dimensional detectors, and are not designed for any special remote sensing operation modes. A two-dimensional detector also have a larger area than a one-dimensional array, which increases the incoming signal. Therefore, the most suitable operation seems to be the simple “staring mode” with a two-dimensional detector instead of push-broom or whisk-broom mode, in contrast to many other optical remote sensing systems. From now on, it will be assumed that the camera has a two-dimensional detector.

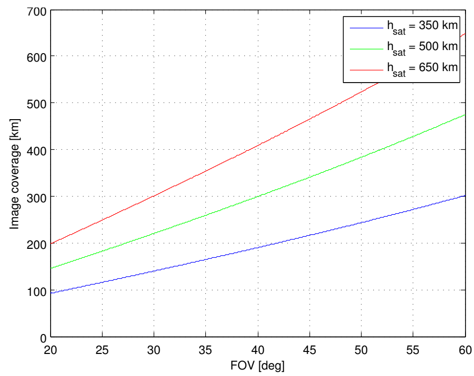
In order to cover a larger area at once, it will be convenient to obtain sequences of images with a suitable overlap, instead of many single images from various locations. How often these sequences can be obtained will depend on the power available and the downlink capacity, as further discussed in Chapter 5.

3.3.2 Calculations of Camera Parameters

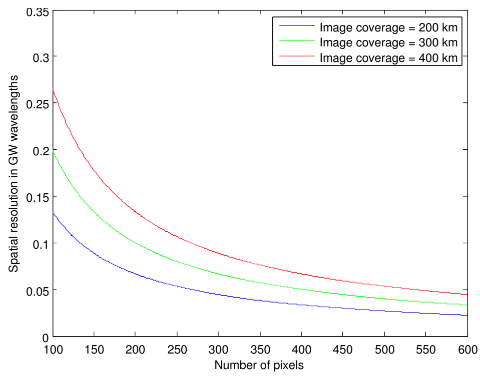
Based on the previous discussion of the gravity wave properties and the satellite orbit, some assumptions are made, as listed in Table 3.2. In order to compute the required number of pixels and FOV of the camera from the requirements in Table 3.2, orbital parameters of the satellite also have to be taken into account. The formulas used for these calculations are given in Appendix C. In order to get a better overview of the problem, a spreadsheet with all the parameters and formulas affecting the camera requirements was developed. This became a useful tool which made it possible to simulate different scenarios, for instance by varying the altitude of the satellite. A snapshot of the spreadsheet itself can be found in Appendix C, and the most interesting imaging parameters are plotted in Figure 3.5 for varying camera and satellite parameters. They will be further discussed below.

3.3.3 Image Coverage

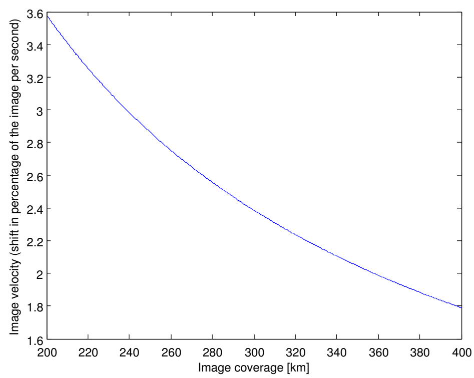
Since the gravity waves is such a large-scale phenomenon, the image coverage is a more crucial parameter than the resolution. To be able to see the wave patterns properly, a coverage of about 10-20 wavelengths per image is suitable. If a mean gravity wave wavelength of 26 km is assumed, this results in a required coverage of about 260-520 km. The image coverage will depend on the detector size and focal length as well as the distance to the target. As shown in Appendix C, the relationship between detector size and focal length results in a FOV, which might be a more intuitive parameter. The resulting image coverage as a function of FOV for different orbits is shown in Figure 3.5(a). From this, it seems like a FOV of around 40-45° would be a suitable choice, because this would provide a suitable image coverage for a wide range of orbital altitudes. Since the orbital altitude is not known yet, an image coverage of 300 km will be assumed in the further discussion.



(a)



(b)



(c)

Figure 3.5: Image coverage, spatial resolution and image velocity for varying satellite and camera parameters. (a) Image coverage as a function of FOV for different satellite altitudes. (b) Spatial resolution given in GW wavelengths per pixel as a function of number of pixels in the detector array, for different values of the image coverage. (c) Image velocity as a function of image coverage, assuming a velocity with respect to the airglow of $V' = 7.16$ km/s

This could for instance correspond to a satellite altitude of 500 km and FOV of 40°.

3.3.4 Spatial Resolution

The main requirement for the spatial resolution, or rezel size, Δx $[\frac{\text{km}}{\text{px}}]$, is simply that the gravity wave patterns are clearly distinguishable. As mentioned in Section 2.3, it is assumed that the shortest gravity wave wavelength one wants to detect is 15 km. For an image coverage of 300 km, this corresponds to a spatial frequency of 20 cycles per image. To ensure that this frequency will be visible in the image, the image must be sampled at the Nyquist rate [25] or higher. This means that sampling frequency must be at least twice the largest frequency one wants to detect, or equivalently:

$$\Delta x_\lambda = \frac{\Delta x}{\lambda_{\text{GW}_{min}}} < \frac{1}{2} \tag{3.12}$$

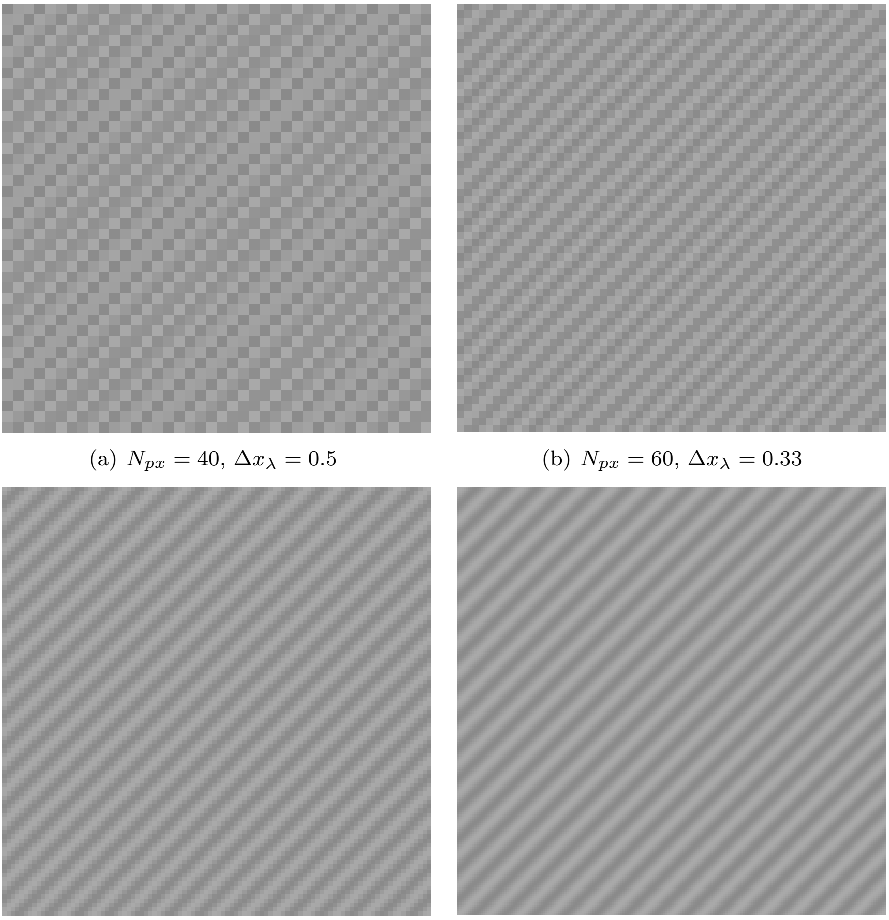
where Δx_λ is the *spatial resolution in wavelengths*. If $\lambda_{\text{GW}_{min}} = 15$ km, this means that the rezel size, Δx, must be smaller than 7.5 km.

To assure a reasonable perceptual quality, it is however advisable to choose a finer resolution than the minimum that is required by the Nyquist criterion. An illustration of the appearance of different values for Δx_λ is given in Figure 3.6, which shows sine images of 15 cycles per image, sampled with different numbers of pixels. From this example it seems that a Δx_λ of 0.1-0.2 is required to obtain a reasonable quality, which is much lower than 0.5. This implies a maximum rezel size of 1.5-3 km if $\lambda_{\text{GW}_{min}} = 15$ km.

Figure 3.5(b) shows the resulting Δx_λ as a function of N_{px} for different values of the image coverage. If it is assumed that $\Delta x_\lambda = 0.2$ gives a sufficient quality, and the image coverage is 300 km, a detector with only 100 pixels in each direction would actually be sufficient. But if the coverage is 400 km and $\Delta x_\lambda = 0.1$ is required, the detector must have at least 267 pixels in each direction.

As mentioned in Section 2.4, the spatial resolution may also be limited by diffraction. This will depend on properties of the optics. For now it is assumed that the spatial resolution will be limited by the rezel size, since the resolution of the detector will be quite coarse. This should however be taken into consideration when designing the optics.

It should also be noted that the number of pixels the detector is partitioned into will affect the pixel pitch. From this point of view the number of pixels should be kept low. A larger pitch gives more incoming photons per pixel, improving the DSNR of the detector. For detectors with a high resolution and a small pixel pitch, the pixel size can be increased artificially by *binning* [19], i.e. combining the signal from several neighbouring pixels, in order to increase the DSNR at the expense of spatial resolution.



(a) $N_{px} = 40$, $\Delta x_\lambda = 0.5$

(b) $N_{px} = 60$, $\Delta x_\lambda = 0.33$

(c) $N_{px} = 100$, $\Delta x_\lambda = 0.2$

(d) $N_{px} = 200$, $\Delta x_\lambda = 0.1$

Figure 3.6: Synthetic image with diagonal sine wave, sampled with different numbers of pixels N_{px} resulting in different wavelength resolutions Δx_λ

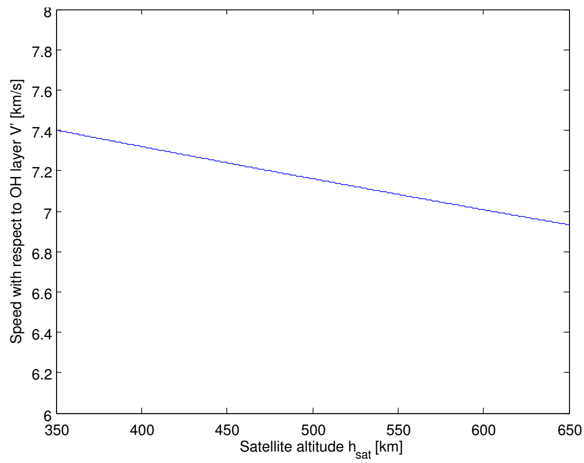


Figure 3.7: Speed of the satellite w.r.t the OH layer, as a function of altitude.

3.3.5 Image Speed

As discussed in Section 2.4, the image might get blurry due to the speed of the satellite. The *image speed* $[\frac{\text{image shift}}{\text{second}}]$, i.e. how fast the satellite moves compared to the coverage area of the image, is an important parameter when investigating the impact of the blur.

The image speed depends on the image coverage and the speed of the satellite with respect to the OH layer, V' [km/s]. This speed is calculated in Appendix C, and plotted as a function of orbital altitude in Figure 3.7. It turns out that it decreases relatively slowly. For the following calculations it is therefore regarded as approximately independent on orbital altitude, and the value corresponding to a 500 km orbit, $V' = 7.16$ [km/s], will be assumed.

Figure 3.5(c) shows the image speed as a function of the image coverage. The image shift is measured in percentage of the total image, to make it independent on resolution. The image coverage may correspond to different combinations of FOV and altitude, as already shown in Figure 3.5(a).

It is hard to say what impact the different image velocities have on the image quality, since this depends on the image content. The values of the image velocity will be used in simulations in Section 4.3.3 to investigate this further.

3.4 Summary of Camera Requirements

To ensure a sufficient resolution and image coverage, the choice of detector and optics must be made jointly. The uncertainties of the orbital altitude must also be taken into account. The number of pixels should be chosen large enough to provide the required resolution, but also be kept low to give a large pitch and good DSNR. A detector size of 128×128 pixels might be sufficient, but 256×256 seems like a safer choice for the time being. This is by the way the same detector size as NASA intended to use on their Waves Explorer, which was mentioned in Section 2.4.5.

According to the discussion in Section 3.3, Table 3.2 shows a summary of the preliminary camera parameters that will be assumed in the following of this report.

Table 3.2: Summary of resulting camera parameters for some given assumptions:

Assumed physical parameters:	
Orbital altitude	500 km
Mean GW wavelength	26 km
Min. GW wavelength	15 km
Assumed requirements:	
Number of GWs per image	10-20
Spatial resolution in wavelengths	0.1-0.2
Resulting image requirements:	
Image coverage	300 km
Spatial resolution	1.5-3 km/px
Resulting camera parameters:	
Field of View	40°
Number of pixels in detector	256×256

3.5 The Search for a Suitable Off-the-shelf Camera

It is beyond the scope of this thesis to find and integrate a suitable camera. Nonetheless, to investigate whether it is likely to find a suitable camera in terms in weight and size, a few candidates have been considered.

Generally, there are a much wider range of light-weight silicon cameras available than InGaAs cameras, but as already mentioned, Si sensors are not able to detect the wavelengths that are relevant for this application. Most of the InGaAs cameras that are available are made for scientific applications, and are heavy and power demanding because of cooling. There are however a few exceptions, and the specifications for the three most relevant cameras that were found are enclosed in Appendix D, in addition to the datasheet for a suitable detector.

The three cameras that have been considered are XSW-640 from Xenics, SU640HSX-1.7RT from Goodrich and a SWIR camera developed by Optigo Systems. They all have suitable size, weight and resolution, but have some disadvantages. The Xenics camera seems promising when it comes to size, weight and power consumption, but the specifications are preliminary and might change. According to Xenics, the camera will not be commercially available until the last quarter of 2012. The specifications for the Goodrich camera are also preliminary, and due to export restrictions it is probably impossible to buy it from outside the USA. The Optigo camera is promising with respect to power consumption and operating temperature, but is not commercially available at the moment.

A sensor from Hamamatsu has also been considered. Buying a sensor and integrating it can provide a tailor-made solution, but require a lot of work and expertise. Additionally, the resolution of this sensor is a bit low (128×128 px), but other sensors should also be considered if it is decided that one should try to build a camera.

Since no suitable and readily available camera has been found so far, the exact properties of the camera that will be used remains unknown. The conclusion so far is that it should be possible to find a commercially off-the-shelf camera with suitable weight and size, but the candidates found so far all have their issues.

Chapter 4

Image Enhancement

Noise can be a big problem for infrared sensors, as indicated in the previous chapter. To have images with sufficient signal-to-noise ratio is important for the interpretation of the image, but also vital for the performance of any image compression algorithm. It is assumed that a long integration time will be necessary to get a sufficient DSNR, and that the background signal of the detector must be removed somehow before compression. The combination of high speed and long integration time may introduce blur in the image, but if the integration time is made shorter, the DSNR will get worse, which can be crucial for the image quality when the signal is weak.

To enable development and simulation of image enhancement and compression algorithms, some simple test images will be generated according to the assumed properties of the satellite orbit, the camera and the gravity waves phenomenon. How the background signal and noise from the detector can be removed will also be discussed. In order to investigate how the speed of the satellite will affect the quality of the image, a model of the motion blur degradation is presented along with simulations of the blur with synthetic test images in MATLAB. There are several solutions that aim to give a better trade-off between DSNR and motion blur, as briefly mentioned in Section 2.4. Two different post-processing strategies will be discussed in this chapter; restoration of motion blur by deconvolution and image averaging with motion compensation. The performance of the two will be investigated through simulations, and compared in order to decide which one is the most feasible for the NUTS application.

Some of the parameters regarding camera, satellite and gravity waves that were found in Chapter 3 are repeated in Table 4.1.

Table 4.1: Assumed satellite and camera parameters:

Image coverage	im_{cov}	300 km
Number of pixels in detector	N_{px}	256×256
Mean GW wavelength	$\lambda_{\text{GW}_{\text{mean}}}$	26 km
Min. GW wavelength	$\lambda_{\text{GW}_{\text{min}}}$	15 km
Speed w.r.t. OH layer	V'	7.16 km/s

4.1 Signal-to-Noise Ratio Metrics for Image Processing

Depending on the application, there are many different definitions of SNR, as already mentioned. The Detector Signal-to-Noise Ratio was defined in Section 3.2.1, but from a signal processing point-of-view, there are other definitions of SNR that are more practical. The definitions are stated with respect to a one-dimensional discrete signal $s(n)$, but are easily extended to images.

In signal and image processing, SNR is usually given as a power ratio. One of the most common definitions is

$$\text{SNR}_{\text{image}} = \frac{\sigma_{\text{s}}^2}{\sigma_{\text{n}}^2} \tag{4.1}$$

where σ_s^2 is the variance of the pure and noiseless source signal, and σ_n^2 is the variance of the noise, as stated for instance in [26].

In the context of image enhancement and compression, the *Mean Squared Error* (MSE), defined as $\text{MSE} = \frac{1}{N}\sum_{i=0}^{N-1}(s(n) - \hat{s}(n))^2$, is often used to measure the restored or compressed image's resemblance to the original. This can also be stated as an SNR metric, as given in [26]:

$$\text{SNR}_{\text{MSE}} = \frac{\sigma_{\text{s}}^2}{\text{MSE}} \tag{4.2}$$

When it comes to measurement of visual pleasantness, SNR generally does a rather poor job. An image can look terrible and have a better SNR than an image that looks good to a human eye. It is however hard to incorporate the complexity of visual perception in a simple SNR metric. The *Peak-to-peak Signal to Noise Ratio* (PSNR) is often used to evaluate the quality of compressed images, and can be defined as in [26]:

$$\text{PSNR} = \frac{\text{s}_{\text{peak}}^2}{\text{MSE}}. \tag{4.3}$$

where s_{peak}^2 is the peak-to-peak value of s

4.2 Synthetic Test Images and Noise Removal

When working with algorithms for image enhancement and compression, it is useful to know something about what the images will look like. An attempt is therefore made to summarize the information that is available so far, and come up with some examples of synthetic test images with suitable parameters.

4.2.1 Assumptions

Ground based observations of gravity waves have provided some information about their structure and nature. As already metioned in Section 2.3, the wavelengths have been measured to have a range of 15-40 km with a mean of 26 km. A wavelength of 15 km is assumed as the worst-case scenario when it comes to requirements for resolution and quality.

The gravity wave patterns in the airglow are assumed to closely approximate sine waves, with amplitudes of about 5-10% of the radiation intensity level. As discussed in Section 3.2.6, the background radiation from Earth will most likely be blocked by water vapour, and the aurora can be considered as nearly invisible in the infrared. Of course there may show up other sources of radiation that has not been considered, but these will probably have a small effect.

When it comes to signal strength, it is hard to find any information about the radiation intensity from the OH airglow. It is also beyond the scope of this report to calculate the resulting number of photons that will hit the lens and the detector. It is therefore hard to say anything about the expected intensity of the signal. The digital output of the camera should preferably have a fine quantization, between 12 and 16 bit per pixel seems common. However, images with 8 bits per pixels will be used for the coming implementations and simulations, for speed and simplicity.

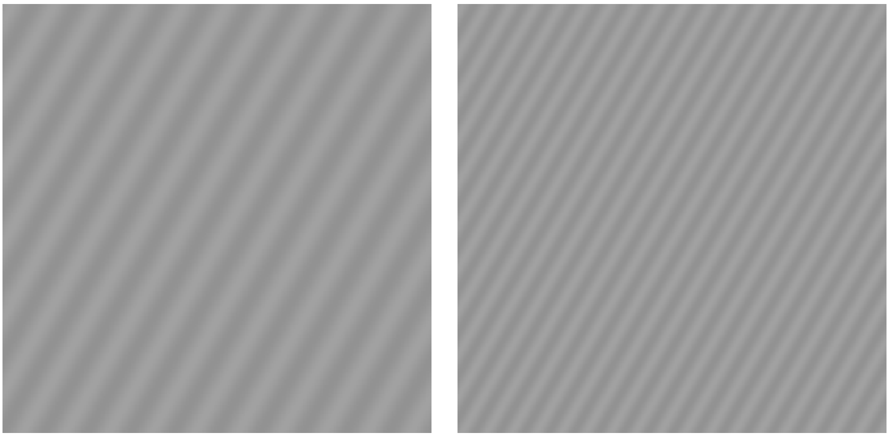
The different noise sources in InGaAs detectors have been treated thoroughly in Section 3.2.3, and this provides knowledge of how the detector noise will affect the images and how it can be removed. But since the intensity of the signal remains unknown, it is hard to give any values of the DSNR. However, different scenarios can be investigated by assuming high or low DSNR.

4.2.2 Synthetic Test Images

Based on the assumptions above, simple sine test images without noise were generated in MATLAB. The mean was set arbitrarily at 0.6 of the intensity range, which corresponds to a pixel value of 153 for an 8-bit image. The amplitude was set to 5 percent of the mean to mimic the intensity variations in the airglow. The frequency of the sine was determined by the desired GW wavelength. The angle can be chosen arbitrarily, since this will vary when the satellite rotates. Test images with frequencies corresponding to the mean and minimum gravity wave wavelengths are shown in Figure 4.1. The minimum wavelength corresponds to the worst-case scenario for many of the coming simulations, and will therefore be used most often.

4.2.3 Background Subtraction

The image in Figure 4.3(a) was synthesized in MATLAB to illustrate the impact of the background signal of the detector. It is composed by a background offset



(a) (b)

Figure 4.1: Synthetic test images without noise. (a): A sine with 11.5 cycles/image, corresponding to the mean GW wavelength (26 km). (b): A sine with 20 cycles/image, corresponding to the minimum GW wavelength (15 km)

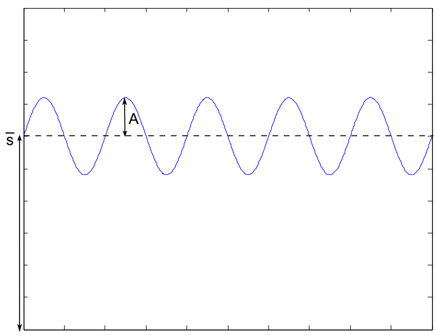


Figure 4.2: Illustration of a sine signal with mean value $\bar{s}$ and amplitude A. $A \propto \bar{s}$

level, fixed pattern noise and thermal noise according to the results in Table 3.1 for an exposure time of 1 second and a temperature of 0 degrees. The numbers where however scaled such that offset level of the sine test image from Figure 4.1(a) is twice the level of the background offset. The resulting summation of the background signal and the sine image is shown in Figure 4.3(c). It is seen that the mean level of the signal is very high, and that the sine pattern is almost drowned in noise.

Fortunately, most of the background signal in the detector is relatively stationary, and can therefore be removed by background subtraction. If the background offset level and the fixed pattern noise is subtracted from the image in Figure 4.3(c), only the small thermal noise component remains. This is not even visible in the image, and is therefore not shown.

The experiment in Section 3.2.5 did however show that the background offset level and the standard deviation of the fixed pattern noise is highly dependent on temperature. The background image can easily be measured by taking a picture with closed shutter and correct exposure time prior to the recording of an image or image series, assuming that the temperature will stay constant for a short period of time.

4.2.4 Synthetic Test Images with Noise

Since most of the disturbing detector background can be removed by a simple subtraction operation, it is assumed that the noise of the image will be dominated by dark noise and photon shot noise. The Detector Signal-to-Noise Ratio can then be expressed as in (3.10). However, due to all the uncertainties regarding signal strength, the factor K that was introduced in (3.11) will be used to describe the DSNR in the following discussion.

In order to generate noisy test images that corresponds to specific values of the normalised DSNR, the noise must be scaled according to the “signal” of the test image. By rearranging (3.11), the standard deviation of the noise is given by

$$\sigma_n = \frac{\bar{s}}{K \cdot \sqrt{t_{int}}} \tag{4.4}$$

where K is the SNR factor from Section 3.2.4, and $\bar{s}$ is the average value of the "signal" in the test image. The resulting test images for different values of K is shown in Figure 4.4.

It could be convenient to know the relation between the SNR metrics DSNR and $\text{SNR}_{\text{image}}$ for the test images. For a sine signal like the one in Figure 4.2, with an amplitude proportional to the offset level of the sine

$$\sigma_s = \frac{A}{\sqrt{2}} = \frac{r \cdot \bar{s}}{\sqrt{2}} \tag{4.5}$$

where r is the proportionality factor. $r = 0.05$ will be assumed for the gravity wave images in accordance with previous discussions.

Inserting (4.5) and (4.4) into (4.1) results in the following relation between the two SNR metrics for the signal in Figure 4.2:

$$\mathrm{SNR_{image}} = \frac{\mathrm{r}^2}{2} \cdot \mathrm{K}^2 \cdot \mathrm{t_{int}} = 0.00125 \cdot \mathrm{DSNR}^2 \tag{4.6}$$

The value of K will however remain unknown until it has been decided on which camera to use.

4.2.5 Noise Removal

Integration time and image averaging

As already indicated in Section 3.2.4, the DSNR can be increased by applying a longer integration time. Figure 4.5 shows test images generated in a similar manner as Figure 4.4, but for varying integration time. Figure 4.6 shows what values for $\mathrm{SNR_{image}}$ that can be obtained for different values of the SNR factor K by varying the integration time.

The effect shown in Figure 4.6 and Figure 4.5 can also be obtained by adding several images of short exposure time, as further discussed in Section 4.5.

Mean filtering

If there still are some noise left in the image after background subtraction and image averaging, one could apply an arithmetic or geometric mean filter, as discussed in [25]. Mean filters have a lowpass effect that smooths the image, and works best on random noise like Gaussian and uniformly distributed noise. This type of filter will introduce blur, and the size of the blur kernel must be chosen carefully.

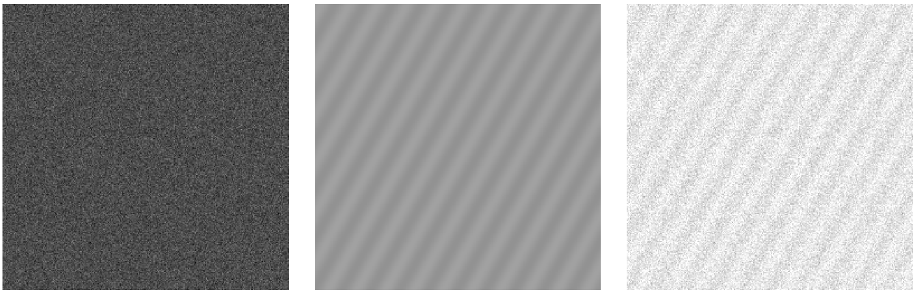
Median filtering

Median filters are non-linear filters of the so-called order-statistic type. As discussed in [25], a median filter replaces a pixel value with the median of its neighbourhood. This often gives a less blurry result than mean filters, and is particularly good for removing so-called salt-and-pepper noise.

A median filter could be applied to remove noise from defective pixels that appear as white or black spots in the image.

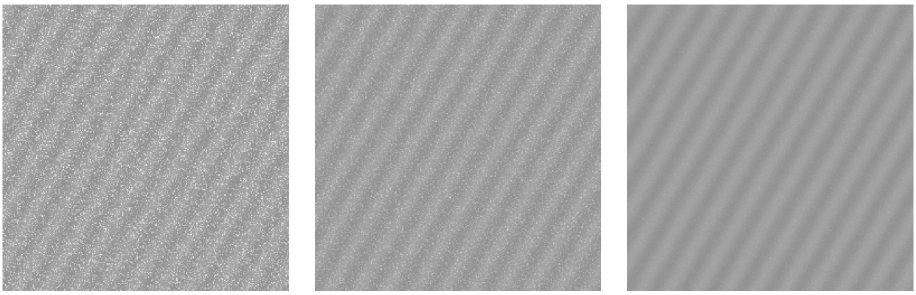
4.3 Motion Blur in Images

Motion blur is caused by the combination of long exposure time and movement of the camera while the image is taken. Photons originating from one particular point of the target is spread over several pixels, which causes the characteristic blurry lines. Whether the motion is caused by linear motion of the camera, a shaking hand or a moving object within the image, changes the properties of the blur. For the linear motion case, the blur is usually considered spatially invariant, but this will not hold if different parts of the image move with different speeds or directions.



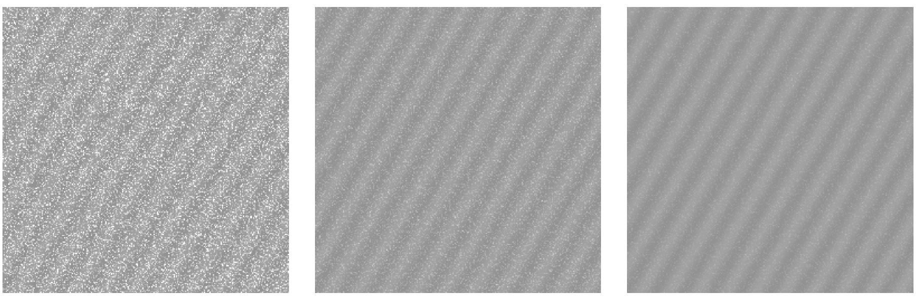
(a) (b) (c)

Figure 4.3: An example of what detector background offset and noise can look like. The values are according to the experimental results, and the mean value of the sine is set to twice the level of the background offset. (a): Detector background (offset, fixed pattern noise and dark noise), (b): Sine signal (without photon noise) (c): Detector background + sine signal



(a) K = 5 (b) K=10 (c) K=50

Figure 4.4: Test images illustrating image quality for different values of the SNR factor.



(a) integration time = 0.1 s (b) integration time = 1 s (c) integration time = 5 s

Figure 4.5: Test images illustrating image quality for different integration times. $K = 10$ for all three images.

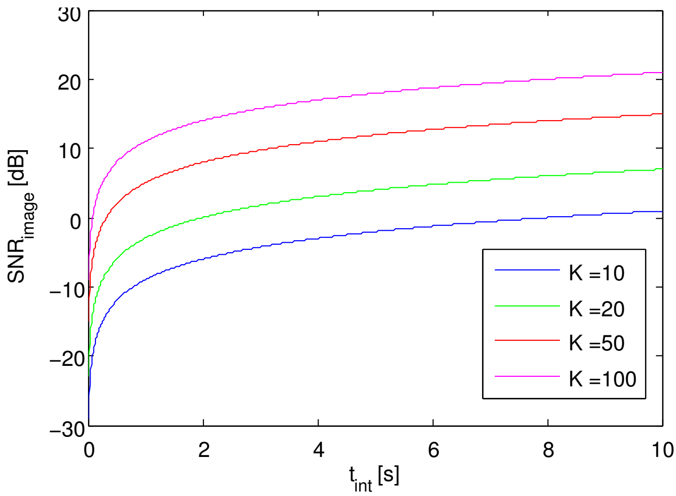


Figure 4.6: An illustration of how the SNR varies with integration time.

4.3.1 Image Degradation Model

Generally, if the degradation of an image is linear and spatially invariant, it can be described in the spatial domain as a convolution [25]:

$$g(x, y) = f(x, y) * h(x, y), \tag{4.7}$$

where $g(x, y)$ is the degraded image, $f(x, y)$ the original image, $h(x, y)$ is the degradation function, often called the *Point Spread Function* (PSF), and $*$ is the convolution operator. To account for additive noise, this model can be extended with a noise term $n(x, y)$:

$$g(x, y) = f(x, y) * h(x, y) + n(x, y) \tag{4.8}$$

which has the following frequency domain equivalent:

$$G(f_x, f_y) = F(f_x, f_y)H(f_x, f_y) + N(f_x, f_y) \tag{4.9}$$

where the capital letters are the corresponding Fourier transforms of the terms in (4.8).

4.3.2 Modelling the Blur Filter

It can be useful to find a good model of the motion blur filter in order to do simulations of motion blur for different speeds and exposure times. Additionally, the restoration problem can usually be solved more exactly when the degradation function is known.

If the motion causing the blur is linear, e.g. no acceleration, the degraded image can be expressed as given in [25]

$$g(x, y) = \int_{0}^{t_{\text{int}}} f(x - x_0(t), y - y_0(t))\,\mathrm{d}t \tag{4.10}$$

given a uniform linear motion

$$x_0(t) = \frac{a}{t_{\text{int}}} \cdot t = v_{\text{im}} \cdot t \tag{4.11}$$

$$y_0(t) = \frac{b}{t_{\text{int}}} \cdot t = v_{\text{im}} \cdot t \tag{4.12}$$

Where $x_0(t)$ and $y_0(t)$ are the time-varying motion components, t_{int} is the exposure time, v_{im} is the image velocity and a and b are the displacement in the x and y directions at time t. As shown in [25], this results in the following frequency domain degradation function:

$$H(f_x, f_y) = \frac{t_{\text{int}}}{\pi(f_x a + f_y b)} \sin(\pi(f_x a + f_y b))e^{-j\pi(f_x a + f_y b)} \tag{4.13}$$

For horizontal blur, this corresponds to a one-dimensional rectangular filter of length a in the spatial domain. When discretized, it can be modelled as a uniform one-dimensional array of length a in MATLAB. The low-pass nature of (4.13) results in attenuation for high frequencies, which is the reason for the blurry effect.

The frequency response of the blur filter depends on the speed of the image, which was discussed and calculated in Section 3.3. Assuming the parameters in Table 4.1, results in a shift of 2.4% of the image per second, i.e. $v_{\text{im}} = 0.024 = 2.4\%$. The blur filter frequency response also depends on the integration time, as shown in the plot of the amplitude of the one-dimensional blur filter in Figure 4.7. The position of the zeros in $H(f_x, f_y)$ depends on the displacements a and b; for a large displacement the first zero will occur at a lower spatial frequency. A plot of the first-zero-frequency vs. integration time is shown in Figure 4.8(a), and Figure 4.8(b) shows the same relation for the corresponding spatial wavelengths as seen from the satellite.

4.3.3 Motion Blur Simulations

As indicated above, the impact of motion blur in an image will depend highly on its frequency content in the direction of the blur. The position of the zeros in Figure 4.7 can give an indication of how short the exposure time must be to preserve content up to a certain frequency, but it is hard to know how much attenuation in $H(f_x, f_y)$ that can be allowed before the blur becomes slightly visible, and for which attenuation the information in the image is lost. Therefore, MATLAB simulations were performed on simple test images in order to investigate the possible motion blur effect in images taken from the satellite with different integration times. The simulations were based on filtering in the spatial domain, according to (4.7).

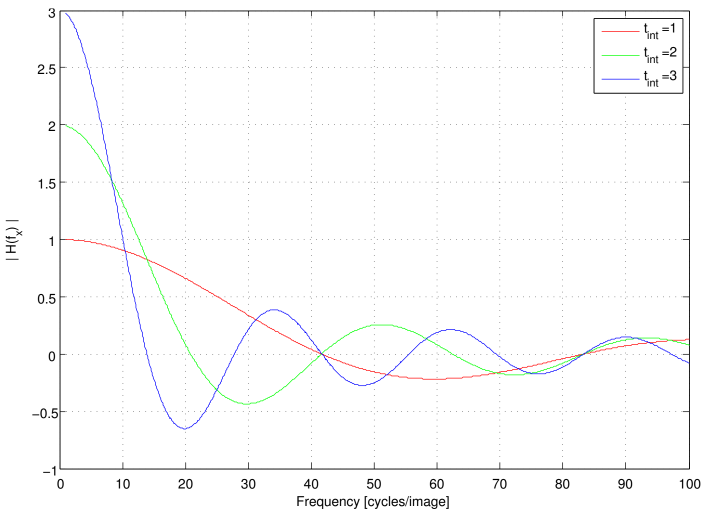
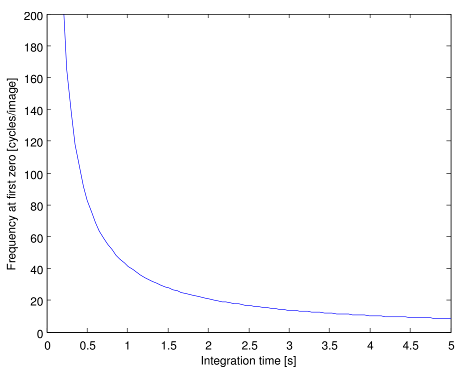
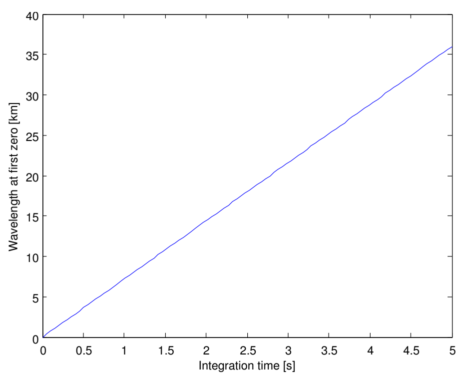


Figure 4.7: Frequency responses of one-dimensional motion blur filters for different integration times. (The curves corresponds to integration times of 1, 2 and 3 seconds and $v_{\mathrm{im}} = 0.024$)



(a)



(b)

Figure 4.8: Positions of the first zero in the blur filter frequency response, for increasing blur lengths corresponding to increasing integration times for the parameters in Table 4.1. (a) shows the zeros for spatial frequencies measured in cycles per image, while (b) shows the same relation translated into wavelengths seen from the satellite, measured in kilometers.

First of all, the spatial motion blur filter had to be estimated for different exposure times. For simplicity, the motion was assumed to be constant and horizontal with respect to the image. As mentioned above, this should correspond to a motion blur filter consisting of a one-dimensional uniform array. The length of the array, in the following called *blur length*, is equal to the number of pixels the image moves during the exposure time. To make the calculations independent on resolution, this shift was measured as a ratio of the image width instead of pixels, just like v_{im}. Once again $v_{\mathrm{im}} = 0.024$ was assumed.

The synthetic motion blur filter was created by means of a built-in MATLAB function named `fspecial()`, using the `'motion'` option. This returns a filter which approximates the linear motion of the camera, given a blur length and orientation of the motion. For the case of horizontal motion, `fspecial()` returns a vector with uniform elements that sums to one, and length equal to the input blur length.

Simple sinusoidal test images with increasing frequencies were generated in order to illustrate the impact of the blur filter for different spatial frequencies. The images shown in the simulation results are separated into equally big sections with frequencies of 5, 10, 15, 20, 25 and 30 cycles/image (see Figure 4.9).

The filtering operation was performed with `imfilter()`, with the blur filter and the test image as inputs, creating synthetic motion blur in the image. This should give a fairly good approximation to linear motion blur, except from close to the edges of the image: Blur that is generated by real motion should also depend on content beyond the field of view of the image due to the movement. The edges were therefore cut after filtering.

Simulation results

Figure 4.9 shows the test images with synthetic blur corresponding to different integration times, plotted together with the frequency response of the blur filters and the original image for comparison. For an exposure of 1 second, a weak blur occur for the highest frequencies, but the sine patterns are fully visible. However, if the exposure time is increased to 2 seconds, the image section with a frequency of 20 is completely wiped out, because this is very close to a zero in the blur filter frequency response. It is also interesting to observe that the higher frequencies seems less distorted, but a closer comparison with the original image reveals that the sine pattern actually is inverted. This is due to the undershoot of the amplitude response, which gives a phase inversion. For an exposure time of 3 seconds, this effect occurs for the section with frequency of 20, while there is a zero blurring the section with $f = 15$. For such a long exposure, all the frequencies are blurred to some extent, except the lowest one.

The test image in Figure 4.1(b) representing the minimum gravity wave wavelength has a frequency of 20 cycles per image. As indicated by the results in Figure 4.9, an exposure time of 2 seconds would wipe out this frequency completely. Assuming that this is the highest frequency one needs to observe in the image, it seem that the integration time must be limited to around 1 second, depending on the required quality. The image quality for different integration times are further discussed in Section 4.6.1.

For now it is concluded that motion blur will occur for integration times of more than 1 second, and that action must be taken if longer integration times are required due to a weak signal.

4.4 Restoration of Motion Blurred Images by Deconvolution

One way of solving the joint motion blur and noise problem, is to apply a long integration time to get a stronger signal, and allow some blur in the image which is removed by post processing.

Motion blur can be considered as filtering with a spatially invariant degradation function, as already discussed in Section 4.3. Restoration of the image after this type of degradation can generally be done by some sort of *deconvolution*. As indicated by the name, this approach aims at reversing the filtering in (4.8). It can be divided into two cases: *blind* and *non-blind deconvolution*. If the PSF is known, restoration of the image can be done by non-blind deconvolution, which is generally a much easier problem than blind deconvolution.

The general goal of image restoration is to find an estimate of the original image, denoted as $\hat{f}$ in the spatial domain and $\hat{F}$ in the frequency domain, which is as close to the original as possible.

4.4.1 Blind Deconvolution

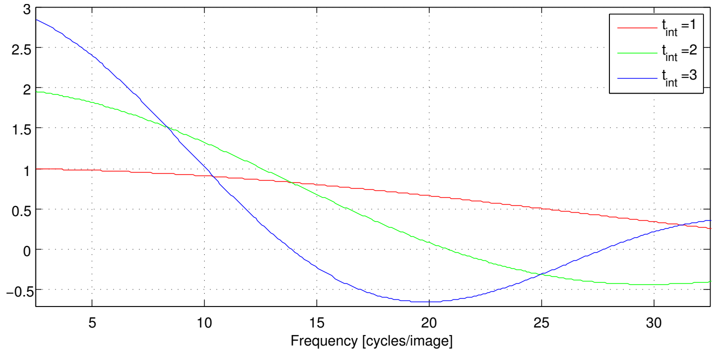
In the general case both the original image, degradation function and the noise term are unknown, and the ill-posed problem of the blind deconvolution has to be solved. A countless number of different algorithms have been developed to solve this problem, often optimized to work for a certain kind of image degradation assuming some statistical properties of the image or the degradation function. Some examples are the concept proposed in [27], which utilizes blur in different directions, and the algorithm presented in [28], which unifies blur filter estimation and image restoration into one algorithm. The common problem for all these approaches is that they do not work as well when the motion blur is too strong.

4.4.2 Non-blind Deconvolution

Fortunately, the problem of non-blind deconvolution is less ill-posed than blind deconvolution. In this case, only the original image and the noise term remain unknown, and the degradation function is known or estimated before the deconvolution.

The most obvious approach to this problem is direct inverse filtering:

$$\hat{F}(f_x, f_y) = \frac{G(f_x, f_y)}{H(f_x, f_y)} \tag{4.14}$$



(a) Frequency response blur filter

(b) Original image

(c) Exposure time 1 s

(d) Exposure time 2 s

(e) Exposure time 3 s

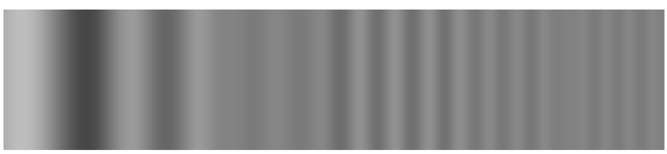


Figure 4.9: Simulation of motion blur for different exposure times: (a) shows the frequency response of the motion blur filter, b) shows the original test image, c)- e) shows the result from blurring with filters corresponding to different exposure times. The frequency axis in the plot corresponds to the placement of the frequencies in the test image.

By inserting (4.9), this can be further expressed as

$$\hat{F}(f_x, f_y) = F(f_x, f_y) + \frac{N(f_x, f_y)}{H(f_x, f_y)} \tag{4.15}$$

Because of the noise term, the resulting estimate would not be exact. The main problem with this approach in practice is that $H(f_x, f_y)$ usually has zeros and low values at higher frequencies. The noise term will then be amplified and dominate the estimate, and the result is usually not as intended.

One way to avoid the problems of the inverse filtering approach is to replace it by a Wiener filter [25] or an iterative restoration method such as the Lucy-Richardson Algorithm [29]. The Wiener filter requires knowledge or estimation of the statistical properties of the noise and the original image. This is not required when using the Lucy-Richardson Algorithm, which iteratively restores the image based on knowledge of the degradation function only.

