SOLAR OCCULTATION IMAGING OF DUST IN THE MARTIAN ATMOSPHERE

by

Ryan Robski

Submitted in partial fulfillment of the requirements
for the degree of Master of Science

at

Dalhousie University
Halifax, Nova Scotia
November 2012

DALHOUSIE UNIVERSITY

DEPARTMENT OF PHYSICS & ATMOSPHERIC SCIENCE

The undersigned hereby certify that they have read and recommend to the Faculty of Graduate Studies for acceptance a thesis entitled "SOLAR OCCULTATION IMAGING OF DUST IN THE MARTIAN ATMOSPHERE" by Ryan Robski in partial fulfillment of the requirements for the degree of Master of Science.

Dated: November 22, 2012

Supervisor: ____________________

Readers: ____________________

____

DALHOUSIE UNIVERSITY

DATE: November 22, 2012

AUTHOR: Ryan Robski

TITLE: SOLAR OCCULTATION IMAGING OF DUST IN THE MARTIAN ATMOSPHERE

DEPARTMENT OR SCHOOL: Department of Physics & Atmospheric Science

DEGREE: M.Sc. CONVOCATION: May YEAR: 2013

Signature of Author

Table of Contents

List of Tables

List of Figures

Abstract

As part of the ExoMars space programme, the 2016 Trace Gas Orbiter mission was announced. The Martian Atmospheric Trace Molecule Occultation Spectrometer (MATMOS) was a proposed Fourier transform spectrometer and solar imager concept pair that would provide for trace gas detection and aerosol observation of the Martian atmosphere. Martian aerosols namely CO_2 crystals, water-ice crystals, and dust have been observed during past missions; however, observations have failed to fully characterize their physical and optical properties.

This thesis presents an analysis of the ability of the proposed imager to determine the pointing of the spacecraft independent of the spectrometer. Furthermore, proof of concept is presenting showing the ability to, in laboratory conditions, characterize the precision and stability of the imager. Finally, window regions in the transmittance spectrum of the Martian atmosphere are determined simulating the Martian atmosphere and viewing geometry.

List of Abbreviations and Symbols Used

α: Angstrom Coefficient

δ: Angular Diameter

λ: Wavelength

τ: Optical Depth or Integration Time

d: Diameter

I: Attenuated Radiation Intensity

I_0: Initial Radiation Intensity

k_e: Extinction Coefficient

R: Object Distance

r: Radius

r_{eff}: Effective Radius

v_{eff}: Effective Variance of Distribution

x: Size Parameter

z: Length of Optical Path

ADC: Analog-to-digital converter

APE: Absolute Pointing Error

AVAR: Allan Variance

DNL: Digital Non-linearity

ESA: European Space Agency

FOV: Field of View

FTS: Fourier Transform Spectrometer

IFOV: Instrument Field of View

INL: Integral Non-linearity

JIDT: Joint Instrument Definition Team

JPL: Jet Propulsion Laboratory

LSB: Least Significant Bit

MATMOS: Mars Atmospheric Trace Molecule Occultation Spectrometer

NASA: National Aeronautics and Space Administration

NOMAD:	Nadir and Occultation for Mars Discovery
PI:	Principal Investigator
TGO:	ExoMars Trace Gas Orbiter

Acknowledgements

First and foremost, I would like to acknowledge and thank my supervisor, Dr. James Drummond. I am enormously grateful for his guidance and mentorship over the past two years and for challenging my mind in new and provoking ways. Also, perhaps most importantly, I would like to thank Jim for his patience. Having tested them sufficiently, I can confirm their existence and assure you they are in good working condition.

I would also like to thank my fellow research group members and office mates. Thank you to Lisa LeBlanc and Yan Tsehtik for making me feel welcoming in the group and providing me with the support I too often required. Thanks to Alexey Tikhomirov for his insight and his related work, some of which appears in this thesis. To Jonathan for our countless conversations, debates, sanity checks, and for generally being another graduate student with whom I could share my experiences and frustrations. And to Chris Perro, Steven D'Andrea, Kim Sakamoto, Jason Hopper, and Colin-Pike-Thackray, all of whom have made my time at Dalhousie both interesting and enjoyable.