4.4.3 The Lucy-Richardson (LR) Algorithm

The Lucy-Richardson algorithm is an iterative non-linear restoration algorithm. In contrast to the direct inverse filtering and Wiener filtering, this algorithm utilizes the known degradation function in an update equation and restores the image iteratively instead of performing the deconvolution directly. The algorithm is based on a maximum-likelihood formulation, which gives the resulting update function:

$$\hat{f}_{k+1}(x, y) = \hat{f}_k(x + y) \left[h(-x, -y) * \frac{g(x, y)}{h(x, y) * \hat{f}_k(x, y)} \right] \tag{4.16}$$

Where $\hat{f}_k$ is the estimate of the original image for the k-th iteration, as given in [29]. A derivation can be found in [30].

The main disadvantage of the *Lucy-Richardson* (LR) algorithm and non-linear methods in general is that they are more computationally intensive than direct methods such as the Wiener filter and direct deconvolution. But whether this is a problem or not depends on the application. If the image is small and there is a lot of computational power available, this is usually not an issue.

It can also be hard to determine whether (4.16) has converged or not, and the number of iterations often has to be decided through manual observation of the result. If the algorithm is stopped too early, the result may still be blurry, but too many iterations is a waste of computational power and may even cause additional noise and artefacts as shown in [29].

The LR algorithm usually gives a better result than the Wiener filter in the case of blur removal when there is little or no noise present. But there are a few issues to cope with, that are discussed in the following sections.

4.4.4 The Boundary Value Problem

Boundary artefacts is a well-known problem for image restoration algorithms based on deconvolution, caused by the limited field of view of the image. This is known

(a) (b) (c)

Figure 4.10: Illustration of typical boundary artefacts, from [31]. (a) The original image with field of view within the white lines. (b) Blurred image padded with mean values along the boundaries. (c) Restored image with boundary artefacts.

as the *boundary value problem*[1]. It causes disturbing ripples along the edges in the restored image that increases in strength and propagation as the size of the blur filter increases.

The limited size of the image causes discontinuity problems both in the spatial domain and the frequency domain, as discussed in [31]. In the spatial domain, the convolution operator depends on information outside the image, and is therefore dependent on some kind of extrapolation beyond the field of view of the image. A simple zero padding would provide wrong information and cause discontinuities at the border. The problem in the frequency domain is that the Discrete Fourier Transform assumes periodicity of the data. A simple periodic extension in 2D implies a splice between both the right-hand and left-hand sides and the top and bottom of the image, which causes discontinuities across the borders unless the image is uniform along the edges. These artificial discontinuities creates Gibbs oscillations [32], or ripples, in the restored image as shown in Figure 4.10.

Several methods have been developed to deal with this problem, with varying complexities and results. One of the simplest is the MATLAB function `edgetaper()`, which blurs the edges of the image to avoid discontinuities. This works well as long as the degradation is not too strong, but some information along the edges of the image is lost. Simple linear interpolation between the edges of the image may also be done, which preserves the image content and gives continuity across the borders. Another approach is to extend the image in such a way that the continuity at the image borders is preserved, as in the case of *reflective boundary conditions* [33]. This can be further extended to keeping the gradient continuous, as for *anti-reflective boundary conditions* [34]. Especially the latter gives very good results, but the problem with these algorithms is that they result in a large input image for the restoration algorithm, which makes it computationally expensive. A similar boundary condition approach based on *tile-generation* was discussed in [31], aiming at reduced complexity.

[1]Used in accordance with [31] and others. Not to be confused with the boundary value problem of partial differential equations.

4.4.5 Other Artefacts

Ringing artefacts also tend to show up along strong edges within the image. Even small errors in the estimated degradation function may cause such artefacts, because it gets mixed with the noise and is thereby modelled incorrectly in the LR algorithm, as discussed in [28].

Many algorithms are proposed to deal with this problem as well, usually involving complicated mathematical regularisation problems. One example is the algorithm in [28], which proposes a unified probabilistic model of both blind and non-blind deconvolution to handle errors in the estimated degradation function. Global and local priors based on image statistics are proposed in order to preserve edges and at the same time keep the constant regions smooth. It shows exceptional results even for blind deconvolution, but the derivation is mathematically extensive.

4.4.6 Sensitivity to Deviations in Speed and Orientation

The LR algorithm requires a good estimation of the degradation function to work properly. Even small errors in the estimated blur filter may cause artefacts in the restored image, as discussed in [28]. The estimation of the blur filter in Section 4.3 assumes a straight movement at a constant and known speed.

The length of the blur filter is computed by means of the integration time and image speed, which will depend on distance to the airglow layer and the speed of the satellite. Ideally, the speed of the satellite should be known and constant if the orbital altitude is known. However, orbital perturbations may cause variations in the speed. Additionally, inaccuracy of the ADCS system may cause the pointing direction of the camera to vary, which also might influence the image velocity.

The estimated blur filter might also be erroneous due to wrong information about the direction of the motion blur. The direction of blur within the image will depend on the orientation of the satellite with respect to the direction of velocity. This is assumed to be known with a certain precision, but it will depend on the final specifications of the ADCS of the satellite.

For now, the inaccuracies of the ADCS system is assumed to be negligible. It is hard to foresee how much orbital perturbations will affect the speed, but a worst-case deviation of $\pm 10\%$ is assumed for the remainder of this report.

4.5 Image Averaging With Motion Compensation

Removing noise generally seems easier than reverting motion blur in images. From this perspective, an alternative and completely different approach for solving the motion blur and noise problem was investigated. The idea is to obtain several images with short integration times and low DSNR to avoid motion blur, and combine these afterwards to get a longer integration time and better SNR in total.

The concept of combining images in order to increase the quality is already utilized in many different applications, for instance in *High Dynamic Range imaging* (HDR), which is used more and more in digital cameras to enhance the dynamic range of an image. The approach is to take a sequence of differently exposed images of the same scene, and combine them afterwards to obtain one image with improved dynamic range. The combination of several independently exposed images will also lead to an increase in SNR, if the detector noise is independent and Gaussian, as discussed in [35]. Another example is so-called *image stacking*, which is a method that has already been used for decades by astrophotographers. The SNR is increased by averaging several images obtained of a constant scene.

Combination of images is widely used to improve the SNR, but if the scene is moving, some sort of motion compensation must be applied between the images to avoid motion blur. In this section, the concept of image averaging and motion compensation will first be discussed in general, and then a discussion of the feasibility for our application follows.

4.5.1 Noise Reduction by Image Averaging

Assume that $\{f_i(x,y)\}$ is a set of N noisy images formed by addition of an underlying noiseless image $g(x,y)$ and different realisations of independently distributed Gaussian noise $n_\mathrm{i}(x,y)$ with zero mean and variance σ_n^2.

$$f_\mathrm{i}(x,y) = g(x,y) + n_\mathrm{i}(x,y) \tag{4.17}$$

Using the metric in (4.1), the SNR of f_i can be expressed

$$\mathrm{SNR}_{\mathrm{image,f}} = \frac{\sigma_\mathrm{g}^2}{\sigma_\mathrm{n}^2} \tag{4.18}$$

The combined image $\bar{f}(x,y)$ is formed by averaging the images in $\{f_i(x,y)\}$:

$$\bar{f}(x,y) = \frac{1}{N}\sum_{i=1}^{N} f_\mathrm{i}(x,y) \tag{4.19}$$

As discussed in [25], the averaging operation reduces the variance of the noise with a factor of N, and the SNR of the combined image is therefore N times higher than the SNR of each single image:

$$\mathrm{SNR}_{\mathrm{image},\bar{\mathrm{f}}} = \frac{\sigma_\mathrm{g}^2}{\frac{1}{\mathrm{N}}\sigma_\mathrm{n}^2} = \mathrm{N}\cdot\mathrm{SNR}_{\mathrm{image,f}} \tag{4.20}$$

It is however important to assure that the underlying noiseless image is the same for the whole set. This means that misalignments between these images must be taken care of before averaging. This is further discussed in Section 4.5.2.

Noise reduction by image averaging has exactly the same effect on the SNR as increasing the integration time like discussed in Section 3.2.4. The SNR will be proportional to the total integration time $t_\mathrm{tot} = N \cdot t_\mathrm{int}$ for the whole series of images. Taking one picture with an exposure of 1 s should therefore have the same effect as taking ten pictures with exposures of 0.1 s.

4.5.2 Motion Compensation

If the scene changes during exposure, either due to movement of the camera or movement of objects in the scene, care must be taken when combining the frames in order to avoid a blurry result.

In the case of a uniform and known movement, for instance a camera moving with constant and known speed, a fairly simple compensation can be done by shifting the images $\delta = v_{\text{im}} \cdot t_{\text{frame}}$ pixels, where t_{frame} is the inverse of the frame rate. The required shift is not necessarily an integer number of pixels, and a suitable interpolation scheme should therefore be chosen, as will be discussed in Section 4.5.3.

On the other hand, if the movement is unknown or not uniform, motion compensation can be a very complicated problem. Motion estimation is further discussed in Section 5.3.

4.5.3 Interpolation Methods

In general, interpolation is the process of estimating an unknown value from known points. This is widely used in image processing, for instance in connection with resizing, rotation or translation of an image, or any other operation where the new pixel values does not fit the old sampling grid any more. According to the sampling theorem [32], any band-limited signal can be perfectly recovered as long as it is sampled with sufficiently high sampling frequency. In practice, the quality of the resulting image depends on the interpolation algorithm.

Three different interpolation methods are introduced in [25]: Nearest neighbour interpolation, bilinear interpolation and bicubic interpolation. The main concept of all three is described below.

Nearest Neighbour Interpolation is the simplest form of interpolation. The intensities of the pixels in the new image is simply set as the intensity of the nearest neighbour in the original sampling grid. The disadvantage of this approach is that artefacts occur along edges and in areas with fine detail.

Bilinear Interpolation is a bit more complicated, but gives a much better result. With this approach, intensities of the pixels in the new image are given by a weighted average of the four nearest neighbours. We want to find $f(x, y)$, which is the intensity value in a point (x, y) on the new sampling grid. This is obtained by

$$f(x, y) = a + bx + cy + dxy \tag{4.21}$$

where the coefficients can be found by solving a set of four linear equations containing the values of the four nearest neighbours of (x, y).

Bilinear interpolation gives a significant improvement compared to the nearest neighbour approach with a modest increase of complexity, but tends to blur sharp edges.

Bicubic Interpolation is an even better, but a bit more computationally expensive interpolation algorithm than bilinear interpolation. In this approach, the sixteen nearest neighbours of each pixel is taken into account to find the pixel values in the new image. Instead of linear interpolation, this algorithm is based on fitting cubic polynomials subsequently in the x- and y-direction. The advantage of bicubic interpolation is that it produces less blurred edges than bilinear interpolation.

There also exists algorithms which take more surrounding pixels into consideration, and are therefore also much more computationally expensive. Bicubic interpolation is however considered as a good combination of processing time and output quality, and is the standard algorithm used in for instance Adobe Photoshop.

4.5.4 Feasibility for NUTS

There are several advantages of applying image averaging with motion compensation on the images from the infrared camera. Instead of one image with long exposure, a series of subsequent images with short exposure times can be obtained. Shorter exposure time for each image means that the images have a lower DSNR, but is without motion blur. It also prevents the detector to saturate due to dark current. The short exposure time results in low SNR, but is efficiently improved by averaging the images, which results in a longer exposure time in total. The images must however be motion compensated properly before averaging to avoid the motion blur. This can be done by simply shifting the images as long as the estimate of the speed of the satellite is good enough.

In practice, such an image sequence can be obtained with a video camera with low frame rate. Most commercially available infrared cameras already have the ability of recording video, and many of them do not have the option for sufficiently long integration times to obtain still images with satisfactory DSNR. It should however be noted that video cameras always have a required reset time between each frame, and the integration time will therefore be shorter than the frame duration.

Assuming that the motion compensation align the images perfectly, and that no other disturbances change the images, the graphs in Figure 4.6 can indicate the expected improvement in SNR for different values of the total integration time. Increasing the total integration time from 1 to 5 seconds will for instance give an improvement of almost 7 dB.

A simple implementation followed by simulations and further discussion is done in Section 4.6.2.

4.6 Implementation and Simulation of Image Enhancement Algorithms

The algorithms discussed in the previous sections were implemented in MATLAB in order to perform simulations. Some simplifying assumptions were made when implementing the algorithms, in order to demonstrate proof-of-concept and evaluate their performance.

4.6.1 Restoration of Motion Blurred Images by Deconvolution

In order to demonstrate removal of motion blur from images by means of deconvolution, a simplified algorithm was implemented in MATLAB and applied to some simple test images.

Implementation in MATLAB

The built-in LR algorithm in MATLAB, `deconvlucy()`, was used to restore the motion blurred images. For most of the simulations, the degradation filters causing the motion blur was assumed to be perfectly estimated. The exact same blur filters as the ones generated during the motion blur simulations were therefore used as input to the `deconvlucy()` function. The number of iterations was set to 15 after a quick inspection of the resulting images, but was not optimized further.

As discussed in Section 4.4, the boundary value problem should be appropriately handled to avoid boundary artefacts in the restored image. For simplicity, the built-in solution in MATLAB, `edgetaper()`, was used for the simulations. It outputs an image with blurred edges according to a user-specified PSF, for instance a Gaussian filter. The size and standard deviation of the PSF should be chosen according to the extent of blur in the motion blurred image.

A test script was developed in order to vary the integration time and type of test images used in the simulations, as well plot functions for image sets to make it easier to compare results. Only synthetic images were used in the simulations, to have a better control on the image content.

Performance for Varying Exposure Time

To investigate the LR algorithm's capability of restoring different degrees of motion blur, and find out how long exposure times that can be allowed, simulations were performed with synthetically blurred test images corresponding to different integration times in the same manner as in Section 4.3.3. For simplicity, only horizontal motion blur was applied, and ideal conditions (known speed and no noise) was assumed.

First, the restoration algorithm was applied to the blurred images in Figure 4.9 to investigate the performance for different combinations of integration time and image frequencies. The result is shown in Figure 4.11, again plotted together with the original image and the frequency response of the blur filter. For the image with exposure time of 1 second, the slight blur in Figure 4.9 is gone, and the image looks perfectly recovered for all frequencies. For an exposure time of 2 seconds on the other hand, the image is still blurry for the frequency closest to the zero in the frequency response. The same holds for the image with an exposure time of 3 seconds, where most of the image remains blurry. It is interesting to note that the image sections that was inverted in Figure 4.9 has been restored back to their correct phase, and they are the frequencies that looks sharpest in image Figure 4.11 (d) and (e). It may seem like the restoration algorithm has a sharpening effect on these frequencies, which may cause distortions, especially when the sharpness vary a lot between the frequencies, as is the case in this example. These effects may be due to an incorrect number of iterations for the LR algorithm, which may cause artefacts if set too high, as mentioned in Section 4.4.3. A lower number of iterations could however lead to more blur.

Simulations were also performed with the noiseless test image in Figure 4.1(b), with a sine frequency of 20 cycles per image. The worst-case image for horizontal motion blur would be a sine image with vertical wavefronts, but the angle of the waves was set to 10° to make it a bit less regular. As above, the test image was blurred with filteres corresponding to different integration times, and restored afterwards. This was done for many different integration times to evaluate the performance of the algorithm for this particular image. Figure 4.12 shows sections of the blurred and recovered images for three examples of integration time, together with the corresponding section of the original image. This test image has a low contrast in the first place, which makes it even more sensitive to strong blur than the test image in Figure 4.11, but weak blur may be harder to spot. For an exposure time of one second, the recovered image shows a slight improvement from the blurred version, but the wave patterns are clearly visible in both cases. For 1.6 seconds however, the wave pattern starts getting quite dim, but the recovered image still shows a good quality. For 2 seconds, the wave pattern has not been recovered at all.

The cut-off for performance of the restoration algorithm applied to the test image seems to lie somewhere in between 1.6 and 2 second exposure. Simulations were done with other parameters than the ones shown in Figure 4.12, but it was hard to determine a definite cut-off value, since this should depend on the required quality. The quality of the blurred and recovered images could have been measured in many ways, but the SNR-metric in (4.2) was chosen for simplicity. The SNR was computed for blurred and recovered images for various exposure times, using the same test image as in Figure 4.12. The results are shown in Figure 4.13 together with the normalised frequency responses, $|H(f = 20)|/t_{\text{int}}$, for different integration times. The SNR of the recovered image is relatively constant around 25 dB for short integration times, and is actually lower than the SNR for the blurred image for the lowest integration times. From $t_{\text{int}} = 1.7$ s, it drops quickly towards a minimum at $t_{\text{int}} = 2.1$ s. The SNR of the blurred image on the other hand, drops quickly in the beginning and has a steady decrease. An interesting thing to notice is that when the value of the frequency response is zero, as indicated by the dotted line, the SNR of the blurred and the recovered image is equal, i.e. the restoration algorithm has no effect at all. But the SNR of the recovered images rise again for higher frequencies, in accordance with the observations in Figure 4.11.

From simulations with images, SNR = 20 dB seem to be a reasonable quality criterion for this type of images. As seen from Figure 4.13, this results in a maximum integration time of 1.1 seconds without processing, and 1.8 seconds if the restoration algorithm is used. Thus, a longer integration time can be allowed if applying this type of post-processing, but just to some extent.

Performance under non-ideal conditions

Until now, noise-free test images and a perfectly known blur filter has been assumed. Simulations were also performed with noisy images and inaccurate blur lengths to investigate how the algorithm performs under non-ideal conditions.

The whole point of the motion blur restoration algorithm is to increase the integration time in order to get a stronger signal and better SNR with respect to detector noise. But if the signal is very weak in the first place, there might still be some noise present for longer exposure times as well. Simulations were therefore performed with the same synthetic test images as in the previous section, but Gaussian noise was added after the blurring process to imitate detector noise. The restoration algorithm was then applied to the blurry and noisy images. Figure 4.14 shows the results for an exposure time of 1.6 seconds, and noise levels corresponding to SNR factors of $K = 50$ and $K = 100$. Even though these images have a relatively low noise level, it is very visible in the recovered images. It seems like the LR algorithm amplifies the noise a bit.

In order to investigate how sensitive the LR algorithm is to errors in the estimated blur filter, images were restored with blur filters corresponding to another image velocity than the one used to synthesise the motion blur. Figure 4.15 shows the recovered images for deviations of -10% and $+10\%$ in the estimated speed of the satellite, since this is assumed to be the worst-case scenario. It is seen that when the estimated speed is smaller than the actual one, the blur is not fully recovered. However, when the estimated speed is higher, a sharpening effect is seen for some of the frequencies. This may cause distortions in the recovered image. The impact of these effects is smaller for shorter exposure times.

Summary

Under ideal conditions, the algorithm can increase the maximum integration time a little bit, but not as much as hoped for. The main problem is that some detector noise must be expected, and this can get amplified. Since the reason for increasing the integration time in the first place was to get a better SNR, this is not a very satisfactory outcome.

The parameters of the LR has not at all been optimized, but this section was first of all meant as a feasibility study. A better result with respect to noise might be obtained using a Wiener filter instead of the LR algorithm, but it was not investigated further for this case. Due to the zeros of the blur filter, there will often be some information in the image that is impossible to recover for strong blurs, which puts a general limitation to the deconvolution approach.

Since artefacts appear so easily when using the LR algorithm, for instance due to noise, inaccurate blur filters or wrong number of iterations, it should probably not be applied on-board the satellite. However, compressing and sending blurry images to the ground station may not be a good option either. If lossy compression is applied, artefacts caused by the compression algorithm can be amplified in the same manner as the noise.

This type of motion blur removal is probably better suited for removal of weak motion blur to make an image look more visually pleasant, than to extend the allowed integration time for a scientific application.

4.6.2 Image Averaging With Motion Compensation

To demonstrate how image averaging can be applied as a principle to avoid motion blur, a simple version working on a sequence of synthetic test images was implemented in MATLAB.

The image averaging is in itself a simple concept, but in order to test it on an image sequence that is as realistic as possible, the test script that was made had to incorporate parameters regarding the satellite and camera. Since there still is a lot of uncertainty regarding the values of these parameters, it was important to have the option to vary them when performing simulations.

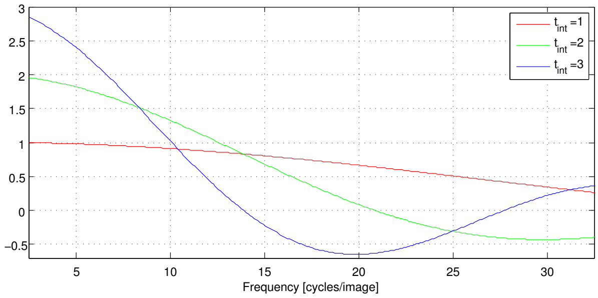
Care should be taken to choose a suitable frame rate of the video; the integration time should be kept short enough to avoid blur completely, but it should also be chosen as long as possible not to waste too much time on resetting the detector, and to increase the signal strength compared to readout noise of the detector.

Implementation and Test Script

As mentioned, a script was developed for simulation and generation of image sequences, or video, for different combinations of satellite and camera parameters. `video_sim` reads parameters from a text file, and generates different sets of *video parameters* (for instance resolution, frame rate and number of frames). Different sets of image parameters (for instance SNR, and amplitude and intensity of the sine wave) for generation of synthetic test images as discussed in Section 4.2.4, are also generated. High resolution images with different parameters are made for use in the generation of the videos.

The synthetic image sequence, or video, is generated in `video_maker()` by sliding a window over a synthetic test image with high resolution, to give a sequence of low-resolution images. Noise is added after the low-resolution image is obtained, in order to make it independent from frame to frame. `video_maker()` also takes a set of video and image parameters as input, in order to provide the desired properties in terms of for instance frame rate, DSNR, gravity wave wavelength and satellite speed.

`video_sim` utilize `video_maker()` to generate different videos according to the parameters that are specified. Since these parameters also are of interest for the functions using the video, the parameter sets are exported to a text file. An example is shown in Appendix E.4.



(a) Frequency response blur filter

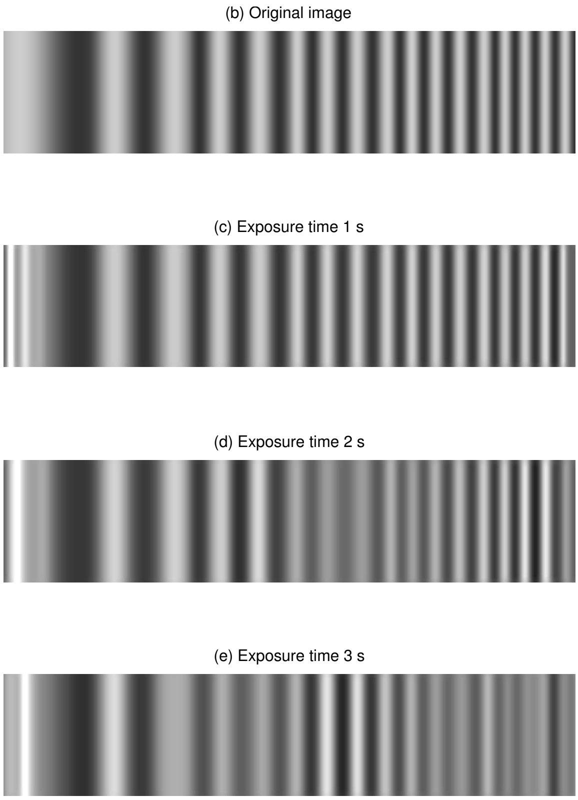


Figure 4.11: Restoration of motion blurred images for different exposure times: (a) shows the frequency response of the motion blur filter, b) shows the original test image, c)- e) shows the recovered versions of images with motion blur corresponding to different exposure times. For comparison, the blurred images can be found in Figure 4.9. The frequency axis in the plot corresponds to the placement of the frequencies in the test image.

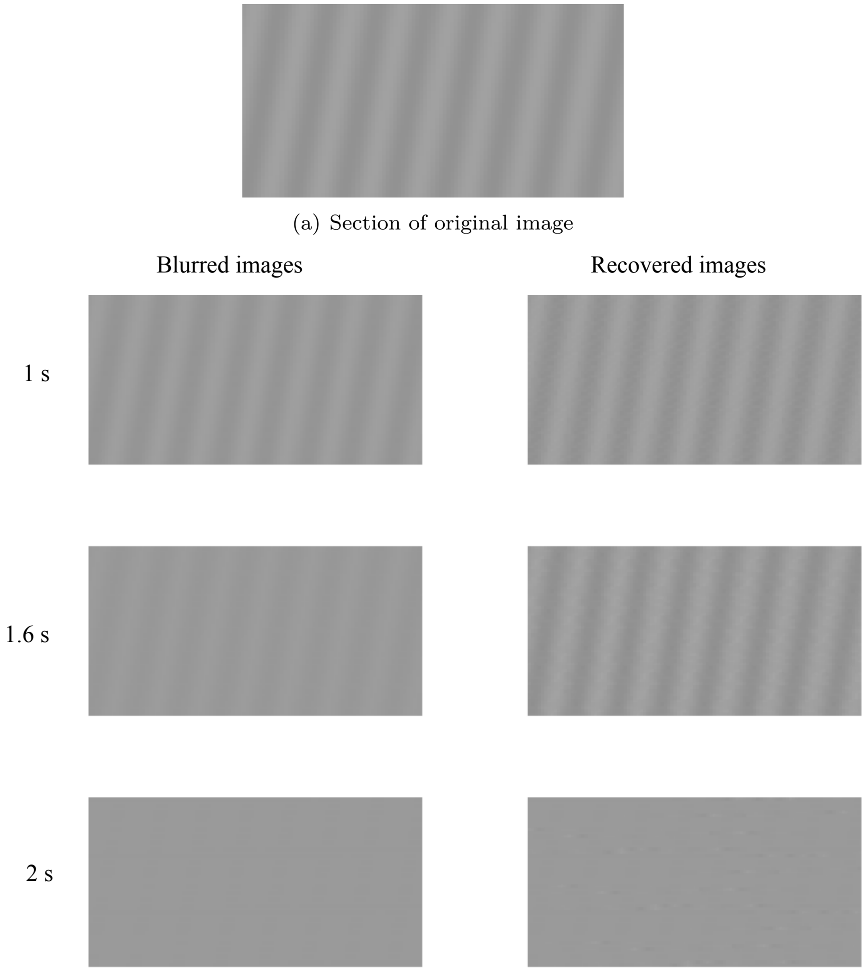


Figure 4.12: Examples of the performance of the deconvolution algorithm for different integration times (for a test image with f = 20 cycles per image)

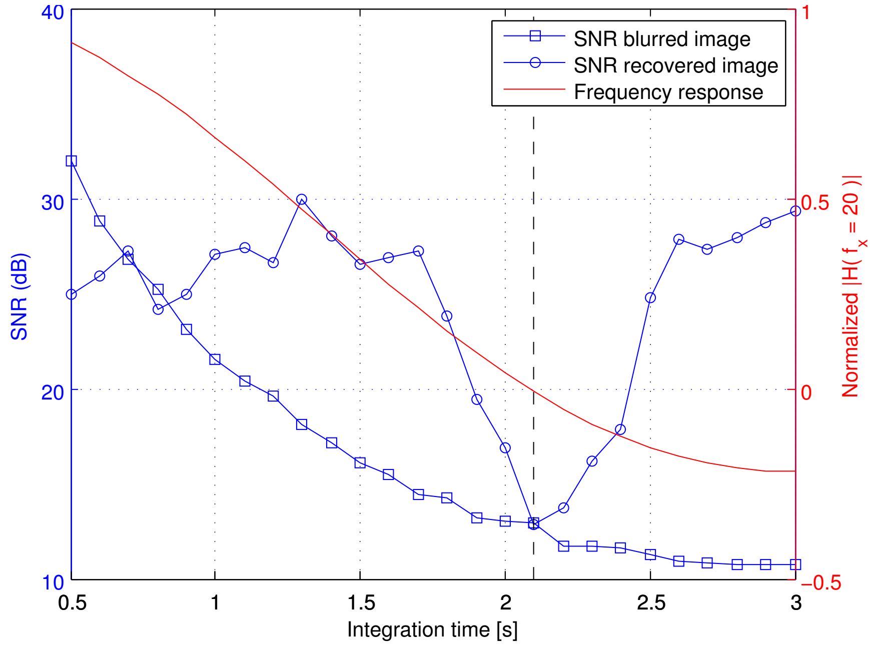


Figure 4.13: SNR (using the metric in (4.2)), for the blurred and recovered images when the test image of 20 cycles per image is used. The blur filter frequency response for f=20 at the different integration times is also shown.

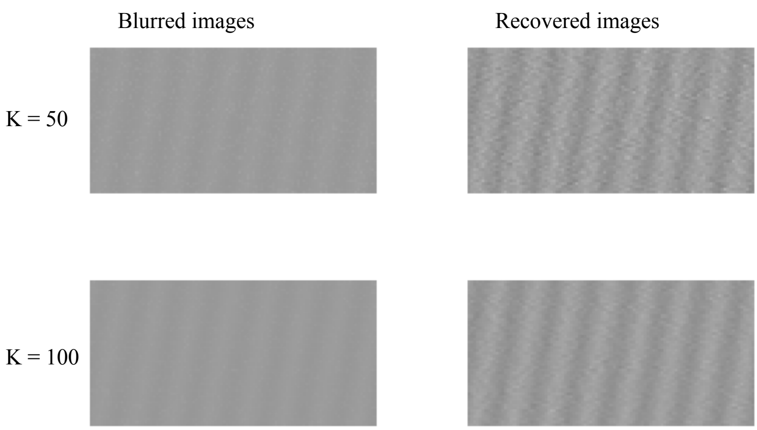


Figure 4.14: Deconvolution with noise for SNR factors of $K = 50$ and $K = 100$, and exposure time of 1.6 seconds.

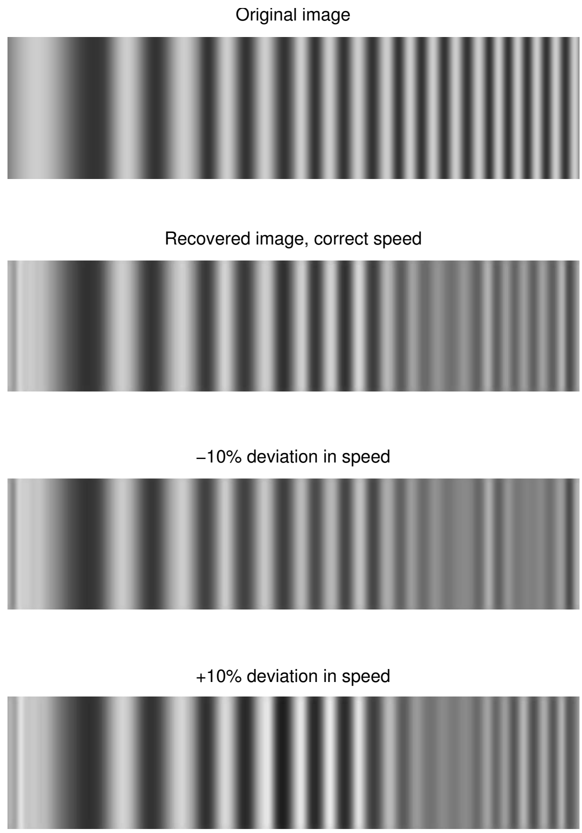


Figure 4.15: Recovered images for errors in the estimated speed. The motion blur corresponds to an integration time of 1.6 seconds.

`video_frame_comb()` performs image averaging and motion compensation on a video sequence. The frames of the video are shifted according to the shift specified in the video parameters, and interpolated onto a common grid with bilinear interpolation (using the built-in functions `meshgrid()` and `interp2()`). Areas along the edges where the images do not overlap are cut away afterwards.

A simple framework for simulations with different videos and parameters is provided with the script `frame_comb_sim`.

Simulations

Various simulations were performed assuming different parameters for the satellite and the camera. However, for the results presented here, the parameters from Table 4.1 and an orbital altitude of 500 km have been used. The frame rate was set to 5 frames per second, with an effective exposure time of 0.19 seconds. Videos with different values of the SNR factor K was generated. An overview of the parameters can be found in Appendix E.4.

Two examples of what the images looked like after averaging is shown in Figure 4.16. The noise is reduced significantly in both cases, but for an SNR factor as low as $K = 5$ many frames had to be averaged to get a reasonable quality, and even after 25 frames (corresponding to a total integration time of 4.75 seconds) the image is still quite noisy. For higher values of K, fewer frames were needed, but the qualities where difficult to compare due to very small differences. Therefore, only two examples with low SNR were given here.

As long as the motion compensation is perfect, the SNR of the images for different exposure times can be computed with (4.1), which was illustrated in Figure 4.6 for different values of K.

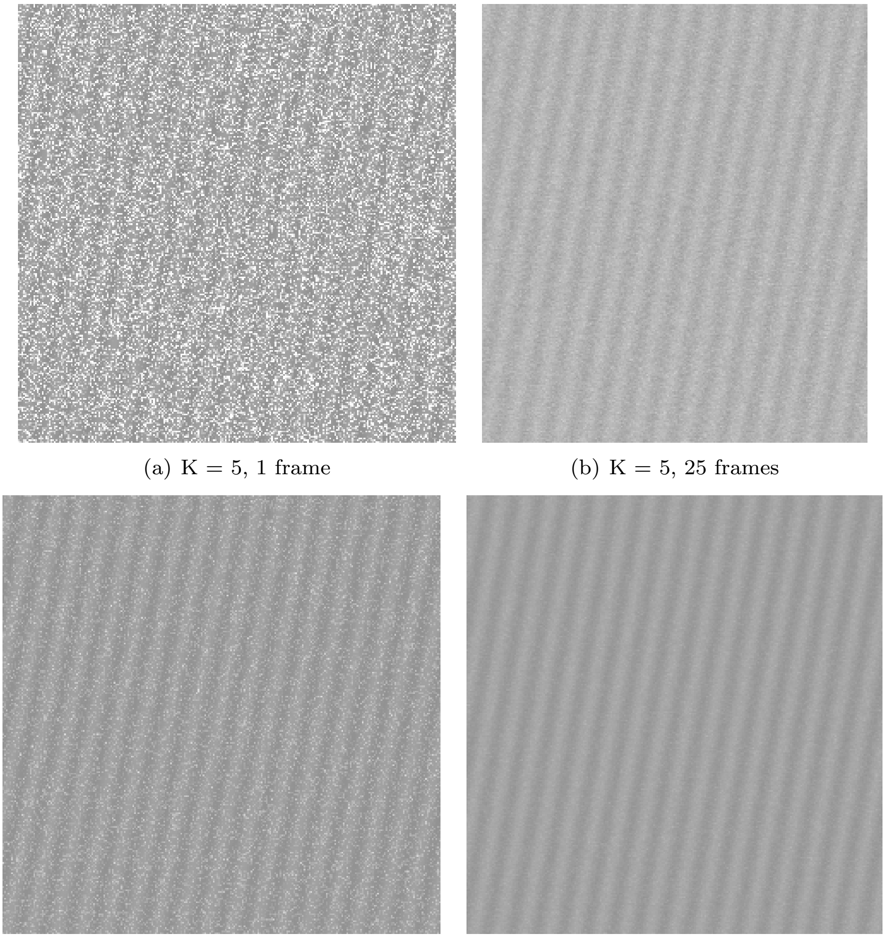
If the speed of the satellite is not known exactly, the motion compensation will apply the wrong shift to the images, and blur could potentially occur. Simulations were performed with a deviation in speed of $\pm 10\%$, and a video duration of 5 seconds (25 frames). No visible degradation could be seen for this case, and images are therefore not included. The effect may however become visible if very long sequences are used.

4.7 Comparison of Algorithms

Two different algorithms have been discussed in this chapter as a possible solution to the motion blur and noise problem; *motion blur restoration by deconvolution* and *image averaging with motion compensation*.

The deconvolution approach may look like a good idea in theory, but simulations showed that it was incapable of restoring images with strong blur, and the integration time could therefore not be extended as much as hoped for. Additionally, the algorithm is very sensitive to errors in the estimated blur filter, which may cause problematic artefacts.

The image averaging and motion compensation approach is much simpler than deconvolution, and not limited by long integration times. The main requirement for



(a) K = 5, 1 frame

(b) K = 5, 25 frames

(c) K = 20, 1 frame

(d) K = 20, 10 frames

Figure 4.16: Simulation of image averaging for SNR factors of $K = 5$ and $K = 20$, $t_{int} = 0.19$, frame rate of 5 frames per second and perfect motion compensation.

this algorithm is that the motion compensation must be accurate enough to avoid blur. Simulations showed that the algorithm is much less sensitive to errors in the speed estimate than the deconvolution approach. Even if the motion compensation is very inaccurate, it only introduces a bit of blur, which is less dramatic than the artefacts created by deconvolution. Additionally, the LR deconvolution algorithm turned out to amplify the noise, while the image averaging approach gives the opposite effect. It is also less computationally extensive than the LR algorithm.

One of the few drawbacks of the image averaging approach is that the signal of each image can become very weak and drown in detector readout noise if the integration time is too short. The video should therefore be obtained with relatively low frame rate. The exposure time should however be kept short enough to be sure to avoid motion blur. A numerical value for the exposure time can first be determined when a camera is obtained, and should be configurable in order to adapt to the signal strength.

Image averaging is clearly the best of the two approaches, and it seems to be a good option for our application. Hopefully, it will provide a good enough SNR, and enable more efficient compression, as will be further discussed in the coming chapter.

Chapter 5

Compression

As mentioned in Section 3.3.1, it is preferred to obtain sequences of overlapping images of the gravity waves rather than single images taken at independent locations. Motion blur and noise should preferably be removed before compression, and it is assumed that the image averaging approach will be applied for this purpose as discussed in Chapter 4. This means that there will be sequences of images with short exposure times, that are combined to give a new sequence of images with higher SNR and lower frame rate, which will be compressed and transmitted to the ground station. This will be further discussed in Section 6.1. The discussion in this chapter will regard compression of the sequence of combined images with low frame rate and good SNR.

As discussed in Section 2.2, an average download capacity of 4.9 Mb/day can be assumed for an orbital altitude of 500 km, but it will be assumed that only half of this capacity is available for the payload data. If a resolution of 256×256 pixels and an output of 8 bits per pixel is assumed for the images from the infrared camera, a sequence of 10 uncompressed images will comprise 5.24 Mb. It will not be possible to download this sequence in one day, maybe not even in two days. A simple compression algorithm would make it possible to download much more images, which is desirable since the lifetime of the satellite is quite limited, and it may take a few attempts to get good images of the phenomenon.

The image sequence can either be regarded as a stream of independent images, or as low rate video, depending on how much the images overlap. If the images are taken with significant overlap, there is a lot of redundant information that can be taken advantage of by video coding. This also assures a continuous scan of the area. Generally, there is more to gain in coding video than still images. The strategy will therefore be to compress the image sequence as low rate video.

This chapter will describe a simple video compression scheme for compression of gravity wave image sequences. The design is not complete, but suggests possible algorithms that can be used in different stages of a typical compression system. The main focus is on the design of a three-dimensional differential coder, but a suitable quantizer, bitcoder and motion compensation scheme is also suggested for the sake of completeness.

First, the underlying principles for compression algorithms are presented. Differential coding is first derived for the one-dimensional case, and then extended to two and three dimensions for use in image and video coding. It continues with a discussion of motion compensation for video compression, followed by discussion of quantizer design for the differential coder and optimal bit coding of the output. In the end, an overview of the complete compression system will be given followed by a presentation of simulations.

5.1 Background

To give a full introduction to compression and information theory is beyond the scope of this report, but an overview of the some fundamental principles is given below to support the discussion in the coming sections. A more detailed treatment can be found in textbooks on information theory and data compression, for instance [26] and [36]. It is assumed that the reader has basic knowledge of statistical signal processing. If not, a brief introduction is given in [37].

5.1.1 Information Theory

Information theory gives the general mathematical foundation that is necessary for a discussion of compression methods. First of all, it defines the bounds on what is theoretically achievable, which can be used as goals for practical algorithms.

In information theory, the *entropy* is used as a measure of average information, or unpredictability, of an uncorrelated source. The entropy for a discrete source is defined as

$$H = -\sum_i p_i \log p_i \tag{5.1}$$

where p_i is the probability of the ith source symbol. According to Shannon's source coding theorem [36], the entropy gives the minimum value for the average number of bits required for representing a source without introducing errors. In practice this implies a lower bound to what can be obtained with lossless compression.

With lossy compression algorithms it is possible to trade lower bit rate against higher distortion, and in this way get bit rates lower than the entropy of the source. The best achievable trade-off between bitrate and distortion is given by the *rate-distortion function*, as discussed in [26]. The rate-distortion function gives the lowest achievable rate R for a given distortion constraint D. It can be difficult to compute it for an arbitrary continuous source, but for a memoryless Gaussian source it is given by

$$R(D) = \begin{cases} \frac{1}{2}\log_2 \frac{\sigma^2}{D} & \text{for} \quad \mathrm{D} < \sigma^2 \\ 0 & \text{for} \quad \mathrm{D} > \sigma^2 \end{cases} \quad \text{[bits]} \tag{5.2}$$

as shown for instance in [26]. It is seen that if a higher D, i.e. more distortion, is allowed, a lower rate can be achieved, but only to a certain limit. If the distortion is higher than the variance of the signal, there is no point in transmitting anything, and the rate is zero. The Gaussian case also gives the upper bound for any continuous memoryless source. This will however only hold for memoryless sources (uncorrelated signals), but it illustrates the principle of rate-distortion. A discussion of rate-distortion for correlated signals is given for instance in [38].

5.1.2 Lossless Compression

Lossless compression schemes does not involve any loss of information, and the original data can therefore be recovered from the compressed data without errors. The main principle in any compression algorithm is to exploit *redundancies* in the data. The two types of redundancies that can be exploited in lossless compression algorithms for digital signals are *coding redundancy* and *spatial/temporal redundancy*.

Coding redundancy implies that the information is represented with more bits than necessary. *Entropy coding* schemes such as *Huffmann coding* and *arithmetic coding* [36] exploit statistical properties of the source in order to get a code rate as close to the entropy as possible.

Spatial and temporal redundancy implies that the samples of the source are correlated in space and/or time. If this is not taken into account, information will be replicated. Various *decomposition* or *decorrelation* schemes can be applied to remove correlation in the signal to ensure a more efficient representation. Examples of such algorithms range from simple run-length coding [26] to differential coding (discussed further in Section 5.2.1) and more complex wavelet transforms [25].

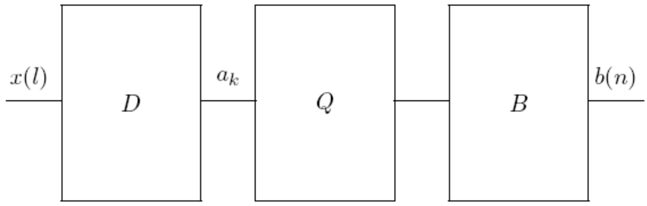
The expressions for the entropy and the rate-distorion function given in this section only hold for uncorrelated sources. A treatment of correlated sources is given for instance in [38].

5.1.3 Lossy Compression

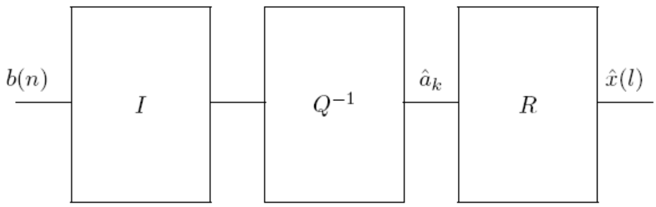
As indicated by the name, lossy compression schemes implies some loss of information. Since errors are introduced in the compression, the original data can not be perfectly recovered as in lossless compression. But a perfect reconstruction is not always necessary, because the data often contain *irrelevant information*, which can be seen as a third type of redundancy. The degree of irrelevancy may be hard to measure, and depends on the application. For compression of photographies and audio, errors in the human perception play an important role. In other cases, some of the information is just not relevant for the intended use of the data. In any case, it is important to bear the application in mind when deciding what types of errors to introduce. The errors introduced are often referred to as a distortion, measured in terms of MSE and SNR.

As already mentioned in Section 5.1.1, the ideal trade-off between rate and distortion in lossy compression is limited by the rate-distortion function. The goal of lossy compression algorithms is therefore to come as close to this bound as possible with a suitable complexity.