Finally, I would like to thank the Natural Sciences and Engineering Research Council of Canada and the Dalhousie Physics & Atmospheric Science department for financial support.

Chapter 1

Introduction

1.1 Motivation for Space Exploration

Space exploration is a costly endeavour, however, the rewards make it worth every penny. Space exploration, and science more broadly, helps us to gain a better understanding of the universe we live in. Asking and attempting to answer questions such as "Where did we come from?", "How was the earth formed?", and "What else is out there?" have always helped to push our imagination and innovation forward. This in turn has also led to significant technological developments that have made their way into universities, the private sector, and into the lives of individuals.

1.2 Why Study Mars?

Humans have an inherent sense of curiosity. Through scientific research we are always looking to push boundaries of understanding and explore new frontiers. After the historically significant "Space Race" of the mid 20th century, wherein we saw the greatest acceleration in space curiosity and exploration that culminated in a successful manned mission to the Moon, it seems natural that our next frontier would be our next closest neighbour: Mars.

Understanding the Martian environment, both atmosphere and geology, help us to focus our investigations and ask the right questions. For instance, in 1784 a British astronomer named Sir William Herschel famously wrote of the dark coloured oceans he observed on the Martian surface. He openly postulated that the inhabitants of Mars "probably enjoy a similar situation to our own." [10] Of course, we know now that based on the surface conditions of low pressure and cold temperatures on Mars that it would be impossible for water to exist in a liquid state.

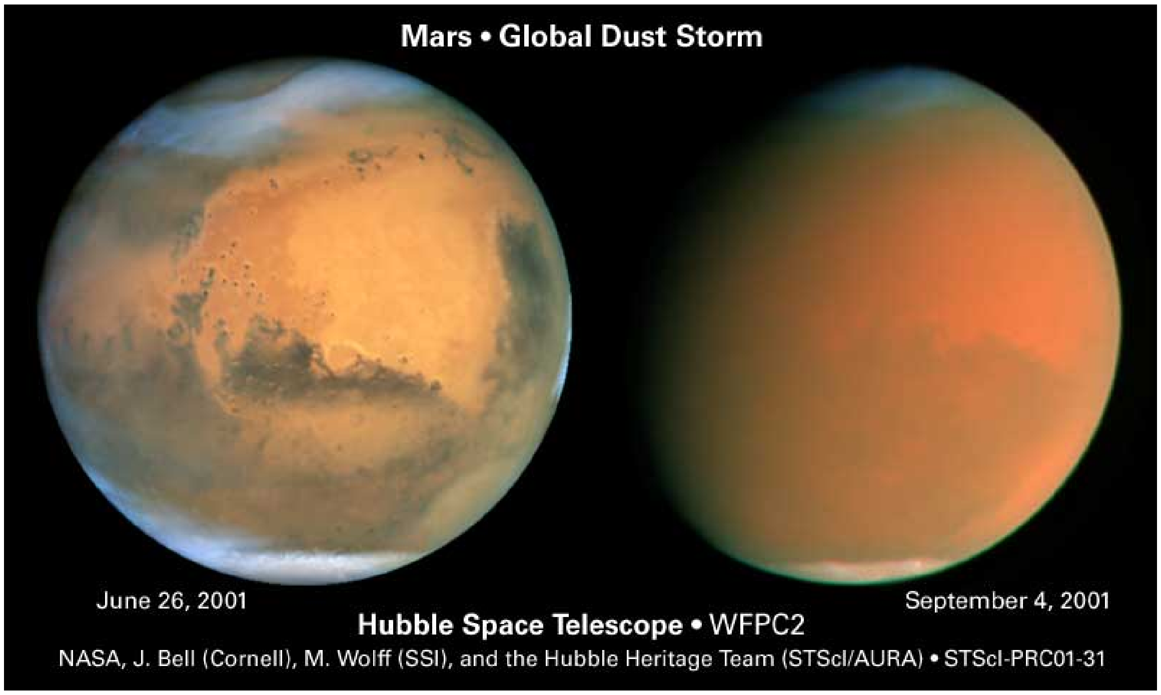

Mars • Global Dust Storm
June 26, 2001
September 4, 2001
Hubble Space Telescope • WFPC2
NASA, J. Bell (Cornell), M. Wolff (SSI), and the Hubble Heritage Team (STScI/AURA) • STScI-PRC01-31

Figure 1.1: An image of the Martian atmosphere before and during the 2001 global dust storm, taken from the Hubble Telescope. (Image from J. Bell/NASA)

Martian Dust

One of the most stark differences between the lower atmospheric climate conditions on Mars compared to those of Earth is the consistent presence of atmospheric dust. The suspension of this granular aerosol is a planet-wide phenomenon of varying in size and distribution.

Although the presence of the Martian dust was established by early observations of Mars, the shape and size of the dust particles is not well known. Published postulated distributions centre on a mean radius of about 1-1.5 μm [2]. Additionally, the particle size distribution has not been well characterized leading to many competing models. Although the dust is a persistent feature, it regularly varies in concentration on both diurnal and seasonal cycles. Moreover, localized dust devil phenomena and global planet-encircling dust storms have been observed. The exact cause of these events is unknown. Figure 1.1 shows two images of Mars, one viewed during one such dust storm.

Challenges exist as a result of our limited knowledge of the dust. Given its persistence and abundance, dust will continue to appear in all atmospheric sounding and remote sensing experiments. As such, priority must be given to furthering our understanding of the dust, its impact on the Martian climate, and its implications on our ability to study other aspects of Mars. Without this, we leave a large unknown completely unconstrained to run though all other data.

Gaseous Composition

Of specific interest to society is the question of "Are we alone?". Based on the prerequisite conditions for biological activity, as we know it, to exist, the most likely candidate within the constraints of our solar system is Mars.

Further, fuelling the mystery is the observation of methane present on Mars. Methane has been reportedly observed by Earth-based instruments in relative abundances of about 10 parts per billion (ppb) in some regions [11]. Further, there has been reported detection from satellite measurements and observed spacial variability of methane with abundances ranging from 0-30 ppb [12]. Despite independent reports of its existence, many groups are still very sceptical of the measurements and methane existing on the Martian surface. Even accepting its presence, experts are divided on its origin.

What makes localized observations of methane puzzling is that given the consistent surface winds and good vertical mixing we observe, the methane should be well-mixed into the atmosphere. Also, methane is disassociated by solar ultra-violet light. Given the thin atmosphere and absence of a absorbing ozone layer such as the one on Earth, it is logical to conclude that, short of any unknown chemical pathways, methane must be being replenished in the atmosphere. Moreover, the majority of methane on Earth is produced by biological sources, thus making this observation further intriguing. Figure1.2 shows an image of methane measurements on the Martian surface.

This leads many to believe that the methane must either be being produced actively on the surface or being released from a subterranean source, either from bacterial activity or geothermal processes. Either way, higher resolution spatial mapping will help to identify good candidate locations for future ground missions to explore these possible sources.

This thesis will discuss some of the design aspects of a pair of instruments intended to go to Mars and observe the atmosphere directly, looking at the dust and the