As discussed in [38], practical lossy coding schemes can often be divided into



(a) Generic encoder



(b) Generic decoder

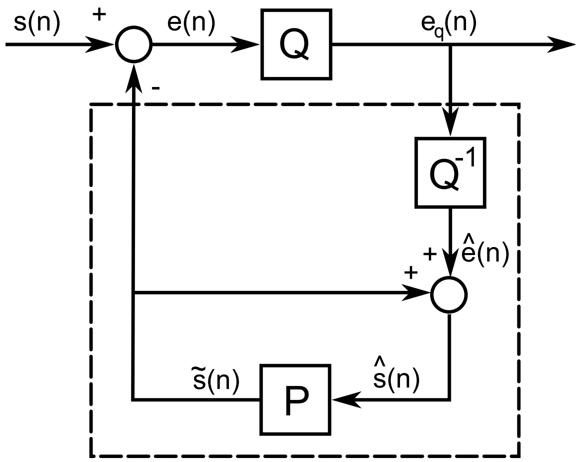
Figure 5.1: Generic encoder and decoder structure. From [38].

three subsystems, as shown in Figure 5.1. On the encoder side, the *decomposition* unit (block D) composes the signal into a set of coefficients by some transform, to enable more efficient scalar quantisation. The coefficients are then quantized in block Q, and the quantization levels are coded to a minimum bit representation in block B. At the decoder, the incoming bitstream is decoded back to quantization levels in unit I, which is mapped to approximations of the coefficients at the inverse quantizer (unit Q^{-1}). The last unit is the reconstruction unit which performs the inverse transform to obtain an approximation of the original signal.

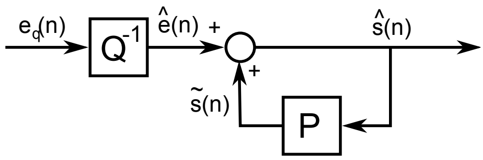
The building blocks in the compression system must usually be co-designed to get the best performance. The optimal properties of the quantizer will for instance depend on the statistical properties of the output from the decomposition unit, but also on the desired format of the final output and how the quantizer levels are coded.

5.2 Differential Coding

Differential coding is a simple compression scheme which is widely used in speech, image and video coding. The basic concept is to encode the difference between samples instead of the sample values. This works as a decomposition of the signal as mentioned in Section 5.1, leading to a smaller variance, and fewer bits are therefore needed to represent the information. The most common algorithm for differential



(a) Encoder



(b) Decoder

Figure 5.2: Block diagram for DPCM. (a) shows the encoder, and (b) shows the decoder.

coding in practice is called *Differential Pulse Code Modulation* (DPCM), which includes a prediction of the samples in order to reduce the variance even further. The principles of DPCM is discussed in for instance [39] and [26]. An introduction to the one-dimensional case with emphasis on linear prediction is given below.

The basic 1D-predictor in Section 5.2.1 can be extended into two and three dimensions and be utilized in image and video coding

5.2.1 Differential Pulse Code Modulation (DPCM)

The basic principle of DPCM is illustrated by the block diagrams in Figure 5.2. The two main building blocks are the quantizer and the predictor denoted by Q and P. Their functionality will be explained in more detail later. The difference between the incoming signal sample $s(n)$ and the predicted sample $\tilde{s}(n)$ is denoted by $e(n)$ and is often called the prediction residual or prediction error. The prediction error is quantized and represented by quantization levels, denoted by $e_q(n)$, which is the output of the DPCM encoder. $e_q(n)$ is also fed into the feedback loop that performs inverse quantization to map it back to an approximation of the prediction error $\hat{e}(n)$, and reconstructs $s(n)$ in order to perform the prediction of the next sample.

As long as there is no quantizer, differential coding can be a lossless coding scheme, but this is often not the case in practice. The quantizer in Figure 5.2(a) introduces errors that will accumulate in the decoding process. This problem is solved by making sure that the encoder and decoder both use the reconstructed signal $\hat{s}(n)$ to perform the prediction. This is done by integrating a decoder in the encoder, as indicated by the dotted box in Figure 5.2(a). The compression will still be lossy, but the error will not accumulate anymore. This is often referred to as closed-loop DPCM. In contrast to the structure in Figure 5.1, closed-loop DPCM performs decomposition and quantization in the same block.

Prediction

The purpose of the predictor in Figure 5.2 is to remove redundant information in the signal by predicting it from previous samples. The predictor should be chosen such that the error is as small as possible, to enable good compression. The variance of the prediction error can be written as

$$\sigma_e^2 = \mathrm{E}\left[e^2(n)\right] = \mathrm{E}\left[\left(s(n) - \tilde{s}(n)\right)^2\right] \tag{5.3}$$

To find the optimal predictor is a very complex problem, but a few assumptions are made to make it possible to estimate it based on statistical properties of the signal:

The first assumption restricts the prediction function to a linear combination of N previous samples

$$\tilde{s}(n) = \sum_{i=1}^{N} a_i s(n-i) \tag{5.4}$$

The second assumption is fine quantization

$$\hat{s}(n) = \hat{e}(n) + \tilde{s}(n) \approx e(n) + \tilde{s}(n) = s(n) \tag{5.5}$$

such that $\hat{s}(n)$ can be used in the prediction instead of $s(n)$

$$\tilde{s}(n) = \sum_{i=1}^{N} a_i \hat{s}(n-i) \tag{5.6}$$

The optimal predictor given these assumptions is found by computing the prediction coefficients $\{a_i\}$ that gives the minimal σ_e^2. Inserting (5.6) into (5.3), we get

$$\sigma_e^2 = \mathrm{E}\left[\left(s(n) - \sum_{i=1}^{N} a_i \hat{s}(n-i)\right)^2\right], \tag{5.7}$$

which is differentiated with respect to each of the a_i and set equal to zero to find the minima. As shown in for instance [39] this results in a set of $N+1$ linear equations called the normal equations:

$$r_s(n) - \sum_{i=1}^{N} a_i r_s(n-1) = \sigma_e^2 \delta(n), \qquad \text{for } n = 0, \ldots, N \tag{5.8}$$

where $r_s(n)$ is the covariance of the input signal.

For higher order predictors, (5.8) can be written on matrix form and solved by linear algebra. In the case of a first-order predictor, the solutions for the prediction coefficient and the minimum prediction error variance is

$$a = \frac{r_s(0)}{r_s(1)} \tag{5.9}$$

$$\sigma_e^2 = r_s(0) - a_1 r_s(1) \tag{5.10}$$

A covariance function that is often assumed for the input signal is

$$r_s(k) = \sigma_s^2 \rho^{|k|} \tag{5.11}$$

where ρ is the one-step correlation coefficient, which gives

$$a = \rho \tag{5.12}$$

$$\sigma_e^2 = \sigma_s^2(1 - \rho^2) \tag{5.13}$$

$$\tag{5.14}$$

Thus, a strong correlation of the input signal gives a low prediction error variance, and efficient compression.

Quantization

The assumption made in (5.5), means that the prediction is performed ignoring the quantization error. To legitimate this assumption, the quantizer should be designed to minimize the quantization noise according to the statistical properties of the resulting prediction error. Quantization design is further discussed in Section 5.4.

5.2.2 Differential Coding in Image Compression

Natural images usually have a strong correlation between adjacent pixels, which is easily exploited with differential coding. DPCM for images works similarly to the encoding and decoding of one-dimensional signals described in Section 5.2.1, but the input samples must come from a scan of the image, usually from top left to bottom right. If samples that are used in the prediction come from the same scan line, the algorithm works similarly to the encoding and decoding of one-dimensional signals. But the DPCM-algorithm can also be extended into a two-dimensional version that also takes the nearest pixels from the scan line above into account, and in this way exploits both the horizontal and vertical correlation. Since only information that is known to the decoder should be used, only the pixels that are already scanned and predicted can be taken into account in the prediction process. The 2D predictor reduces the variance of the prediction error compared to the 1D version.

By extending (5.6) to two dimensions, one gets the expression for a 2D linear predictor as given in [40]:

$$\tilde{f}(x, y) = \sum_{(i,j)\in W} a_{i,j} \hat{f}(x - i, y - j) \tag{5.15}$$

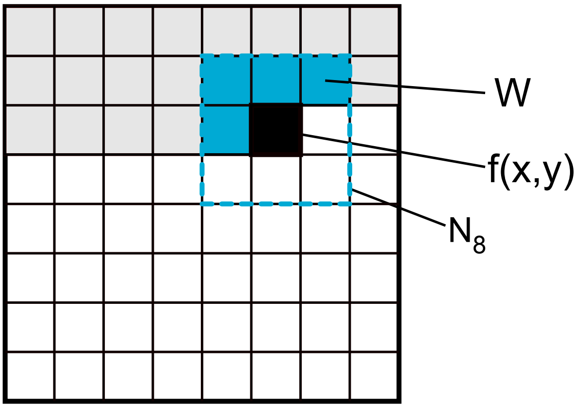


Figure 5.3: Illustration of 2D differential coding. The grey pixels are the ones who are already scanned and predicted, the black pixel is the current pixel $f(x,y)$, the blue region W represents the prediction window and the dotted blue square represents the N_8 neighbourhood of (x,y).

where W is the *prediction window*; a set of pixels that are already scanned, with coordinates relative to the current pixel (x,y). The prediction window can for instance be defined as the four pixels from the N_8 neighbourhood [1] of (x,y), giving $W = \{(0,1),(1,1),(1,0),(-1,1)\}$ as illustrated by the blue pixels in Figure 5.3. This results in the following first-order two-dimensional predictor:

$$\tilde{f}(x,y) = a_{0,1}\hat{f}(x,y-1) + a_{1,1}\hat{f}(x-1,y-1) + a_{1,0}\hat{f}(x-1,y) + a_{-1,1}\hat{f}(x+1,y-1) \tag{5.16}$$

A commonly used statistical model for images is a two-dimensional covariance function separable in the vertical and horizontal directions:

$$\mathrm{Cov}\,[x,y] = r(x,y) = r_v(x)r_h(y) = \sigma_f^2 \rho_v^{|x|}\rho_h^{|y|} \tag{5.17}$$

where ρ_v and ρ_h are the vertical and horizontal one-step correlation coefficients given by $\rho_v = \frac{r(1,0)}{\sigma_f^2}$ and $\rho_h = \frac{r(0,1)}{\sigma_f^2}$.

The optimal prediction coefficients can be found in a similar manner as described in Section 5.2.1. The set of normal equations for the two-dimensional case can then be expressed as

$$r(k,l) - \sum_{(i,j)\in W} a_{i,j} r(k-i,l-j) = \sigma_e^2 \delta(k,l)\,, \qquad \text{for } (k,l) \in W' \tag{5.18}$$

where W' is a set of pixels including the prediction window and (0,0). This set of

[1] The D_8 distance measure between the pixels p and q with coordinates (x,y) and (s,t) is defined as $D_8(q,p) = max(|x-s|,|y-t|)$. The pixels with $D_8 = 1$ from (x,y) is called the N_8 neighbourhood of (x,y). N_8 is indicated with dotted blue lines in Figure 5.3. It is referred to [25] for a general discussion of neighbourhoods and distance measures.

equations can be conveniently expressed on matrix form:

$$\Gamma \mathbf{a} = \boldsymbol{\gamma} \tag{5.19}$$

$$\sigma_e^2 = \sigma_s^2 (1 - \mathbf{a}^T \boldsymbol{\gamma}) \tag{5.20}$$

where the content of Γ, $\boldsymbol{\gamma}$ and $\mathbf{a}$ depends on the prediction window. Assuming the covariance function in (5.17), and the predictor in (5.16) results in the following vectors and matrices:

$$\boldsymbol{\Gamma} = \begin{bmatrix} 1 & \rho_v\rho_h & \rho_h & \rho_h \\ \rho_v\rho_h & 1 & \rho_v & \rho_v\rho_h^2 \\ \rho_h & \rho_v & 1 & \rho_h^2 \\ \rho_h & \rho_v\rho_h^2 & \rho_h^2 & 1 \end{bmatrix} \tag{5.21}$$

$$\mathbf{a} = \begin{bmatrix} a_{1,0} \\ a_{0,1} \\ a_{1,1} \\ a_{1,-1} \end{bmatrix}, \boldsymbol{\gamma} = \begin{bmatrix} \rho_v \\ \rho_h \\ \rho_v\rho_h \\ \rho_v\rho_h \end{bmatrix} \tag{5.22}$$

Solving for $\{a_{i,j}\}$ results in the following optimal prediction coefficients and corresponding prediction error variance:

$$a_{0,1} = \rho_h,\ a_{1,1} = -\rho_h\rho_v,\ a_{1,0} = \rho_v,\ a_{-1,1} = 0 \tag{5.23}$$

$$\sigma_{e_{min}}^2 = \sigma_f^2 (1 - (\rho_v^2 + \rho_h^2 - \rho_v^2\rho_h^2)) \tag{5.24}$$

and (5.16) is thus reduced to a predictor with three coefficients.

5.2.3 Differential Coding in Video Compression

Temporal differential coding is commonly used in international video standards in order to exploit the strong correlation between adjacent frames. Each pixel is then predicted from the corresponding pixels in the previous frames. In order to reduce the variance of the prediction error, some form of motion compensation is usually applied before prediction, as further discussed in Section 5.3.

The most common practice is to apply differential coding between each frame, and then code the prediction error with subband or transform coding as discussed in [26]. However, if the image compression also is done with differential coding, it is advantageous to apply differential coding in both the spatial and temporal dimensions at once to avoid unnecessary quantization errors. This results in a three-dimensional prediction, as further discussed below.

3D differential coding

Although it has probably been done before, no sources describing three-dimensional differential coding was found. The expressions for the predictor and the optimal prediction coefficients were therefore obtained by extending the expressions from

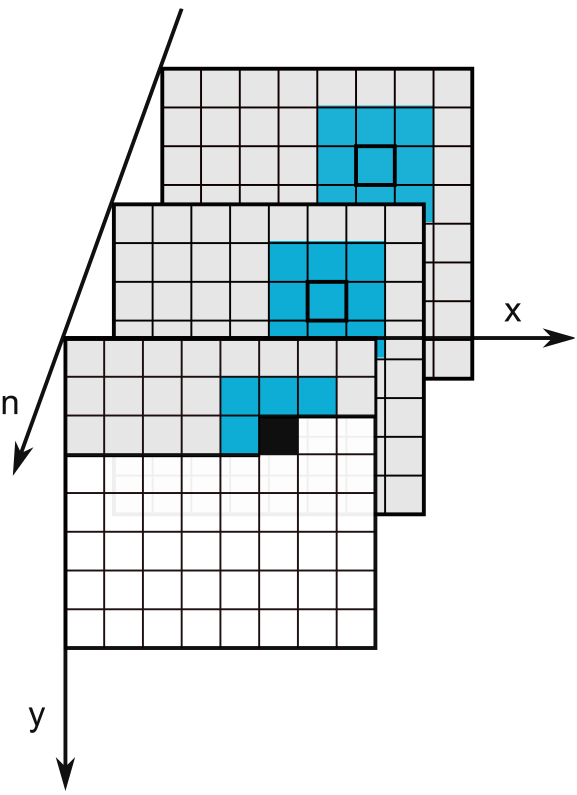


Figure 5.4: Illustration of 3D differential coding of video. The foremost frame is the current one at time n. The black pixel is the current pixel $f(x, y, n)$, that is to be predicted from a three-dimensional set of pixels from both the current and previous frames. An example of such a set is indicated by the blue pixels.

Section 5.2.2 one dimension further. The three-dimensional predictor can then be given by

$$\tilde{f}(x, y, n) = \sum_{(i,j,k) \in W} a_{i,j,k} \hat{f}(x - i, y - j, n - k) \tag{5.25}$$

where W is a three-dimensional prediction window formed by a set of pixels from both current and previous frames. An example of such a set is illustrated with the blue pixels in Figure 5.4, but a smaller prediction window is usually sufficient. One could for instance choose a prediction window based on the three nearest neighbours, giving $W = \{(1, 0, 0), (0, 1, 0), (0, 0, 1)\}$, and get the following predictor:

$$\tilde{f}(x, y, n) = a_{1,0,0} \hat{f}(x - 1, y, n) + a_{0,1,0} \hat{f}(x, y - 1, n) + a_{0,0,1} \hat{f}(x, y, n - 1) \tag{5.26}$$

The normal equations is also easily extended one dimension further in the same manner as in (5.18)

$$r(x, y, n) - \sum_{(i,j,k) \in W} a_{i,j,k} r(x - i, y - j, n - k) = \sigma_e^2 \delta(x, y, n) , \qquad \text{for } (x, y, n) \in W' \tag{5.27}$$

As in the one- and two-dimensional case, a stationary separable covariance is assumed:

$$r(x, y, n) = \sigma_f^2 \rho_v^{|x|} \rho_h^{|y|} \rho_t^{|n|} \tag{5.28}$$

where ρ_t is the one-step correlation coefficient in time. This gives

$$\mathbf{\Gamma} = \begin{bmatrix} 1 & \rho_v \rho_h & \rho_v \rho_t \\ \rho_v \rho_h & 1 & \rho_h \rho_t \\ \rho_v \rho_t & \rho_h \rho_t & 1 \end{bmatrix} \tag{5.29}$$

$$\mathbf{a} = \begin{bmatrix} a_{1,0,0} \\ a_{0,1,0} \\ a_{0,0,1} \end{bmatrix}, \gamma = \begin{bmatrix} \rho_v \\ \rho_h \\ \rho_t \end{bmatrix} \tag{5.30}$$

which can be inserted into (5.19) and (5.20) to find the optimal prediction coefficients $a_{i,j,k}$, and the prediction error σ_e^2. Solving this with linear algebra does unfortunately not result in an expression as simple and intuitive as for the one- and two-dimensional cases discussed earlier. But it can easily be solved if numerical values for the correlation coefficients are known. Assuming $\rho_v = \rho_h = 0.9$ and $\rho_t = 0.95$ and solving in MATLAB results in

$$a_{1,0,0} = 0.25, \quad a_{0,1,0} = 0.25, \quad a_{0,0,1} = 0.50 \tag{5.31}$$

Estimation of prediction coefficients

Since the underlying statistics of the image sequences is unknown, the optimal prediction coefficients should be calculated with correlation coefficients ρ_v, ρ_h and ρ_t that is estimated from samples in the image sequence. A possible estimator for the covariance of a $(M \times N \times P)$ array is formed by rewriting the two-dimensional estimator in [39]:

$$\hat{r}(x,y,n) = \frac{1}{(M-x)(N-y)(P-n)} \cdot \sum_{x'=1}^{M-x} \sum_{y'=1}^{N-y} \sum_{p'=1}^{P-n} f_0(x',y',n') f_0(x'+x, y'+y, n'+n) \tag{5.32}$$

where $f_0(x,y,n) = f(x,y,n) - \hat{\mu}_f$. This results in the following estimators for the correlation coefficients:

$$\hat{\rho}_v = \frac{\hat{r}(1,0,0)}{\sigma_f^2} = \frac{1}{\sigma_f^2(M-1)NP} \sum_{x'=1}^{M-1} \sum_{y'=1}^{N} \sum_{p'=1}^{P} f_0(x',y',n') f_0(x'+1,y',n') \tag{5.33}$$

$$\hat{\rho}_h = \frac{\hat{r}(0,1,0)}{\sigma_f^2} = \frac{1}{\sigma_f^2 M(N-1)P} \sum_{x'=1}^{M} \sum_{y'=1}^{N-1} \sum_{p'=1}^{P} f_0(x',y',n') f_0(x',y'+1,n') \tag{5.34}$$

$$\hat{\rho}_t = \frac{\hat{r}(0,0,1)}{\sigma_f^2} = \frac{1}{\sigma_f^2 MN(P-1)} \sum_{x'=1}^{M} \sum_{y'=1}^{N} \sum_{p'=1}^{P-1} f_0(x',y',n') f_0(x',y',n'+1) \tag{5.35}$$

These estimators assume stationarity for the whole image series, which may often not be the case in practice. Videos often have a varying correlation between frames; it will be high for slow-varying scenes and low for abrupt motion and scene-changes. To mitigate this problem, it is common to make the differential coding adaptive. If short-time stationarity is assumed, the signal can be partitioned into shorter segments, and the optimal prediction coefficients can then be estimated for each segment. Since the decoder must use the same prediction coefficients as the encoder, these must be sent as side information, which requires extra bits. Additionally, the estimation of the prediction coefficients requires more memory and computational power in the encoder than if fixed prediction coefficients are used. Whether the additional complexity and overhead in an adaptive scheme pays off will depend on the application. If the image statistics are assumed to be relatively stationary and low complexity is important, an algorithm using fixed prediction coefficients is probably sufficient. This is assumed to be the case for the NUTS application.

Anchor Frames and Bidirectional Prediction

One problem with differential coding of video is that the whole encoded image sequence depends on the first frame. This is inconvenient for practical video applications because it makes random access impossible, and it is often useful to split

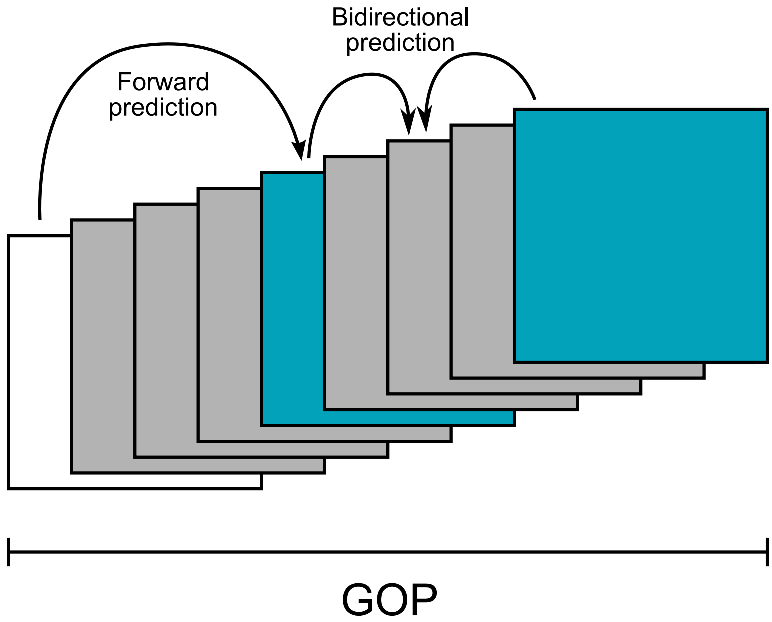


Figure 5.5: Example of a Group of Pictures (GOP) with I- P- and B-frames, illustrating forward and bidirectional prediction. The white frame is an I-frame, the blue ones are P-frames, and the grey ones are B-frames.

the video into shorter sequences to fit it into a packet format for transmission. Additionally, noise from the transmission will propagate through all the differentially coded frames, which makes it necessary to reset the coding now and then. Therefore, it is quite common to make sure that with a certain interval there are frames that are coded without any reference to past frames.

As discussed in [26], three frame types are defined in the MPEG-1 and -2 standards: intracoded I-frames, predictively coded P-frames and bidirectionally predicted B-frames. The different frame types are organized in a *Group of Pictures* (GOP), which is the smallest random access unit in the video sequence. The I-frames enable random access, but have a low compression rate because they are coded without any reference to other frames. The number of I-frames is a trade-off between compression rate and convenience, but there must be at least one I-frame in each GOP. The P-frames are coded predictively from the last P- or I-frame, and has a much better compression efficiency than the I-frames. The B-frames are bidirectionally predicted from the two nearest P- or I-frames, which is even more efficient than forward prediction. P- and I-frames are also referred to as anchor frames. An example of a possible GOP is shown in Figure 5.5.

The general 3D predictor in (5.25) already enables bi-directional prediction, it is just a matter of defining the prediction window W in such a way that it includes pixels from the two nearest anchor frames. For a bidirectional predictor with a prediction window of four pixels, a possible set could be

$$W = \{(x-1, y, n), (x, y-1, n), (x, y, n-k), (x, y, n+l)\}$$

where k and l are the distances to the previous and following anchor frames respectively.

It is important to remember that the prediction only should be based on information known to the decoder, that is, the frames that are already predicted. I- and P-frames must be encoded and decoded first, in order to perform the bidirectional prediction of the B-frames in between. The encoding/decoding order of a GOP is therefore not the same as the display order when B-frames are included.

As discussed in [41], introducing B-frames will often improve the overall compression efficiency, but this depends on among other things the accuracy of the motion compensation. As mentioned, the B-frames have a better compression efficiency than P-frames. Additionally, they are not used for prediction of any other frames, and can therefore tolerate more error since the quantization error does not propagate further. But using B-frames also leads to reduced compression efficiency for the P-frames because it increases the difference between the predicted frame and the reference frame. How many B-frames to insert, and whether they should be used or not, will therefore depend on the application.

5.3 Motion Estimation for Video Coding

The problem with temporal differential coding for video, is that motion corrupts the temporal pixel-by-pixel correlation, which makes the coding less effective. For videos with abrupt motion, differential coding may even lead to an expansion instead of compression. To deal with this problem, it is common to apply some kind of motion compensation, to match corresponding parts of successive frames.

5.3.1 Block Matching

There are various ways to estimate the motion in a sequence of images. The one most commonly used for video compression purposes is called block matching, and is described in detail in [42]. The concept of block matching is to partition a frame into $m \times n$ subblocks, and then search for the best match of each subblock in the previous frame. In the simplest form of block matching, it is common to assume rectangular non-overlapping blocks of a fixed size, and pure translational motion which is uniform within each block.

Block size

For each subblock, the search for the best match is performed within a search window of size $p \times q$ as illustrated in Figure 5.6. A correlation window of size $m \times n$ is slided over positions within the search window as shown in Figure 5.7, to compute the corresponding matching criteria for different offsets. The relative position with the best match gives the displacement vector for that subblock.

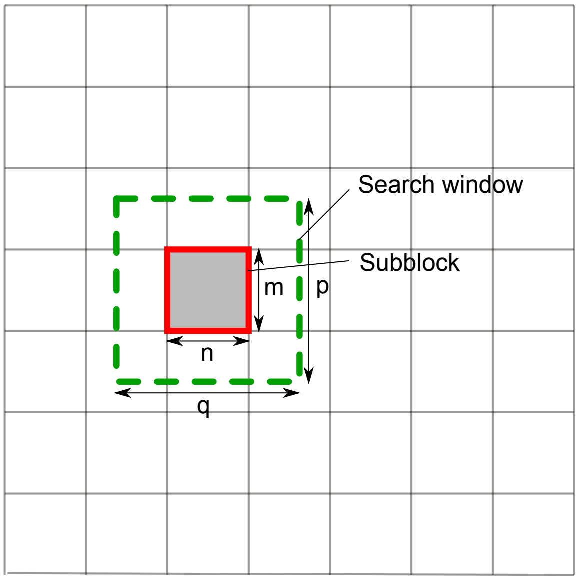


Figure 5.6: Illustration of block matching.

The blocksize affects the precision of the motion estimation, and should be chosen carefully. In order to approximate rotation, zooming or non-uniform motion in the image, the block size needs to be small. However, since small block size leads to more subblocks and motion vectors, this leads to a heavier computational burden and more side information. 16×16 pixels is considered to be a suitable compromise for general video, and is used in many international video standards, for instance MPEG. But it should be noted that the optimal block size depends on resolution, image content and frame rate and should be chosen according to the suitable precision for the application in question.

The size of the search window is chosen according to the maximal displacement in all four directions (d_N, d_E, d_S and d_W in Figure 5.7). The search window should be as small as possible to give fewer computations of the matching criterion, and to prevent matching with similar regions other places in the image.

Matching criteria

The criteria for the *best match* can be defined in several ways. One possible matching criteria is to maximise the correlation between blocks, but this expression is heavy to compute. An other option is to minimize the dissimilarity, or the average error, between the two images. The error between the subblock in the current frame f_k and the shifted correlation window in the previous frame f_{k-1} can be

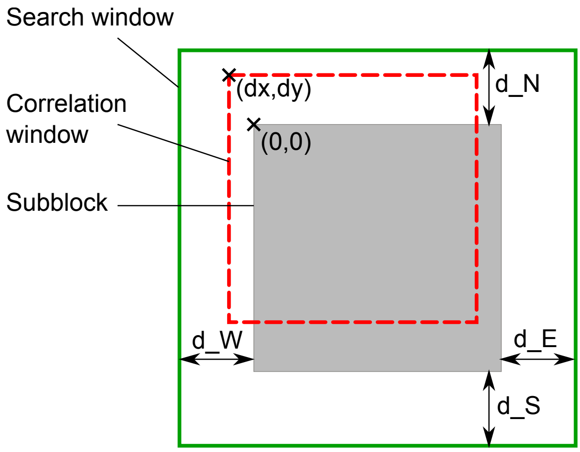


Figure 5.7: Illustration of how the search within a subblock is done to find the best match.

expressed as

$$D(dx, dy) = \frac{1}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} M(f_k(i,j), f_{k-1}(i+dx, j+dy))) \tag{5.36}$$

where $M(u,v)$ is an error metric and $dx \in (-d_W, d_E)$ and $dy \in (-d_N, d_S)$ are the shifts of the correlation window in the x- and y-direction as illustrated in Figure 5.7. Various error metrics have been proposed in literature, among them the MSE and the *Mean Absolute Difference* (MAD) given by

$$M_{\mathrm{MSE}}(u,v) = (u-v)^2 \tag{5.37}$$

and

$$M_{\mathrm{MAD}}(u,v) = |u-v| \tag{5.38}$$

Due to its simple expression, MAD is commonly used. To get an even more computationally efficient error measure, one can also use the *Sum of Absolute Difference* (SAD), which is the same as MAD just without the $1/mn$ factor. The matching criterion is then given by

$$D(dx, dy) = SAD(x,y) = \sum_{i=1}^{m} \sum_{j=1}^{n} |f_k(i,j) - f_{k-1}(i+x, j+y)))| \tag{5.39}$$

Searching procedures

In the search for the correlation window with the best match, the chosen search procedure will affect the computational burden. A full search will result in $(d_W + d_E \times d_N + d_S)$ computations of $D(dx, dy)$ in addition to the same amount of comparison operations to find the minimum, for each subblock. For a frame with 512×512 pixels, partitioned into 32 16×16 subblocks with 32×32 search windows, this leads to 8192 comparison operations and computations of the matching criterion per frame. A full search gives the best accuracy, but alternative search methods have been developed to decrease the computational burden, as discussed in [42]

In order to increase the accuracy of the estimated motion vector, spatial interpolation can be applied to get sub-pixel precision. This will of course lead to more computation, and more bits will be needed for the coding of the motion vector. The opposite operation, subsampling, can also be done to decrease the computational burden at the expense of lower accuracy.

5.4 Quantizer Design

The quantizer module in Figure 5.2 makes DPCM a lossy coding scheme, which means that precision can be traded for a higher compression ratio. The goal of quantizer design is to simultaneously minimize the output rate and the quantization error for a source with given properties. The solutions range from simple scalar uniform quantizers to complex adaptive quantizers and vector quantizers. There will always be a trade-off between performance and complexity, and a simple quantizer combined with entropy coding can give a sufficient result for many applications.

For simplicity, only scalar quantizers will be discussed in this section, with emphasis on the optimization of uniform quantizers for DPCM. To enable this discussion, basic concepts will be treated first. Coding of the output of the quantizer will be discussed later, in Section 5.5. For a complete treatment of quantizers, it is referred to [26].

5.4.1 Quantization

The quantization operation divides the range of the time discrete source $s(n)$ into L *quantization intervals*, I_k, which is bounded by the *decision boundaries* b_k and represented with the corresponding representation levels $\hat{s}_k$, as illustrated in Figure 5.8. The expression for this operation is given by

$$Q[s(n)] = \hat{s}_k, \qquad \text{if} \quad s(n) \in I_k = (b_k, b_{k+1}] \,, \tag{5.40}$$

where $k = 1 \dots L$.

Figure 5.8 shows a *uniform quantizer*, which means that the quantization intervals are of equal lengths, and the representation levels are placed in the middle of each interval. For uniform quantizers, the step size Δ denotes the spacing between the decision boundaries, which is the same as the distance between the representation levels. Generally, the quantization intervals can be of different lengths, and the representation levels can have any position inside the intervals.

Quantizers can be either of the midtread or midrise type. As illustrated in Figure 5.9, the midtread quantizer has a representation level in zero, while the midrise has a decision level at zero. The midtread type is often preferred, to assure

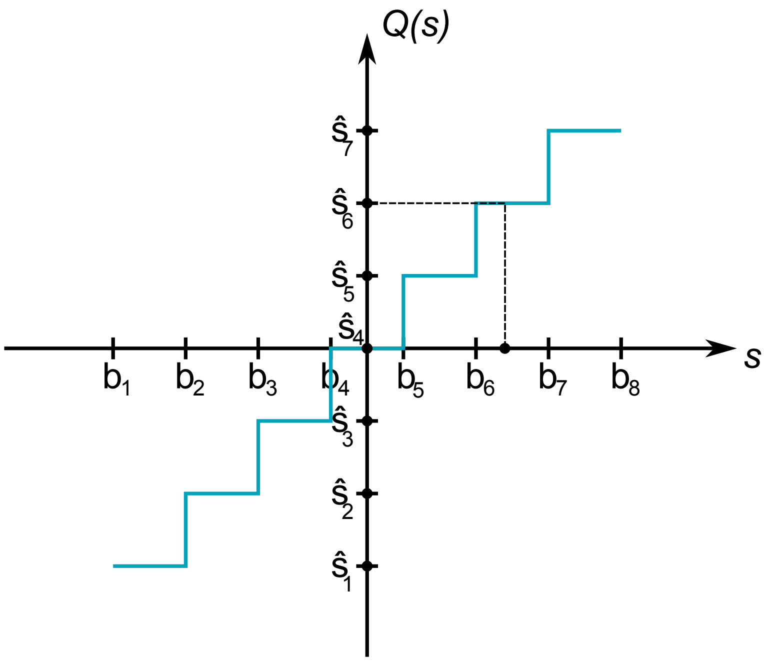


Figure 5.8: Characteristic function of a quantizer with L=7

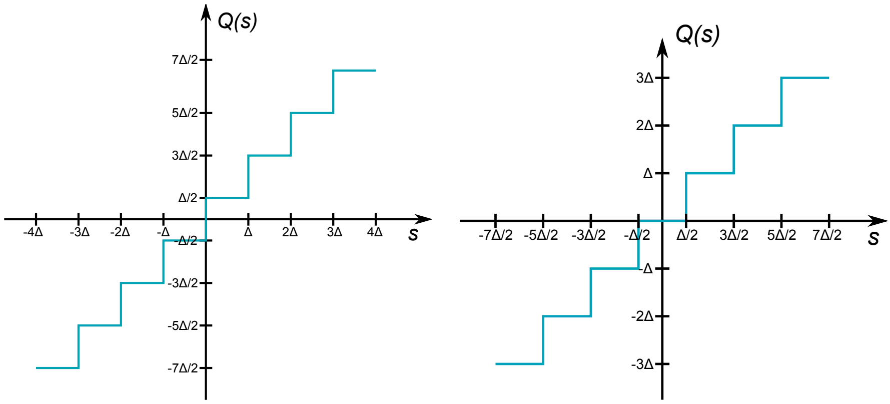
correct representation of zero values in the source, but it results in an odd number of representation levels. If the quantizer output is coded with a constant word length of B bits, codewords will be wasted unless $L = 2^B$. An asymmetric number of representation levels can be assigned to make sure this is fulfilled. For the further discussion, a uniform midtread quantizer is assumed. Expressions for a midrise quantizer can be found in [26], but the discussion is quite similar for the two cases.

5.4.2 Quantization noise

As opposed to sampling, quantization is a non-invertible process which produces inevitable errors. This error can be expressed as the difference between the continuous sample value and the corresponding quantized level:

$$q(s) = s(n) - Q(s(n)) \tag{5.41}$$

$q(s)$ is stochastic in nature due to its dependence of the source, and is referred to as quantization noise because it is often modelled as additive noise on the signal. As shown in Figure 5.10, it is limited by the distance between the representation levels and the decision boundaries for the inner quantization intervals, but if the source is unbounded the error in the two outer intervals can become infinitely large. This type of error is referred to as *overload noise*, while the error for the inner intervals is referred to as *granular noise*.



(a) Midrise uniform quantizer

(b) Midtread uniform quantizer

Figure 5.9: Uniform quantizers of the (a) midrise and (b) midtread type.

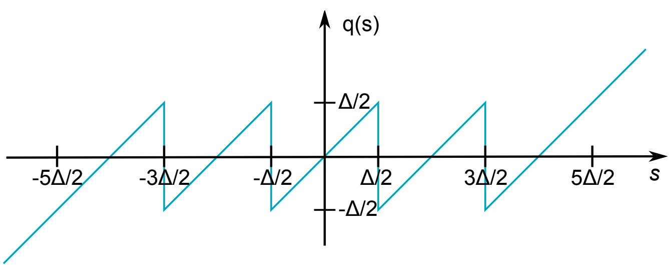


Figure 5.10: Quantization noise for a midtread quantizer with five quantization levels.

The Mean Squared Error of the quantizer, or the variance of the quantization noise, is an important means of measuring the error, and can be computed by

$$\sigma_q^2 = \int_{-\infty}^{\infty} q^2(s) p_s(s)\, \mathrm{d}s \tag{5.42}$$

$$= \sum_{k=1}^{L} \int_{b_k}^{b_{k+1}} (s - \hat{s}_k)^2 p_s(s)\, \mathrm{d}s \tag{5.43}$$

The integral must be partitioned into L terms due to the discontinuities in $q(s)$. Note that $b_1 = -\infty$ and $b_{L+1} = \infty$ for unbounded sources. The *Signal-to-quantization noise ratio* (SQNR) is given by

$$SQNR(dB) = 10 \log_{10} \left(\frac{\sigma_s^2}{\sigma_q^2} \right) \tag{5.44}$$

As seen from (5.43), the variance of the quantization noise depends on the interaction between the quantizer properties (L, b_k and $\hat{s}_k$) and the probability distribution of the source. Various strategies have been proposed to optimize the positions of the decision boundaries and representation levels according to the distribution of the source, in order to minimize the quantization error. Optimal quantizers are usually non-uniform, i.e. b_k and $\hat{s}_k$ are not equally distributed along the range of the quantizer. Finding an optimal quantizer is a very complex problem and depends on good knowledge and/or estimation of the statistical properties of the signal, but can be done for instance by means of the Lloyd-Max algorithm [26].

If fixed-length binary codewords are used to represent the quantizer output, the rate simply depends on the number of quantization intervals:

$$R = \lceil \log_2 L \rceil \quad \text{[bits]} \tag{5.45}$$

But if variable-length codewords are allowed, the rate will depend on the positions of the representation levels and decision boundaries as well as the probability distribution of the source:

$$R = \sum_{k=1}^{L} l_i P(\hat{s}_k) \tag{5.46}$$

$$= \sum_{k=1}^{L} \int_{b_k}^{b_{k+1}} p_S(s) ds \tag{5.47}$$

where $P(\hat{s}_k)$ is the probability of occurrence for the representation level $\hat{s_k}$.

Uniform Quantizer and Uniformly Distributed Source

If a uniformly distributed source is assumed, the resulting expression for the quantization variance is quite simple. $s(n)$ is then limited by a maximum amplitude $s_{\max}$, which gives a finite range of the signal and the following probability distribution

$$p_s(s) = \frac{1}{2s_{\max}} \qquad \text{for} \quad -s_{\max} < s(n) \le s_{\max} \tag{5.48}$$

This results in an optimal step size $\Delta = \frac{2s_{\max}}{L}$. The quantization noise will be uniformly and equally distributed within each quantization interval, and (5.43) can therefore be simplified into

$$\sigma_q^2 = L \int_0^{\Delta} s^2 \frac{1}{2s_{max}} \,\mathrm{d}s \tag{5.49}$$

$$= \frac{1}{\Delta} \int_{-\frac{\Delta}{2}}^{\frac{\Delta}{2}} s^2 \,\mathrm{d}s \tag{5.50}$$

$$= \frac{\Delta^2}{12} \tag{5.51}$$

Uniform Quantizer and Non-uniformly Distributed Source

Non-uniformly distributed sources are generally not bounded, and as already mentioned this results in overload noise. The total quantization noise is given by

$$\sigma_q^2 = \sigma_{\text{granular}}^2 + \sigma_{\text{overload}}^2 \tag{5.52}$$

where

$$\sigma_{\text{granular}}^2 = \sum_{i=2}^{L-1} \int_{d_{i-1}}^{d_i} (s - y_i)^2 p_S(s) \,\mathrm{d}s \tag{5.53}$$

$$\sigma_{\text{overload}}^2 = 2 \int_{d_{L-1}}^{\infty} (s - y_M)^2 p_S(s) \,\mathrm{d}s \ , \tag{5.54}$$

where it is assumed that both the source distribution and the quantizer are symmetric. The impact of the overload noise thus depends on the tails of the probability distribution, as illustrated in Figure 5.11. The granular noise can be approximated with (5.49), if the quantization levels are small compared to the variance of the source. This will not be the case for low-rate applications with coarse quantization.

5.4.3 Design of a Uniform Quantizer for DPCM

For a uniform quantizer, the design parameters are reduced to the number of levels L and the step size Δ. These parameters should be designed according to the probability distribution of the signal to be quantized, to reach a good trade-off between quantization noise and final output rate. How this should be done depends on the application.

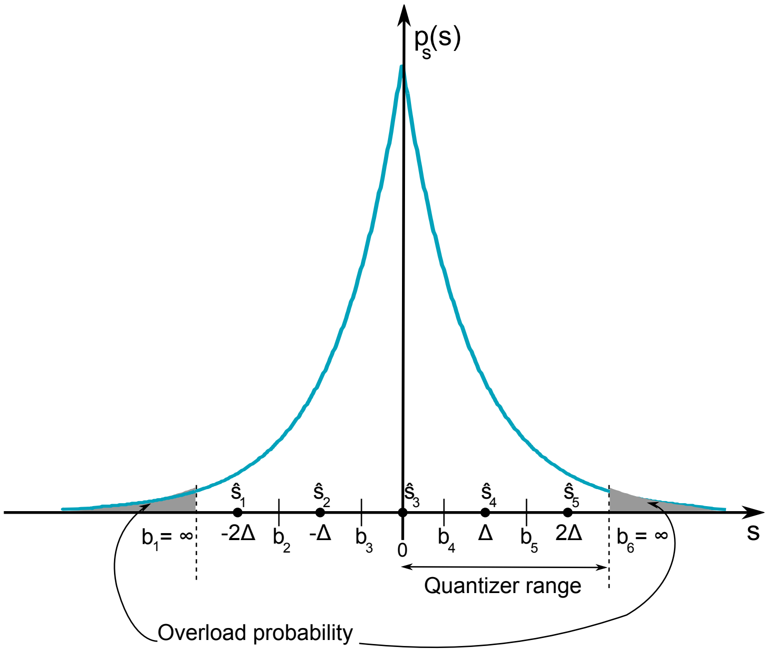


Figure 5.11: Illustration of overload noise for a Laplace distributed source.

The prediction error signal in DPCM is often modelled as Laplacian distributed:

$$f(e|\mu, \beta) = \frac{1}{2\beta} \exp - \frac{|s - \mu|}{\beta} \tag{5.55}$$

where the parameters μ and β are the mean and the scale. The standard deviation is directly given by the scales as $\sigma = \sqrt{2}\beta$.

An additional design parameter, the *loading factor* λ, is often defined to describe the trade-off between step size and range. The loading factor is defined as the ratio of the maximum value within the granular region (as illustrated with the dotted line in Figure 5.11) to the standard deviation of the source:

$$\lambda = \frac{\text{range}}{\sigma_e} = \frac{b_{L-1} + \Delta}{\sigma_e} \tag{5.56}$$

Often, a constant word length of B bits is required for coding of the quantizer output, which restricts the number of levels in the quantizer to $L = 2^B$. To find the optimal Δ for the given L in terms of quantization noise, σ_q should be minimized with respect to Δ for the current probability distribution. This trade-off between granular and quantization noise is not equally important when designing a quantizer whos output is coded with variable word length. Theoretically, the granular noise can then be avoided completely by having an infinite number of quantization intervals, and code the less probable ones with many bits if they ever

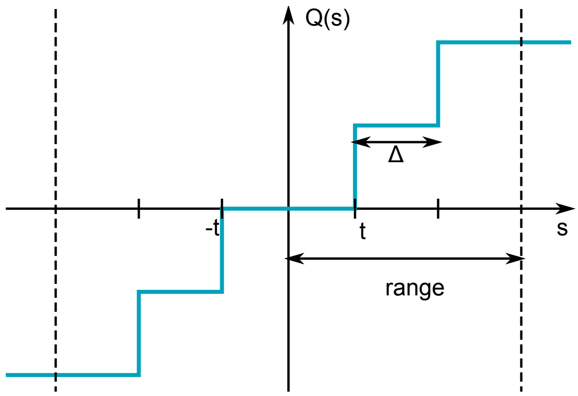


Figure 5.12: Design parameters for a dead-zone quantizer

appear. In practice it is easier to implement a quantizer with a finite number of levels, but the range of the quantizer can be relatively wide to avoid overload noise.

For uniform midtread quantizers, it is quite common to introduce a so-called *dead-zone*, which means that the step size of the zero-interval is widened, and the quantizer is not strictly uniform any more. This actually introduces additional noise, but it also results in a removal of distortions around the zero-level, an the result can therefore look more visually pleasing than the uniformly quantized version. Since the distribution of the prediction error signal in DPCM is very concentrated around zeros, a dead-zone can make a run-length coding scheme more effective, and can therefore reduce the final output rate. Run-length coding is further discussed in Section 5.5. The design parameters for a uniform dead-zone quantizer is shown in Figure 5.12. The threshold t defines the one-sided width of the dead-zone interval, while Δ gives the step size for the other intervals. The dead-zone quantizer is a bit harder to optimize than the plain uniform quantizer, because the final bit rate after encoding has to be taken into account as well. If a certain SQNR is required, the parameters t and Δ can be chosen such that this requirement is fulfilled, and further simulations or calculations can be performed to optimize in terms of output rate.

5.5 Coding for Minimum Bit Representation

As mentioned in Section 5.1, the quantizer levels should in some way be coded for minimum bit representation. How this should be done depends on many factors, for instance the desired format of the output. If a variable wordlength is allowed, the quantizer levels can be entropy coded with Huffman code or arithmetic coding as described in [26] to reduce the bitrate. But if a constant wordlength is required, more care must be taken in the quantizer design itself.

If the output of the quantizer contains long runs of zeros, it can be effectively represented by run-length coding. This principle is for instance applied in the facsimile standard as described in [26]. The basic principle of run-length coding is to encode the lengths of runs with the same value instead the values itself. How the corresponding value for each run is encoded depends on the application.

One variety of run-length coding is called *Stack-run* (SR). The main principle of SR encoding is to code the zero-runs and the non-zero values binary with two different alphabets, and then encode the stream of the four different symbols binary or with entropy coding. This scheme is discussed in detail in [43], which applies it for efficient encoding of wavelet coefficients. It can also be useful for differential coding, especially for low rate applications. If a dead-zone quantizer is applied, it can be expected that most of the output values will be zero, which can be efficiently encoded with SR encoding.

If a major part of the signal is represented by zero, the SR encoder will efficiently reduce the bit rate. On the other hand, if most of the quantizer output values take other values than zero, the SR encoder may increase the bit rate since it employs a four-symbol alphabet. The non-zero values are therefore represented with twice as many bits as if they were binary coded directly.

5.6 Suggestion for a Complete Compression Algorithm

This section gives a suggestion for a complete compression system based on differential video coding for the NUTS application.

A three-dimensional DPCM scheme was chosen for compression of the gravity wave image sequences, due to its simplicity compared to other video coding schemes. The gravity waves image sequences are expected to have a high correlation both spatially and temporarily, at least if most of the detector noise can be removed before compression. A three-dimensional predictor will be used for most of the pixels, while a two-dimensional one will be used where there are no data from earlier samples available. For the 3D-prediction, a prediction window with the three nearest neighbours will be used, resulting in the predictor in (5.26). The three first terms of the 2D predictor in (5.16) was chosen for the first frame of the sequence. For the DPCM algorithm to work optimally, the input signal should be zero mean. The mean value of the images should be subtracted before compression. This can either be sent as side information or be omitted completely.

Since the satellite is moving fast and the frame rate of the image sequence should be relatively low, some kind of motion compensation needs to be applied. If the speed of the satellite is known accurately enough, this could be done by shifting the whole image with a constant number of pixels, as for the image averaging in Section 4.5. This can be incorporated into the DPCM algorithm simply by shifting the coordinates of the pixel from the previous frame that is used for the prediction, if a precision of integer pixels is good enough. However, if the speed of the satellite is not accurately known, a simplified version of the block matching scheme in Section 5.3 can be applied. If the velocity is assumed to be constant in the whole image and close to the estimated speed, the block matching can in principle be performed using only one block in the middle of the image, with a narrow search window around the estimated shift. The estimated motion vector for this block will then be used to shift the whole image. For robustness, a grid of for instance 9 blocks can be used in the same manner, and the average of the estimated motion vectors can be used. However, if there is any significant rotation or non-uniform movement between the images, motion compensation with a constant vector may lead to inaccurate results. But full block matching as described in Section 5.3 is computationally demanding, and will lead to a huge increase in complexity for the compression algorithm. It is advised that the motion compensation is kept as simple as possible to keep the complexity of the algorithm low.

If the differential coding works as intended, the variance of the prediction error images will be much lower than for the original images. This means that the range of the quantizer can be quite small, and fewer bits are needed to encode the output. A midtread uniform quantizer with dead-zone was chosen for quantization of the prediction error signal, in order to enable Stack-run coding and low bitrate.

The predictors and quantizer depend on several parameters that should be optimized with respect to the statistics of the image sequence. The correlation coefficients of the image sequence can be estimated using the expressions in (5.33),(5.34) and (5.35), and the prediction coefficients for the two- and three-dimensional predictor can be computed from these estimated values by solving the normal equations. The dead-zone threshold and range of the quantizer should be determined according to the distribution of the prediction error images. If the standard deviation of the prediction error images is computed, the loading factor will determine the range of the quantizer, and a *dead-zone loading factor* $\lambda_{\mathrm{dz}} = \frac{t}{\sigma_e}$ will set the dead-zone threshold in a similar way. Simulations can be performed to find the loading and dead-zone loading factors and the number of levels that give the desired trade-off between output rate and distortion.

Assuming that the statistics of the gravity wave images are relatively stationary, the parameters discussed above can have fixed values known to both the encoder at the satellite, and the decoder at the ground station. However, since there are a lot of uncertainties regarding what the images will look like, it could be advantageous to estimate the parameters from images obtained by the camera while the satellite is in orbit. The suggested solution is to transmit the first images obtained by the satellite without any compression, and use these to estimate the optimal parameters for the compression algorithm. The parameters can then be sent to the satellite and used for efficient encoding. This option does of course require the possibility to change the parameters of the algorithms when the satellite is in orbit.

It is also suggested to apply Stack-run coding, in order to reach low bit rates. The performance of the SR encoder will depend on the distribution of the quantizer output. If the quantizer output turns out to have a wider distribution than expected, simple binary coding will perform better. The optimal type of bit coding for the SR symbols depends on their distribution. If the distribution is relatively uniform, simple binary encoding may be just as good as Huffman coding.

5.7 Implementation and Simulations in MATLAB

The proposed three-dimensional DPCM algorithm were implemented in MATLAB, and simulated for different input images and parameters. The implementation is not complete, and is first of all meant as a proof-of-concept, to demonstrate that this scheme might be feasible for our application. An attempt has been made to choose reasonable values for the parameters, but most of the parameter values can be optimised further, and also be adapted to the image material, to improve the performance of the algorithm.

An overview of the MATLAB functions in the implementation is given in Figure 5.13, and their content is further discussed below.

5.7.1 Prediction and quantization

A simple one-dimensional DPCM algorithm was modified to work in three dimensions, as discussed in Section 5.6. `dpcm3D_encode` incorporates the complete 3D DPCM encoder, taking a zero-mean three-dimensional array, representing the image sequence, as input. Other inputs are the quantization parameters and prediction coefficients. The output is also a three-dimensional array, representing the prediction error image sequence.

As mentioned in Section 5.6, a 2D predictor will be used for the first frame of the sequence. Since both the two- and three-dimensional predictors depend on values of previous pixels vertically and horizontally, the pixels in the first row and column of each frame will not be predicted and their value must be encoded as it is. This results in three different prediction categories for the pixels in a sequence; those without prediction, those with 2D prediction and those with 3D prediction, as illustrated in Figure 5.14(a). Quantizers have been implemented, with different parameters for the three different prediction categories. Uniform dead-zone quantizers are used for the 2D and 3D predicted pixels, while a plain uniform quantizer with wider range is applied for the pixels without prediction.

The quantization parameters and the prediction coefficients are estimated in separate functions: `set_q_param`, `est_2Dpredcoeffs` and `est_3Dpredcoeffs`. For simulation purposes, these are included in the same script as the DPCM encoder, providing updated estimates for each image sequence. This should however not be the case in the final implementation, which should use static parameters known to both the encoder and decoder, as discussed in Section 5.6.

The estimation of the prediction coefficients is performed by estimating the correlation coefficients from the image sequence as described earlier, and using them to generate and solve the normal equations containing the prediction coefficients.

The three different quantization parameters (L, Δ and t) are determined by running an image sequence through the complete DPCM encoder without quantization, and then estimate the standard deviation of the outputs corresponding to the three different prediction categories. These estimates are used to determine the quantization parameters as discussed in Section 5.6.

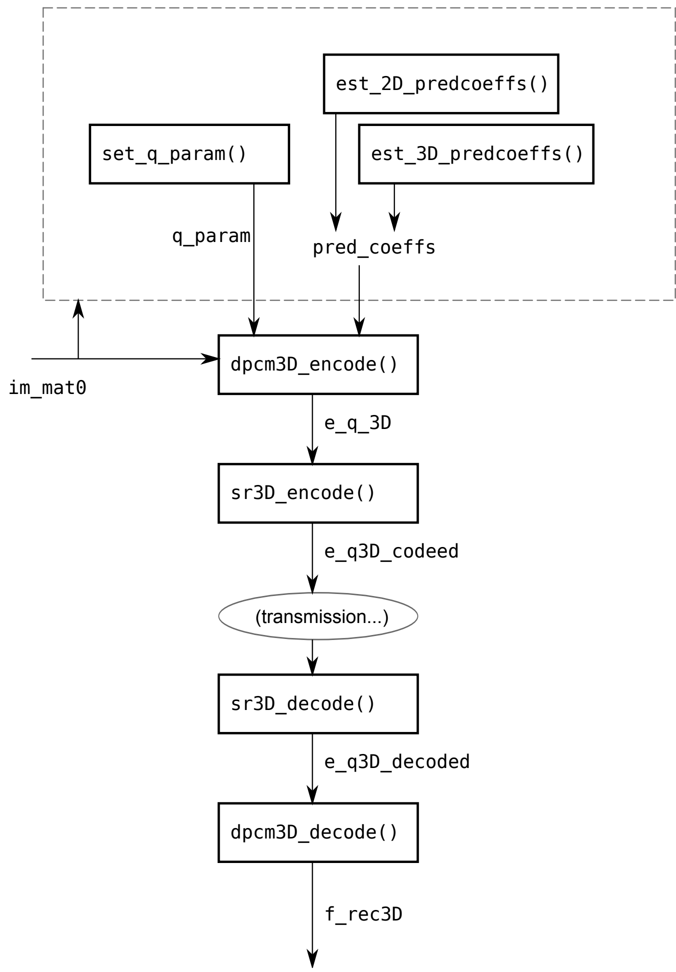


Figure 5.13: Overview of the implementation of the compression system in MATLAB.

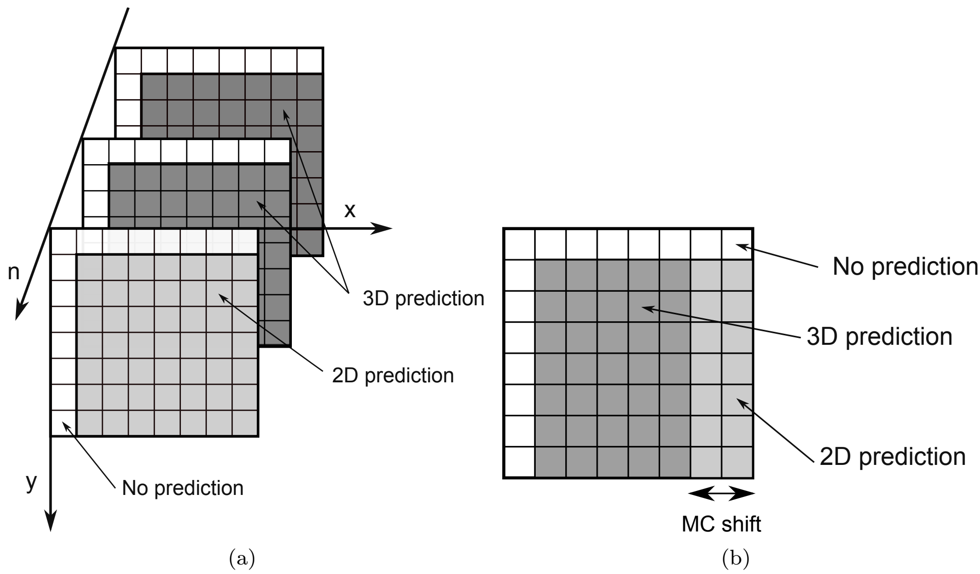


Figure 5.14: Illustration of the three different prediction categories. (a) shows a sequence of three images without motion compensation. (b) shows a frame where motion compensation is performed by applying a shift "MC shift" between the images, which results in an area where 3D prediction can not be applied.

5.7.2 Motion Compensation

The motion compensation has not been implemented yet, but based on the knowledge of the satellite's motion, it is assumed that a simple scheme with a constant shift for the whole image will be sufficient. The prediction has been implemented such that a shift of an integer number of pixels can be applied for the pixel in the previous image that is used for prediction, as shown in Figure 5.15. Due to the motion, there is an area along the edges that the previous image does not cover, as shown in Figure 5.14(b) for a horizontal shift of two pixels. The 2D prediction scheme must therefore be applied in this area.

5.7.3 Bitcoding

For the SR encoding and decoding, a MATLAB implementation made by Anna Kim based on [43] was used. The same encoding was used for the whole image sequence. It would probably be better to use another encoding scheme for the pixels in the edges that are not predicted, but this would complicate the decoding. The SR symbols were binary coded.

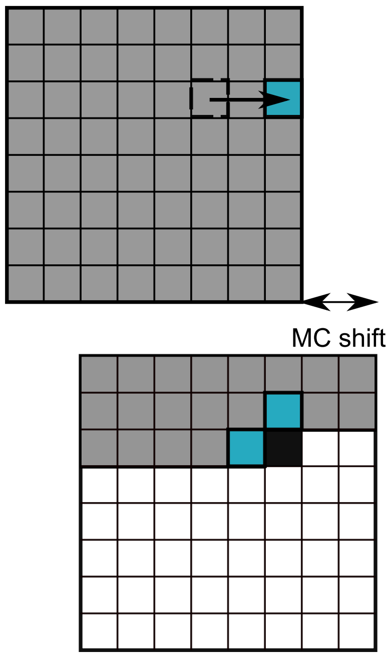


Figure 5.15: Illustration of 3D-prediction with simple motion compensation (MC). The turquoise squares indicate the pixels used for the prediction (the prediction window) of the current pixel (indicated with a black square). The current image is shifted with "MC shift" compared to the previous image, and the turquoise pixel in the previous frame is therefore shifted in order to correspond to the image content of the current pixel.

5.7.4 Simulations

Some simulations with representative test images and parameters were performed, but the parameters have not been optimized. The test script `dpcm_demo` generates test images, performs 3D DPCM with quantization and SR encoding, and subsequent decoding. Some of the functions used have a debug option that enable plots and display of parameters for simulation and debugging purposes.

Test Image Sequence

Ideally, an image sequence with a shift between the images should have been used, to simulate the motion, but since the motion compensation has not been implemented yet, images that are already aligned had to be used. The test image sequence that was generated consisted of five sine images, with Gaussian noise and a small random shift in angle, to ensure that there is some difference between them. The test images were generated as discussed in Section 4.2, with possibility to vary the SNR factor K and the number of periods of the sine wave. For the simulations discussed in this section, sine waves with 11.5 periods per image (corresponding to the mean GW wavelength) and an angle of 10° was used, and an SNR factor of K=50 was used unless other values are specified.

Prediction Error Signal and Quantizers

Figure 5.16 shows distributions of the prediction error signal for the different prediction categories, as well as an example of corresponding quantizers. The top histogram of Figure 5.16(a) shows the distribution of the pixel values along the edges that are not predicted. This corresponds to the distribution of the whole image before prediction, and looks roughly uniform. The distribution of the prediction error of the 2D- and 3D-predicted pixels, in the middle and bottom histograms of Figure 5.16(a), looks much narrower, and resemble laplacian distributions. It was expected that the three-dimensional predictor would perform better than the two-dimensional, and the slightly narrower distribution of the 3D-predicted pixels confirm this. The difference is however not that large, especially not for the distributions of the quantized prediction error in Figure 5.16(c). The distribution of the 3D-predicted pixels is slightly askew, but the reason for this is not known. Figure 5.16(b) shows the quantizers that were used for the simulations, which corresponds to row (c) in Table 5.1.

Quantization Parameters

Different quantizer parameters were tried relatively arbitrarily, to investigate how the PSNR and output rate varies with different values for the dead-zone threshold t and step size Δ. For all the simulations, a loading factor of $\lambda = 5$ was used for the quantizers to avoid overload noise. The number of levels, L, and the dead zone loading factor λ_{dz} was used to adjust the dead-zone threshold t and the step size Δ. The results are summarized in Table 5.1, and the corresponding quantizer

Table 5.1: Simulation results for four different quantizers.

	Parameters			**Results**		
	L	λ_{dz}	$\frac{2t}{\Delta}$	PSNR [dB]	Entropy [bit/px]	Output rate [bit/px]
(a)	5	1	1	50.7	1.44	1.04
(b)	7	0.71	1	53.4	1.78	1.27
(c)	7	1.25	2	50.9	1.25	0.83
(d)	7	1.66	3	48.8	0.94	0.61

characteristics are shown in Figure 5.17 while the distribution of the quantized 3D-prediction errors are given in Figure 5.19.

The results in Table 5.1 clearly shows that by increasing the width of the dead-zone, the output rate is lowered due to increased efficiency of the SR coding, but the PSNR also gets lower. The dead-zone quantizer in row (c), which has a dead-zone that is twice as big as the step size, has a rate of only 0.83 bit/px, but also a PSNR of more than 50 dB. This shows that a rate of less than 1 bit per pixel is possible while still maintaining a good quality. The corresponding recovered image is shown in Figure 5.18(c).

Noisy Images

Simulations for test images with different SNR factors were also tried, in order to investigate how the performance of the DPCM algorithm would vary. A test image for an SNR factor of $K = 10$ is shown in Figure 5.18(b). As illustrated in Figure 5.20, the signal cannot be decorrelated properly, and the distribution of the prediction signal remain as wide as the source signal. First, this prediction error was quantized with the parameters of row (b) in Table 5.1. This resulted in a low rate, but a poor PSNR of the recovered image since the prediction error signal could not be reconstructed properly. The reconstructed image is shown in Figure 5.18(d). Then, a uniform quantizer with $L = 11$ was used. This resulted in better PSNR, but a very high rate since the SR coding did not function properly anymore.

5.8 Summary and Discussion

The implementation and simulations of the three-dimensional DPCM algorithm with dead-zone quantizer and SR coding showed that even with such a simple algorithm, bitrates of less than 1 bit per pixel can be achieved, with an acceptable image quality. A bitrate of 0.83 bits per pixel, as obtained with one of the quantizers in the simulations, results a compression factor of 9.64 compared to a 8-bit representation, and gives 54.4 Kb per image with a resolution of 256 × 256 pixels. A sequence of 10 images can then be represented by 0.54 Mb. This gives the opportunity to either download more image sequences, or to obtain longer sequences with the same download capacity as before. However, the number of images that

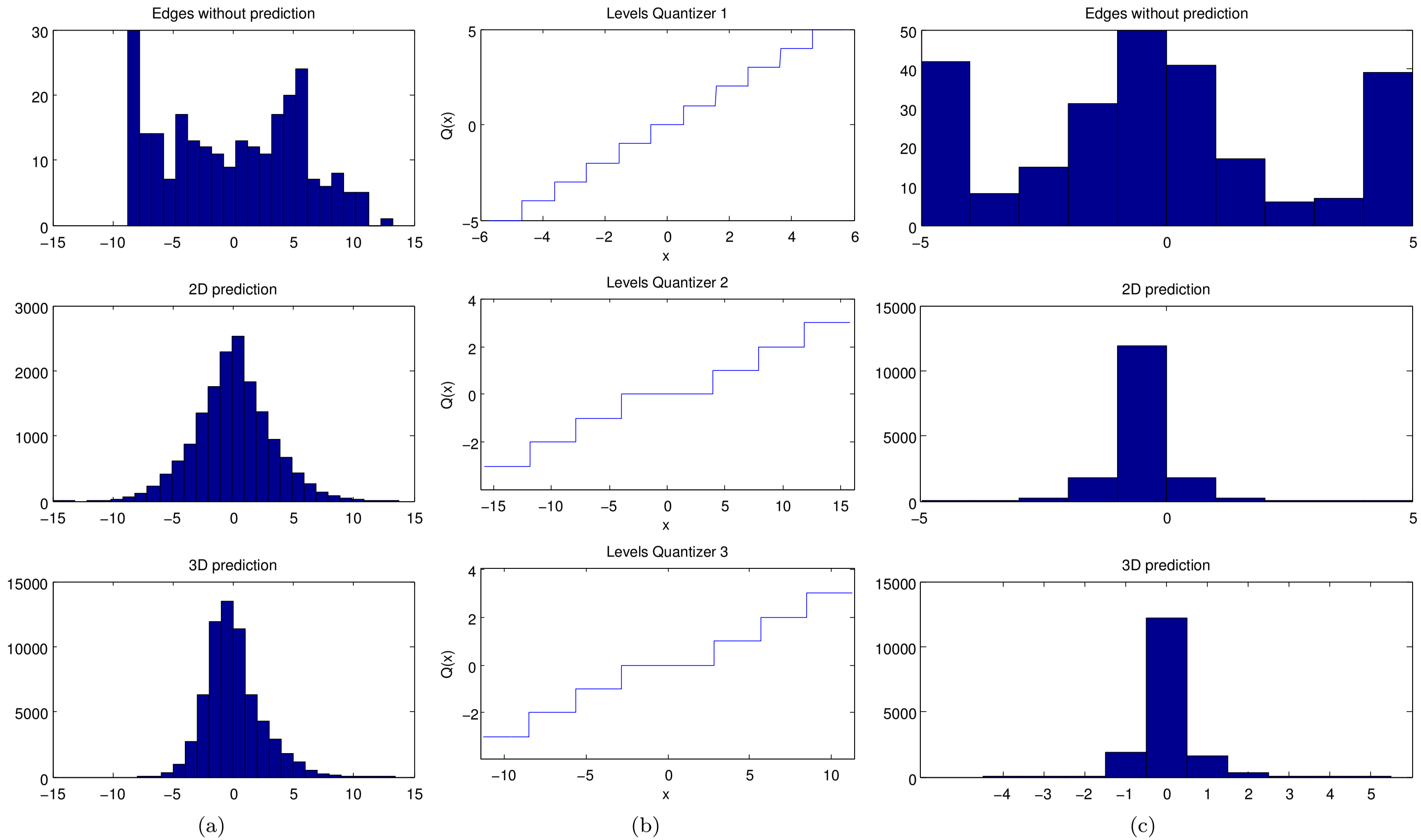
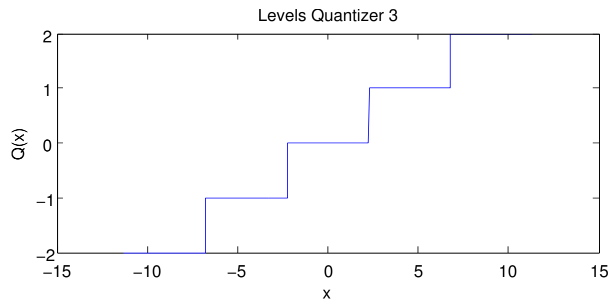
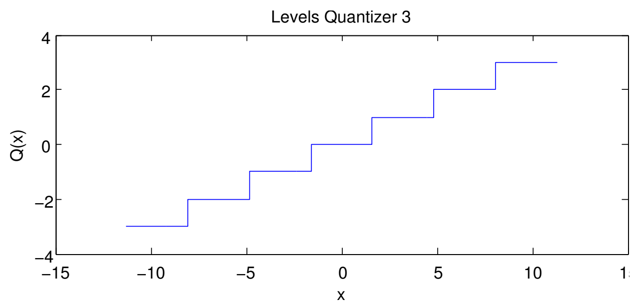


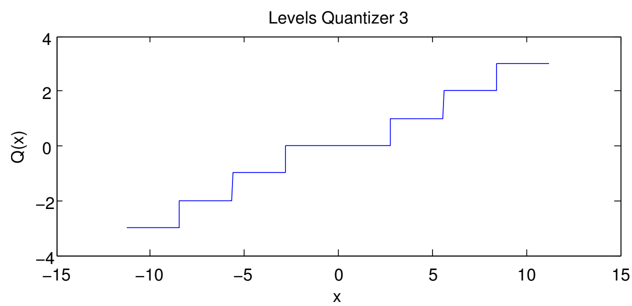
Figure 5.16: Simulation of the 3D DPCM algorithm. (a) shows the distribution of the prediction error for the different prediction categories, without any further quantization. (b) shows the characteristic function of the three quantizers. (c) shows the distribution of the quantized prediction error for the different prediction categories.



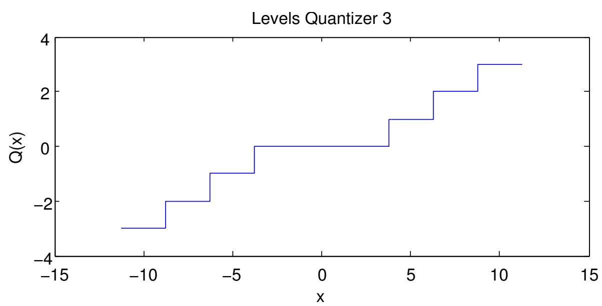
(a)



(b)



(c)



(d)

Figure 5.17: Quantizers with different parameters as given in Table 5.1

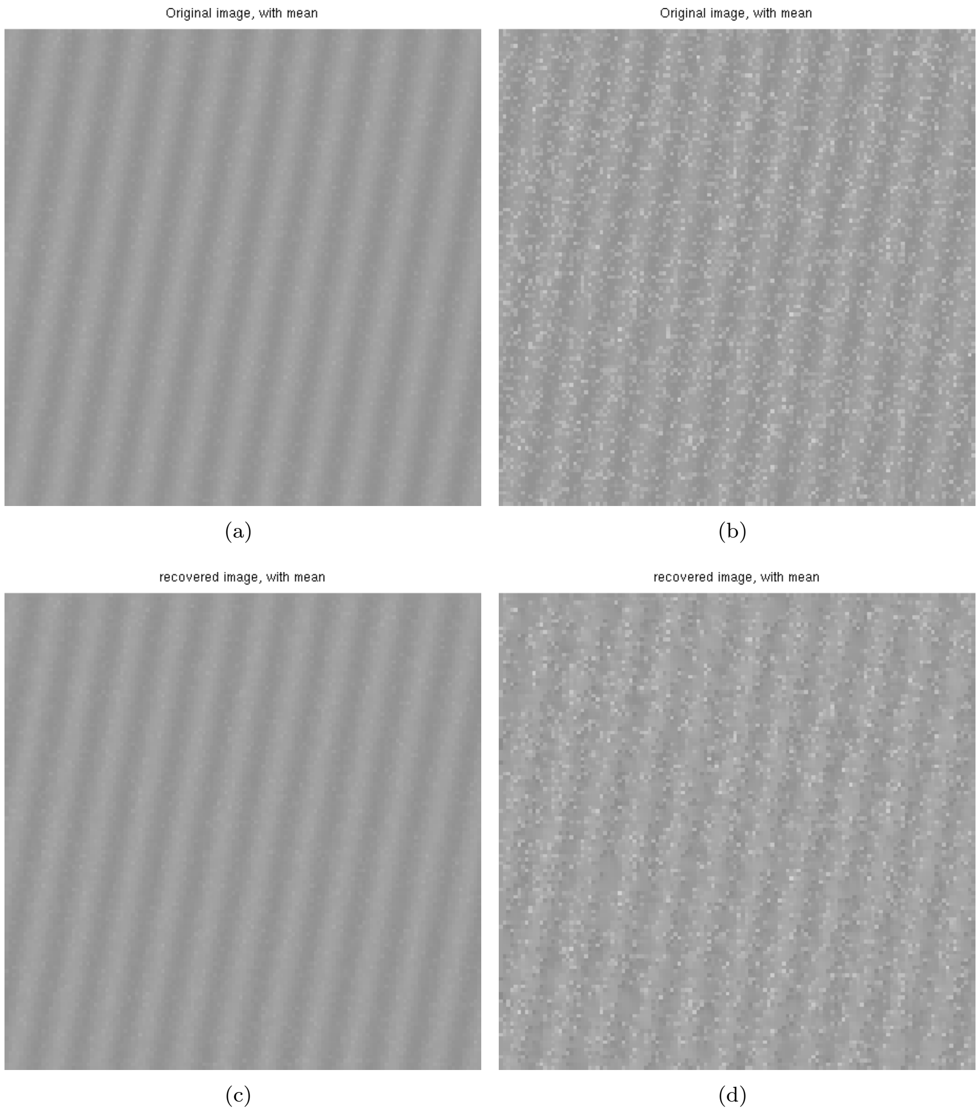


Figure 5.18: Original and recovered images after compression. (a): Test image with $K = 50$. (b): Test image with $K = 10$. (c): Recovered version of (a) after compression. (d): Recovered version of (b) after compression.

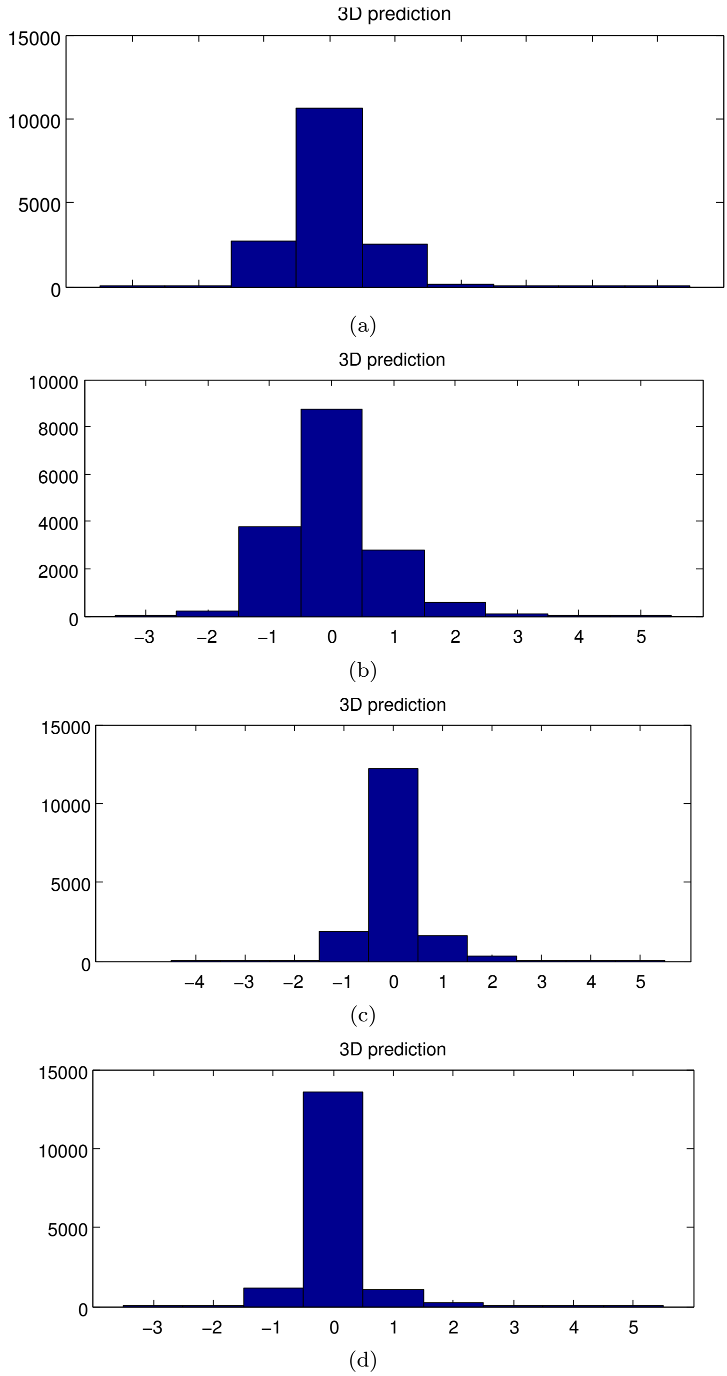


Figure 5.19: Distribution of quantized prediction error (3D) for the quantizers in Figure 5.17 and Table 5.1

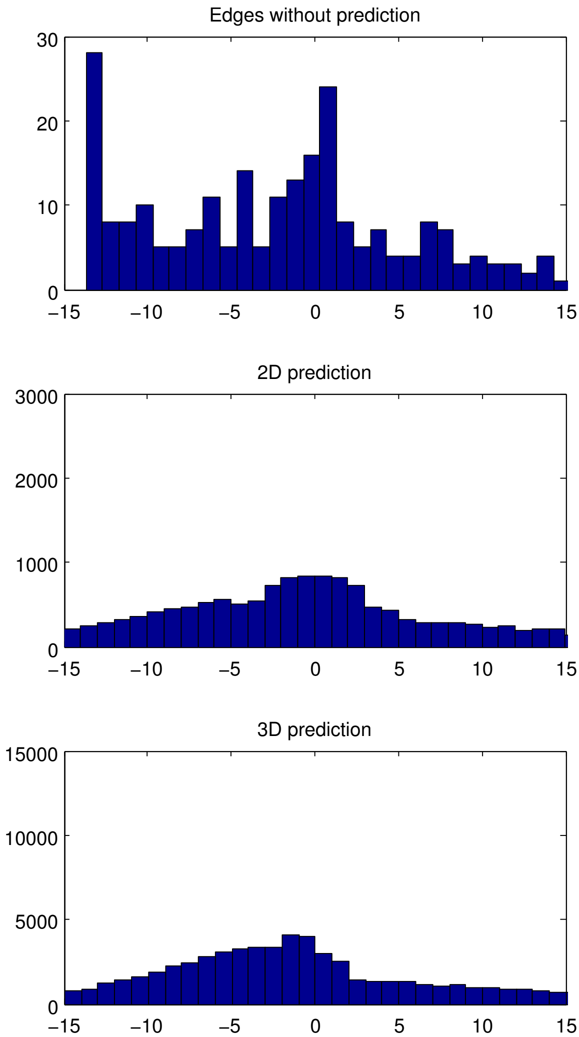


Figure 5.20: Distribution of the prediction error signal (without quantization) for the test image in Figure 5.18(b)

can be transmitted is still quite limited due to the low download capacity.

The algorithm should be optimised further by performing simulations with different parameters for the dead-zone quantizer, in order to get the desired image quality at the lowest possible rate. The simulations were performed without taking motion compensation into account. However, as long as the motion compensation works well, the results should not be very different from the ones obtained here. But there will be a section along the edges where the 3D-predictor cannot be applied, as discussed earlier, due to the fact that the images do not overlap. This can lead to some decrease in the performance of this compression algorithm, especially if the shift between the images is large.

As shown in the simulations, the DPCM algorithm does not work that well on noisy images, which leads to an increased output rate to get the quality required. This confirms the importance of noise removal before compression.

The implementation and simulations in this section is by no means complete. It has however been demonstrated that a simple differential coding scheme might be feasible for compression of payload data from NUTS. The foundation that has been laid regarding the implementation of a three-dimensional DPCM algorithm can hopefully be useful when the complete algorithm is to be implemented on the satellite in the future.

Chapter 6

Summary and Conclusion

6.1 Suggestion for a Complete Signal Processing System

This section attempts to provide an overview of the payload system, with main focus on obtaining a complete signal processing strategy to prepare the images for transmission and assure a sufficient quality. A schematic of the complete system is provided in Figure 6.1.

A suitable InGaAs camera must be found as soon as possible, and be integrated with optics and other modules of the satellite. The optics and detector must be chosen such that a suitable image coverage and resolution can be obtained. For most of the discussion in this thesis, an image coverage of 300 km and a detector with 256×256 pixels have been assumed. An optical bandpass filter should also be applied, in order to eliminate background radiation from Earth, as discussed in Section 3.2.6. InGaAs sensors are known to have a strong background signal due to dark current, which must be removed in some way to get sufficient image quality.

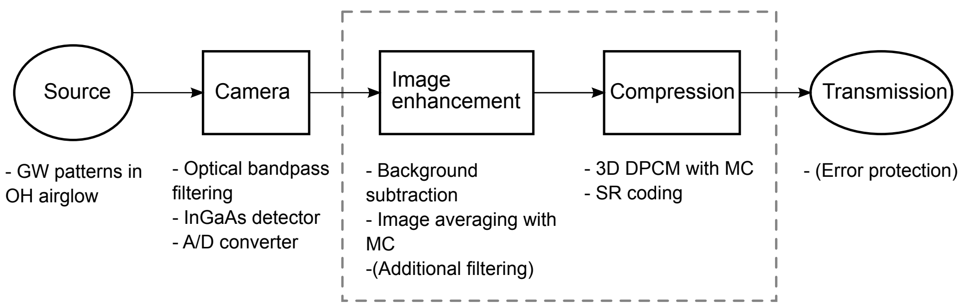


Figure 6.1: Overview of the whole system

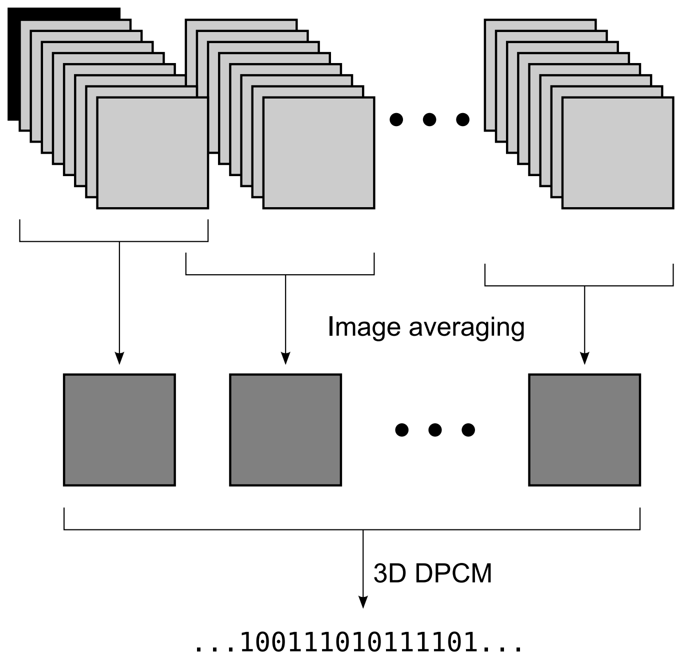


Figure 6.2: Image enhancement and compression

The A/D converter of the camera is assumed to have a fine quantizer, but this also means a high number of bits per pixel in the output image from the camera.

Several sequences of images with short exposure times should be obtained as shown at the top of Figure 6.2. Each sequence is to be combined by image averaging to yield one image for transmission. The number of images should be chosen to give a total integration time that is long enough to give sufficient DSNR. Furthermore, the integration time for each image should be short enough to prevent motion blur. The first image of the sequence can be a calibration image (indicated by the black square in Figure 6.2) obtained with closed shutter to measure the background signal of the detector. How often this calibration should be done depends on the temperature variations.

Background subtraction should be performed by subtracting the calibration image from the following images in the sequence, as suggested in Section 4.2.3. Image averaging with motion compensation will then be applied to rest of the images, as discussed in Section 4.5. If necessary, additional filtering (lowpass and/or median filter) may be applied to reduce the noise further, as discussed in Section 4.2.5.

A sequence of overlapping *combined* images (indicated by the dark grey squares in Figure 6.2) should be obtained to provide a scan of a desired area. This sequence can be regarded as a video with low frame rate, and should be compressed with the 3D DPCM algorithm combined with SR coding, as discussed in Section 5.6. There are several parameters regarding the prediction and quantization in this algorithm that should be adjusted according to the image material. It is suggested that there should be an option to transmit uncompressed images to the ground station, in order to determine suitable parameters for the algorithms, as discussed in Section 5.6.

As shown by the simulations of the DPCM algorithm in Section 5.7.4, it will not be able to compress noisy images efficiently. This is the reason why the image averaging should be performed *before* compression, and must therefore be implemented on-board the satellite. One should however have the possibility to turn off the averaging and adjust the exposure time, in case the algorithm fails or the incoming signal is completely different than expected.

It is important to have accurate information about the motion of the satellite, both regarding orbital velocity and any possible rotation. The image averaging algorithm depends on precise motion compensation to avoid blur, and this is also important for the compression algorithm to function optimally. Some form of motion estimation could be applied for the compression algorithm, but this is not trivial to implement, and would increase the complexity. For the images with short exposure time that is combined with image averaging, it is probably hard to perform any motion estimation at all, due to the noise. Since there still are some uncertainties regarding the specification of the ADCS system, this should be further investigated before the final enhancement and compression algorithms are implemented. It may also be convenient to adjust some parameters regarding motion compensation after launch, in case the ADCS system works differently than expected.

The image enhancement and compression algorithms contain several parameters that must be optimized further, but many important factors are still unknown. Most of the implementation and optimization can be done when a launch is scheduled and the camera has been found, but there are also several parameters that must be adjusted after launch. This requires the possibility to send commands that can change the parameters in question while the satellite is in orbit.

As shown in the discussion of the downlink capacity in Section 2.2, an average downlink capacity of 4.9 Mb per day can be assumed, where approximately half of it can be used for payload data, i.e. 2.45 Mb per day. This corresponds to an image transfer rate of less than 5 uncompressed images[1] per day. Assuming that video compression can provide 0.83 bits per pixel (as in Table 5.1), the image transfer rate is increased to 45 images per day. Whether this should be obtained as one long sequence or several short ones depends on the kind of data that is desired for further analysis of the images. The compression will however be most efficient for long sequences of images with significant overlap.

Several factors have been mentioned that might degrade the performance of the compression algorithm compared to the simulations performed in Section 5.7.4, but the parameters have not been optimized yet. All in all, it is therefore believed that the result from these simulations can give a good indication to what can be achieved. One should however investigate the algorithm's vulnerability to channel errors, and what kind of protection that is necessary for the NUTS downlink. If necessary, a forward error correcting code should be applied, but this will reduce

[1] Assuming 256×256 pixels and 8 bit per pixel.

the image transfer rate. A turbo-code could for instance be applied to the payload date, since this type of error correction provides a low complexity at the encoder side [44] and a low overhead, but this needs to be investigated further.

6.2 Conclusion

In this thesis, several issues regarding the NUTS payload camera have been discussed. The main focus has been on suggesting suitable signal processing algorithms that can assure good quality and compression.

A suitable and available InGaAs infrared camera is yet to be found, and many parameters regarding the camera are therefore still unknown. The studies and experiments concerning InGaAs sensors in Chapter 3 indicated that a significant background signal, both in terms of offset and noise, can be expected. Long integration time and background subtraction will therefore be necessary in order to ensure a satisfying image quality.

The simulations of motion blur in Section 4.3.3 show that even though a low resolution is required to observe the large-scale gravity wave patterns, motion blur will be a problem for long integration times due to the high speed of the satellite. The maximum integration time that can be allowed will depend on the required image quality. It should at least be kept below 1 second to preserve gravity wave patterns down to a wavelength of 15 km.