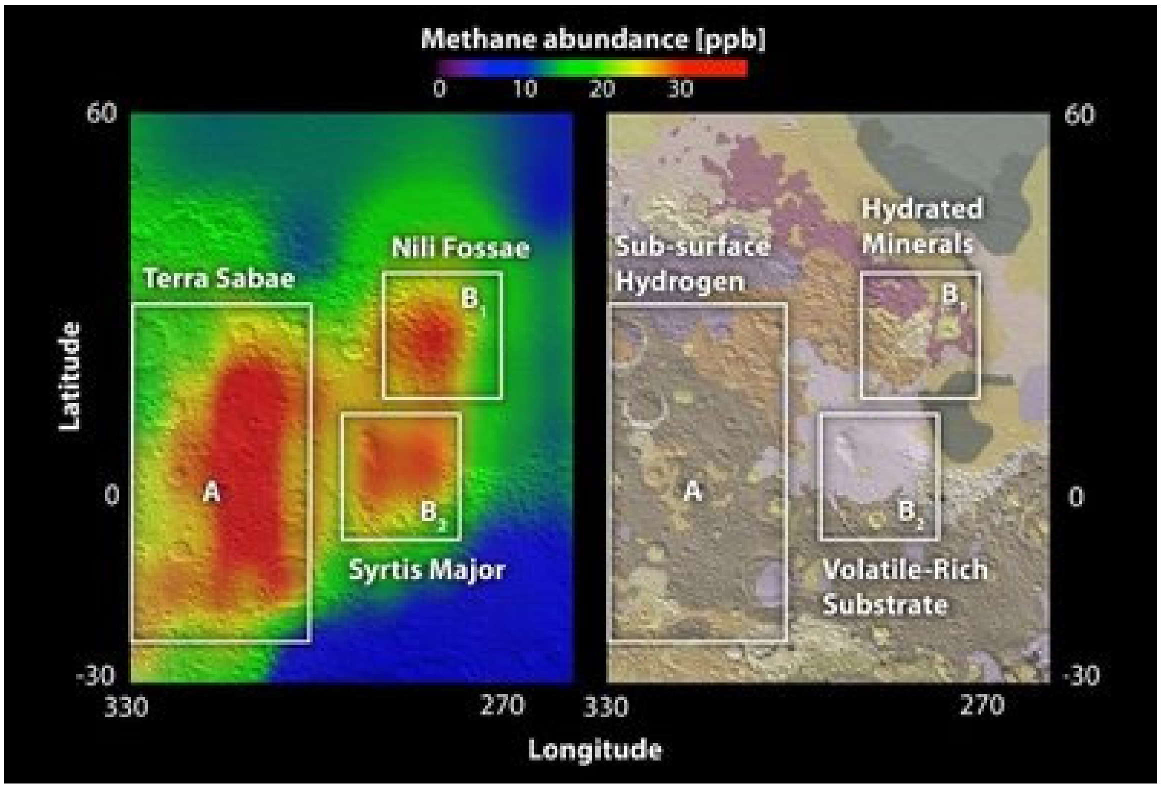


Figure 1.2: **Left**: A contour plot showing detected surface methane distribution on Mars. **Right**: A surface image of the same region. (Image from Susan Tawdry/NASA)

gaseous composition. These instruments will help us to understand some of the many unknowns of the Martian atmosphere.

Chapter 2

Background

2.1 Early Mars Missions

Early missions to study Mars began in the late 1960s and early 1970s with a series of attempts by the USSR. After a number of failed launches, communications errors, and some mixed successes, NASA was the first to successfully place a man-made satellite, the Mariner 9, in orbit of Mars [13]. Aboard the Mariner 9 was an Infrared Interferometer Spectrometer (IRIS) used to examine a broad range of infrared wavelengths. Given that relatively little was known of Mars and its composition at the time, one of the goals of IRIS was to search for new molecular species. IRIS reported the strong affect the dust had on its data gathering ability, as well as water vapour observations, especially around the polar region [14].

Several years later in 1976, NASA commissioned another mission for a joint orbiter and lander launch. This mission, named Viking, was equip to search for microbial life on the surface, measure the water vapour abundance in the column, look at mineral and surface composition, and measure the atmospheric temperature [13].

After a long hiatus through the 1980s, Mars exploration was renewed with the launch of NASA's Pathfinder lander, launched in 1997. Since then Mars has been the subject of many robotic missions launched by collaborators from several countries. This has led us to where we are today with our understanding of the red planet, but has also allowed shown us how much we have left to learn.

2.2 Known Martian Characteristics

Mars is the fourth planet from the Sun in our solar system and a neighbour of Earth. Aside from its proximity and terrestrial nature, many of the known characteristics of Mars sharply contrast those of Earth, despite often being thought of as similar to Earth and a possible host candidate for microbial life. (See table 2.1 for a comparison of some physical quantities.)

The atmosphere of Mars is quite thin. The surface pressure, varying significantly with location and with the seasonal changes, is on the order of 400-900 Pa, less than one percent of that on Earth [1]. Through seasonal temperature changes, both water and carbon dioxide will condense and sublimate on alternating polar regions of Mars. These processes may vary the surface pressure by up to 30% on an annual basis [15]. Given the thin atmosphere, refraction plays much less of a role for light being transmitted through the atmosphere than for the similar case on Earth [16].