Image averaging with motion compensation was concluded to be the best strategy for avoiding motion blur and at the same time get a sufficient SNR with respect to detector noise. Simulations showed that this is a more reliable and flexible strategy than the deconvolution approach.

The processing required for image averaging and motion compensation should be done on-board the satellite before compression, in order to provide a sufficient SNR for the compression algorithm. A sequence of overlapping combined images can be obtained to provide a scan of a desired area, and should be encoded as video to enable efficient compression and transmission of as many images as possible to the ground station.

Through simulations with synthetic test images, it has been indicated that video coding with three-dimensional DPCM combined with a dead-zone quantizer and SR coding can provide a bit rate lower than 1 bit per pixel for a sequence of gravity wave images. The simulations performed in Section 5.7.4 show bit rates down to 0.61 bit/px with an acceptable quality, but 0.83 bit/px was required to get a PSNR above 50 dB. The parameters of the dead-zone quantizer should however be optimized further, to provide a good trade-off between distortion and bitrate. Accurate motion compensation and low noise levels are vital factors that will affect the performance of this algorithm.

This work has only provided suggestions for a suitable camera type and algorithms for the payload. A lot of work still remains to be done to get an operational payload, and many of the remaining tasks have been mentioned earlier in this thesis:

- Obtain a suitable InGaAs camera

- Integrate the camera with optics and electronics
- Investigate the assumed motion of the satellite, implement motion compensation (and estimation if necessary)
- Optimize and perform further simulations of the quantization and SR decoding, enable adjustment of parameters in orbit
- Implement the image enhancement and compression algorithms on a microcontroller on the satellite

Bibliography

[1] G. R. Swenson, “The waves explorer,” *Research Proposal to NASA. Submitted by: The Board of Trustees of the University of Illinois*, 1998.

[2] K.Nielsen, M.J.Taylor, R. Hibbins, and M. Darvis, “Climatology of short-period mesospheric gravity waves over halley, antarctica(761s, 271w),” *Journal of Atmospheric and Solar-Terrestrial Physics*, 2009.

[3] Z. Li, A. Z. Liu, X. Lu, G. R. Swenson, and S. J. Franke, “Gravity wave characteristics from OH airglow imager over maui,” *JOURNAL OF GEOPHYSICAL RESEARCH, VOL. 116, D22115,*, 2011.

[4] R. Birkeland, “NUTS-1 mission statement,” 2011.

[5] R. Birkeland, E. K. Blom, and E. Narverud, “Small student satellite,” 2006.

[6] The CubeSat Program, Cal Poly SLO, *CubeSat Design Specication rev. 12*, 2011.

[7] R. R. J. Muylaert and C. Asma, “The QB50 project,” in *4th European CubeSat Symposium, Book of Abstracts* (R. Reinhard, ed.), Von Karman Institute for Fluid Dynamics, 2012.

[8] W.G.Rees, *Physical Principles of Remote Sensing.* Cambridge University Press, 2001.

[9] S. Marholm, “Antenna systems for NUTS.” Specialization Project Report, NTNU, published at `http://nuts.cubesat.no/publications-and-reports`, 2011.

[10] S. Marholm, “Antenna systems for nuts,” Master’s thesis, NTNU, 2012. Will be published in Autumn 2012.

[11] J. Friedel and S. McKibbon, “Senior project: Thermal analysis of the cubesat CP3 satellite,” 2011.

[12] D. G.Andrews, *An Introduction to Atmospheric Physics.* Cambridge University Press, 2010.

[13] A. G. Villafranca, J. Corbera, F. Martin, and J. F. Marchan, “Limitations of hyperspectral earth observation on small satellites,” *Journal of Small Satellites*, vol. 1, 2012.

[14] “Telescope - payload swisscube.” École Polytechnique Fédérale de Lausanne, published on the SwissCube website. Retrieved 09.12.2011.

[15] N. Scheidegger, “SwissCube payload system engineering draft.” École Polytechnique Fédérale de Lausanne, published on the Swisscube webpage, 2006.

[16] R. Vandersmissen, “Night-vision camera combines thermal and low-light-level images,” *Photonik International*, vol. 2, 2008.

[17] Princeton Instruments imaging group, *Introduction to Scientific InGaAs FPA cameras*, 2012.

[18] R. D. Fiete and T. Tantalo, “Comparison of SNR image quality metrics for remote sensing systems,” *Optical Engineering*, vol. Volume 40, p. 574, 2001.

[19] G. C. Holst, *CCD Arrays, Cameras and Displays.* SPIE Press, 1996.

[20] S. S. Rønning, “Notes on signal to noise ratio.” Unfinished.

[21] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye, *Probability and Statistics for Engineers and Scientists.* Prentice-Hall, 2002.

[22] R. Guntupalli and R. Allen, “Evaluation of ingaas camera for scientific near infrared imaging applications,” in *Proc. SPIE 6294, 629401*, 2006.

[23] Oral statement from Patrick Espy.

[24] P.J.Espy, W.R.Pendleton, G. Sivjee, and M. Fetrow, “Vibrational development of the n_2^+ meinel band system in the aurora,” *Journal of Geophysical Research*, vol. 92, pp. 11257–11261, 1987.

[25] R. C. Gonzales and R. E. Woods, *Digital Image Processing.* Pearson, 2008.

[26] K. Sayood, *Introduction to Data Compression.* Morgan Kaufmann, 2005.

[27] A. Rav-Acha and S. Peleg, “Restoration of multiple images with motion blur in different directions,” in *Proceedings of the Fifth IEEE Workshop on Applications of Computer Vision*, 2000.

[28] Q. Shan, J. Jia, and A. Agarwala, “High-quality motion deblurring from a single image,” in *SIGGRAPH ’08 ACM SIGGRAPH 2008 papers*, 2008.

[29] R. C. Gonzales, R. E. Woods, and S. L.Eddins, *Digital Image Processing Using MATLAB.* Pearson, 2004.

[30] W. H. Richardson, "Bayesian-based iterative method of image restoration," *Journal of the Optical Society of America, Volume 62*, 1972.

[31] R. Liu and J. Jia, "Reducing boundary artifacts in image deconvolution," in *Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on*, pp. 505 –508, oct. 2008.

[32] J. G. Proakis and D. G. Manolakis, *Digital Signal Processing.* Pearson, Prentice Hall, 1996.

[33] R. Chan, M. Ng, and W. Tang, "A fast algorithm for deblurring models with neumann boundary conditions," *SIAM J. SCI. COMPUT., Vol. 21*, 1999.

[34] M. Donatelli, C. Estatico, A. Martinelli, and S. Serra-Capizzano, "Improved image deblurring with anti-reflective boundary conditions and re-blurring," *Institute of Physics Publishing, Inverse Problems 22*, 2006.

[35] A. A. Bell, C. Seiler, J. N. Kaftan, and T. Aach, "Noise in high dynamic range imaging," in *Proc. 15th IEEE Int. Conf. Image Processing ICIP 2008*, pp. 561–564, 2008.

[36] J. C. A. van der Lubbe, *Information Theory.* Cambridge University Press, 1997.

[37] B. Gajić, "Introduksjon til statistisk signalbehandling." Lecture notes TTT4120 Digital Signal Processing, 2006.

[38] T. A. Ramstad, "Image communication." Lecture notes, TTT05 Digital bildekommunikasjon, 2011.

[39] A. K. Jain, *Fundamentals of digital image processing.* Englewood Cliffs, N.J. : Prentice-Hall, 1989.

[40] D. Daut, R. Fries, and J. Modestino, "Two-dimensional DPCM image coding based on an assumed stochastic image model," *Communications, IEEE Transactions on*, vol. 29, pp. 1365 – 1374, sep 1981.

[41] R. Krishnamurthy, J. Woods, and P. Moulin, "Frame interpolation and bidirectional prediction of video using compactly encoded optical-flow fields and label fields," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 9, pp. 713 –726, aug 1999.

[42] Y. Q. . Shi and H. Sun, *Image and Video Compression for Multimedia Engineering.* CRC Press, 2008.

[43] M.-J. Tsai, J. Villasensor, and F. Chen, "Stack-run coding for low bit rate image communication," in *Image Processing, 1996. Proceedings., International Conference on*, vol. 1, pp. 681 –684 vol.1, sep 1996.

[44] J. H. Weber, "Error-correcting codes." Lecture notes in ET4030 Error-Correcting Codes at TU Delft, 2007.

[45] K. Wakker, *Astrodynamics-I*. Delft University of Technology, 2010.

Appendix A

Overview of The NUTS Subsystems

This is a draft of a document written by the master students working on the NUTS project Spring 2012.

The NUTS Subsystems

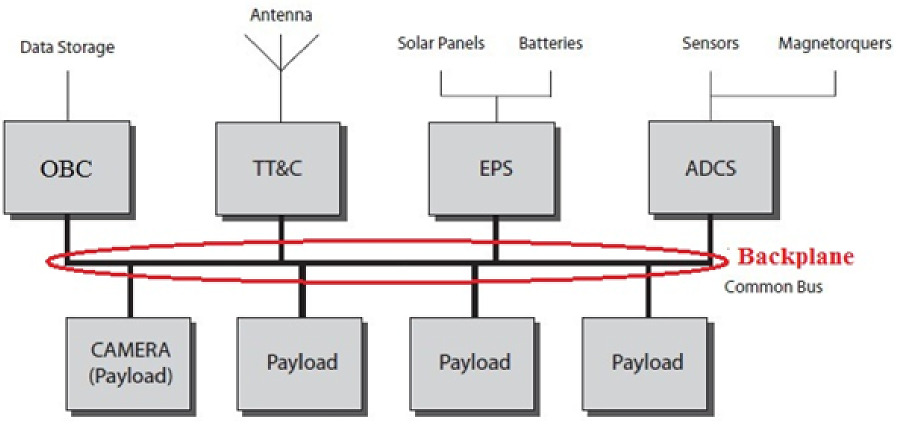
<u>Version: draft</u>

About the Project

The Norwegian University of Science and Technology (NTNU) Test Satellite (NUTS) project is aiming to launch a nanosatellite into Low Earth Orbit (LEO) by 2014. The satellite is a double CubeSat, measuring 10 cm x 10 cm x 20 cm and weighing less than 2.66 kg, which conforms to the CubeSat Standard. The satellite will carry an IR-camera for atmospheric observations as its main payload.

The NUTS project was started in September 2010, and is a part The Norwegian Satellite Program, ANSAT, run by NAROM (Norwegian Centre for Space-related Education). This program involes three educational establishments, namely the University of Oslo (UiO), Narvik University College (HiN) and NTNU. The program is developed with the intention to stimulate cooperation between different educational institutions in Norway and with the industry. The students will experience team work and hands-on training.

System Overview



Mechanical Structure

The satellite structure is typically 10-15% of the total satellite weight, where the main task is to make it possible to install and keep all the components connected. The most critical phase is the launch where the major forces and vibrations occur. After launch, the structure has a more passive role where the proper thermal and electrical conductivity is of interest, in addition to the ability to protect internal components against radiation.

Unlike previous CubeSat projects, which expand »typically have used aluminum, this project aims to utilize composite materials (carbon fiber/epoxy). Although some minor components have been made of carbon fiber in the past, launching a CubeSat with an all-composite primary

structure has not yet been done.
Composite materials have been used in the aircraft and aerospace industry several years and have an ever-growing popularity. One its main advantages having a superior relationship between stiffness and weight than other materials. In addition, due to the manufacturing method of long-fiber composites, we have the ability to create materials with very high anisotropic properties. Together, the high stiffness and the ability to tailor mechanical properties give rise to significant potential weight savings. Typical weight savings from aluminum to composites is highly dependent on the circumstances, but is about 30%.

The development of the frame this spring has been the development of secondary structure (attachment between primary structure and components), as well as tests to detect interactions between composite frame and P-POD. It has also been developed simulation and test methods for dynamic testing of the satellite later in the project.

+figur x 2
+referanse til masterrapport

On-Board Computer (OBC)
-tekst fra Dan Erik

Attitude Determination and Control System (ADCS)

An attitude determination and control system (ADCS) is important for the orientation control of a satellite. Without reliable attitude estimates, mission objectives may be severely compromised. It is important that the satellite rotates in a controlled way and that one of its sides points towards the Earth. This way, the infrared camera will be able to take pictures of the gravity waves. An ADCS system consists of two main parts; estimation and control. The attitude of the satellite is estimated through information from sensors, and the orientation is controlled to a known reference by using actuators.

A sun sensor, a magnetometer and a gyroscope will be used as sensors in the NUTS CubeSat. The solar panels can in theory be used instead of sun sensors, but as most sun sensors are low cost and light weight, buying them will be more convenient. A vector pointing towards the Sun is measured by the sun sensor, while the magnetometer measures a magnetic field vector pointing towards the Earth. The gyroscope gives the angular velocity of the satellite. The vector measurements are used as input for the attitude estimation.

The satellite will be controlled by magnetourqers, which will affect the local magnetic field. Therefore it is important that the attitude estimation and the attitude control are strictly separated. Since the estimated attitude will be inaccurate during control, the results would be useless. By switching the attitude determination off, power can be saved. Therefore a short start-up time of the estimation method is preferred. The number of coil windings for the magnetorquers are yet to be designed.

Estimation methods are needed to determine the current attitude. An extended quaternion estimation (EQUEST) method and a nonlinear observer have been developed and implemented in order to test the different qualities of the methods. Due to limited space, weight and power, estimation methods used for larger satellites are less suited for implementation in CubeSats. The EQUEST method obtains a solution in one time-step, which makes it fast. However it is very sensitive to disturbances. The nonlinear observer has a slower start-up phase, but can cope better with noise. A combination of the two methods, where the EQUEST method is used to find the initial values for the nonlinear observer, can be considered if the power of the ADCS is sufficient.

If the estimated attitude deviates from the wanted reference, the orientation of the satellite needs to be changed. Two phases for the control must be considered; the detumbling phase and the stabilization phase. After the satellite is launched into orbit, it will get an initial spin around its centre of gravity relative to the Earth. After a while, the satellite will be stabilized, and the rotation will slow down. However, gravity forces from the Sun, the moon and different planets will still cause the satellite to rotate. In addition, magnetic field disturbances and other factors will influence the satellite. A dissipative controller has been investigated for the detumbling control, but unless more testing is done, a more familiar B-dot controller will be used. For the stabilization, optimization controllers have been evaluated. No definite decision for the choice of stabilization controller is made.

Radio and Antenna Systems

The radio system is a major part of the Telemetry, Tracking and Command (TT&C) system. The satellite will receive commands from the ground station and transmit payload and housekeeping data to it through two radio links. The radio waves will have center frequencies of approximately 437 MHz and 146 MHz which are both located within amateur radio frequency bands. The transmitted power will be less than a watt for both links.

The data will be modulated onto the radio waves as a stream of 9600 bits per second (bps) by applying a Frequency Shift Keying (FSK) modulation scheme. In the other end of the link, be it the satellite or the ground station, the received signal will be demodulated such the data can be further processed. Moreover, the transmitted data will be protected by an error-correcting code (ECC). On the 437 MHz frequency there will also be a beacon transmitting a simple morse code. This will be helpful as a first means for the ground station to locate the satellite.

Since the satellite will be at different angles as seen from the ground station, it is important that the radiated power is somewhat evenly distributed in all directions (the satellite antenna has a near-isotropic pattern). This will ensure a robust and long-lasting link and it will also be a redundancy in case of an ADCS failure when one cannot know the attitude of the satellite. To achieve a near-isotropic pattern it has been decided to use two crossed dipole antennas, one for each frequency. The antennas will be made of measuring tape and wrapped around the satellite during launch. 30 minutes after the satellite has been ejected from the P-POD the antennas will be deployed such that one antenna is located on the nadir plane and the other at the zenith plane.

The radio system is also designed to take into account atmospheric and ionospheric propagation effects such as attenuation and Faraday rotation (rotation of the electric field vector through ionized gases). The ground station will track the satellite with mechanically steered antennas and account for the Doppler shift due to the mutual speed between the ground station and the satellite.

For more information about the radio systems, see [1,2]

Power System

The power system of the NUTS satellite is divided into two parts; a power distribution system, the backplane, and a power condition system, the Electrical Power System (EPS). The power system is a crusial part of the satellite because without power the satellite will not be able to operate.

The backplane is the medium used to connect the different modules together, and provides communication and power interfaces for the rest of the system. It also provides protection by allowing individual modules to be isolated, reset or powered off. The design is based on a single I^2C bus with bus repeaters for each submodule, with the ability to isolate individual modules from the system in case of a malfunction. Power is distributed with dual 3:3V and dual 5V busses working in active redundancy, ensuring continued operation should a voltage converter fail. Power distribution for each module consists of three parts: power supply or-ing, current-limit switch and power monitor, and is integrated into the backplane. The state of the power switches and bus repeaters are controlled from two master modules, and a watchdog timer ensures return to a default state should both master modules be disabled. For more information on the backplane, see [3].

The Electrical Power System (EPS) is an important part of the NUTS satellite in that it provides power to the rest of the systems of the satellite. The primary tasks of the EPS module are to charge the batteries with energy from the solar cells efficiently and safely, and to provide two regulated 3.3 V and two regulated 5.0 V power rails to the backplane connector. The secondary task is to provide telemetric data about provided power from the solar cells and the state-of-charge of the batteries.

The EPS module provides an efficient charging of the batteries through the SPV1040, which integrates the strategies of maximum power point tracking (MPPT) and constant current - constant voltage (CCCV). The MPPT strategy tracks the solar cell's most efficient operating point, as the temperature and irradiation of the cell changes, utilizing the maximum potential of the solar cells. The CCCV strategy allows efficient and safe charging of the $LiFePO_4$ batteries. The power to the backplane is provided with fixed output voltage step-down converters from Texas Instruments. Power monitoring is implemented by using current monitor sensors on the output of each charger circuit and the batteries. For more information on the EPS, see [4].

The NUTS satellite will carry 18 GaAs solar cells for energy harvesting and 2 x battery pack consisting of 4 $LiFePO_4$ cells.

The Infrared Camera Payload

The main payload will be an infrared camera for observing gravity waves in the upper atmosphere. Gravity waves, created by air blowing over mountains and weather phenomena, propagate throughout the atmosphere and drive the large scale flows in the middle atmosphere. Despite this their properties are poorly understood, mainly due to a lack of observational data. At an altitude of about 90 km in the atmosphere we find a layer of OH molecules that emit short-wave infrared radiation. When gravity waves propagate through this layer wave patterns in the radiation intensity are observed.

By taking series of images with an infrared camera pointing towards the Earth, the wavelength, direction and phase speed of the gravity waves can be observed through intensity variations in the OH airglow layer. This method for gravity wave observation has been employed in several ground based observations, but never for a satellite mission. Due to limited space, weight, power and downlink data rate, several challenges arise.

A lightweight uncooled InGaAs camera with suitable optics and readout electronics must either be designed or bought commercially off-the-shelf. Due to the large scale of the gravity waves, a relatively low resolution will be sufficient.

To ensure sufficient image quality for compression and interpretation, some processing of the images will be performed on-board the satellite before transmission. For InGaAs detectors, long exposure times are usually required due to the high noise levels in the detector. But since the satellite has such a high speed, the images with long exposure will be distorted by motion blur. In order to avoid this, the camera will be operating as a video camera with low frame rate, and series of images with short exposures will be combined into blur-free images with improved signal-to-noise ratio. To be able to download more images per pass over the ground station, series of combined images will be compressed with a video compression scheme before transmission. The image processing and compression algorithms require relatively constant and known speed, low rotation and good pointing stability.

More information about the camera and image processing can be found in [TBD: cite Marianne's thesis]

[1] S. Marholm, *Specialization project: Antenna Systems for NUTS*. Norwegian University of Science and Technology (NTNU), 2011.

[2] S. Marholm, *Masters thesis: Antenna Systems for NUTS*. Norwegian University of Science and Technology (NTNU), 2012.

[3] D. De Bruyn, *Masters thesis: Power Distribution and Conditioning for a Small Student Satellite.* Norwegian University of Science and Technology (NTNU), 2011.

[4] L. Jacobsen, *Masters thesis: Electrical Power System of the NTNU Test Satellite.* Norwegian University of Science and Technology (NTNU), 2012.

Appendix B

Presentation Held at the European CubeSat Symposium

Observation of Gravity Waves From a Small Satellite by Means of an Infrared Camera

Snorre Rønning and Marianne Bakken

NTNU Norwegian University for Science and Technology

Introduction

The NTNU Test Satellite (NUTS)

- Double cubesat
- Launch planned in 2014
- 10-15 master students at NTNU working on it
- Infrared camera payload for observation of gravity waves

Outline:

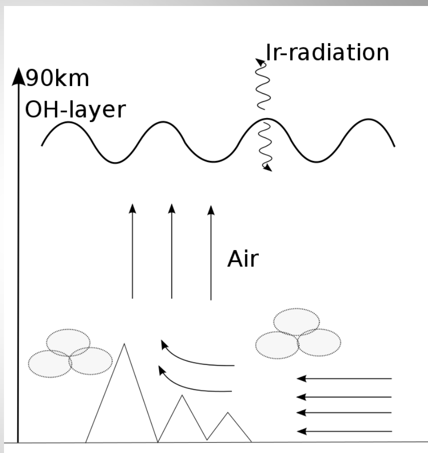
- Atmospheric Gravity Waves
- The Infrared Camera
- The Motion Blur Problem

Atmospheric Gravity Waves

What are they?

How to study them?

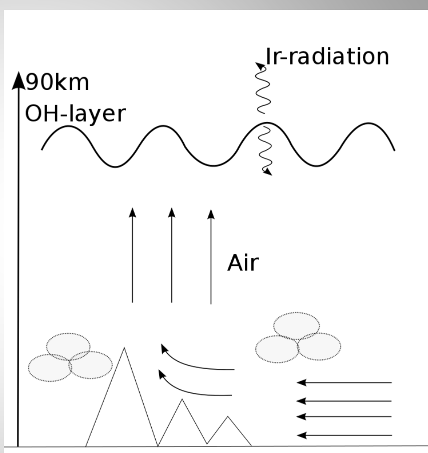
- Hydroxyl (OH) layer at 90 km
  - SWIR, 1434 nm and 1381 nm
- Wavelength, phase speed, intensity



Atmospheric Gravity Waves

Why do we want to study them?

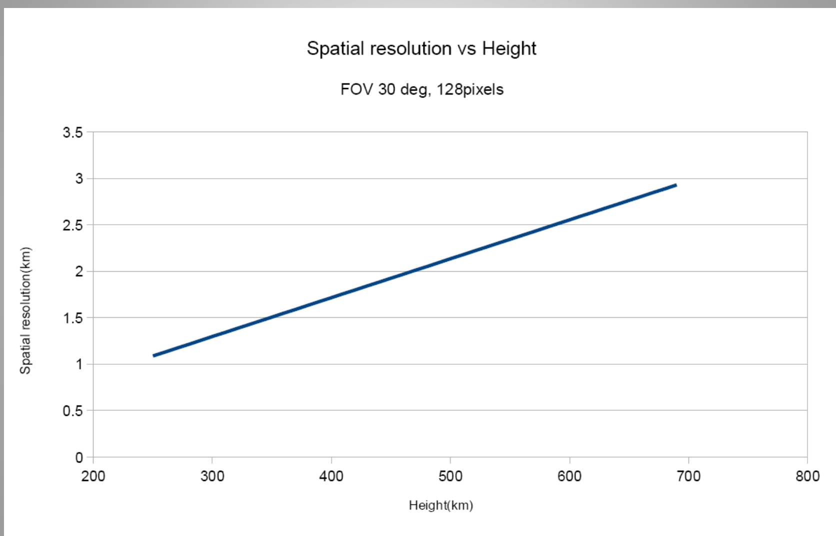
- Momentum deposition
- Global meridional circulation
- Weather models
- Global coverage by satellite



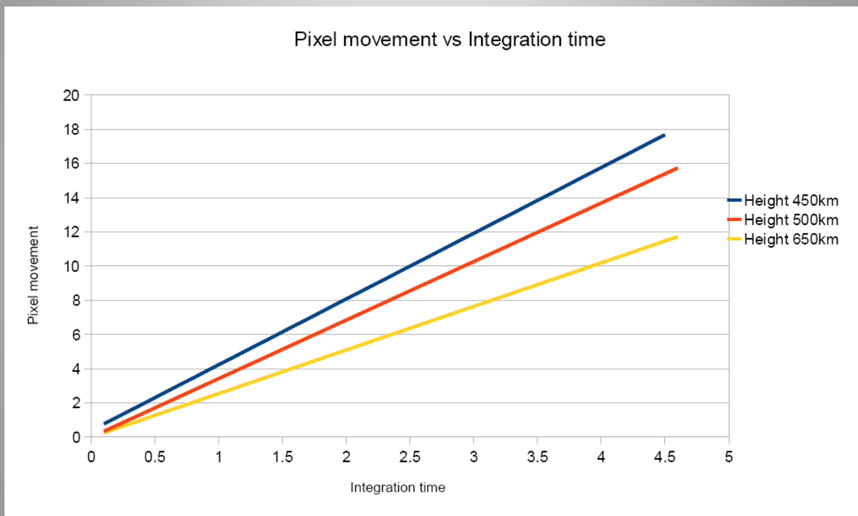
The Infrared Camera

- Camera type
  - Uncooled detector required
  - Suitable detector type: InGaAs
- Camera requirements
  - Resolution: Sufficient to distinguish the wave patterns
  - Integration time: A trade-off between noise and motion blur
  - Optics: Wide Field-of-view
- Background radiation blocked by atmospheric absorption

Camera and Satellite Parameters



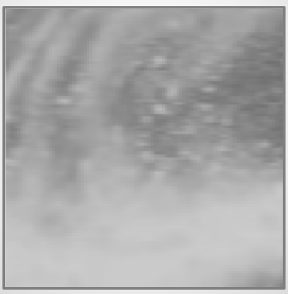
Camera and Satellite Parameters



The Motion Blur Problem

- High speed and long exposure → motion blur
- MATLAB simulations (Alt.: 450 km, Resolution: 128x128 px, Field of view: 30°):

Original image



Blurred image
(4 px = 1 s exposure)

Blurred image
(9 px = 2 s exposure)

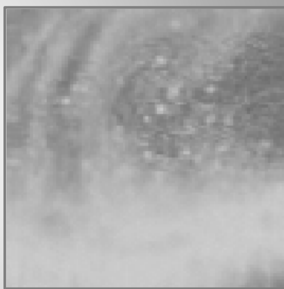
Marianne Bakken, mariba@stud.ntnu.no http://nuts.iet.ntnu.no

Restoration of Motion Blur

- The image content can be restored with signal processing
- Images restored in MATLAB by means of Richardson-Lucy algorithm

Original image

Blurred image
(9 px = 2s exposure)



Restored image

QUESTIONS?

Sponsors:

Kongsberg Seatex, supporting the trip to this conference

KONGSBERG

NTNU Department of Electronics and Telecommunications

NAROM

NORDIC SEMICONDUCTOR

Wondercode

ASTRIUM AN EADS COMPANY

agi

ATMEL

Appendix C

Calculation of Camera and Satellite Parameters

In the following, formulas used for computation of auxiliary parameters needed for the camera requirements are presented.

C.1 Orbital Mechanics

The orbital period of a satellite is given by Kepler's third law [45]:

$$T = 2\pi\sqrt{\frac{(r_e + h_{sat})^3}{\mu}}, \tag{C.1}$$

Where $\mu = 398601\,\frac{\mathrm{km}^3}{\mathrm{s}^2}$ is the gravitational parameter of the Earth, $r_e = 6371\,\mathrm{km}$ is the mean earth radius and h_{sat} is the altitude of the satelitte. Then the orbital velocity follows as the circumference O of the orbit divided by T:

$$V = \frac{O}{T} = \frac{2\pi(r_e + h_{sat})}{T} \tag{C.2}$$

C.2 Imaging Parameters

Figure C.1 illustrates the geometry used in the following calculations. It is seen that the image coverage x can be expressed by

$$x = \frac{ud}{f} \qquad [\mathrm{m}] \tag{C.3}$$

where f is the focal length, u is the distance to the target, and d is the detector size.

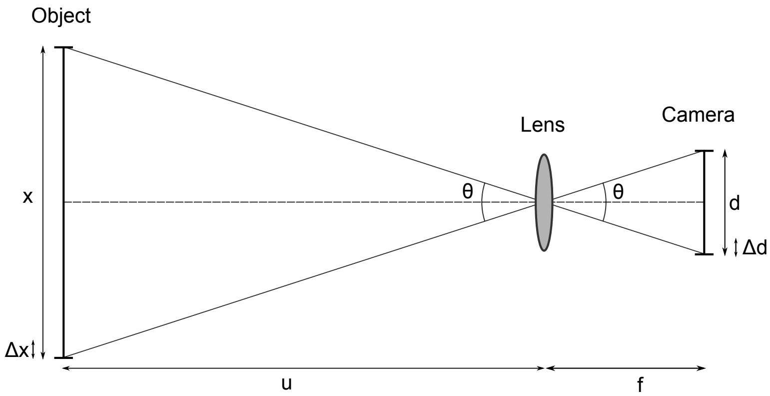


Figure C.1: A simple illustration of imaging geometry

The pixel pitch will be given by $\Delta d = \frac{d}{N_{px}}$. The rezel size can thus be calculated as

$$\Delta x_{rezel} = \frac{u \Delta d}{f} \qquad \text{[m]} \quad \text{(C.4)}$$

(C.3) and (C.4)may also be expressed by the field of view θ instead of the focal length, which might be more intuitive:

$$x = 2u \tan \frac{\theta}{2} \qquad \text{[m]} \quad \text{(C.5)}$$

$$\Delta x_{rezel} = \frac{2u \tan \frac{\theta}{2}}{N_{px}} \qquad \text{[m]} \quad \text{(C.6)}$$

The *image coverage in wavelengths* can be defined as:

$$x_\lambda = \frac{x}{\lambda_{\text{gravity waves}}} \qquad \text{(C.7)}$$

The spatial resolution is also limited by the optics is according to [8]:

$$\Delta x_{optics} = \frac{u \lambda_{rad}}{D} \qquad \text{[m]} \quad \text{(C.8)}$$

The satellite velocity with respect to the airglow layer will be slightly reduced compared to the orbital velocity given in (C.2):

$$V' = \frac{O'}{T} = \frac{2\pi(r_e + h_{OH})}{T} \qquad \text{[m/s]} \quad \text{(C.9)}$$

Where $h_{OH} = 89$ km is the altitude of the OH airglow layer. This must be taken into account when computing the image velocity, i.e. how many pixels the camera will move per second:

$$V_{im} = \frac{V'}{\Delta x} \qquad \text{[px/s]} \quad \text{(C.10)}$$

Payload Calculations			
Parameters		**Results**	
Satellite			
Earth radius [km]	6371	Orbital Period [s] (4.1)	5,61E+03
Gravitational parameter [km^3/s^2]	398601	Velocity [m/s] (4.2)	7,64E+03
Height satellite [km]	450	Velocity w.r.t airglow layer [m/s] (5.10)	7,24E+03
Distance satellite-airglow [km]	361		
Camera spec			
FOV spec. [deg]	45	Image coverage [m] (5.6)	2,99E+05
Array size [px x px]	128	Spatial resolution [m] (5.7)	2,34E+03
Integration time [sec]	0,03	Image velocity [px/s] (5.11)	3,1
		Relative image velocity [image width/s] (5.11)	2,42E-002
Gravity waves			
Mean Wavelength gravity waves [km]	20	Spatial resolution in wavelengths [m] (5.8)	0,16
Minimum Wavelength gravity waves [km]	15	Image coverage in wavelengths [ ](5.9)	14,95
Mean Speed gravity waves [m/s]	25		
Height airglow layer [km]	89		
IR_wavelength [m]	1,50E-06		
Video:			
Frame rate [fps]	10	Time per frame	1,00E-001
Number of frames	50	Movement per exposure [px]	1,02E-001
Reset time [s]	1,00E-002	Movement per frame [px]	3,10E-001
		Movement per total image [px]	1,55E+001
		Max integration time	9,00E-002
		Corresponding "still image integration time"	4,50E+001
		Total imaging duration [s]	5,00E+000

Figure C.2: Spreadsheet for calculations

C.3 Spreadsheet for Calculations

The equations relating the satellite parameters and the camera specification with the resulting imaging and video parameters were inserted into a spreadsheet to easier get an overview of the system. Figure C.2 shows the spreadsheet for a fixed set of parameters. The intact spreadsheet with formulas can be found in the enclosed CD/zip-file.

Appendix D

Datasheets for Cameras and Sensors

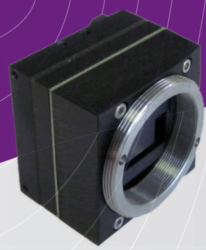
Preliminary

Imagine the invisible

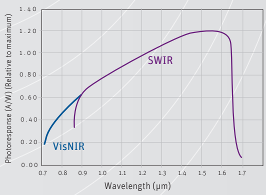
Modules & components

XSW-640

High resolution
uncooled SWIR infrared module



Ready-to-integrate SWIR infrared module consuming ultra-low power



Xenics' XSW-640 camera module is an extremely compact and versatile core for easy and swift integration in your SWIR imaging configuration.

The XSW-640 camera module detects short wave infrared radiation between 0.9 and 1.7 µm with a wide dynamic range.

Typical OEM applications include infrared imaging for man–portable and unmanned (airborne and land-based) vehicle payloads, night vision, border security, Search & Rescue and more.

Designed for use in

Person identification

Camouflage detection

Vision enhancement: looking through haze with SWIR

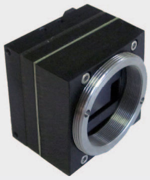
Key features

- Made in Europe
- High resolution
- Easy connectivity
- Small 20 µm pixel pitch

OEM applications

- Night vision
- SWIR sights
- Border security
- Driver assistance
- Search & Rescue

Ready-to-integrate



Specifications

Array Specifications	XSW-640
Array type	Uncooled InGaAs
Spectral band	0.9 to 1.7 µm
# pixels	640 x 512
Pixel pitch	20 µm
Pixel operability	› 99 %

Module Specifications	XSW-640
Lens (not included)	
Optical interface	Multiple lens mounts
Imaging performance	
Frame rate	50 Hz
A to D conversion resolution	14 bit
Interfaces	
Connector type	Samtec 40 pin QTE
Digital output	Digital output following BT.601-6/BT.656-5 standard Parallel uncompressed video data
Digital control	Serial LVCMOS 3 V interface using XSP protocol
Trigger	In and out
GPIO	Extended GPIO via I2C
Power requirements	
Power consumption	2.0 W
Power supply	3.3 V
Physical characteristics	
Shock	70 G, 2 ms halfsine profile
Vibration	4.5 G, (5 Hz to 500 Hz)
Ambient operating temperature	0 °C to 50 °C
Dimensions	45 W x 45 H x 20 L mm^3
Weight module	60 g

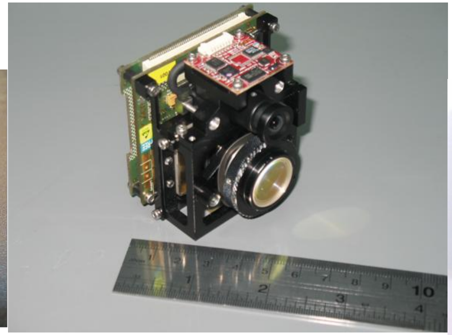
www.xenics.com
www.sinfrared.com

Xenics Headquarters
Ambachtenlaan 44, BE-3001 Leuven, Belgium
T +32 16 38 99 00 • sales@xenics.com

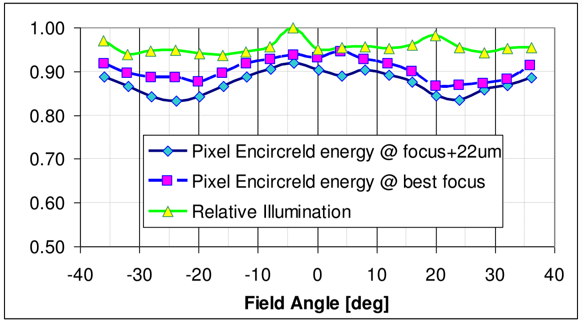
Xenics Infrared Solutions
ISO 9001:2008 certified

OPTIGO SWIR BUILDING BLOCKS

A-thermalized lens



Miniature Image processor



- Weight < 150gr
- -30C < T < +60C
- Dual Band
  - ✓CMOS/CCD
  - ✓SWIR
- Cam-Link™ outputs
- 1000 FPS , 320×256
- 250 FPS, 640×512
- P < 2.5 Watt

- Real Time image processing @ 1Gbps
- P < 6W
- Serial Comm outputs
- Output Video interface
- USB-2 outputs
- Nav. Data inputs
- Battery operated

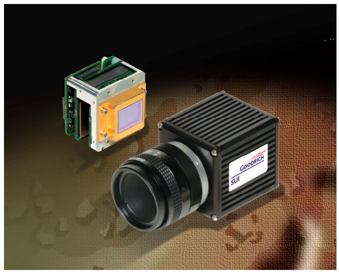
G. Tidhar, Proc SPIE 6940 (2008)

SU640HSX-1.7RT

Preliminary

Mil-Rugged High Sensitivity InGaAs SWIR Camera with Advanced Dynamic Range Enhancements

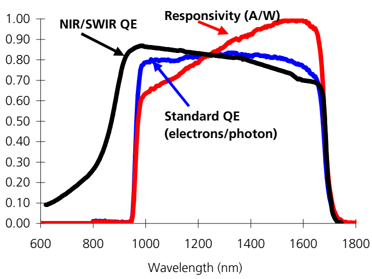
GOODRICH



The compact **SU640HSX-1.7RT** is a Mil-Rugged InGaAs video camera featuring high-sensitivity and wide operating temperature range. It provides real-time daylight to low-light imaging in the Short Wave Infrared (SWIR) wavelength spectrum for persistent surveillance, laser detection, and penetration through fog, dust, and smoke. In addition, the camera employs on-board Automatic Gain Control (AGC), proprietary dynamic-range enhancement technology, and built-in non-uniformity corrections (NUCs), allowing it to address the challenges of urban night imaging **without blooming**. Simultaneous RS170 analog and Camera Link® digital output provide a means for plug-and-play video and high quality 12-bit images for image processing or transmission. The light-weight, compact size, and low power consumption enables easy integration into surveillance systems, whether hand-held, mobile, or aerial. Optional **NIR/SWIR technology** is available to extend the sensitivity of Goodrich cameras down to 0.7 µm, offering the advantage of both Near Infrared (NIR) and Short Wave Infrared wavelength response.

APPLICATIONS

- Low-light level imaging
- Covert surveillance with passive 24 hr/7 day operation
- Driver Vision Enhancement (DVE)
- Imaging through atmospheric obscurants
- OEM version for easy integration into UAVs, handheld, or robotic systems
- Laser spotting and tracking



FEATURES

- Highest sensitivity available in 0.9 to 1.7 µm spectrum; NIR/SWIR, from 0.7 to 1.7 µm
- Images from partial starlight to direct sun illumination
- 640 x 512 pixel format, 25 µm pitch
- Compact OEM module size < 3.8 in^3
- Enclosed module size < 9.5 in^3
- Low power, < 2.7 W at 20 °C
- All solid-state InGaAs imager
- On-board non-uniformity corrections
- Simultaneous digital & analog outputs
- Advanced Automatic Gain Control (AGC)
- Selectable contrast enhancement modes
- Region of Interest (ROI) windowing mode
- FCC CE and MIL-461F certified
- MIL-STD-810G certified
- Operation from -40 °C to 70 °C
- Environmental Stress Screening

3490 U.S. Route 1 • Princeton, New Jersey 08540
Phone: (609) 520-0610 • Fax: (609) 520-0638
www.sensorsinc.com • sui_sales@goodrich.com

seeing **beyond**™

Doc. No. 4110-0252 Rev. 2

Effective Date: 13 JUN 2011

MECHANICAL SPECIFICATIONS

Model:	Enclosed	OEM
Module dimensions Width x Height x Depth	2.1 x 2.1 x 2.55 inches 52.1 x 52.1 x 64.7 mm (with I/O connectors, no lens or mount)	1.64 x 1.5 x 1.6 inches 42 x 38 x 41 mm
Weight (no lens)	< 270 g	< 90 g (analog out)
Lens Mount	C-mount adapter in M42x1 mount	M42x1 mount bracket
Included Lens	f/1.4, 50 mm, 18° FOV width, M42x1-mount	none
Camera Link Connector	3M SDR26 Connector	none
I/O Connector	3M SDR14 Connector	none
Interface Connector	Not applicable	Harwin Datamate M80-5020805
Pixel Pitch	25 µm	
Focal Plane Array Format	640 x 512 pixels	
Active Area	16 mm x 12.8 mm x 20.5 mm diagonal	

ENVIRONMENTAL & POWER SPECIFICATIONS

Operating Case Temperature	-40 °C to 70 °C
Storage Temperature	-54 °C to 85 °C
Humidity	100 % Non-condensing
Power Requirements: AC Adapter Supplied DC Voltage Typical Power	 100-240 VAC, 47-63 Hz +9-16 V <2.7 W at 20 °C ambient, <4 W @ 40 °C
Functional Shock, Thermal Shock, Random Vibration, Storage Temperature, Temperature/Altitude Combine, Humidity, Transportability	MIL-STD-810G compliant
Conducted & Radiated Emissions	CE FCC Part 15, MIL-STD-461F
Mean Time Between Failure	>10,000 hours, MIL-HDBK-217F N2
Fungus-Inert Material	MIL-HDBK-454B

ELECTRICAL SPECIFICATIONS

Optical Fill Factor	100 %
Spectral Response	Standard, 0.9 µm to 1.7 µm NIR/SWIR, 0.7 µm to 1.7 µm
Quantum Efficiency	Standard, > 65 % from 1 µm to 1.6 µm NIR/SWIR, > 65 % from 0.9 µm to 1.6 µm
Mean Detectivity, D* [1]	> 3.51 x 10^{13} cm√Hz/W
Noise Equivalent Irradiance [1]	< 3.46 x 10^{8} photons/cm^2·s
Noise (RMS)[1]	< 50 electrons
Full Well (Typical) In OPR0	12 x 10^{6} electrons
Dynamic Range (Typical)[4]	> 3000:1
Operability [2]	> 99 %
Exposure Times[3]	60 µs to 33 ms in 12 steps
Image Correction	2-point (offset and gain) pixel by pixel, user selectable
Digital Output Format	12 bit Camera Link® (SDR connector for enclosed version, ribbon for OEM version)
Analog Output Format	Buffered EIA170 compatible video, 30 fps (both versions)
Digital Output Frame Rate	30 fps (faster frame rates in windowed operation)
Scan Mode	Continuous, or 4 externally triggered modes, or ROI windowing mode

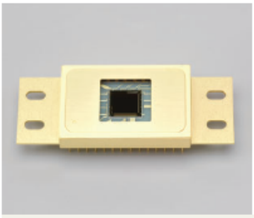
[1] λ = 1.55 µm, exposure time = 33.2 ms, Highest Sensitivity OPR setting, no lens, x1 digital gain with enhancement, AGC, and correction off.
[2] The fraction of pixels with responsivity deviation between +/- 35 % from the mean
[3] The 12 pre-configured exposure times include factory stored non-uniformity corrections. Additional exposure times are programmable via RS-232 commands.
[4] In high dynamic range OPR settings.

seeing beyond™

HAMAMATSU
PHOTON IS OUR BUSINESS

PRELIMINARY

InGaAs area image sensor



G11097-0707S

Image sensor with 128 × 128 pixels developed for two-dimensional infrared imaging

The G11097-0707S has a hybrid structure consisting of a CMOS readout circuit (ROIC: readout integrated circuit) and back-illuminated InGaAs photodiodes. Each pixel is made up of an InGaAs photodiode and a ROIC electrically connected by an indium bump. A timing generator in the ROIC provides an analog video output and AD-TRIG output which are easily obtained by just supplying a master clock (MCLK) and master start pulse (MSP) from external digital inputs.
The G11097-0707S has 128×128 pixels arrayed at a 50 μm pitch and their signals are read out from a single video line. Light incident on the InGaAs photodiodes is converted into electrical signals which are then input to the ROIC through indium bumps. Electrical signals in the ROIC are converted into voltage signals by charge amplifiers and then sequentially output from the video line by the shift register. The G11097-0707S is hermetically sealed in a metal package together with a one-stage thermoelectric cooler to deliver low-cost yet highly stable operation.