	Mars	Earth
Mean Radius	3390 km	6371 km
Mass	6.4185x10^{23} kg	5.9736x10^{24} kg
Orbital Semi-Major Axis	228x10^{6} km	150x10^{6} km
Orbital Eccentricity	0.093	0.017
Orbital Period	687 days	365.24 days
Axis Tilt	25.19°	23.44°
Rotational Period	1479.6 minutes	1440.0 minutes
Typical Surface Gravity	3.7 m/s^2	9.8 m/s^2
Typical Surface Pressure	636 Pa	101300 Pa
Mean Surface Temperature	210 K	287 K
Typical Scale Height	11 km	8 km

Table 2.1: Comparison of similarities and differences of Martian characteristics to those of Earth [1].

The bulk of the Martian atmosphere is composed of carbon dioxide – in excess of 95% by mole fraction. Other major contributors, by mixing ratio, are: nitrogen, argon, oxygen, carbon monoxide. Trace gases in parts per million and billion concentrations also include: water vapour, nitrogen oxide, their isotopologues, and others in lesser concentrations [1]. See Table 2.2.

As with Earth, the surface temperature on Mars varies significantly with latitude and with daily and seasonal cycles. One mission in lower mid-latitudes, around 23 degrees North, reported diurnal temperature variation ranging between 184 K and 242 K [1]. Given this, and Mars' smaller size and thus weaker gravitational pull, Mars has a typical scale height of about 11 km, comparable to that of Earth's 8.5 km at mid latitudes. This low temperature and pressure environment also results in the absence of a liquid phase for many molecular species, notably water (see figure 2.1).

Table 2.2: Largest gaseous contributors, by concentration, to the Martian atmosphere [1].

Species	Concentration
Carbon Dioxide (CO_2)	95.32%
Nitrogen (N_2)	2.7%
Argon (Ar)	1.6%
Oxygen (O_2)	0.13%
Carbon Monoxide (CO)	0.08%
Water (H_2O)	210 ppm
Nitrogen Oxide (NO)	100 ppm
Neon (Ne)	2.5 ppm
Hydrogen-Deuterium-Oxygen (HDO)	850 ppb
Krypton (Kr)	300 ppb
Xenon (Xe)	80 ppb