Features

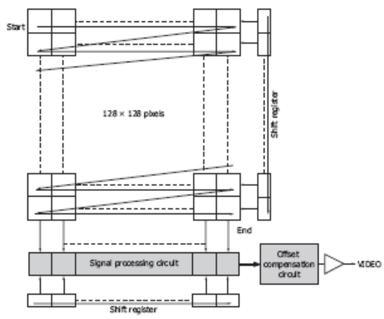
- **Spectral response range: 0.95 to 1.7 μm**
- **Excellent linearity by offset compensation**
- **High sensitivity: 1600 nV/e-**
- **Simultaneous charge integration for all pixels (global shutter mode)**
- **Simple operation (built-in timing generator)**
- **One-stage TE-cooled**

Applications

- **Thermal imaging monitor**
- **Laser beam profiler**
- **Near infrared image detection**
- **Foreign object detection**

Block diagram

A sequence of operation of the readout circuit is described below.
In the readout circuit, the charge amplifier output voltage is sampled and held simultaneously at all pixels during the integration time determined by the low period of the master start pulse (MSP) which is as a frame scan signal. Then the pixels are scanned and their video signals are output.
Pixel scanning starts from the basing point at the upper left in the right figure. The vertical shift register scans from top to bottom in the right figure while sequentially selecting each row.
For each pixel on the selected row, the following operations are performed:
① Transfers the sampled and held optical signal information to the signal processing circuit as a signal voltage.
② Resets the amplifier in each pixel after having transferred the signal voltage and transfers the reset voltage to the signal processing circuit.
③ The signal processing circuit samples and holds the signal voltage ① and reset voltage ②.
④ The horizontal shift register scans from left to right in the right figure, and the voltage difference between ① and ② is calculated in the offset compensation circuit. This eliminates the amplifier offset voltage in each pixel. The voltage difference between ① and ② is output as the output signal in the form of serial data.
The vertical shift register then selects the next row and repeats the operations from ① to ④. After the vertical shift register advances to the 128th row, the MSP, which is a frame scan signal, goes low. After that, when the MSP goes high and then low, the reset switches for all pixels are simultaneously released and the next frame integration begins.



www.hamamatsu.com

1

Structure

Parameter	Specification	Unit
Image size	6.4 × 6.4	mm
Cooling	One-stage TE-cooled	-
Number of total pixels	16384 (128 × 128)	pixels
Number of effective pixels	16384 (128 × 128)	pixels
Pixel size	50 × 50	μm
Pixel pitch	50	μm
Package	28-pin metal (refer to dimensional outline)	-
Window	Borosilicate glass with anti-reflective coating	-

Absolute maximum ratings (Ta=25 °C, unless othewise noted)

Parameter	Symbol	Value	Unit
Supply voltage	Vdd	-0.3 to +5.5	V
Clock pulse voltage	V(MCLK)	Vdd + 0.5 max.	V
Operating temperature	Topr	-10 to +60	°C
Storage temperature	Tstg	-20 to +70	°C

Note: This product must be used within the range of the absolute maximum ratings. Product quality may suffer if any item of the absolute maximum ratings is exceeded even momentarily.

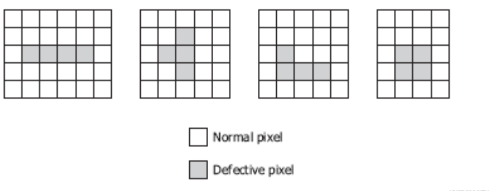
Electrical and optical characteristics (Element temperature=25 °C, Ta=25 °C, Vdd=5 V, PD_bias=4.5 V)

Parameter	Symbol	Condition	Min.	Typ.	Max.	Unit
Spectral response range	λ		-	0.95 to 1.7	-	μm
Peak sensitivity wavelength	λp		-	1.55	-	μm
Photo sensitivity	S	λ=λp	0.7	0.8	-	A/W
Conversion efficiency	CE	Cf=0.1 pF	-	1600	-	nV/e^-
Saturation charge	Qsat		-	1.3	-	Me^-
Saturation output voltage	Vsat		-	2	-	V
Photo response non-uniformity*1	PRNU	After subtracting dark current, Integration time 5 ms	-	±10	±20	%
Dark voltage	VD		-	20	100	V/s
Dark current	ID		-	2	10	pA
Dark signal non-uniformity	DSNU		-	20	50	V/s
Readout noise	Nr	Integration time 10 ms	-	600	1200	μV rms
Dynamic range	DR		1600	3300	-	-
Defective pixel*2	-		-	-	1	%

*1: Measured at one-half of the saturation, excluding first and last pixels

*2: Pixels with photo response non-uniformity (integration time 5 ms), readout noise, or dark current higher than the maximum value
One or less cluster of four or more contiguous defective pixels

<Examples of four contiguous defective pixels>



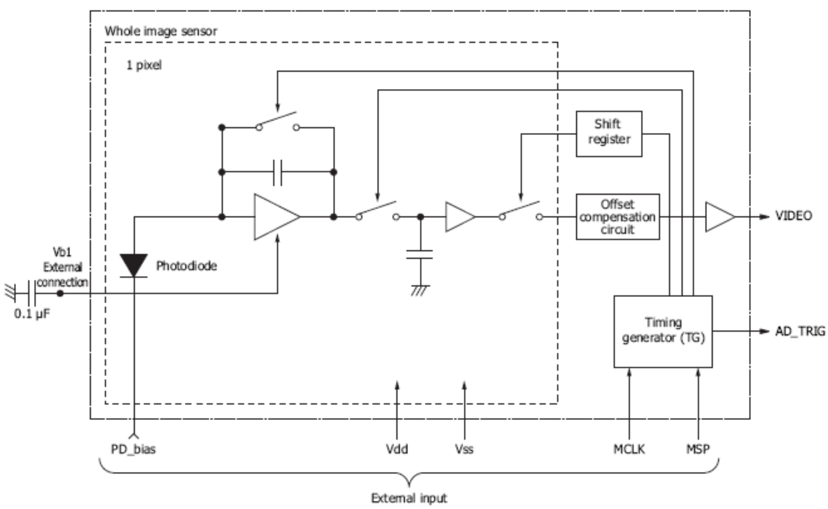
InGaAs area image sensor | **G11097-0707S**

Electrical characteristics (Ta=25 °C)

Parameter		Symbol	Min.	Typ.	Max.	Unit
Supply voltage		Vdd	4.9	5	5.1	V
Supply current		I(Vdd)	-	30	60	mA
Ground		Vss	-	0	-	V
Element bias		PD_bias	4.4	4.5	4.6	V
Element bias current		I(PD_bias)	-	-	1	mA
Clock frequency		f	-	-	40	MHz
Clock pulse voltage	High	V(MCLK)	Vdd - 0.5	Vdd	Vdd + 0.5	V
	Low		0	0	0.5	V
Clock pulse rise/fall times		tr(MCLK) tf(MCLK)	0	10	12	ns
Clock pulse width		tpw(MCLK)	10	-	-	ns
Start pulse voltage	High	V(MSP)	Vdd - 0.5	Vdd	Vdd + 0.5	V
	Low		0	0	0.5	V
Start pulse rise/fall times		tr(MSP) tf(MSP)	0	10	12	ns
Start pulse width		tpw(MSP)*3	0.001	-	10	ms
Start (rise) timing		t1	10	-	-	ns
Start (fall) timing		t2	10	-	-	ns
Output setting time		t3	-	-	50	ns
Video output voltage	High	V_H	-	3.2	-	V
	Low	V_L	-	1.2	-	
Video data rate		fV	-	f/8	-	MHz

*3: Integration time max.=10 ms

Equivalent circuit



HAMAMATSU
PHOTON IS OUR BUSINESS

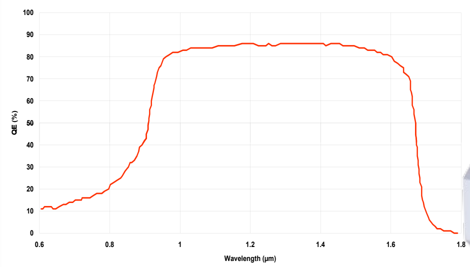
3

ANDOR™ TECHNOLOGY

iDus InGaAs 1.7 µm

600 nm - 1.7 µm
Spectroscopy InGaAs PDA

Spectroscopy



Features and Benefits

- **0.6 to 1.7 µm**
  Operating wavelength range
- **Peak QE of > 85%**
  High detector sensitivity
- **TE cooling to -90°C** [*1]
  Negligible dark current without the inconvenience of LN_2
- **UltraVac™** [*2]
  Permanent vacuum integrity, critical for deep cooling and sensor performance
- **Single window design**
  Delivers maximum photon throughput
- **25 µm pixel width option**
  Ideal for high-resolution NIR spectroscopy
- **Simple USB 2.0 connection**
  USB plug and play – no controller box. Inputs & Outputs: External Trigger, Fire and Shutter TTL readily accessible. I^2C for the more adventurous user
- **Software selectable output amplifiers**
  Allows user to optimize operation with choice of High Dynamic Range (HDR) or High Sensitivity (HS) modes of operation
- **Minimum exposure time of 1.4 µs**
  Enables higher time-resolution and minimization of dark current contribution for applications with reasonable signal level

Andor's iDus InGaAs detector array for Spectroscopy

Andor's iDus InGaAs 1.7 array detector series provides the most optimized platform for Spectroscopy applications up to 1.7 µm. The TE-cooled, in-vacuum sensors reach cooling temperatures of -90°C where best Signal-to-Noise ratio can be achieved.
Indeed dark current will improve moderately below -90°C where scene black body radiation will dominate, while Quantum Efficiency of the sensor will be greatly impacted at these lower temperatures and lead to a lower Signal-to-Noise ratio.

Specifications Summary

Active pixels	**512 or 1024**
Pixel size (W x H)	**25 x 500 or 50 x 500 µm**
Pixel well depth (typical) **High Dynamic Range mode** **High Sensitivity mode**	 **170 Me^-** **5 Me^-**
Maximum cooling [*1]	**-90°C**
Maximum spectra per sec	**193**
Read noise (typical)	**580 e^-**
Dark current (typical)	**11.7 ke^-/pixel/sec**
Minimum exposure time	**1.4 µs**

Key Specifications •3

Model number	DU490A	DU491A	DU492A
Sensor options	512 pixels, 25 µm pitch	1024 pixels, 25 µm pitch	512 pixels, 50 µm pitch
Active pixels	512	1024	512
Pixel size	25 x 500	25 x 500	50 x 500
Cooler type		DU	
Wavelength range		600 nm - 1.7 µm	
Minimum exposure time •4		1.4 µs	
Minimum temperatures •5 Air cooled Coolant chiller, coolant @ 16°C , 0.75l/min Coolant chiller, coolant @ 10°C, 0.75l/min		 -70°C -85°C -90°C	
Max spectra per second (100 kHz readout)	193	97	193
System window type		Single quartz window, uncoated	
Digitization		16 bit	

Advanced Specifications •3

Dark current ke⁻/pixel/sec @ max cooling •6	10.1	10.1	18.9
Pixel well depth (Me⁻) •7 High Dynamic Range mode High Sensitivity mode		 170 5	
Read noise (e⁻) •8 High Sensitivity mode High Dynamic Range mode		 580 8150	
Sensitivity (e⁻/count) High Dynamic Range mode High Sensitivity mode		 2800 90	
Blemishes •9	0	≤10	≤5
Linearity		Better than 99%	
Insertion delay from external trigger		2.95 µs ± 0.1 µs	

Have you found what you are looking for?

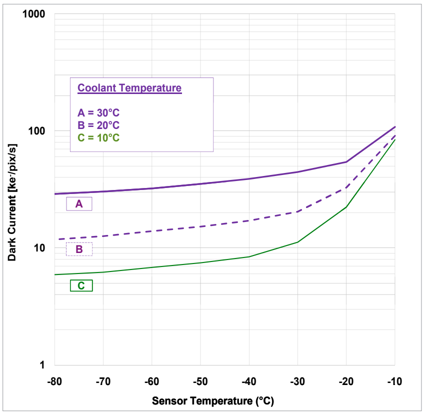
Need extended NIR response? The iDus InGaAs 2.2 µm series offer three array formats.

Need to work below 1 µm? The iDus 401 & 420 series offer Deep Depletion NIR optimized sensors.

Need a customized version? Please contact us to discuss our Customer Special Request options.

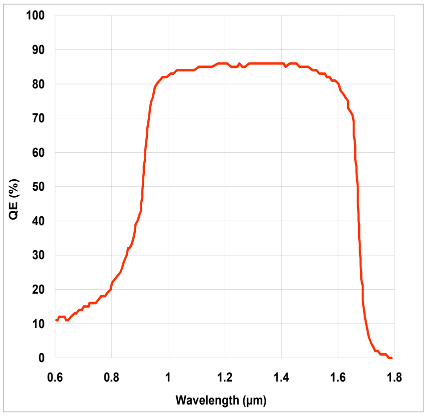
The iDus InGaAs series combines seamlessly with Andor's research grade Shamrock Czerny-Turner spectrographs. These instruments are available on request with gold or silver coated optics for optimised NIR operations.

System Dark Current v Temperature [10]

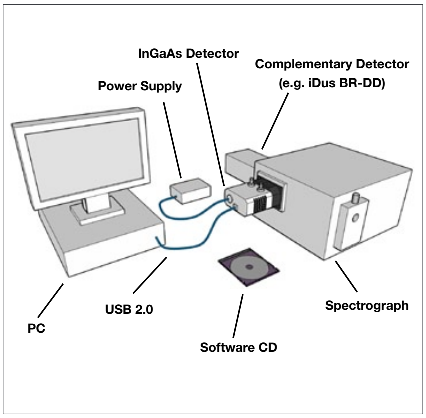


Quantum Efficiency Curve [11]

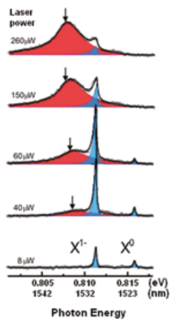
20°C

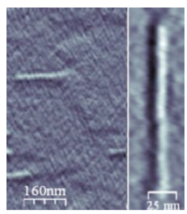


Typical Setup



Typical Application





Conductivity behaviour study of Single Semiconductor Quantum Wires. Spectra acquired with an Andor InGaAs array detector.

Courtesy of: Dr. Benito Alén, Instituto de Microelectrónica de Madrid, Spain.

Creating The Optimum Product for You

How to customize the iDus InGaAs 1.7 :

Step 1.

The iDus InGaAs 1.7 comes with 3 options for sensor types. Please select the sensor which best suits your needs.

Step 2.

Please select which software you require.

Step 3.

For compatibility, please indicate which accessories are required.

DU 490A- 1.7

example shown

Step 1.

Choose sensor array

490: 25 µm x 250 µm, 512 pixel array
491: 25 µm x 250 µm, 1024 pixel array
492: 50 µm x 250 µm, 512 pixel array

Step 2.

The iDus InGaAs requires at least one of the following software options:

Solis for Spectroscopy A 32-bit application compatible with 32 and 64-bit Windows (XP, Vista and 7) offering rich functionality for data acquisition and processing. AndorBasic provides macro language control of data acquisition, processing, display and export. Control of Andor Shamrock spectrographs and a very wide range of 3rd party spectrographs is also available, see list below.

Andor SDK A software development kit that allows you to control the Andor range of cameras from your own application. Available as 32 and 64-bit libraries for Windows (XP, Vista and 7) and Linux. Compatible with C/C++, C#, Delphi, VB6, VB.NET, LabVIEW and Matlab.

Step 3.

The following accessories are available:

XW-RECR Coolant re-circulator for enhanced cooling performance.
ACC-XW-CHIL-160 Oasis 160 Ultra Compact Chiller Unit (tubing to be ordered separately)
ACC-6MM-TUBING-2xxxxM 6 mm tubing option for ACC-XW-CHIL-160
SR-ASZ-0033 SR-750 Adapter Flange for InGaAs detector.
SR1-ASZ-8044 SR-163 Adapter Flange for InGaAs detector
ACC-SD-VDM1000 Shutter Driver for NS25B Bistable Shutter (not needed for Shamrock spectrographs)
ACC-SHT-NS25B Bistable Shutter, Standalone (not needed for Shamrock spectrographs)

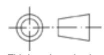
Spectrograph Compatibility

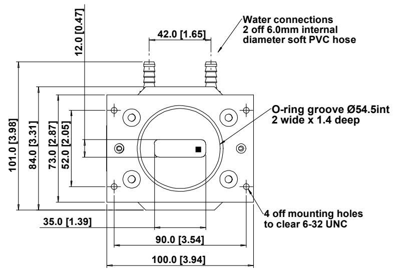
The InGaAs series is fully compatible with Andor's Shamrock spectrograph (163 - 750 nm focal lengths) family. Shamrock spectrographs are supplied with Al/MgF_2 mirror coatings as standard, gold or silver optics are avaialable on request. Spectrograph mounting flanges and software control are available for a wide variety of 3rd party spectrographs including, McPherson, JY/Horiba, PI/Acton, Chromex/Bruker, Oriel/Newport, Photon Design, Dongwoo, Bentham, Solar TII and others.

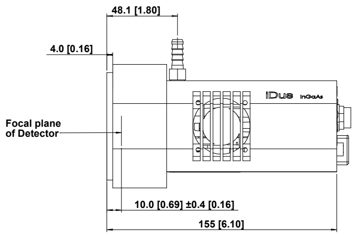
InGaAs mounted on a Shamrock 163 mm spectrograph, ideal combination for NIR Photoluminescence Spectroscopy.

Product Drawings

Dimensions in mm [inches]

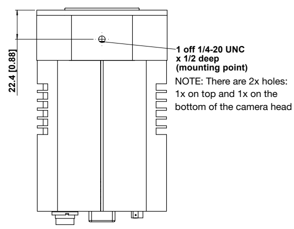






■ = position of pixel 1,1

Weight: 2 kg [4 lb 8 oz]



Mounting hole locations

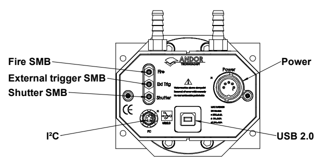
Connecting to the InGaAs

Camera Control
Connector type: USB 2.0

TTL / Logic
Connector type: SMB, provided with SMB - BNC cable
1 = Fire (Output), 2 = External Trigger (Input), 3 = Shutter (Output)

I²C connector
Compatible with Fischer SC102A054-130
1 = Shutter (TTL), 2 = I²C Clock, 3 = I²C Data, 4 = +5 Vdc, 5 = Ground

Minimum cable clearance required at rear of camera
90 mm



Rear connector panel

Applications Guide

	DU490-1.7	DU491-1.7	DU492-1.7
NIR Absorption-Transmission-Reflection Spectroscopy	✓	✓	✔
NIR Photoluminescence	✓	✓	✔
1064 nm Raman Spectroscopy	✓	✔	✓

✓ = Suitable
✔ = Optimum

Order Today

Need more information? At Andor we are committed to finding the correct solution for you. With a dedicated team of technical advisors, we are able to offer you one-to-one guidance and technical support on all Andor products. For a full listing of our local sales offices, please see: andor.com/contact

Our regional headquarters are:

Europe
Belfast, Northern Ireland
Phone +44 (28) 9023 7126
Fax +44 (28) 9031 0792

Japan
Tokyo
Phone +81 (3) 3518 6488
Fax +81 (3) 3518 6489

North America
Connecticut, USA
Phone +1 (860) 290 9211
Fax +1 (860) 290 9566

China
Beijing
Phone +86 (10) 5129 4977
Fax +86 (10) 6445 5401

Items shipped with your camera:

1x 2m BNC - SMB conection cable
1x 3m USB 2.0 cable Type A → Type B
1x Set of Allen keys (7/64" & 3/32")
1x Power supply (PS-25) with mains cable
1x Quick launch guide
1x CD containing Andor user guides
1x Individual system performance booklet
1x CD containing either Solis software or SDK (if ordered)

Footnotes: **Specifications are subject to change without notice**

1. Typically obtainable at ambient temperature of 20°C, coolant chillers operating with 10°C coolant @ 0.75l/min.
2. Assembled in a state-of-the-art facility, Andor's UltraVac™ vacuum process combines a permanent hermetic vacuum seal (no o-rings), with a stringent protocol and proprietary materials to minimize outgassing. Outgassing is the release of trapped gases that would otherwise degrade cooling performance and potentially cause sensor failure.
3. Figures are typical unless otherwise stated.
4. The InGaAs sensor starts to 'open' to light up to approximately 1 µs before the rising edge of the Fire pulse. It then starts to 'close' to light up to 1 µs before the falling edge of Fire. This ensures that the camera is 100% responsive by the time the Fire pulse has risen and closed by the falling edge. These figures only need to be taken into account for extremely short exposures.
5. The standard PS-25 power supply is suitable for air cooling and deep cooling. Measured at ambient temperature of 20°C.
6. Measured using 10°C water and 10°C target/scene.
7. At exposures below 20 µs, well depth will be reduced by approximately 1/3 of typical value stated.
8. Noise is measured on a single pixel.
9. Blemishes as stated by sensor manufacturer.
10. The coolant temperature is also representative of the scene temperature that the camera is exposed to during these measurements.
11. Quantum efficiency of the sensor at 20°C as measured by the sensor manufacturer.

Minimum Computer Requirements:
- 3.0 GHz single core or 2.4 GHz multi core processor
- 2 GB RAM
- 100 MB free hard disc to install software (at least 1 GB recommended for data spooling)
- USB 2.0 High Speed Host Controller capable of sustained rate of 40 MB/s
- Windows (XP, Vista and 7) or Linux

Operating & Storage Conditions
Operating 0°C to 20°C ambient (air cooling)
Operating 0°C to 30°C ambient (deep cooling)
Relative Humidity < 70% (non-condensing)
Storage Temperature -25°C to 50°C

Power Requirements
110 - 240 Vac, 50 - 60 Hz

SInGaAs17SS 1111 R2

Appendix E

MATLAB code

E.1 Download Capacity

This code is the result of a cooperation between Sigvald Marholm and the author.

pass_duration.m

```
% This is a script used to gain knowledge about the duration of the passes

%clear all
close all

% SIMULATION INPUT GOES HERE (AND IN STK)
altitudes = [350 500 650];          % different orbital altitudes
thresholds = [21 28 34];            % different threshold elevation angles
%thresholds = [0 0 0];              % to simulate TOTAL visibity

                                    % one angle for each altitude.
schemes = strvcat('r','g','b');     % Different plot colors (and dots etc.)

% Loads the data. If the variable already exist, the program will assume it
% is from the previous run, and save time by not loading it again. The
% "clear all" command at the top of the script must be commented for that.
if(~exist('data'))
    for i=1:length(altitudes)
        data{i}= read_stk_elev([num2str(altitudes(i)) 'c.txt']);
        disp(['Datafile for altitude ' num2str(altitudes(i)) ' km loaded.']);
    end
else
    disp(['Data already loaded. If not true; run "clear data".']);
end

for ind=1:length(altitudes)

    intervals = threshold_stk_elev(data{ind},thresholds(ind));

    if(isempty(intervals))  % Elevation never passes threshold
        start=0;
        stop=0;
    else
        start = datenum(intervals(:,1:6));
        stop = datenum(intervals(:,7:12));
    end

    duration_d = stop-start;          % This is the duration of the passes in days.
    duration_h = duration_d*24;       % ... and in hours
    duration_m = duration_h*60;       % ... and in minutes.
    duration_s = duration_m*60;       % ... and in seconds.

    N = size(intervals,1); % The number of passes during the
```

```
                                % simulation time (1 week)

    % Length on the passes (in ascending order) are store to outside the
    % loop in this variable for all altitudes.
    dur{ind} = sort(duration_m);

    disp(['Simulation for altitude ' num2str(altitudes(ind)) ' km finished.']);

end

% Creating legends
legs = [];
for alt=1:length(altitudes)
    legs = strvcat(legs,[num2str(altitudes(alt)) ' km']);
end

for ind=1:length(altitudes)
    altstr = num2str(altitudes(ind));
    durt = dur{ind};
    mindur = num2str(min(durt));
    maxdur = num2str(max(durt));
    meddur = num2str(median(durt));
    meandur = num2str(mean(durt));
    disp(['Pass duration information for ' altstr ' km altitude:']);
    disp(['  Minimum duration: ' mindur ' min.']);
    disp(['  Maximum duration: ' maxdur ' min.']);
    disp(['  Mean duration:    ' meandur ' min.']);
    disp(['  Median duration:  ' meddur ' min.']);

    figure
    hist(durt,5);
    title(['Distribution of duration for ' altstr ' km altitude']);
    xlabel('Duration [min]');
    ylabel('Number of passes');
end
```

plot_data_down.m

```
% This is a script that plots the downloaded data per average day for
% different orbital heights and threshold elevation angles.

%clear all
close all

% SIMULATION INPUT GOES HERE (AND IN STK)
altitudes = [350 500 650];       % different orbital altitudes
thresholds = 0:90;             % different threshold elevation angles
zoom = 15:40;                    % Make a plot for these angles only
schemes = strvcat('r','g','b'); % Different plot colors (and dots etc.)

% D and I is hold the average kB downloaded during an average day and an
% average pass, respectively, for various altitudes and thresholds.
D = zeros(length(altitudes),length(thresholds));
I = D;

% Loads the data. If the variable already exist, the program will assume it
% is from the previous run, and save time by not loading it again. The
% "clear all" command at the top of the script must be commented for that.
if(~exist('data'))
    for i=1:length(altitudes)
        data{i}= read_stk_elev([num2str(altitudes(i)) 'c.txt']);
        disp(['Datafile for altitude ' num2str(altitudes(i)) ' km loaded.']);
    end
else
    disp(['Data already loaded. If not true; run "clear data".']);
end

for alt=1:length(altitudes)

    for thr=1:length(thresholds)

        intervals = threshold_stk_elev(data{alt},thresholds(thr));

        if(isempty(intervals))  % Elevation never passes threshold
            start=0;
            stop=0;
        else
            start = datenum(intervals(:,1:6));
            stop = datenum(intervals(:,7:12));
        end

        duration_d = stop-start;        % This is the duration of the passes in ...
             days.
```

```
            duration_h = duration_d*24;      % ... and in hours
            duration_m = duration_h*60;      % ... and in minutes.
            duration_s = duration_m*60;      % ... and in seconds.

            T = sum(duration_s);    % Total duration of pass in seconds
            N = size(intervals,1); % The number of passes during the
                                    % simulation time (1 week)

            R = 9600;         % Assuming bitrate of 9600 bps
            W = R*T;          % Bits downloaded during simulation time (1 week)
%           W = (W/8)/1024; % W is now kB per week.
            W = W/(1024^2); % W is no Mb (megabit) per week.
            D(alt,thr) = W/7; % D is Mb per average day.
            I(alt,thr) = W/N; % I is Mb download per average pass.

        end

        disp(['Simulation for altitude ' num2str(altitudes(alt)) ' km finished.']);

end

% Creating legends
legs = [];
for alt=1:length(altitudes)
    legs = strvcat(legs,[num2str(altitudes(alt)) ' km']);
end

% Plot Data per Day
figure
hold on
grid on
for alt=1:length(altitudes)
    plot(thresholds,D(alt,:),schemes(alt,:));
end
xlabel('Minimum Elevation Angle [degree]');
ylabel('Average Downlink Capacity [Mb/day]');
% title('Average Data per Day for Different Criterea');
legend(legs);

% Plot Data per Day (with zoom)
first = find(thresholds==zoom(1));
last = find(thresholds==zoom(end));

figure
hold on
grid on
for alt=1:length(altitudes)
    plot(thresholds(first:last),D(alt,first:last),schemes(alt,:));
end
xlabel('Minimum Elevation Angle [degree]');
ylabel('Average Downlink Capacity [Mb/day]');
% title('Average Data per Day for Different Criterea');
legend(legs);
% a = [18,24,30];
% b = [D(1,19) D(2,25) D(3,31)];
a = [21,28,34];
b = [D(1,22) D(2,29) D(3,35)];
plot(a,b,'ok');

% Plot Data per Pass
figure
hold on
grid on
for alt=1:length(altitudes)
    plot(thresholds,I(alt,:),schemes(alt,:));
end
xlabel('Minimum Elevation Angle [degree]');
ylabel('Average Downlink Capacity [Mb/pass]');
% title('Average Data per (Usable) Pass for Different Criteria');
legend(legs);
```

plot_stk_elev.m

```
function plot_stk_elev(data)

N = size(data,1);

% Remove spurious elevations caused by a bug in STK
data = stk_remove_spurious(data);

% This line eliminates all rows that are filled with just zeros. They are
% separators to separate between the passes.
data(all(ismember(data,[0 0 0 0 0 0 0]),2),:)=[];
```

```

% Takes in the date vector (first six elements) and converts it to MATLABs
% serial date format.
time_axis = datenum(data(:,1:6));

plot(time_axis,data(:,7));
datetick('x','HH:MM');

end
```

read_stk_elev.m

```
function [data sep] = read_stk_elev(fname)
% Function to read STK data files containing elevation. For now it only
% supports elevation. The usage is as follows:
%
% You set up a scenario with a satellite and a ground station facility in
% STK and you export the elevation as seen from the ground station to a
% .dat file. You have to export ONLY the elevation and not azimuth and
% range since read_stk_elev() doesn't support that. If done correctly, all
% the lines in the .dat file should look like the following:
%
%    Elevation (deg) 47   1.1.2012 03:36:15,000    2,27845993613939
%
% The first number is simple an index, the number of the sample.
% Subsequently follows the date and time, and finally the elevation angle
% in degrees.
%
% There is an issue with the format of the .dat file STK produces; they
% cannot be read by MATLAB. To fix that, open them in a text-editor and
% save them with UTF-8 encoding before using read_stk_elev().
%
% Import the data to a MATLAB matrix called data by
%
%    data = read_stk_elev(fname)
%
% where fname is a variable (of type string) containing the filename (or
% URL) of the .dat-file exported from STK.
%
% data will be a N x 7 matrix where N is the number of samples in the
% .dat-file from STK. The first dimension is the number of samples whereas
% the second dimension holds different information (fields) about the
% samples:
%
%    1 - Year
%    2 - Month
%    3 - Day
%    4 - Hour
%    5 - Minute
%    6 - Second
%    7 - Elevation
%
% For example, data(23,7) yields the elevation of the 23rd sample.
%
% Note that between the passes, when the satellite is not empty, STK
% produces an empty (separator) sample. For these samples all the fields
% are set to 0 in the data matrix. In some cases, it will be useful for the
% user to know which samples are separators. Instead of testing for it, the
% user can get a list of separators, sep, in the following way:
%
%    [data sep] = read_stk_elev(fname)
%
% Made by:
%    Marianne Bakken <mariba@stud.ntnu.no>, MSc. thesis for NUTS 2012
%    Sigvald Marholm <marholm@stud.ntnu.no>, MSc. thesis for NUTS 2012
%
% Copyright 2012, NUTS - NTNU Test Satellite
%



% A typical line may look like the following, followed by regexp blocks and
% name of variables they will be stored into.

% Elevation (deg)    47   1.1.2012 03:36:15,000    2,27845993613939
% ---%s----- -%s--   %d   ---%s--- -----%s-----    --------%s--------
%                  index   date        time              elevation

% The two first blocks are just waste
% The %d can be used as index in a matrix
% The three last blocks are read as strings to be parsed later

pattern='%s %s %d %s %s %s';
```

```

fid = fopen(fname);
unparsed = textscan(fid, pattern);
fclose(fid);

% Read out the regexp blocks
index = unparsed(3);
date = unparsed(4);
time = unparsed(5);
elevation = unparsed(6);

% MATLAB stores them as a 1-element cell-array for some reason.
% Read out the one cell element.
index = index{1};
date = date{1};
time = time{1};
elevation = elevation{1};

N = size(index,1); % The output is a Nx7 matrix

% Reads out the data in a Nx7 matrix of the form data(index,parameter)
% where index is the same index as in the file, parameter is a value of
% the following:
%
%   1 - year
%   2 - month
%   3 - day
%   4 - hour
%   5 - minute
%   6 - second
%   7 - elevation
%
% So for example data(23,:) yields the time and elevation for sample 23.

% Initialize data matrix and sep vector
data = zeros(N,7);
sep = [];

for i=index.'

    date_ = date(i);
    time_ = time(i);
    elevation_ = elevation(i);

    % For some reason MATLAB stores this into 1-dimensional cell arrays as well
    date_ = date_{1};
    time_ = time_{1};
    elevation_ = elevation_{1};

    if(isempty(elevation_))
        % STK Produces an empty row with just an index when the satellite
        % is not visible. Reg.exp. functions run into trouble if they try
        % to parse an empty string. The convention will be that the matrix
        % is left with zeros at these places. A vector sep is available as
        % an output for the user. This is an easy way for the user to keep
        % track of where these samples are.
        sep = [sep i];
    else

        % Replace delimiters with space
        date_ = regexprep(date_,'\.',' ');          % '1.1.2012' => '1 1 2012'
        time_ = regexprep(time_,':',' ');           % '03:36:15,000' => '03 36 ...
             15,000'

        % Replace decimal , with .
        time_ = regexprep(time_,',','.');           % '03 36 15,000' => '03 36 ...
             15.000'
        elevation_ = regexprep(elevation_,',','.'); % '2,27845993613939' => ...
             '2.27845993613939'

        % Read out 'day month year' from date_
        temp = textscan(date_,'%d %d %d','CollectOutput',1);
        temp = temp{1};

        data(i,1) = temp(3);    % year
        data(i,2) = temp(2);    % month
        data(i,3) = temp(1);    % day

        % Read out 'hour minute second' from time_
        temp = textscan(time_,'%f %f %f','CollectOutput',1);
        temp = temp{1};

        data(i,4) = temp(1);    % hour
        data(i,5) = temp(2);    % minute
        data(i,6) = temp(3);    % second

        % Read out 'elevation' from elevation_
        temp = textscan(elevation_,'%f','CollectOutput',1);
        temp = temp{1};

```

```
            data(i,7) = temp(1);     % elevation

        end

    end

    end
```

stk_remove_spurious.m

```
function data = stk_remove_spurious(data)
% This function remove spurios spikes that STK produces due to a bug in the
% elevation angle at the beginning/end of a pass.

N = size(data,1);

% This is a logical vector (filled with ones and zeros) that indicate where
% there is an increase from a given sample to the next one.
increase = data(1:end-1,7)<data(2:end,7);

% This logical vector indicate where there is a decrease from a given
% sample to the next one.
decrease = [~increase; 0];

% Now it represents whether a given sample has increased w.r.t. the
% previous sample. I.e. if increase(34)==1 then data(34,7) is larger than
% the previous elevation, data(33,7).
increase = [0;increase];

% This is a logical vector representing all the separators
sep = all(ismember(data,[0 0 0 0 0 0 0]),2);

% These are logical vectors representing the samples before and after the
% separators
presep = [sep(2:end);0];
postsep = [0;sep(1:end-1)];

% This is a logical vector representing the elements where the sample
% before the separator increased w.r.t. the one before that. That is a
% spurious elevation angle, a bug from STK. We know that it is below 1, we
% will set it to 0 to fix it.
% spur = or(and(presep,increase),and(postsep,~increase));
spur = or(and(presep,increase),and(postsep,decrease));
data(spur,7)=0;

end
```

threshold_stk_elev.m

```
function intervals = threshold_stk_elev(data,threshold)
% Given a set of elevation data from STK simulations and read with
% read_stk_data(), this function determines the time intervals when the
% elevation is higher than a certain threshold, i.e. 10 degrees.
%
% Usage:
%
%   intervals = threshold_stk_elev(data,threshold)
%
% where data is the output data matrix from read_stk_data() and threshold
% is the elevation threshold.
%
% The output is a matrix where each interval is represented by one row of
% 12 elements. The first 6 elements of the row are the year, month, day,
% hour, minute and second of the start of the period. The 6 last elements
% represent the end of the period in the same manner.
%
% Example output:
%
% intervals =
%
%   1    1    2012    13   43   23   1    1    2012    13   48   43
%   1    1    2012    15   32   11   1    1    2012    15   42   12
%
% This example shows that the data contains two intervals where the
% elevation is higher than the user-specified threshold elevation. The
% second interval, for example, starts on 1.1.2012 15:32:11 and ends
```

```
% 15:42:12 the same day.

intervals = [];

% The algorithm works by iterating through all samples and detect when the
% elevation increase above, or decrease below, the given threshold. When
% that happens, it writes to the intervals array. It remembers whether or
% not the elevation is above the threshold by a flag called above.

above = 0;  % Nominally not in an interval
N = size(data,1);

data=stk_remove_spurious(data);

for i=1:N

    if( (data(i,7)>=threshold) && ~above  )

        if(~all(data(i,1:6)==[0 0 0 0 0 0]))    % Ignore empty samples

            % Sometimes STK produces spurious output (unbelievably enough).
            % When the elevation passes below 1 degree it starts typing them in
            % exponential form, i.e. 7,442341e-001 except that sometimes it
            % doesn't get the exponent right. Instead the exponent becomes 0,
            % and there is a single sample before the satellite goes down where
            % the elevation is supposedly 7 degrees. This function handles that
            % by checking that the next sample is also above the threshold.

            if (i+1)<=N % Check that there is a next sample and ignore if not.

                if( data(i+1,7)>=threshold )

                    % An interval starts at this sample. Add a new row in
                    % intervals and add the time in the six first columns.

                    intervals(end+1,1:6) = data(i,1:6);
                    above = 1;
                end
            end
        end

    elseif( (data(i,7)<threshold) && above )

        % An interval stops at this sample. Add the time to the six last
        % columns in the last row.
        intervals(end,7:12) = data(i,1:6);
        above = 0;

    elseif( all(data(i,1:6)==[0 0 0 0 0 0]) && above )

        % if threshold = 0 the above mechanism to detect an end of the
        % interval doesn't work since STK doesn't save samples with
        % negative elevation, and hence all samples will have values >=0.
        % Instead, STK saves a separator sample between every pass where
        % all fields (year, etc.) are simply zero. When such an sample
        % occur, and the above flag is still set, you know that the
        % PREVIOUS sample was the last one.

        intervals(end,7:12) = data(i-1,1:6);
        above = 0;

    end

end
```