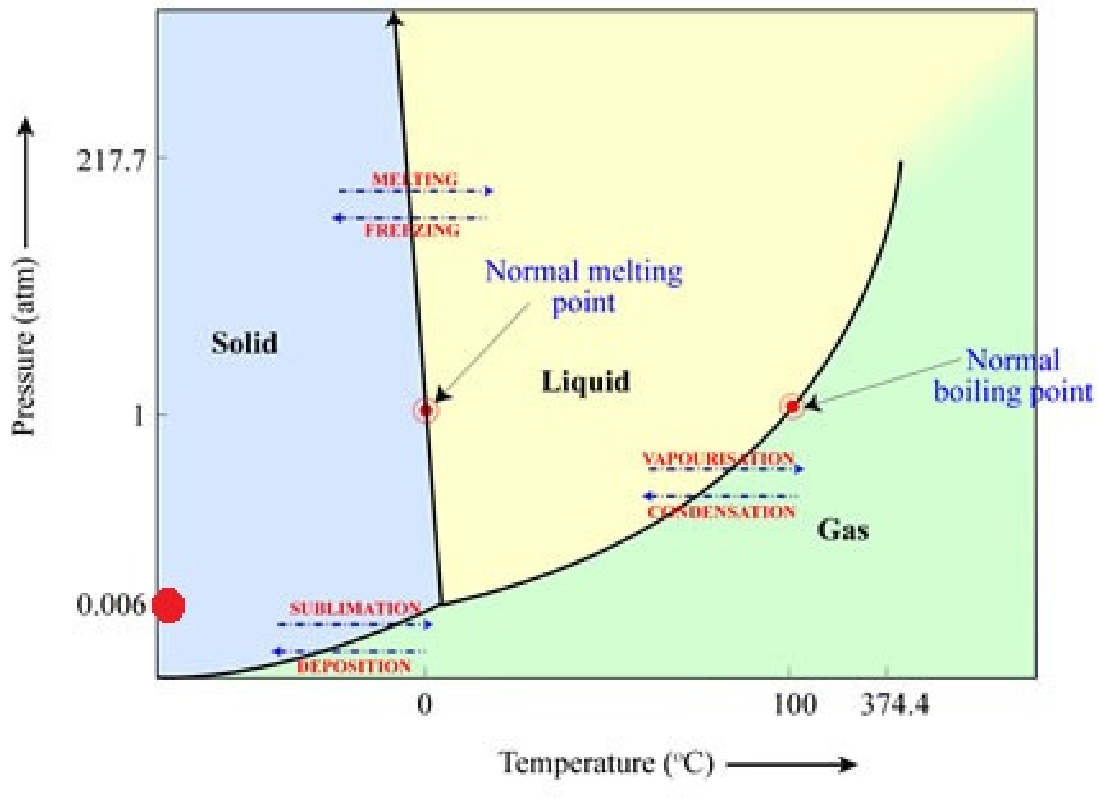


Figure 2.1: Phase diagram for water. The red dot represents the typical surface conditions on Mars [5].

Table 2.3: Areocentric longitude values and their associated equinox or solstice.

L_S=0^o	Vernal Equinox
L_S=90^o	Summer Solstice
L_S=180^o	Autumnal Equinox
L_S=270^o	Winter Solstice

Mars rotates with a tilt of 25.19^o, similar to that of Earth. Each solar day on Mars is called a "Sol" and is roughly equal to a solar Earth day, longer by about 39.6 minutes. As a result of the axis tilt, Mars experiences seasons in the same way we do on Earth, with alternating hemispheres being oriented more directly towards the sun. In order to report the seasonal conditions on Mars as a function of an entire seasonal cycle, the degree of progression is commonly displayed in terms of the areocentric longitude L_S. See table 2.3 for defined areocentric longitudinal values.

Martian Aerosols

As light (or radiation) passes through a region containing gas molecules or some physical obstruction, say aerosols, one of three things can happen: the light will continue unobstructed, it will be scattered, or it will be absorbed. The latter two in combination are referred to as extinction or attenuation [17].

The loading of dust in the ray path is often referred to in terms of its monochromatic optical depth (or optical thickness). Optical depth is a dimensionless quantity that, for a given wavelength, is defined as:

$$\tau = ln(\frac{I_0}{I})$$

where I_0/I is the ratio of the total intensity of radiation incoming to the intensity that is not scattered or absorbed at that particular wavelength [17]. This relationship is known as Beer's Law.

The optical depth can also be expressed as:

$$d\tau = -k_e dz$$

where k_e is the extinction coefficient (or extinction efficiency) and dz is a piece of the optical path through which the light travels [17]. The extinction coefficient is wavelength dependent and thus should be expressed for each wavelength in a polychromatic ray.

Aerosols in the Martian atmosphere have been observed mainly in three forms: CO_2 ice crystals, H_2O ice crystals, and suspended dust particles.

Given that carbon dioxide is abundant in the Martian atmosphere and the characteristically low temperatures, it is not surprising to observe CO_2 "ice" aerosols. Such crystals have been widely reported in the colder upper atmosphere of Mars in polar regions [18] [19], at mid-latitudes [20], and in the sub-tropical region [21]. Such aerosols are observed as detached, cirrus-like clouds like those observed on Earth. The clouds exist primarily in the night and early morning before sublimating in response to radiative heating. The reported shape and size of these CO_2 crystals is varied [2].

Water-ice crystal aerosols have been observed and reported primarily in the northern polar region [22], but also extending as far as into the northern equatorial region through spring and early summer seasons during the Martian aphelion due to the highly elliptical orbit of Mars in which the global temperature can vary by up to 20 K over its orbital path [23] [24]. Further, there are large asymmetries in the water vapour concentrations, and consequently water ice aerosol appearances, between the northern and southern hemispheres. Clancy et al. (1996) postulate this may be due to a coupling of the water vapour saturation altitude with the Hadley circulation at solstice whereby a disproportionate amount of atmospheric water vapour is pumped towards the aphelion summer hemisphere, presently the north.

Martian Dust

Perpetually suspended dust is one of the most significant defining characteristics of the Martian atmosphere. The dust is lifted from the Martian face by surface winds and distributed throughout the boundary layer by horizontal and vertical currents. The dust is a strong absorber of both solar and thermal radiation and thus has significant impacts on climate through radiative forcing. Its concentrations also has diurnal and seasonal variations [25].