E.2 DPCM Algorithm and Test Script

autocorrcoeffs_2D.m

```
function [corr_coeffs power]= autocorrcoeffs_2D(M)
    %input: Array M (1D, 2D or 3D), zero-mean
    %output: - vector corr_coeffs with correlation coefficients
    %          (along 1st, 2nd and 3rd dimension)
    %        - estimated power (variance) of M
    N = length(M(:)); %total number of samples
    shifts = [1 0];
    r = zeros(1,2); %initiating covariance vector
    for i = 1:2
        r(i) = sum(sum(M(1:end-shifts(1),1:end-shifts(2))...
        .*M(shifts(1)+1:end,shifts(2)+1:end)))/N;
```

```
        shifts = circshift(shifts,[0 1]);
    end
    power = sum(sum(sum(M.^2)))/N;
    corr_coeffs = r/power;
end
```

autocorrcoeffs_3D.m

```
function [corr_coeffs power] = autocorrcoeffs_3D(M)
    %input: 3D array, zero-mean
    %output: - vector corr_coeffs with correlation coefficients
    %          (along 1st, 2nd and 3rd dimension)
    %        - estimated power (variance) of M

    N_tot = length(M(:)); %total number of samples
    shifts = [1 0 0]; %indicating which dimension to shift
    r = zeros(1,3); %initiating covariance vector
    for i = 1:3
        N_r = prod(size(M)-shifts); %total number of samples in estimation
        r(i) = sum(sum(sum(M(1:end-shifts(1),1:end-shifts(2),1:end-shifts(3))...
        .*M(shifts(1)+1:end,shifts(2)+1:end,shifts(3)+1:end))))/N_r;
        shifts = circshift(shifts,[0 1]);
    end
    power = sum(sum(sum(M.^2)))/N_tot;
    corr_coeffs = r/power;
end
```

deadzone_quantizer.m

```
function q_level = deadzone_quantizer(x,q_param)
    %% --Quantization with uniform deadzone quantizer
    % input: x - input signal (one sample)
    %        dz_offset - (two-sided) width of dead-zone interval
    %        range - one-sided range
    %        L - number of quantization levels
    % output: quantization level for sample x
    %% -------
    L = q_param.L;
    range = q_param.range;
    dz_offset = q_param.dz_offset;

    % Compute step size outside deadzone
    step_size = (2*range-dz_offset)/(L-1);
    % Compute maximum quantization level
    %max_q_level = (dz_offset+step_size*(L-1)-step_size*0.5)/2;
    max_q_level = range-0.5*step_size;
    %clip input outside the quantizer range:
    x_clipped = min(abs(x),max_q_level);
    %map to one-sided uniform midtread with step_size=1
    x_mapped = max(0,(x_clipped-0.5*(dz_offset-step_size))/step_size);
    % add sign back:
    x_signed = sign(x).*x_mapped;
    % shift signal to get all-positive values:
    x_pos = x_signed+L+1;
    % round to nearest integer and add sign:
    q_level_pos = round(x_pos);
    % shift signal back
    q_level = q_level_pos-(L+1);
end
```

dpcm3D_decode.m

```
function f_rec = dpcm3D_decode(e_q,pred_coeffs,q_param,MC_shift)
    imsize = size(e_q);
    f_rec = zeros(imsize);

    q_param_nopred = q_param{1};
    q_param2D = q_param{2};
```

```
    q_param3D = q_param{3};

    a2D = pred_coeffs{1};
    a3D = pred_coeffs{2};

    x_mc = MC_shift; %Assume shift in the x-direction common to all frames

    % Coordinates for 3D prediction window:
    m = [1 0 -x_mc]; % image x-direction
    n = [0 1 0]; % image y-direction
    p = [0 0 1]; % time direction
    % Coordinates for 2D prediction window:
    m2D = [1 0 1]; % image x-direction
    n2D = [0 1 1]; % image y-direction

    for t = 1: imsize(3) % For each frame
        for y = 1: imsize(1) % For each row
            f_p = 0;    %predicted value
            for x = 1 : imsize(2) % For each pixel
                if ((~(y <= 1 || x < 1 || x == imsize(2)) && (t <= 1))... %due ...
                        to prediction window
                    || (~(y <= 1 || x == imsize(2)) && x >= (imsize(2)-x_mc))) ...
                        %due to motion compensation
                    pred_mode = '2D';
                elseif (~(y <= 1 || x < 1 || x == imsize(2)) && (t > 1))
                    pred_mode = '3D';
                else
                    pred_mode = 'none';
                end

                %--Reconstruction
                    if(strcmp(pred_mode, 'none'))
                        e_q_rec(y,x,t) = inv_uniform_quantizer(e_q(y,x,t), ...
                            q_param_nopred);
                    elseif(strcmp(pred_mode, '2D'))
                        e_q_rec(y,x,t) = inv_deadzone_quantizer(e_q(y,x,t), ...
                            q_param2D);
                    elseif(strcmp(pred_mode, '3D'))
                        e_q_rec(y,x,t) = inv_deadzone_quantizer(e_q(y,x,t), ...
                            q_param3D);
                    end
                % e_q_rec(y,x,t) = e_q(y,x,t); %testing without Q
                f_rec(y,x,t)  = e_q_rec(y,x,t)  + f_p;

                %--Prediction---
                %Update prediction signal (for next sample)
                f_p_temp = 0;
                if(strcmp(pred_mode, '2D'))
                    %2D prediction for the first frame:
                    for l = 1:length(a2D)
                        f_p_temp = f_p_temp+a2D(l)*f_rec(y-n2D(l),x-m2D(l)+1,t);...
                            %indexes relative to current px!
                    end
                elseif(strcmp(pred_mode, '3D'))
                    %3D prediction for the rest:
                    for k = 1:length(a3D)
                        %f(y,x+1,t) %testing
                        %f(y,x+1,t-1)%testing
                        f_p_temp = ...
                            f_p_temp+a3D(k)*f_rec(y-n(k),x-m(k)+1,t-p(k));%%indexes...
                            relative to current px!
                    end
                end

                f_p = f_p_temp;
            end
        end
    end
end
```

dpcm3D_encode.m

```
function e_q = dpcm3D_encode(f, pred_coeffs, q_param, MC_shift)
    %DPCM encoder with 3D predictor, order decided by prediction window W
    %Input signal: f (3D matrix), input image sequence (video), must have zero ...
        mean!
    %Output signals: e_q (3D matrix), quantized prediction error
    %                a (double vector), prediction coefficients

    if(~iscell(q_param))
        quantize = 0;
    else
        quantize = 1;
```

```
        q_param_nopred = q_param{1};
        q_param2D = q_param{2};
        q_param3D = q_param{3};
    end

    a2D = pred_coeffs{1};
    a3D = pred_coeffs{2};

    x_mc = MC_shift; %Assume shift in the x-direction common to all frames

    f_size=size(f);
    f_rec = zeros(f_size); %internal reconstructed signal
    % Coordinates for 3D prediction window:
%without MC
%     m = [1 0 0]; % image x-direction
%     n = [0 1 0]; % image y-direction
%     p = [0 0 1]; % time direction
%     % Coordinates for 2D prediction window:
%     m2D = [1 0 1]; % image x-direction
%     n2D = [0 1 1]; % image y-direction
    m = [1 0 -x_mc]; % image x-direction
    n = [0 1 0]; % image y-direction
    p = [0 0 1]; % time direction
    % Coordinates for 2D prediction window:
    m2D = [1 0 1]; % image x-direction
    n2D = [0 1 1]; % image y-direction

    %Encoding:
    for t = 1: f_size(3) % For each frame
        for y = 1: f_size(1) % For each row
            f_p = 0;    %predicted value
            for x = 1 : f_size(2) % For each pixel
                if ((~(y <= 1 || x < 1 || x == f_size(2)) && (t <= ...
                    1))... %due to prediction window
                    || (~(y <= 1 || x == f_size(2)) && x >= ...
                        (f_size(2)-x_mc))) %due to motion compensation
                    pred_mode = '2D';
                elseif (~(y <= 1 || x < 1 || x == f_size(2)) && (t > 1))
                    pred_mode = '3D';
                else
                    pred_mode = 'none';
                end
                %Difference signals
                e = f(y,x,t)-f_p;      %prediction error

                %--Quantization and inverse quantization:
                if(quantize == 0) %quantization turned off
                    e_q(y,x,t) = e;
                    e_q_rec(y,x,t) = e;
                elseif(strcmp(pred_mode, 'none'))
                    e_q(y,x,t) = uniform_quantizer(e,q_param_nopred);
                    e_q_rec(y,x,t) = inv_uniform_quantizer(e_q(y,x,t), ...
                        q_param_nopred);
                elseif(strcmp(pred_mode, '2D'))
                    e_q(y,x,t) = deadzone_quantizer(e,q_param2D);
                    e_q_rec(y,x,t) = inv_deadzone_quantizer(e_q(y,x,t), ...
                        q_param2D);
                elseif(strcmp(pred_mode, '3D'))
                    e_q(y,x,t) = deadzone_quantizer(e,q_param3D);
                    e_q_rec(y,x,t) = inv_deadzone_quantizer(e_q(y,x,t), ...
                        q_param3D);
                end

                %--Reconstruction (internal decoding)
                f_rec(y,x,t)  = e_q_rec(y,x,t) + f_p;

                %--Prediction---
                %Update prediction signal (for next sample)
                f_p_temp = 0;
                if(strcmp(pred_mode, '2D'))
                    %2D prediction for the first frame:
                    for l = 1:length(a2D)
                        f_p_temp = ...
                            f_p_temp+a2D(l)*f_rec(y-n2D(l),x-m2D(l)+1,t); ...
                            %indexes relative to current px!
                    end
                elseif(strcmp(pred_mode, '3D'))
                    %3D prediction for the rest:
                    for k = 1:length(a3D)
                        f_p_temp = ...
                            f_p_temp+a3D(k)*f_rec(y-n(k),x-m(k)+1,t-p(k));%%indexes...
                             relative to current px!
                    end
                end

                f_p = f_p_temp;
            end
        end
    end
```

dpcm_demo.m

```
% DPCM parameter testing script
 clear
 close all

%--- Prepare parameters and inputs ---

%Define images:
make_test_images_dpcm %make test images sequence without motion compensation
%frame_shift = [0 3];
%make_test_images_dpcm_mc %make test image sequence with motion compensation

im_mat = video_mat;
nr_px = numel(video_mat);
nr_px_frame = size(video_mat,1)^2;
nr_frames = size(video_mat,3);
nr_rows = size(video_mat,1);
%MC_shift = round(frame_shift(2));
MC_shift = 0;
%Prediction windows:
W3D = [1 0 0; 0 1 0; 0 0 1];
W2D = [1 0; 0 1; 1 1];
%Subtract mean:
im_mat_mean = mean(mean(im_mat));
for i = 1:size(im_mat,3)
    im_mat0(:,:,i) = im_mat(:,:,i)-im_mat_mean(1,1,i);
end
%Compute prediction coefficients
pred_coeffs = cell(1,2);
[a2D power2D] = est_2Dpredcoeffs(im_mat0(:,:,1), W2D,'debug');
[a3D power3D] = est_3Dpredcoeffs(im_mat0, W3D,'debug');
pred_coeffs{1} = a2D;
pred_coeffs{2} = a3D;


%---Estimation of Quantization parameters-----

L1 = 11;
L2 = 7;
L3 = 7;
dz_l = 1.25;
q_param = set_q_param(L1,L2,L3, dz_l,im_mat0,pred_coeffs, MC_shift,'debug');

%-----DPCM Encode----------
e_q3D = dpcm3D_encode(double(im_mat0), pred_coeffs, q_param,MC_shift);
%SR-encoding
e_q3D_coded = sr3D_encode(e_q3D);
rate = length(e_q3D_coded)/length(e_q3D(:))

%-----Decode----------
e_q3D_decoded = e_q3D; %no SR-coding
%-SR-decoding:
%e_q3D_decoded = sr3D_decode(e_q3D_coded,nr_rows, nr_frames);

f_rec3D = dpcm3D_decode(e_q3D_decoded,pred_coeffs,q_param,MC_shift);
for i = 1:size(im_mat,3)
    f_rec3D(:,:,i) = f_rec3D(:,:,i)+im_mat_mean(1,1,i);
end
im_mat_rec3D = uint8(f_rec3D);


%Recovered sequence vs. original sequence
%step_im_sequence(im_mat_rec3D)
figure
imshow(im_mat_rec3D(:,:,2));
title('recovered image, with mean')
figure
imshow(uint8(im_mat(:,:,2)));
title('Original image, with mean')

%--Histograms:
e_q3D0 = [e_q3D(1:end,1,1)' e_q3D(1, 1:end, 1)];
e_q3D1 = e_q3D(2:end,2:end,1);
e_q3D2 = e_q3D(2:end,2:end,2);
figure
subplot(3,1,1)
hist(e_q3D0(:),max(e_q3D0(:))-min(e_q3D0(:)))
title('Edges without prediction')
%title('Histogram of quantizer output, not predicted pixels 1st frame')
subplot(3,1,2)
hist(e_q3D1(:),max(e_q3D1(:))-min(e_q3D1(:)))
title('2D prediction')
%title('Histogram of quantizer output, 2D predicted pixels 1st frame')
subplot(3,1,3)
hist(e_q3D2(:)+0.5,max(e_q3D2(:))-min(e_q3D2(:)))
title('3D prediction')
```

```
%title('Histogram of quantizer output, 3D predicted frames')


%-- Compute entropy and SNR:
%e_q3D = e_q3D(2:end, 2:end, 2:end);
bins = max(e_q3D(:))-min(e_q3D(:));
histogram = hist(e_q3D(:),bins);
prob = histogram./sum(histogram); %probability distribution
figure
entropy = 0;
for m=1:bins
    if prob(m) > 0
        entropy = entropy - prob(m)*log2(prob(m));
    end
end
entropy

diff_sig = uint8(im_mat(2:end,2:end,2:end))-im_mat_rec3D(2:end,2:end,2:end);
PSNR=20*log10(255/std(double(diff_sig(:))))
```

est_2Dpredcoeffs.m

```
function [a norm_power] = est_2Dpredcoeffs(f,W, debug)

    if(nargin < 3)
        debug = 0;
    else
        debug = 1;
    end

    %Compute autocorrelation coefficients:
    rho = autocorrcoeffs_2D(f);
    rho_h=rho(1);
    rho_v=rho(2);

    W_0 = [0 0; W];
    x_0 = W_0(:,1);
    y_0 = W_0(:,2);
    m = W(:,1);
    n = W(:,2);
    A = zeros(length(W_0),length(W));
    r_w = zeros(length(W_0),1);
    for i = 1:length(W_0)
        r_w(i) = rho_v^x_0(i)*rho_h^y_0(i);
        for j = 1:size(W,1)
                A(i,j) = rho_v^abs((x_0(i)-m(j)))*rho_h^abs((y_0(i)-n(j)));
        end
    end

    a = A(2:end,:)\r_w(2:end);
    %power = r_w(1)*a'*r_w(2:end);
    norm_power = 1-a'*r_w(2:end);

    %Scaling to sum to one:
    weight = sum(a);
    a = a/weight;
    if(debug == 1)
        disp('2D prediction parameters')
        rho
        A
        r_w
        weight
        a
        norm_power
        gain = 1/norm_power
    end
end
```

est_3Dpredcoeffs.m

```
function [a norm_power] = est_3Dpredcoeffs(f,W,debug)
    % W - prediction window: matrix with coordinates as row vectors
    % f -input array (3D)

    if(nargin < 3)
        debug = 0;
```

```
    else
        debug = 1;
    end
    %----Compute autocorrelation coefficients:----
    rho = autocorrcoeffs_3D(f);
    rho_h=rho(1);
    rho_v=rho(2);
    rho_t=rho(3);

    % ---- LPC analysis -----
    % Defining windows and counters
    W_0 = [0 0 0; W];
    x_0 = W_0(:,1);
    y_0 = W_0(:,2);
    n_0 = W_0(:,3);
    i = W(:,1);
    j = W(:,2);
    k = W(:,3);
    A = zeros(size(W_0,1),size(W,1));
    r_w = zeros(size(W_0,1),1);
    % Generating matrix A and vector r_w:
    for m = 1:size(W_0,1) % for all (x,y,n)
        r_w(m) = rho_v^x_0(m)*rho_h^y_0(m)*rho_t^n_0(m); %vil dette bli noe ...
            annet enn 1?
        for n = 1:size(W,1) % for all (i,j,k)
            A(m,n) = ...
                rho_v^abs(x_0(m)-i(n))*rho_h^abs(y_0(m)-j(n))*rho_t^abs(n_0(m)-k(n));
        end
    end
    % Solving for a and power:
    a = A(2:end,:)\r_w(2:end);
    %power = r_w(1)*a'*r_w(2:end)
    norm_power = 1-a'*r_w(2:end);

    %Scaling to sum to one:
    weight = sum(a);
    a = a/weight;

    if(debug == 1)
        disp('3D prediction parameters')
        rho
        A
        r_w
        weight
        a
        norm_power
        gain = 1/norm_power
    end
end
```

inv_deadzone_quantizer.m

```
function x_q = inv_deadzone_quantizer(q_level,q_param)
    %% Map quantizations level back to corresponding representation value
    % input: q_level - quantization level (integer)
    %        dz_offset - (two-sided) width of dead-zone interval
    %        range - one-sided range
    %        L - number of quantization levels
    % output: x_q - representation value
    L = q_param.L;
    range = q_param.range;
    dz_offset = q_param.dz_offset;

    step_size = (2*range-dz_offset)/(L-1);
    %   x_q - representation level
    q_level_unsigned = abs(q_level);
    %x_mapped = max(0,(-0.5*(dz_offset-step_size))/step_size);
    x_q_unsigned = q_level_unsigned*step_size+0.5*(dz_offset-step_size);
    x_q = sign(q_level)*x_q_unsigned;
end
```

inv_uniform_quantizer.m

```
function x_q = inv_uniform_quantizer(q_level,q_param)
    %% Map quantization levels back to corresponding representation value
    % input: q_level - quantization level (integer)
```

```
    %          range - one-sided range
    %          L - number of quantization levels
    % output: x_q - representation value
    L = q_param.L;
    range = q_param.range;

    step_size = 2*range/L;
    x_q = q_level*step_size;
end
```

make_sine_image_for_video.m

```
function im_sine = make_sine_image_for_video(video_param, image_param)
    %-----------
    % Input: the structs video_param and image_param
    %-----------

    % Extracting the sine parameters:
    intensity = image_param.intensity;
    amplitude = image_param.sine_amplitude;
    angle = image_param.sine_angle;
    % Extracting the video_parameters:
    im_size = video_param.highres;
    %satcam_param = video_param.satcam_param;
    num_of_periods = satcam_param.image_cov_wl;

    % Making the sine image:
    im_sine = make_sinus_image(im_size, num_of_periods, intensity, ...
        amplitude, angle);
end
```

make_sinus_image.m

```
function im_sinus = make_sinus_image(im_size, num_of_periods, intensity, ...
    amplitude, angle)
%making an image with sinusoidal stripes of with mean "intensity" and
%amplitude "amplitude"

im_sinus=zeros(im_size);
N = size(im_sinus,2);
M = size(im_sinus,1);
kx = cos(angle*pi/180)*num_of_periods/N;
ky = sin(angle*pi/180)*num_of_periods/N;
u = (1:N)*(2*kx*pi);
U = ones(M,1)*u;
v = (1:M)*(2*ky*pi);
V = v'*ones(1,N);

im_sinus = amplitude*sin(U+V)+intensity;
im_sinus = uint8(256*im_sinus);
end
```

make_test_images_dpcm.m

```
% Make test images for DPCM (without MC)

% Many images, high res(Very slow when SR-coding):
% N = 10; %number of test images
% im_size = 256;
% number_of_periods = 11.5;
% SNR_factor = 50;
% exp_time = 1;

% Short version for efficient testing:
N = 5; %number of test images
im_size = 128;
number_of_periods = 11.5;
SNR_factor = 50;
```

```
exp_time = 1;

sine_dc = 0.6; % DC level of sine images
sine_amp = 0.05*sine_dc; % Amplitude of sine images
noise_std = sine_dc/(SNR_factor*sqrt(exp_time));
skew_factor = 20;
angle = 10;
sine_angles = angle +skew_factor*rand(1,N); % Angles around 45 degrees;
%sine_angles = 45+ zeros(1,N); %constant angle
im_mat = uint8(zeros(im_size, im_size, N));
for i = 1:N
    im_sine = make_sinus_image(im_size, number_of_periods, sine_dc, sine_amp, ...
        sine_angles(i));
    im_mat(:,:,i) = im_sine+(uint8(256*noise_std*randn(im_size)));
end

video_mat = double(im_mat);
```

make_test_images_dpcm_mc.m

```
% Make test images for DPCM (with MC)
video_param = struct('size_out', [128 128], 'frames', 5, 'frame_shift', ...
     frame_shift, ...
                'frame_rate', 1, 'highres', [128 1024], 'exp_time', 1);
image_param = struct('intensity', 0.6, 'SNR_factor', 50);
%highres_test_im = make_sine_image_for_video(video_param, image_param);
im_size = video_param.highres;
num_of_periods = 20*8;
intensity = 0.6;
amplitude = 0.05*intensity;
angle = 0;
highres_test_im = make_sinus_image(im_size, num_of_periods, intensity, ...
     amplitude, angle);

video_mat = video_maker(highres_test_im, video_param, image_param);
```

plotQ.m

```
function plotQ(q_param, text)
    struct_len = size(fieldnames(q_param),1);
    if(struct_len == 2) %uniform quantizer
        range = q_param.range;
        x = -range:0.01:range;
        x_q = uniform_quantizer(x,q_param);
    else % deadzone quantizer
        range = q_param.range;
        x = -range:0.01:range;
        x_q = deadzone_quantizer(x,q_param);
    end
    plot(x,x_q)
    title(['Levels ' text])
    xlabel('x')
    ylabel('Q(x)')
end
```

set_q_param.m

```
function q_param = set_q_param(L1,L2,L3, dz_loading,f,pred_coeffs,MC_shift,debug)

    if(nargin < 3)
        debug = 0;
    else
        debug = 1;
    end

    loading = 5;

    %encoding without quantization:
```

```
    e_q = dpcm3D_encode(double(f),pred_coeffs,0,MC_shift);

    %Grouping pixels in prediction categories:
    % Group 1: without prediction
    e_q1 = [e_q(1:end,1,1)' e_q(1, 1:end, 1)];
    % Group 2: 2D predicted
    e_q2 = e_q(2:end,2:end,1);
    % Group 3: 3D predicted
    e_q3 = e_q(2:end,2:end,2:end);

    %Compute standard deviation of different prediction categories:
    std_eq1 = std(e_q1(:)); %better to estimate from the whole image?
    std_eq2 = std(e_q2(:));
    std_eq3 = std(e_q3(:));

    %Compute quantization parameters:
    range1 = std_eq1; %compute differently?

    range2 = std_eq2*loading;
    range3 = std_eq3*loading;

    dz_offset2 = 2*std_eq2*dz_loading;
    dz_offset3 = 2*std_eq3*dz_loading;

    q_param = cell(1,3);
    q_param{1} = struct('L', L1, 'range', range1);
    q_param{2} = struct('L', L2, 'range', range2, 'dz_offset', dz_offset2);
    q_param{3} = struct('L', L3, 'range', range3, 'dz_offset', dz_offset3);


    if(debug ==1)
        disp('Quantizer 1')
        std_eq1
        figure
        subplot(3,1,1)
        disp(q_param{1})
        plotQ(q_param{1}, 'Quantizer 1')

        disp('Quantizer 2')
        std_eq2
        subplot(3,1,2)
        disp(q_param{2})
        plotQ(q_param{2},'Quantizer 2' )

        disp('Quantizer 3')
        std_eq3
        subplot(3,1,3)
        disp(q_param{3})
        plotQ(q_param{3},'Quantizer 3')

        %Plotting histogram of different frame outputs:
%           figure
%           hist(e_q1(:),max(e_q1(:))-min(e_q1(:)))
%           title('Histogram of quantizer output frame 1, edges without prediction')
%           figure
%           hist(e_q2(:),max(e_q2(:))-min(e_q2(:)))
%           title('Histogram of quantizer output frame 1, 2D prediction')
%           figure
%           hist(e_q3(:),max(e_q3(:))-min(e_q3(:)))
%           title('Histogram of quantizer for 3D predicted frames')
        figure
        subplot(3,1,1)
        hist(e_q1(:),max(e_q1(:))-min(e_q1(:)))
        title('Edges without prediction')
        axis([-15 15 0 30])
        subplot(3,1,2)
        hist(e_q2(:),max(e_q2(:))-min(e_q2(:)))
        title('2D prediction')
        axis([-15 15 0 3000])
        subplot(3,1,3)
        hist(e_q3(:),max(e_q3(:))-min(e_q3(:)))
        title('3D prediction')
        axis([-15 15 0 15000])
    end
end
```

set_q_param.new.m

```
function q_param = set_q_param(L1,delta2,delta3,tau,f,pred_coeffs,MC_shift,debug)

    if(nargin < 3)
        debug = 0;
    else
```

```
        debug = 1;
    end

    loading = 5;
    %encoding without quantization:
    e_q = dpcm3D_encode(double(f),pred_coeffs,0,MC_shift);

    %Grouping pixels in prediction categories:
    % Group 1: without prediction
    e_q1 = [e_q(1:end,1,1)' e_q(1, 1:end, 1)];
    % Group 2: 2D predicted
    e_q2 = e_q(2:end,2:end,1);
    % Group 3: 3D predicted
    e_q3 = e_q(2:end,2:end,2:end);

    %Compute standard deviation of different prediction categories:
    std_eq1 = std(e_q1(:)); %better to estimate from the whole image?
    std_eq2 = std(e_q2(:));
    std_eq3 = std(e_q3(:));

    %Compute quantization parameters:
    range1_temp = 2*std_eq1; %compute differently?

    range2_temp = std_eq2*loading;
    range3_temp = std_eq3*loading;

    dz_offset2 = 2*std_eq2*tau;
    dz_offset3 = 2*std_eq3*tau;

    step_size1 = delta1*std_eq1

    q_param = cell(1,3);
    q_param{1} = struct('L', L1, 'range', range1);
    q_param{2} = struct('L', L2, 'range', range2, 'dz_offset', dz_offset2);
    q_param{3} = struct('L', L3, 'range', range3, 'dz_offset', dz_offset3);


    if(debug ==1)
        disp('Quantizer 1')
        std_eq1
        figure
        subplot(3,1,1)
        disp(q_param{1})
        plotQ(q_param{1}, 'Quantizer 1')

        disp('Quantizer 2')
        std_eq2
        subplot(3,1,2)
        disp(q_param{2})
        plotQ(q_param{2},'Quantizer 2' )

        disp('Quantizer 3')
        std_eq3
        subplot(3,1,3)
        disp(q_param{3})
        plotQ(q_param{3},'Quantizer 3')

        %Plotting histogram of different frame outputs:
%           figure
%           hist(e_q1(:),max(e_q1(:))-min(e_q1(:)))
%           title('Histogram of quantizer output frame 1, edges without prediction')
%           figure
%           hist(e_q2(:),max(e_q2(:))-min(e_q2(:)))
%           title('Histogram of quantizer output frame 1, 2D prediction')
%           figure
%           hist(e_q3(:),max(e_q3(:))-min(e_q3(:)))
%           title('Histogram of quantizer for 3D predicted frames')
        figure
        subplot(3,1,1)
        hist(e_q1(:),max(e_q1(:))-min(e_q1(:)))
        title('Edges without prediction')
        axis([-15 15 0 30])
        subplot(3,1,2)
        hist(e_q2(:),max(e_q2(:))-min(e_q2(:)))
        title('2D prediction')
        axis([-15 15 0 3000])
        subplot(3,1,3)
        hist(e_q3(:),max(e_q3(:))-min(e_q3(:)))
        title('3D prediction')
        axis([-15 15 0 15000])
    end
end
```

sr3D_decode.m

```
function im_sequence = sr3D_decode(bitcoded_vector,frame_res, nr_frames)
    nr_px = frame_res*frame_res*nr_frames;
    coded_vector = bits2symbol(bitcoded_vector,'char');
    decoded_vector = SRdecode(coded_vector, nr_px,'char');
    decoded_array = reshape(decoded_vector, frame_res,frame_res, nr_frames);
    im_sequence = permute(decoded_array, [2 1 3]); %"transpose" the frames back

%      nr_rows = length(bitcoded_rows);
%      nr_frames = nr_rows/frame_res;
%      im_sequence = zeros(frame_res, frame_res, nr_frames);
%      for k = 1:nr_rows
%          coded_row = bits2symbol(bitcoded_rows{k})
%          row_index = mod(k,fram_res)
%          frame_index = mod(k,
%          im_sequence(
%      end
%
end
```

sr3D_encode.m

```
function bitcoded_vector = sr3D_encode(array_in)
      % Encode three-dimensional array with SRencode
      array_in = permute(array_in,[2 1 3]); %"transpose" each frame
      vector_in = array_in(:); %reshape into vector
      coded_vector = SRencode(vector_in,'char');
      bitcoded_vector = symbol2bits(coded_vector);
end
```

uniform_quantizer.m

```
function q_level = uniform_quantizer(x,q_param)
    %% --Quantization with uniform deadzone quantizer
    % input: x - input signal (one sample)
    %         dz_offset - (two-sided) width of dead-zone interval
    %         range - one-sided range
    %         L - number of quantization levels
    % output: quantization level for sample x
    %% -------
    L = q_param.L;
    range = q_param.range;

    % Compute step size outside deadzone
    step_size = 2*range/L;
    % Compute maximum quantization level
    %max_q_level = (dz_offset+step_size*(L-1)-step_size*0.5)/2;
    max_q_level = range-0.5*step_size;
    %clip input outside the quantizer range:
    x_clipped = min(abs(x),max_q_level);

    %map to one-sided uniform midtread with step_size=1
    x_mapped = max(0,(x_clipped./step_size));
    % add sign back:
    x_signed = sign(x).*x_mapped;

    % shift signal to get all-positive values:
    x_pos = x_signed+L+1;
    % round to nearest integer and add sign:
    q_level_pos = round(x_pos);
    % shift signal back
    q_level = q_level_pos-(L+1);
end
```

video_maker.m

```
function video = video_maker(im_in, video_param, image_param)
    size_out = video_param.size_out;
    frames = video_param.frames;
```

```
    frame_shift = video_param.frame_shift;
    exp_time = video_param.exp_time;
    SNR_factor = image_param.SNR_factor;
    intensity = image_param.intensity;

    if(SNR_factor ==0)
        noise_std = 0;
    else
        noise_std = intensity/(SNR_factor*sqrt(exp_time));
    end
    %noise_std = noise_std_normalized*sqrt(exp_time);
    res_ratio = min(size(im_in)./size_out); %The ratio between the low and high ...
        resolution, assuming a rectangular image
    video = zeros([size_out frames]); %Making a 3 dim matrix for video, last ...
        index is frame number
    for i = 1:frames
        y=0;
        x=0;
        shift_highres = frame_shift.*res_ratio;
        size_highres = size_out.*res_ratio;
        y = ((1:size_highres(1))+round(shift_highres(1)*(i-1))); %must round to ...
            make integer index. Any better solutions? Interpolation?
        x = ((1:size_highres(2))+round(shift_highres(2)*(i-1)));
        if((y(end) <= size(im_in,1)) && (x(end) <= size(im_in,2))) %cheking if ...
            the input image is large enough
            im_highres = im_in(y,x);
            im_lowres = imresize(im_highres, size_out);
        else
            i = frames+1; %(noen smartere åmte å avbryte øforlkke åp?)
        end
        video(:,:,i) = im_lowres+(uint8(256*noise_std*randn(size_out)));
        %video(:,:,i) = im_lowres;
    end
end
```

E.3 Stack-run coding

The following code was used for SR- encoding and decoding in the simulations of the DPCM algortihm, and is made by Anna Kim.

bit2int.m

```
function N =bit2int(bits)
% convert bits to integers.

n = length(bits)-1;
w =2.^(n:-1:0);
N = sum(bits.*w);
```

bits2symbol.m

```
function symbols = bits2symbol(bits,FMT)
% convert bits back to symbols
% bits dimension 2xL
L = size(bits,2);

if nargin == 1|| strcmp(FMT,'double')==1
    symbols = zeros(1,L);
    for l = 1:L
        if bits(:,l) ==[1 1]'
            symbols(l) = 3;
        elseif bits(:,l) ==[1 0]'
            symbols(l) = 2;
        elseif bits(:,l) ==[0 1]'
            symbols(l) = 1;
        elseif bits(:,l) ==[0 0]'
            symbols(:,l) = 0;
        else
            symbols(:,l) = -1;
        end
```

```
        end
else
        symbols = repmat('+',1,L);
        for l = 1:L
            if bits(:,l) ==[1 1]'
                symbols(l) = '+';
            elseif bits(:,l) ==[1 0]'
                symbols(l) = '-';
            elseif bits(:,l) ==[0 1]'
                symbols(l) = '1';
            elseif bits(:,l) ==[0 0]'
                symbols(:,l) = '0';
            else
                symbols(:,l) = 's';
            end
        end
end
```

int2bit.m

```
% convert integer to bits. we use little Endian.
% N:     input integer. n: number of bits for output.
% n must be greater or equal to log2(N).
function bits = int2bit(N,n)
if nargin == 1

        if N ==0 || N ==1
            bits = rem(N,2);
        else
            n = floor(log2(N))+1;
            bits = zeros(1,n);
            for i = 1:n
                bits(i) = rem(N,2);
                %N = N-2^(n-i+1);
                %N = N-2^i;%
                N=(N-bits(i))/2 ;
            end
            bits = fliplr(bits);
        end

else
        bits = zeros(1,n);

        if N ==0 || N ==1
            bits(end) = rem(N,2);
        else

            for i = 1:n
                bits(i) = rem(N,2);
                %N = N-2^(n-i+1);
                %N = N-2^i;%
                N=(N-bits(i))/2 ;
            end
            bits = fliplr(bits);
        end



end
```

SRdecode.m

```
function x = SRdecode(Y,output_length,FMT)
%SR_decode This is the decoder of the stack run run-length encoder.
%
%   INPUT: Y is an array of integers contain 3,2,1,0 or strings
%             contain '+,-,1,0'.
%           FMT is the string indicating format of input. 'char' or 'double'
%           are allowed. 'double' is default format.
%           output_length is the desired output length.
%
%   OUTPUT: x is an array of integers.
%
%


```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% by A. Kim
% date created: 02.10.2011
% last change: 02.10.2011
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
x = [];

if nargin == 0
    fprintf('ERROR: no input given');
elseif nargin ==1 || nargin == 2|| strcmp(FMT,'double')==1
    y = Y;
elseif strcmp(FMT, 'char')==1
    % converting string into to double.
    y(Y=='+') = 3;
    y(Y=='-') = 2;
    y(Y=='1') = 1;
    y(Y=='0') = 0;
else
    fprintf('ERROR: unknown input format.');
    y = [];
end




while ~isempty(y)
    if length(y)>1
        % find the '+','-', '1' and '0' positions
        PM_pos = find(y==3| y==2);
        OZ_pos = find(y==1|y==0);

        % y always starts with the symbol '+' or '-'

        if isempty(OZ_pos) % only runs of zeros.
            temp3 = y(1:length(y));
            y = [];
            temp3(temp3==3)=1;

            temp3(temp3==2)=0;

            numbr = bit2int([1 temp3]);

            % check if the length is 2^k-1

            if round(log2((numbr)+1))-log2((numbr)+1)==0
                rl = bit2int(temp3);
                rl_zeros = zeros(1,rl);
            else
                rl_zeros = zeros(1,numbr);
            end


                x = [x rl_zeros];

        elseif isempty(PM_pos) % only 1 or zeros. (not valid code word)
            x = y;
            y = [];
        elseif ~isempty(OZ_pos) && OZ_pos(1)==2 % first codeword is none-zero value

            % check the sign of the none-zero value.
            if y(PM_pos(1))==3
                nz_sign = 1;
            elseif y(PM_pos(1))==2
                nz_sign = -1;
            end


            %convert the binary into decimal.
            % remove the codeword from y

            if length(PM_pos)>1
                temp = [1 y(2:PM_pos(2)-1)];
                y = y(PM_pos(2):length(y));

            else% no other valide codeword in the input.
                temp = [1 y(2:length(y))];
                y = [];
            end


            x = [x nz_sign*(bit2int(temp)-1)];

        elseif OZ_pos(1) ==1 %first codword is 1/0. decode as 1/0.
                x = [x y(1)];

                % remove codeword.
                y = y(2:length(y));

        elseif OZ_pos(1)>2% first code word is run length of zeros.
```

```
                    %if ~isempty(OZ_pos)
                temp3 = y(1:OZ_pos(1)-2);% remember the nz val has MSB.
                y = y(OZ_pos(1)-1:length(y));

                temp3(temp3==3)=1;

                temp3(temp3==2)=0;

                numbr = bit2int([1 temp3]);


                if round(log2((numbr)+1))-log2((numbr)+1)==0 % if the length is ...
                     2^k-1
                    rl = bit2int(temp3); % then all bits are there
                    rl_zeros = zeros(1,rl);
                else % if not, need the one with added bits.
                    rl_zeros = zeros(1,numbr);
                end


                x = [x rl_zeros];
        end

    elseif length(y)==1
        if y == 3 || y == 0% here y = +, then only 1 zero is left.
            x = [x 0];
        elseif y == 2 % here y = -, two zeros are left.
            x = [x 0 0];
        elseif y == 1
            x = [x 1];

        end
            y = [];
    end
end


%check to see if x is at the right length.
if nargin >=2
    if output_length > length(x)
        x = [x zeros(1,output_length-length(x))];
    else
        x = x(1:output_length);
    end
end
```

SRencode.m

```
function Y = SRencode(x,FMT)

% SRencode SR (Stack-Run) run-length encoder, encode the length of zeros and ...
     value of
% non-zero elements. [Tsai'96]
%
%
%
% The SRencode takes input of an array contains zeros and non-zero values
% outputs a string array consists of four symbols: +,-,0,1
%
% all non-zero values are first incremented (or decremented) by 1.
% 0,1 are used to describe the LSB of the non-zero values in binary.
% +,- are used to describe the MSB and sign of non-zero values in binary
%
%
% + is 1 and - is 0 in describing the run-length of zeros in binary.
% the MSB (since it's always 1) is omitted, except when the run length is
% 2^k-1, where k is an integer.
%
%
% INPUT: 'x' e.g. quantization levels, array in double.
%         FMT is a string identifies the output format, can be 'char',
%         or 'double'. default format is 'double'.
% OUTPUT: Y is the output array. when FMT is NOT 'char', '+' becomes 3 and
% '-' becomes 2.
%
% Example: x = [-11 0 0 0 34 0 0 0 0 2 1 0 0 0 5];
%            y = SRencode(x,'char')
%            y = -100+++00011--+1+0+++10
%
%

% %%%%%%%%%%%%%%%%%%%%%%%%%%%%
% by A. Kim
```

```
% date: 28.02.2011
% last change: 24.09.2011
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%
%pre allocate codeword size. -1 is dummy bit.
y = -ones(size(x));
sizey1 = 1; %starting position of the codeword
sizey2 = 0; %end position.
while ~isempty(x) % check if x has any value
     if isempty(find(x~=0, 1))% x has only zeros.
          RLbin_str = int2bit(length(x));
          % checking to see if run-length is 2^k-1
          if round(log2(length(x)+1))-log2(length(x)+1)~=0
               RLbin_str = RLbin_str(2:length(RLbin_str));
          end
          % mapping to '+','-'. where '+' is 3, '-' is 2.
          RLbin_str = RLbin_str +2;
          %determine end position of codeword
          sizey2 = sizey2+length(RLbin_str);
          % remove the encoded zeros.
          x = [];
          % assign to output and update starting position of codeword
          y(sizey1:sizey2)=RLbin_str;
          sizey1 = sizey2+1;
     else % x has both zero and nonzero elements
               if x(1)==0  % when first element is zero, code run length:
                    % determine length of zeros, convert to binary.
                    marker = length(x(1:find(x~=0, 1 )-1));
                    RLbin_str = int2bit(marker);
                    % checking to see if run-length is 2^k-1,if not remove MSB
                    if round(log2(marker+1))-log2(marker+1)~=0
                         RLbin_str = RLbin_str(2:length(RLbin_str));
                    end
                    %change 0, to -, 1 to +
                    RLbin_str = RLbin_str +2;
                    %update codeword start and end positions and output array
                    sizey2 = sizey2+length(RLbin_str);
                    y(sizey1:sizey2)=RLbin_str;
                    sizey1 = sizey2+1;
                    % remove the encoded zeros from input array
                    x = x(find(x~=0,1):length(x));
               else % encode the first nonzero value
                    % first increment the absolute value by 1. retain sign.
                    nz_val = abs(x(1))+1;
                    %NZbin_str = dec2bin(abs(nz_val));
                    NZbin_str = int2bit(nz_val);
                    % change the MSB into '+' or '-'
                    if sign(x(1))>0
                         NZbin_str(1)=3; % '+' is 3
                    else
                         NZbin_str(1)=2;% '-' is 2;
                    end
```

```
                % update codeword positions and output array
                sizey2 = sizey2+length(NZbin_str);
                y(sizey1:sizey2)=NZbin_str;
                sizey1 = sizey2+1;



                % remove the nonzero value from input array
                if length(x)==1  % come to the last element
                    x = [];
                else
                    x = x(2:length(x));

                end


            end

    end





end

% remove the dummy bits -1;

y = y(y>=0);

if nargin==1 || strcmp(FMT,'double')==1
    Y = y;

elseif  strcmp(FMT, 'char')==1

    Y(y==3) = '+';
    Y(y==2) = '-';
    Y(y==0) = '0';
    Y(y==1) = '1';

else
    fprintf('ERROR: unknown output format.');
    Y = [];
end
```

symbol2bits.m

```
function bits = symbol2bits(symbols)
% converts symbols of +,-,0,1 into bits
% uses 2 bits per symbol +: 11 -: 10 1: 01, 0:00
% -1 is the dummy bit. it is transformed into [-1 -1];

bits = zeros(2,size(symbols,2));

for i = 1: size(symbols,2)
    if symbols(i) =='+'|| symbols(i)==3
        bits(:,i) = [1 1]';
    elseif symbols(i)== '-'|| symbols(i) ==2
        bits(:,i) = [1 0]';
    elseif symbols(i)=='1'|| symbols(i) ==1
        bits(:,i) = [0 1]';
    elseif symbols(i) == '0'|| symbols(i) == 0
        bits(:,i) = [0 0]';
    else
        bits(:,i) = ones(size(2,1))*symbols(i);%[-1 -1]';
    end
end
```

E.4 Image Averaging

The function `import_vars.m` is made by Sigvald Marholm.

compute_and_set_image_param.m

```
function image_param = ...
    compute_and_set_image_param(intensity,SNR_factor,sine_angle,sine_amplitude)
    % All quantities are normalised to 1s exposure time
    if(nargin == 0)
        intensity = 0.5;
        SNR_factor = 100;
        sine_angle = 0;
    end

    if(nargin ~=4)
        amplitude_factor = 0.05;
        sine_amplitude = amplitude_factor*intensity;
    end

%     noise_flag = 0;
%     if((nargin > 0))
%         if(intensity == 'no noise')
%             noise_flag = 1;
%             intensity = 10;
%             dark_current = 0;
%         end
%     end
%
%     if(nargin == 0)
%         intensity = 100; %intensity normalized to one second exposure
%         dark_current = 5;
%     end
%     if (nargin < 3)
%         sine_angle = 0;
%     end
%
%
%     if(noise_flag == 1)
%         noise_std = 0;
%     else
%         noise_std = sqrt(intensity+dark_current);
%     end
%

    % Defining the output struct:
    image_param = struct('intensity', intensity, 'SNR_factor', ...
        SNR_factor,'sine_amplitude', sine_amplitude, 'sine_angle', sine_angle);
end
```

compute_and_set_satcam_param.m

```
function satcam_param = compute_and_set_satcam_param(column_number)
% column_number: picks the column (i.e. set of parameters) in sat_param.txt

% Import variables and constants from text files (column number chooses which ...
    column in sat_param to use
import_vars('sat_param.txt', column_number);
import_vars('sat_const.txt',1);


% Compute variables regarding satellite and camera:
T = 2*pi*sqrt((earth_radius+height_sat)^3/g_param); % Satellite orbital period
speed_sat_OH = 2*pi*(earth_radius*1000+height_OH*1000)/T; % satellite speed ...
    w.r.t OH layer
image_cov = (2*((height_sat-height_OH)*1000)*tan((FOV*pi/180)/2)); % Image coverage
image_cov_wl = image_cov/(gw_min_wl*1000); % Image coverage in wavelengths
spat_res = image_cov/array_size; % coverage per pixel [m]
image_speed = speed_sat_OH/spat_res; %satelite speed in pixels per second

% Defining the output struct:
satcam_param = struct('T',T,'speed_sat_OH',speed_sat_OH, 'image_cov', image_cov,...
                'image_cov_wl',image_cov_wl,'spat_res', spat_res,...
                'image_speed', image_speed, 'height_sat', height_sat, 'FOV', ...
                    FOV, ...
                'array_size', array_size, 'reset_time', reset_time);

end
```

compute_and_set_video_param.m

```matlab
function video_param = compute_and_set_video_param(satcam_param,frame_rate, frames)
image_speed = satcam_param.image_speed;
array_size = satcam_param.array_size;
reset_time = satcam_param.reset_time;
% frame_rate = satcam_param.frame_rate;
% frames = satcam_param.frames;

approx_res_ratio = 10; % Approximate ratio between the low and high resolution
%-do something with the ratio stuff?
lowres = array_size*[1 1]; % Resolution of the resulting video frames
frame_shift = (1/frame_rate)*image_speed*[0 1];
highres_shift = round(frame_shift*approx_res_ratio); % Make sure this is an ...
        integer!
res_ratio = highres_shift/frame_shift; % Computed ratio between low and high ...
        resolution
highres = round((lowres(1)*res_ratio)+[0 (highres_shift(2)*frames)]); %The ...
        resolution of the "analog" inherent image
exp_time = 1./frame_rate-reset_time;
% Defining the output struct (including the corresponding satcam_param as a ...
        nested struct):
video_param = struct('size_out',lowres,'frames',frames,'frame_rate',frame_rate,...
                'frame_shift',frame_shift, 'highres',highres,'exp_time', ...
                    exp_time,...
                'satcam_param',satcam_param);
end
```

display_struct_rows_rec.m

```matlab
function display_struct_rows_rec(struct_in)
    % Displays an 1byN array of structs as a table, with the field names in the
    % first column, and struct number in the first row.

    % Make column of row names----------
    %Initialize row names and row counter:
    names{1} =' ';
    counter = 1;
    %Pick up names from the structs recursively:
    [names counter] = struct_names_rec(struct_in,names,counter);

    %making a column with whitespaces to put inbetween the columns:
    empty_col = char(ones(length(names),1)*'  ');
    matrix_out = [];
    matrix_out = [strvcat(names) empty_col];

    % Fill columns with struct values-----------

    M = length(names); %number of rows
    N = size(struct_in,2); %number of columns
    cell_in = struct2cell(struct_in);
    for i = 1:N
        str_cell = cell(M,1);
        %Make header for column:
        str_cell{1} = num2str(i);
        counter=1;

        % Fetch struct values recursively:
        [str_cell counter] = struct_values_rec(cell_in(:,1,i),str_cell, counter);

        % Extend output matrix with one column:
        matrix_out = [matrix_out strvcat(str_cell) empty_col];
    end

    disp(matrix_out)
end

%                   1           2           3
% T                 5792.3301   5792.3301   5730.1231
% speed_sat_OH      7007.4351   7007.4351   7083.5088
% image_cov         423326.2607 423326.2607 247049.1554
% image_cov_wl      21.1663     21.1663     12.3525
% spat_res          3307.2364   1653.6182   965.0358
% image_speed       2.1188      4.2376      7.3402
% height_sat        600         600         550
% FOV               45          45          30
% array_size        128         256         256
% exp_time          0.03        0.03        0.03
% frame_rate        25          25          10
% frames            50          50          30
% my_struct
%  |-field1
%  |-field2
%  '-field3
```

frame_comb_sim.m

```
%% -----Image averaging-------
% Script for simulation of image averaging with motion compensation
%------------------------------
close all
%% Choose and load videos and parameters
clear all
%video_sim %Run video sim the first time to generate test videos and parameters
param_set_name = 'report_set_testing'; % Choose parameter set
load(['parameters_' param_set_name], 'all_param') % Load parameters
load(['videos_' param_set_name], 'videos') % Load videos

% Show the first image of each video as a thumbnail:
thumbnails = show_video_thumbnails(videos);
% Display parameters:
display_struct_rows_rec(all_param)
%% -------------------

%% Simple demo:
index = 3; % Choose which video(s) in the video set to work with
video_duration = 5;
im_comb_stack = cell(1,index);
video_param = all_param(index).video_param;
frames_to_combine = video_duration*video_param.frame_rate;
% Combine frames:
[im_comb im_comb_cropped] = ...
     video_frame_comb(videos{index},video_param,frames_to_combine);
% Plot first frame and averaged image:
im_comb_stack = cell(1,2);
im_comb_stack{1} = thumbnails{index};
im_comb_stack{2} = im_comb_cropped;
plot_im_stack(im_comb_stack,'w', {'First frame', 'Averaged image'})

%% Varying video durations:
index = 2; % Choose which video(s) in the video set to work with
video_duration = [2 5];
im_comb_stack = cell(1,index);
video_param = all_param(index).video_param;
%Combine frames:
im_comb_stack{1} = thumbnails{index};
im_comb_stack_eq{1} = histeq(thumbnails{index},256);
for j = 1: length(video_duration)
    frames_to_combine = video_duration(j)*video_param.frame_rate;
    [im_comb im_comb_cropped] = ...
         video_frame_comb(videos{index},video_param,frames_to_combine);
    im_comb_stack{j+1} = im_comb_cropped;
end
plot_im_stack(im_comb_stack,'w')


%% Testing frame combining with incorrect frame shift values:
index = 3; % Choose which video in the video set to work with
video_param = all_param(index).video_param;
video_duration = 5;
frames_to_combine = video_param.frame_rate*video_duration;
% Make vector with wrong frame_shifts:
frame_shift = video_param.frame_shift;
error_vector = [0.9 1 0.9]';
frame_shift_fake =zeros(length(error_vector),2);
frame_shift_fake(:,1) = zeros(length(error_vector),1);
frame_shift_fake(:,2) = frame_shift(2)*error_vector.*ones(length(error_vector),1);
errors = frame_shift_fake(:,2)-(frame_shift(2).*ones(length(error_vector),1));
%im_comb_fake = cell(1,length(error_vector));
im_stack = cell(1,length(error_vector));
titles = cell(1,length(error_vector));
for i = 1:length(error_vector)
    video_param.frame_shift = frame_shift_fake(i,:);
    titles{i} = num2str(error_vector(i));
    [im_comb_fake im_comb_fake_cropped] = ...
         video_frame_comb(videos{index},video_param,frames_to_combine);
    im_stack{i} = im_comb_fake_cropped;
end
% Plot resulting combined images:
plot_im_stack(im_stack,'q', titles);
```

import_vars.m

```
function import_vars(fname,column)
% import_vars(fname,column)
```

```
%
% This function is primarily written for importing link budget parameters
% but can be used in a more generic manner as-is.
%
% fname is the name of the file to be imported.
%
% The file can have several sections separated by lines of =============
% The variables are stored in the 2nd section like this:
%
% Header Comments (myfile.txt)
%        Set 1    Set 2    Set 3
% ======================================
% cool   100      200      300
% fun    40.4     5.5      6e3
% ======================================
% Footer Comments
%
% The column argument is which column to import as the variable's values.
% In our example, import_vars('myfile.txt',2) will be equivalent to writing
%    cool = 200;
%    fun = 5.5;
%

% Create the regexp pattern that extracts the variable name and value
pattern = '%s';                              % Extract var. name
for i=1:column-1
    pattern = strcat(pattern,' %*f');      % Ignore (column-1) values
end
pattern = strcat(pattern,' %f');           % Extract column value

section = 1;          % Which section of file.
                      % Each section is seperated by =======

fh = fopen(fname);
while 1
    fline = fgetl(fh);

    if(fline==-1)        % End-of-file
        fclose(fh);
        break;
    end

    if(length(fline)>0)

        if(fline(1)=='=')
            section = section + 1;    % Section separation detected
        else

            if(section==2)             % This is where the variables are at
                A = textscan(fline,pattern,'MultipleDelimsAsOne',true);
                varname = A(1);
                varvalue = A(2);
                varname = varname{1}{1};
                varvalue = varvalue{1};
                varvalue = varvalue(1);
                evalin('caller', strcat(varname,'=',num2str(varvalue),';'));
            end

    end
end

end

end
```

make_sine_image_for_video.m

```
function im_sine = make_sine_image_for_video(video_param, image_param)
    %------------
    % Input: the structs video_param and image_param
    %------------

    % Extracting the sine parameters:
    intensity = image_param.intensity;
    amplitude = image_param.sine_amplitude;
    angle = image_param.sine_angle;
    % Extracting the video_parameters:
    im_size = video_param.highres;
    satcam_param = video_param.satcam_param;
    num_of_periods = satcam_param.image_cov_wl;

    % Making the sine image:
```

```
        im_sine = make_sinus_image(im_size, num_of_periods, intensity, ...
            amplitude, angle);
    end
```

make_sinus_image.m

```
function im_sinus = make_sinus_image(im_size, num_of_periods, intensity, ...
    amplitude, angle)
%making an image with sinusoidal stripes of with mean "intensity" and
%amplitude "amplitude"
%angle given in degrees, 0 = vertical stripes

im_sinus=zeros(im_size);
N = size(im_sinus,2);
M = size(im_sinus,1);
kx = cos(angle*pi/180)*num_of_periods/N;
ky = sin(angle*pi/180)*num_of_periods/N;
u = (1:N)*(2*kx*pi);
U = ones(M,1)*u;
v = (1:M)*(2*ky*pi);
V = v'*ones(1,N);

im_sinus = amplitude*sin(U+V)+intensity;
im_sinus = uint8(256*im_sinus);
end
```

plot_im_stack.m

```
% draft for smart subplot-function
function plot_im_stack(im_cell, mode, title_cell)

 n_total = length(im_cell); %number
% Quadratic constellation
if(mode == 'q')
    % Determining the numbers of figures and plots in each figure:
    number_of_figures = ceil(n_total/12);
    n = 12*ones(1,number_of_figures);
    n_last_figure = mod(n_total,12);
    n(number_of_figures) = n_last_figure;

    for i=1:number_of_figures
        % determining the number of rows to plot
        n = n(i);
        if(n < 4)
            rows(i) = 1;
        elseif(n >= 4 && n < 9)
            rows(i) = 2;
        elseif(n >= 9 && n <= 12)
            rows(i) = 3;
        end
    %number of columns:
    columns(i) = ceil(n/rows(i));
    end

elseif(mode == 'l')
    number_of_figures = ceil(n_total/4);
    n = 4*ones(1,number_of_figures);
    n(number_of_figures) = mod(n_total,12);
    rows = n;
    columns = ones(1,number_of_figures);
elseif(mode == 'w')
    number_of_figures = ceil(n_total/4);
    n = 4*ones(1,number_of_figures);
    n(number_of_figures) = mod(n_total,12);
    columns = n;
    rows = ones(1,number_of_figures);
else
    disp('Invalid plot constellation')
end

    %making subplots:
    plot_count = 0;
    for k = 1:number_of_figures
        figure
        for j = 1:n(k)
            plot_count = plot_count+1;
```

```
                subplot(rows(k),columns(k),j)
                imshow(im_cell{plot_count})
                if(nargin==3)
                 title(title_cell{plot_count})
                end
            end
            %set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
        end
    end
```

sat_const.txt

```
 Satellite and camera constants (sat_const.txt)
=================================
earth_radius    6371
g_param        398601
gw_mean_wl     20
gw_min_wl      15
gw_mean_speed     25
height_OH      89
airglow_wl    0.0000015
=================================
  Footer Comments
```

sat_param.txt

```
 Satellite and camera parameters (sat_param.txt)
                          Set 1     Set 2     Set 3    Set 4
========================================================
 height_sat               500       500       350      350
FOV                        40       40        30       30
 array_size               128       256       256      128
 reset_time              0.01      0.01      0.01     0.01
========================================================
```

show_video_thumbnails.m

```
function thumbnails = show_video_thumbnails(videos)
    thumbnails = cell(1,length(videos));
    titles = cell(1,length(videos));

    for index = 1:length(videos);
        %subplot(1,length(videos),index)
        %Pick the first frame of the video as thumbnail:
        thumbnails{index} = uint8(videos{index}(:,:,1));
        titles{index} = num2str(index);
    end

    plot_im_stack(thumbnails, 'q', titles)
end
```

step_im_sequence.m

```
function step_im_sequence(im_matrix)
    % Scale image to fill a larger window:
    approx_res = 600;
    scale_factor = round(approx_res/size(im_matrix,1));
    im_matrix_sc = imresize(im_matrix,scale_factor);
    % Show one image at a time separated by button press:
    figure
    %set(gcf,'Position', get(0,'Screensize'));
```

```
    for i = 1:size(im_matrix,3)
        imshow(uint8(im_matrix_sc(:,:,i)));
        waitforbuttonpress
    end
    close gcf %close the current figure
end
```

struct_names_rec.m

```
function [outer_names counter] = struct_names_rec(struct_in, outer_names, counter)

    cell_in = struct2cell(struct_in);
    names = fieldnames(struct_in);
    struct_length = size(names,1);
    %isstruct_array = ones(struct_length);

    for i = 1:struct_length
        counter = counter+1;
        outer_names{counter} = names{i};
        isstruct_array(i) = isstruct(cell_in{i,1,1});
        if(isstruct_array(i) ==1)
            inner_struct= cell_in{i,1,1};
            [outer_names counter] = ...
                struct_names_rec(inner_struct,outer_names,counter);
        end
    end
end
```

struct_values_rec.m

```
function [str_cell counter] = struct_values_rec(cell_in, str_cell,counter)
    struct_length = length(cell_in);

    for i = 1:struct_length
        isstruct_array(i) = isstruct(cell_in{i});
        counter=counter+1;
        if(isstruct_array(i) == 0)
            str_cell{counter} = num2str(cell_in{i});
        else
            str_cell{counter} = '-------';
            inner_cell = struct2cell(cell_in{i});
            [str_cell counter] = struct_values_rec(inner_cell, str_cell, counter);
        end
    end
end
```

video_frame_comb.m

```
function [im_comb im_comb_cropped] = video_frame_comb(video, video_param, ...
        frames_to_combine)

frame_shift = video_param.frame_shift;
lowres = video_param.size_out;
%frames = video_param.frames;
frames = frames_to_combine;

total_shift = ceil(frames*frame_shift);
im_comb = zeros(lowres+total_shift); %Defining the size of the resulting ...
        combined image
im_comb(1:size(video,1),1:size(video,2))=video(:,:,1); %Placing the firs frame
for i = 2:frames
    % Zero-padding and interpolation:
    im_in = video(:,:,i); % Rename (too similar to interpolation...)
    % input grid:
    x_in = (1:lowres)+((i-1)*frame_shift(2));
    y_in = (1:lowres)+((i-1)*frame_shift(1));
    [X_in Y_in] = meshgrid(x_in,y_in);
    % interpolated grid:
```

```
    x_interp = 1:size(im_comb,2);
    y_interp = 1:size(im_comb,1);
    [X_interp Y_interp] = meshgrid(x_interp,y_interp);
    % Interpolation and zero filling:
    im_interp = interp2(X_in,Y_in,im_in,X_interp,Y_interp);
    im_interp(isnan(im_interp)) = 0;
    % Adding to the resulting image:
    im_comb = im_comb+im_interp;
end
im_comb = uint8(im_comb/frames);
im_comb_cropped = uint8(im_comb(:,total_shift(2):(end-total_shift(2)-1)));

end
```

video_maker.m

```
function video = video_maker(im_in, video_param, image_param)
% size_out and frame_shift are relative to the low resolution image!
%im_in = double(im_in); % any way to cast automatically when called?
% noise_std is the standard deviation of the noise (should be zero if no noise)
% % size_out = get_video_param(video_param,'size_out');
% % frames = get_video_param(video_param,'frames');
% % frame_shift = get_video_param(video_param,'frame_shift');
% % noise_std = get_video_param(video_param,'noise_std');
size_out = video_param.size_out;
frames = video_param.frames;
frame_shift = video_param.frame_shift;
exp_time = video_param.exp_time;
%noise_std_normalized = image_param.noise_std;
SNR_factor = image_param.SNR_factor;
intensity = image_param.intensity;

if(SNR_factor ==0)
    noise_std = 0;
else
    noise_std = intensity/(SNR_factor*sqrt(exp_time));
end
%noise_std = noise_std_normalized*sqrt(exp_time);
res_ratio = min(size(im_in)./size_out); %The ratio between the low and high ...
    resolution, assuming a rectangular image
video = zeros([size_out frames]); %Making a 3 dim matrix for video, last index ...
    is frame number
for i = 1:frames
    y=0;
    x=0;
    shift_highres = frame_shift.*res_ratio;
    size_highres = size_out.*res_ratio;
    y = ((1:size_highres(1))+round(shift_highres(1)*(i-1))); %must round to make...
        integer index. Any better solutions? Interpolation?
    x = ((1:size_highres(2))+round(shift_highres(2)*(i-1)));
%     y = (1:size_out(1)*res_ratio)+frame_shift(1)*res_ratio;
%     x = (1:size_out(2)*res_ratio)+frame_shift(2)*res_ratio;
    if((y(end) <= size(im_in,1)) && (x(end) <= size(im_in,2))) %cheking if the ...
        input image is large enough
        im_highres = im_in(y,x);
        im_lowres = imresize(im_highres, size_out);
    else
        i = frames+1; %(noen smartere åmte å avbryte øforlkke åp?)
    end
    video(:,:,i) = im_lowres+(uint8(256*noise_std*randn(size_out)));
    %video(:,:,i) = im_lowres;
end
end
```

video_sim.m

```
%% -----Video simulation script--------
% Script for simulation of video generation
% Computes and defines sets of different parameters for video generation and stores
% them in the structs satcam_param, video_param and image_param.
% Series of videos are then generated by videomaker2, with the parameter sets ...
    chosen in parameters. T

% To show any of the finished videos (number 1 in this example):
% implay(uint8(videos{1}),0.25*video_param(1).frame_rate)
%% ----------------------------
```

```
close all
clear all

%% Define possible frame rate and video duration:
frame_rate = [5 10];
video_time = 5; %setting number of frames as a constant times frame rate
frames = video_time.*frame_rate;

%% Read parameters from files and generate video parameters
%(Generate video parameters for different combinations of satelite
%parameters and frame rates)
i=1;
k=1;
while(i ~= 0) % runs as long as import_vars manages to read a new column from ...
        sat_param.txt
    try
        % Importing and computing satellite and camera parameters and constants:
        satcam_param(i) = compute_and_set_satcam_param(i);
        % Computing and setting video parameters:
        for j = 1:length(frame_rate)
            video_param(k) = ...
                compute_and_set_video_param(satcam_param(i),frame_rate(j),frames(j));
            k=k+1;
        end
        i=i+1;
    catch me
        i=0;
        %rethrow(me) %for debugging of the things inside try
    end
end

% Display the video parameters:
display_struct_rows_rec(video_param)

%% Define parameters for sine image used for video generation:
%(syntax: dc level, SNR_factor, angle)
image_param(1) = compute_and_set_image_param(0.6, 0, 10);
image_param(2) = compute_and_set_image_param(0.6, 5, 10);
image_param(3) = compute_and_set_image_param(0.6, 20, 10);
image_param(4) = compute_and_set_image_param(0.6, 50, 10);
image_param(5) = compute_and_set_image_param(0.6, 100, 10);

%% Define parameter set combinations for video making:
%(Choose combinations of video parameters and image parameters
% Syntax: [video_param image_param], possible to generate several sets)

name = 'report_set_testing';
parameter_sets = [... %new syntax: video_param image_param
    3 1
    3 2
    3 3
    3 4
    3 5];

%% Making high resolution images and low resolution video:
for j = 1:size(parameter_sets,1)
    %Setting current image and video parameters
    v_index = parameter_sets(j,1);
    im_index = parameter_sets(j,2);
    % Make high resolution image and corresponding video with current parameters:
    highres_images{j} = make_sine_image_for_video(video_param(v_index), ...
        image_param(im_index));
    videos{j} = ...
        video_maker(highres_images{j},video_param(v_index),image_param(im_index));
    % Display all the parameters:
    disp(['----------Parameters video ' num2str(j) '----------------'])
    disp(video_param(v_index))
    disp('     ------------------------')
    disp(video_param(v_index).satcam_param)
    disp('     ------------------------')
    disp(image_param(im_index))
    % Collect all the parameters:
    all_param(j) = struct('video_param', video_param(v_index),...
                    'image_param', image_param(im_index));
end
%% Save videos and corresponding parameters for use in test scripts:
save(['videos_' name], 'videos')
save(['parameters_' name], 'all_param')
```

Video parameters

Example of video parameters sets generated by `video_sim`:

```
                    1              2              3              4              5
video_param         -------        -------        -------        -------        -------
size_out            256  256       256  256       256  256       256  256       256 ...
      256
frames              25             25             25             25             25
frame_rate          5              5              5              5              5
frame_shift         0       1.2255 0       1.2255 0       1.2255 0       1.2255 0 ...
          1.2255
highres             2507  2807     2507  2807     2507  2807     2507  2807     2507...
      2807
exp_time            0.19           0.19           0.19           0.19           0.19
satcam_param        -------        -------        -------        -------        -------
T                   5668.1404      5668.1404      5668.1404      5668.1404      ...
     5668.1404
speed_sat_OH        7160.9689      7160.9689      7160.9689      7160.9689      ...
     7160.9689
image_cov           299183.5326    299183.5326    299183.5326    299183.5326    ...
     299183.5326
image_cov_wl        19.9456        19.9456        19.9456        19.9456        19.9456
spat_res            1168.6857      1168.6857      1168.6857      1168.6857      ...
     1168.6857
image_speed         6.1274         6.1274         6.1274         6.1274         6.1274
height_sat          500            500            500            500            500
FOV                 40             40             40             40             40
array_size          256            256            256            256            256
reset_time          0.01           0.01           0.01           0.01           0.01
image_param         -------        -------        -------        -------        -------
intensity           0.6            0.6            0.6            0.6            0.6
SNR_factor          0              5              20             50             100
sine_amplitude      0.03           0.03           0.03           0.03           0.03
sine_angle          10             10             10             10             10
```

E.5 Motion Blur

make_chirp_image.m

```
function im_chirp = make_chirp_image(f_x)

if(nargin==0)
    f_x = [5 10 20 40 70 100]; %cycles per image
end
f_x
N = length(f_x);
len = 200;
im_chirp = zeros(300,N*len); %Set size and resolution of chirp image
f_x = f_x/(N*len); %cycles per pixel

intensity= 0.5;
amplitude= 0.3;

start_pos = 1;
phase = zeros(1,N);
phase(1) = 0;
for i = 1:N
    end_pos = start_pos+len -1;
    n = 0:len -1;
    sine = amplitude*sin(2*pi*f_x(i)*n+phase(i))+intensity;
    im_chirp(:,start_pos:end_pos) = ones(size(im_chirp,1),1)*sine;
    start_pos = end_pos+1;
    phase(i+1) = phase(i)+2*pi*f_x(i)*len;
end

im_chirp = uint8(256*im_chirp);
```

make_sinus_image.m

```
function im_sinus = make_sinus_image(im_size, num_of_periods, intensity, ...
     amplitude, angle)
%making an image with sinusoidal stripes of with mean "intensity" and
%amplitude "amplitude"
%angle given in degrees, 0 = vertical stripes

im_sinus=zeros(im_size);
```

```
N = size(im_sinus,2);
M = size(im_sinus,1);
kx = cos(angle*pi/180)*num_of_periods/N;
ky = sin(angle*pi/180)*num_of_periods/N;
u = (1:N)*(2*kx*pi);
U = ones(M,1)*u;
v = (1:M)*(2*ky*pi);
V = v'*ones(1,N);

im_sinus = amplitude*sin(U+V)+intensity;
im_sinus = uint8(256*im_sinus);
end
```

make_sinus_image2.m

```
function im_sinus = make_sinus_image2(im_size, period, intensity, amplitude,angle)
%making an image with sinusoidal stripes of with mean "intensity" and
%amplitude "amplitude"
%period is the period of the sine in #pixels

im_sinus=zeros(im_size);
N = size(im_sinus,2);
M = size(im_sinus,1);
kx = cos(angle*pi/180)*(1/period);
ky = sin(angle*pi/180)*(1/period);
u = (1:N)*(2*kx*pi);
U = ones(M,1)*u;
v = (1:M)*(2*ky*pi);
V = v'*ones(1,N);

im_sinus = amplitude*sin(U+V)+intensity;
end
```

motionblur_deconv.m

```
function im_recovered = motionblur_deconv(im_blurred,h_motionblur)
    % Restoration of motion blurred image, with edgetaper and richardson lucy ...
        deconvolution
    num_it = 15;
    h_gaussian = fspecial('gaussian', 15, 7);

    im_edgetaper = edgetaper(im_blurred, h_gaussian);
    im_recovered = deconvlucy(im_edgetaper,h_motionblur,num_it);
end
```

motionblur_maker.m

```
function [im_blurred_mat h_mb] = motionblur_maker(im, blur_len, blur_angle)
    %Simulation of motion blur, returning a matrix with blurred images and
    %blur kernels
    %(former name: mb_sim)

    if(nargin<4)
        blur_angle=0;
    end

    cut_length = ceil(max(blur_len)/3); %Defining how much to cut according to ...
        blur length


    %im_blurred_mat = zeros(size(im,1)-2*cut_length, size(im,2)-2*cut_length, ...
        length(blur_len)+1);
    im_blurred_mat = cell(length(blur_len)+1);

    % Cutting, plotting and saving the original image:
    im_cut = ...
        im((cut_length+1):(end-cut_length),(cut_length+1):(end-cut_length)); ...
        %Cut to compare with blurred image
```

```
        im_blurred_mat{1} = im_cut;
        h_mb{1}=0;
        %{
        figure
        subplot(ceil(length(blur_len)/3),3,1)
        imshow(im_cut)
        title(im_name);
        %}

        % Generating and plotting blurred images:

        for i = 1:length(blur_len)
            % Create blur
            h_mb{i+1} = fspecial('motion', blur_len(i), blur_angle);
            temp = imfilter(im,h_mb{i+1},'conv','replicate');

            % Cut edges to avoid artifacts:
            im_blurred_mat{i+1} = ...
                 temp((cut_length+1):(end-cut_length),(cut_length+1):(end-cut_length));

            % Plot
            %{
            subplot(ceil(length(blur_len)/3),3,i+1)
            imshow(uint8(im_blurred_mat(:,:,i+1)))
            title(['blur length=' num2str(floor(blur_len(i)))]);
            %}
        end
    end
```

motionblur_sim.m

```
%% Simulation of motion blur
 close all
 clear all

%To save figures:
%saveas(gcf, '/home/marianne/Skole/NUTS/report2/matlabfigurer/name', 'fig')
%saveas(gcf, '/home/marianne/Skole/NUTS/report2/matlabfigurer/name', 'pdf')

% number_of_pixels = 512;
% intensity = 0.6;
% amplitude = 0.05*intensity;
% angle = 0;
% number_of_periods = [2 3 4 5 6 10];
% v_im = 0.02; %image velocity relative to image size!
% t_int = 1;
% blur_len = v_im*t_int; %blur length relative to image size!!!
% blur_angle = 0;
%
% for i = 1:length(number_of_periods)
%     im_sinus = make_sinus_image(number_of_pixels, ...
     number_of_periods(i),intensity,amplitude,angle);
%     [im_blurred_mat{i} h_mb{i}] = motionblur_maker(im_sinus, ...
     blur_len*number_of_pixels, blur_angle);
%     im_blurred_stack{i} = im_blurred_mat{i}{2};
% end

%plot_im_stack(im_blurred_stack,'q')
%
%
%% ----Plotting of frequency response:
% v_im = 1;
% t_int = 1;
% u = linspace(0,10,1000);
% H = 1./(pi*u*v_im).*sin(pi*u*v_im*t_int);
% figure(10)
% plot(u,H)
% grid on
% xlabel('Frequency [cycles/image]')
% hold on
%% Computing zeros:
% clear all
% syms v
% syms v_im
% syms t_int
% H = 1/(pi*v*v_im)*sin(pi*v*v_im*t_int);
% %diff_H = diff(H);
% v = solve(H)


%% plotting zeros:
%v_im = 0.0185; %corresponding parameters: 550,45
v_im = 0.024; %corresponding to image coverage 300 km, V' = 7.16 (h_sat = 500)
```

```
t_int = linspace(0,5, 100);
im_cov = 300;
zerocrossings = 1./(v_im*t_int);
zero_wavelengths = im_cov./zerocrossings;
figure
plot(t_int,zero_wavelengths)
xlabel('Integration time [s]')
ylabel('Wavelength at first zero [km]')

figure
plot(t_int,zerocrossings)
xlabel('Integration time [s]')
ylabel('Wavelength at first zero [km]')
axis([0 5 0 200])


v_im = [0.0107 0.0133 0.0185 0.0287 0.0374];
%corresponding parameters: 650,60 - 550,60 - 550,45 - 550,30 - 450,30
im_cov = [648 532 382 247 193]; %corresponding image coverage
t_int = linspace(0,10, 100);
figure
colours = ['b' 'r' 'g' 'c' 'm'];
for i = 1:length(v_im)
    zerocrossings = 1./(v_im(i)*t_int);
    zero_wavelengths = im_cov(i)./zerocrossings;
    plot(t_int,zero_wavelengths,colours(i))
    hold on
end
legend('650,60','550,60', '550,45', '550,30', '450,30')
xlabel('Integration time [s]')
ylabel('Wavelength at first zero [km]')

% %Computing second derivative to find "vendepunkt":
% delta_x = u(2)-u(1);
% ddH = (H(1:end-2)-2*H(2:end-1)+H(3:end))/delta_x^2;
% plot(u(2:end-1),10*ddH)
% grid on
%
% syms u
% H = 1/(pi*u*v_im)*sin(pi*u*v_im*t_int);
% diff_H = diff(diff(H))
% solve(diff_H)

%% Testing with chirp image:
%---parameters----
chirp_freq = [5 10 15 20 25 30];
im_chirp = make_chirp_image(chirp_freq);
t_int = [1 2 3];
v_im = 0.024; %image velocity relative to image size!
blur_len = v_im*t_int; %blur length relative to image size!!!
blur_angle = 0;
number_of_pixels = size(im_chirp,2);
% %---------------------------
% make chirp image:
[im_blurred_mat_chirp h_mb] = motionblur_maker(im_chirp, ...
     blur_len*number_of_pixels, blur_angle);
%plot chirp image:
% titles{1}=['Sine frequencies: ' mat2str(chirp_freq) ', Original image'];
% for i = 1:length(blur_len)
%     title = ['Sine frequencies: ' mat2str(chirp_freq) ', exposure time = ' ...
     num2str(t_int(i)) ];
%     titles{i+1} = title;
% end
% plot_im_stack(im_blurred_mat_chirp,'l', titles)


%% Plot blur and frequency response together
% clear title
% figure
% subplot(5,1,1)
% %freq response:
% u = linspace(1,100,1000);
% color = ['r' 'g' 'b'];
% for i = 1:length(blur_len)
%     H = 1./(pi*u*v_im).*sin(pi*u*blur_len(i));
%     plot(u,H,color(i))
%     hold on
% end
% grid on
% %title('Motion blur filter in the frequency domain')
% xlabel('Frequency [cycles/image]')
% axis([2.5 32.5 -0.7 3])
% legend( ['t_{int} =' num2str(t_int(1))],['t_{int} =' ...
     num2str(t_int(2))],['t_{int} =' num2str(t_int(3))])
% hold off
% %images:
% subplot(5,1,2)
% imshow(im_blurred_mat_chirp{1})
% title('freq: [5 10 15 20 25 30], original')
% subplot(5,1,3)
```

```
% imshow(im_blurred_mat_chirp{2})
% title('freq: [5 10 15 20 25 30], t_{int} = 1')
% subplot(5,1,4)
% imshow(im_blurred_mat_chirp{3})
% title('freq: [5 10 15 20 25 30], t_{int} = 2')
% subplot(5,1,5)
% imshow(im_blurred_mat_chirp{4})
% title('freq: [5 10 15 20 25 30], t_{int} = 3')
%
% titles = {'(b) Original image', '(c) Exposure time 1 s', '(d) Exposure time 2 ...
    s', '(e) Exposure time 3 s'}
% plot_im_stack(im_blurred_mat_chirp,'l', titles)
%
%
%
% %% --- Deblurring a single chirp image:
% blur_index = 4;
% im_blurred = im_blurred_mat_chirp{blur_index};
% h_motionblur = h_mb{blur_index};
% num_it = 15;
% h_gaussian = fspecial('gaussian', 15, 7);
% im_r_plain = deconvlucy(im_blurred,h_motionblur,num_it);
% im_edgetaper = edgetaper(im_blurred, h_gaussian);
% im_r_edgetaper = deconvlucy(im_edgetaper,h_motionblur,num_it);
%
% % plot with frequency response:
% figure
% subplot(4,1,4)
% imshow(im_r_edgetaper)
% subplot(4,1,3)
% imshow(im_blurred)
% subplot(4,1,2)
% imshow(im_blurred_mat_chirp{1})
% subplot(4,1,1)
% u = linspace(min(chirp_freq),max(chirp_freq),1000);
% H = 1./(pi*u*v_im).*sin(pi*u*blur_len(blur_index-1));
% plot(u,H)
% grid on
% xlabel('Frequency [cycles/image]')
% axis([min(chirp_freq)-2.5 max(chirp_freq)+2.5 -0.52 1.9])

% % Deblurring several chirp images:
% titles{1}=['Sine frequencies: ' mat2str(chirp_freq) ', Original image'];
% im_r_mat_chirp{1} = im_blurred_mat_chirp{1};
% for i = 1:length(blur_len)
%     im_blurred = im_blurred_mat_chirp{i+1};
%     h_motionblur = h_mb{i+1};
%     im_r_plain = deconvlucy(im_blurred,h_motionblur,num_it);
%     im_edgetaper = edgetaper(im_blurred, h_gaussian);
%     im_r_edgetaper = deconvlucy(im_edgetaper,h_motionblur,num_it);
%
%     im_r_mat_chirp{i+1} = im_r_edgetaper;
%     title = ['Sine frequencies: ' mat2str(chirp_freq) ', exposure time = ' ...
    num2str(t_int(i)) ];
%     titles{i+1} = title;
% end
% titles = {'(b) Original image', '(c) Exposure time 1 s', '(d) Exposure time 2 ...
    s', '(e) Exposure time 3 s'}
% plot_im_stack(im_r_mat_chirp,'l', titles)
%

%title('recovered image')
%difference between recovered and original image:
% im_diff = im_blurred_mat_chirp{1}-im_r_edgetaper;
% figure
% imshow(im_diff)

%% blur/deblur images:
N_px = 256;
nr_periods = 20;
intensity = 0.6;
amplitude = 0.05*intensity;
angle = 10;

im_sine = make_sinus_image(N_px, nr_periods, intensity, amplitude,angle);

t_int = 1;
v_im = 0.024; %image velocity relative to image size!
blur_len = v_im*t_int;
[im_blurred_sine h_mb] = motionblur_maker(im_sine, blur_len*N_px, blur_angle);

num_it = 15;
h_gaussian = fspecial('gaussian', 15, 7);
%im_r_plain = deconvlucy(im_blurred_sine{2},h_mb{2},num_it);
im_edgetaper = edgetaper(im_blurred_sine{2}, h_gaussian);
im_r_edgetaper = deconvlucy(im_edgetaper,h_mb{2},num_it);
%
% clear title
% figure
% imshow(im_sine)
```

```
% title('original image')
% figure
% imshow(im_blurred_sine{2})
% title('blurred image')
% figure
% imshow(im_r_edgetaper)
% title('recovered image')

%% blur/deblur several images:
t_int = 0.5:0.1:3;
v_im = 0.024; %image velocity relative to image size!
blur_len = v_im*t_int;

[im_blurred_sine h_mb] = motionblur_maker(im_sine, blur_len*N_px, blur_angle);
im_r_mat{1} = im_blurred_sine{1};
im_r_mat_cut{1} = im_blurred_sine{1}(N_px/2-25:N_px/2+25,N_px/2-50:N_px/2+50);
im_blurred_cut{1} = im_blurred_sine{1}(N_px/2-25:N_px/2+25,N_px/2-50:N_px/2+50);
%im_diff_mat{i+1} = im_r_mat_cut{1}-im_r_mat_cut{1};

for i = 1:length(blur_len)
    im_blurred = im_blurred_sine{i+1};
    im_blurred_cut{i+1} = ...
        im_blurred_sine{i+1}(N_px/2-25:N_px/2+25,N_px/2-50:N_px/2+50);

    h_motionblur = h_mb{i+1};

    %im_r_plain = deconvlucy(im_blurred,h_motionblur,num_it);
    im_edgetaper = edgetaper(im_blurred, h_gaussian);
    im_r_edgetaper = deconvlucy(im_edgetaper,h_motionblur,num_it);

    im_r_mat{i+1} = im_r_edgetaper;
    im_r_mat_cut{i+1} = im_r_edgetaper(N_px/2-25:N_px/2+25,N_px/2-50:N_px/2+50);

end

% % SNR and frequency response:
% snr_r = zeros(1,length(t_int));
% snr_blur = zeros(1,length(t_int));
% for i = 1:length(blur_len)
%     im_power = sum(double(im_r_mat_cut{1}(:).^2));
%     %SNR between original and recovered image:
%     im_diff_r = im_r_mat_cut{1}-im_r_mat_cut{i+1};
%     diff_r_power = sum(double(im_diff_r(:).^2));
%     snr_r(i) = im_power./diff_r_power;
%     %SNR between original and blurred image:
%     im_diff_blur = im_blurred_cut{1}-im_blurred_cut{i+1};
%     diff_blur_power = sum(double(im_diff_blur(:).^2));
%     snr_blur(i) = im_power./diff_blur_power;
% end
% u = 20;
% H_20 = 1./(pi*u*v_im).*sin(pi*u*v_im*t_int);
% H_20_norm = H_20./t_int;
% %plot SNR and frequency response together:
% clear title
% figure
% plot(t_int,10*log10(snr_blur),'s-')
% hold on
% [AX,H1,H2] = plotyy(t_int,10*log10(snr_r),t_int,H_20_norm);
% set(get(AX(1),'Ylabel'),'String','SNR (dB)')
% set(get(AX(2),'Ylabel'),'String','Normalized |H( f_x = 20 )|','color','r')
% set(AX(2),'YColor','r')
% set(H1,'Marker','o')
% set(H2,'Color','r')
% grid on
% legend('SNR blurred image','SNR recovered image','Frequency response')
% xlabel('Integration time [s]')
% plot([2.1 2.1],[10 40],'k--')

% plotting of many image segments:
clear title
clear titles
%index = [10 12 14 15 16 17 18 20]; %long int times
index = [1 2 4 6 8 10 12 14]; %short int times
%index = [6 12 17]; % t_int=[1 1.6 2.1], selction for report
t_int(index)
titles{1}='Original image';
plot_r{1} = im_r_mat_cut{1};
plot_blur{1} = im_blurred_cut{1};
plot_combo = cell(1,6);
titles_combo = cell(1,6);
%makin image stacks for plotting:
for i = 1:length(index)
    title = ['Exposure time = ' num2str(t_int(index(i))) ];
    titles{i+1} = title;
    plot_r{i+1} = im_r_mat_cut{index(i)+1};
    plot_blur{i+1} = im_blurred_cut{index(i)+1};

    plot_combo{3*i-2} = plot_blur{i+1}; %dummy image
    titles_combo{3*i-2} = ' ';
    plot_combo{3*i-1} = plot_blur{i+1};
```

```
        titles_combo{3*i-1} =  ' ';

        plot_combo{3*i} = plot_r{i+1};
        titles_combo{3*i} =  ' ';
end
plot_im_stack(plot_blur,'q',titles)
plot_im_stack(plot_r,'q',titles)

%plot_im_stack(plot_combo,'q',titles_combo)

%% Testing for deviations in speed:
%sine test image
N_px = 256;
nr_periods = 20;
intensity = 0.6;
amplitude = 0.5*intensity;
angle = 10;
im_sine = make_sinus_image(N_px, nr_periods, intensity, amplitude,angle);

%blurred image
v_im = 0.024; %image velocity relative to image size!
t_int_test = 1.6;
blur_angle = 0;
blur_len = v_im*t_int_test;
%[im_blurred_sine h_mb] = motionblur_maker(im_sine, blur_len*N_px, blur_angle);
%Recovering image with varying estimated image speed:
%v_im_fake = [1 0.6 0.8 1.1 1.2]*v_im;
v_im_fake = [1 0.9 1.1]*v_im;
blur_len_fake = v_im_fake.*t_int_test;

%chirp image or sine images:
%chirp:
im_test = im_chirp;
N_px = number_of_pixels;
[im_blurred_mat_chirp h_mb] = motionblur_maker(im_test, blur_len*N_px, blur_angle);
im_blurred = im_blurred_mat_chirp{2};
im_r_fake = cell(1,length(blur_len_fake));
im_r_fake{1} = im_chirp;
for i = 1:length(blur_len_fake)
        h_motionblur = fspecial('motion', blur_len_fake(i)*N_px, blur_angle);
        im_r_fake{i+1} = motionblur_deconv(im_blurred, h_motionblur);
end
clear titles
clear title
%titles = {'-40%', '-20%', '+10%', '+20%'};
titles = {'Original image', 'Recovered image, correct speed' '-10% deviation in ...
        speed','+10% deviation in speed',};
plot_im_stack(im_r_fake,'l',titles)
figure
imshow(im_r_fake{2})
%title('correct image speed')
figure
imshow(im_r_fake{1})
%title('original image')

%% Testing with noise:
v_im = 0.024; %image velocity relative to image size!
blur_angle = 0;
blur_len = v_im*t_int;

%make test image without noise:
im_size = 256;
number_of_periods = 20;
t_int = 1.6;
SNR_factor = [50 100];
sine_dc = 0.6; % DC level of sine images per second
sine_angle = 10;
sine_amp = 0.05*sine_dc;
im_sine = make_sinus_image(im_size, number_of_periods, sine_dc, ...
        sine_amp,sine_angle);
%blur image:
[im_blurred h_mb] = motionblur_maker(im_sine, blur_len*im_size, blur_angle);
%add noise:
new_im_size = size(im_blurred{2},1);
noise_std = sine_dc./(SNR_factor*sqrt(t_int));
im_blurred_noisy1 = im_blurred{2}+(uint8(256*noise_std(1)*randn(new_im_size)));
im_blurred_noisy2 = im_blurred{2}+(uint8(256*noise_std(2)*randn(new_im_size)));
%recover:
im_rec_noisy1 = motionblur_deconv(im_blurred_noisy1,h_mb{2});
im_rec_noisy2 = motionblur_deconv(im_blurred_noisy2,h_mb{2});
im_rec = motionblur_deconv(im_blurred{2},h_mb{2});
%plot:
%im_plot = cell(1,5);
% titles = cell(1,5);
% titles = {'','','','','',};
% im_plot{1} = im_blurred{1};
% im_plot{2} = im_blurred{2};
% im_plot{3} = im_blurred_noisy;
% im_plot{4} = im_rec;
% im_plot{5} = im_rec_noisy;
```

```
% plot_im_stack(im_plot,'q',titles)

% cut out segments for plotting:
im_plot = cell(1,4);
im_plot{1} = im_blurred_noisy1;
im_plot{2} = im_rec_noisy1;
im_plot{3} = im_blurred_noisy2;
im_plot{4} = im_rec_noisy2;
N_px = new_im_size;
for i = 1:length(im_plot)
    im_plot_cut{i} = im_plot{i}(N_px/2-25:N_px/2+25,N_px/2-50:N_px/2+50);
end

titles = {'','','','',};
plot_im_stack(im_plot_cut,'q',titles)
```

plot_im_stack.m

```
% draft for smart subplot-function
function plot_im_stack(im_cell, mode, title_cell)

 n_total = length(im_cell); %number
% Quadratic constellation
if(mode == 'q')
    % Determining the numbers of figures and plots in each figure:
    number_of_figures = ceil(n_total/12);
    n = 12*ones(1,number_of_figures);
    n_last_figure = mod(n_total,12);
    n(number_of_figures) = n_last_figure;

    for i=1:number_of_figures
        % determining the number of rows to plot
        n = n(i);
        if(n < 4)
            rows(i) = 1;
        elseif(n >= 4 && n < 9)
            rows(i) = 2;
        elseif(n >= 9 && n <= 12)
            rows(i) = 3;
        end
    %number of columns:
    columns(i) = ceil(n/rows(i));
    end

elseif(mode == 'l')
    number_of_figures = ceil(n_total/4);
    n = 4*ones(1,number_of_figures);
    n(number_of_figures) = mod(n_total,12);
    rows = n;
    columns = ones(1,number_of_figures);
elseif(mode == 'w')
    number_of_figures = ceil(n_total/4);
    n = 4*ones(1,number_of_figures);
    n(number_of_figures) = mod(n_total,12);
    columns = n;
    rows = ones(1,number_of_figures);
else
    disp('Invalid plot constellation')
end

    %making subplots:
    plot_count = 0;
    for k = 1:number_of_figures
        figure
        for j = 1:n(k)
            plot_count = plot_count+1;
            subplot(rows(k),columns(k),j)
            imshow(im_cell{plot_count})
            if(nargin==3)
            title(title_cell{plot_count})
            end
        end
        %set(gcf, 'Position', get(0,'Screensize')); % Maximize figure.
    end
end
```

E.6 Test Images

image_detector_sim.m

```
%% Simulation of detector background and noise

clear all
close all
% numbers for dark offset and fpn from experiment (1s 20deg):
dark_offset = 1123;
std_fpn = 248;
std_dn = 5.54;

im_size = [256 256];
%% 1D

fpn = std_fpn*randn(1,256);
dn = std_dn*randn(1,256);
snr_scale = 2;

%signal
x = 1:256;
num_of_periods = 15;
sine_period = length(x)/num_of_periods;
sine_offset = snr_scale*dark_offset;
sine_amp = 0.05*sine_offset;
sig_sine = sine_offset+sine_amp*sin(2*pi*1/sine_period*x);

% add photon noise:
% std_photon = sqrt(sine_offset);
% photon_noise = std_photon.*randn(1,256);
% sig_noise1 = sig_sine + photon_noise;
% figure
% stem(x,photon_noise)

yrange = 4000;

figure
subplot(1,3,1)
plot(x,sig_sine)
ylabel('Counts')
axis([1 256 0 yrange])

% figure
% plot(x,sig_noise1)
% axis([1 256 0 2*sine_offset])

sig_noise2 = sig_sine + dark_offset + fpn + dn;
sig_dark = dark_offset+fpn+dn;
% figure
% plot(x,sig_dark)
% axis([1 256 0 2*sine_offset])

sig_noise3 = sig_noise2-fpn-dark_offset;

subplot(1,3,2)
plot(x,sig_noise2)
ylabel('Counts')
axis([1 256 0 yrange])
subplot(1,3,3)
plot(x,sig_noise3)
ylabel('Counts')
axis([1 256 0 yrange])

%% 2D:
%Making image with sine signal and background

common_scale = 0.8; %must be scaled down for high background levels

%sine parameters:
number_of_periods = 11.5;
sine_dc = 0.6; % DC level of sine images
sine_dc = sine_dc*common_scale;
sine_angle = 30;
sine_amp = 0.05*sine_dc;
std_phn = sqrt(sine_dc);

snr_scale = 1; %ratio between sine dc level and background level
bg_scale = sine_dc/(snr_scale*dark_offset);

%making image noise:
fpn = std_fpn*randn(im_size); %background fixed pattern noise
dn = std_dn*randn(im_size); %background dark (thermal) noise
%phn = std_phn*randn(im_size); %photon noise
```

```
bg = zeros(im_size)+dark_offset+fpn+dn; %background signal
%bg_normalized = bg./max(bg(:)); %normalizing to one;
bg_normalized = common_scale*bg_scale.*bg;
dn_normalized = common_scale*bg_scale.*dn;
im_bg = uint8(256*bg_normalized); %making an 8-bit image
im_dn = uint8(256*dn_normalized);

im_sig = make_sinus_image(im_size(1), number_of_periods, sine_dc, ...
    sine_amp,sine_angle);
%im_sig_noisy = im_sig+(uint8(256*phn)); %tried to make photon noise, get the ...
    wrong scaling

im_total = im_sig+im_bg;
im_dn_sig = im_sig+im_dn;

figure
imshow(im_bg)
%save_figure(gcf,['bg_k' num2str(snr_scale)])
figure
imshow(im_sig)
%save_figure(gcf,['sig_k2'num2str(snr_scale)])
figure
imshow(im_total)
%save_figure(gcf,['total_k'num2str(snr_scale)])
figure
imshow(im_dn_sig)
```

make_test_im.m

```
%Script for making one single test image with noise
im_size = 256;
number_of_periods = 20;
%
t_int = 1.6;
SNR_factor = 20;
%
sine_dc = 0.6; % DC level of sine images per second
sine_angle = 10;
sine_amp = 0.05*sine_dc;
noise_std = sine_dc./(SNR_factor*sqrt(t_int));
%noise_std = 0;

im_sine = make_sinus_image(im_size, number_of_periods, sine_dc, ...
    sine_amp,sine_angle);
im_noisy = im_sine+(uint8(256*noise_std*randn(im_size)));
figure
imshow(im_noisy)
```