The vertical structure and layering of the dust is also of importance. Although variable given the time of day and season, dust may be expected to maintain a

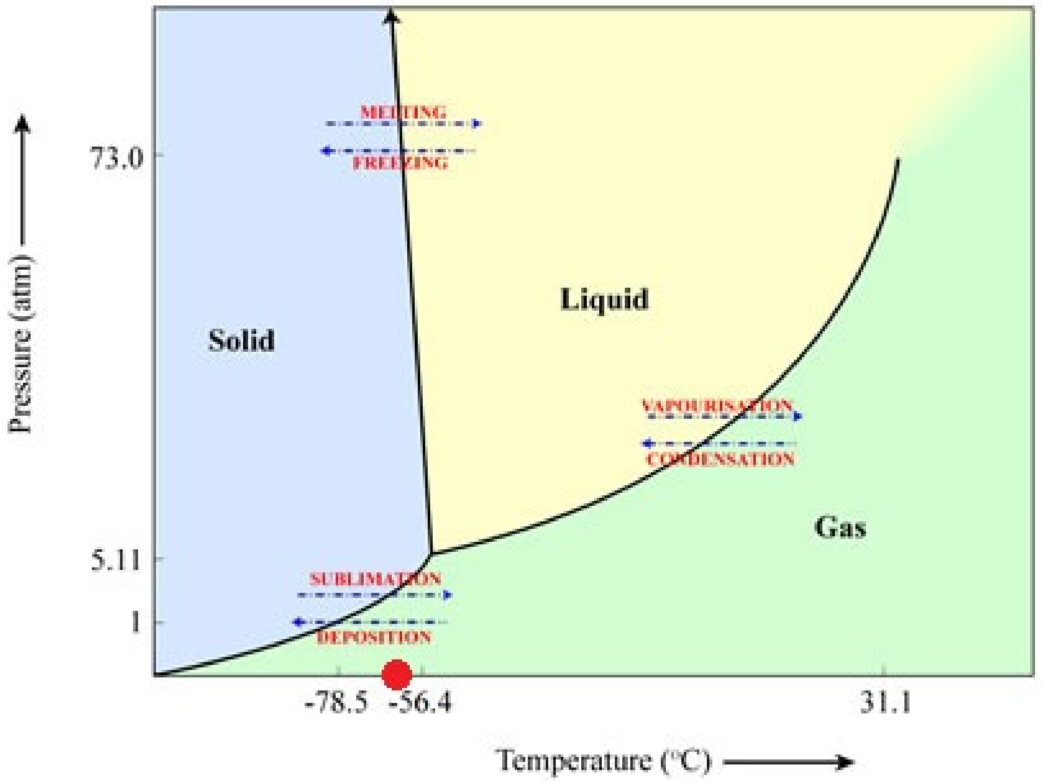


Figure 2.2: Phase diagram for carbon dioxide. The red dot represents the typical surface conditions on Mars [5].

constant presence in the lower boundary layer due to surface wind and vertical mixing [26]. Reported occultation measurements by the Phobos spacecraft show a reduction in the particle number concentrations with increasing altitudes [27]. Further, the effective radius also decreases with altitude from 1.6 μm at 15 km to 0.8 μm at 25 km. This intuitively makes sense as larger dust particles would tend to fall out. Additionally, given that the ground is the source of the dust, it would make sense to observe larger concentrations at lower altitudes. Additionally, models approximate the scale height of the dust between 10 and 15 km [28].

Global dust storms have been observed and studied as an occurrence unique to Mars. Such dust storms appear to occur during southern summer, which corresponds to Mars at perihelion. In 1971, the Mariner 9 spacecraft was able to observe a global dust storm [29]. This was the first time researchers were able to examine such a phenomenon from such a range. Later, in 1977 during the Viking mission, two global dust storms were observed to rise and fall around the perihelion time period [30]. During such storms dust aerosols were observed to have reached heights of 60 km and higher [31] [32]. Later, in 2001, the Mars Global Surveyor collected data on the global dust storm that occurred that year [33]. In many cases it can take months for the dust to return to regular loading levels.

2.3 Questions About Mars

There continues to be much left to uncover about Mars, its environment, and its history.

Water and Hydrological Cycle

One topic of study is the hydrological cycle on Mars. Subterranean ice has been detected using instruments aboard the Mars Odyssey [34]. The ice is reported at higher altitudes, close to the winter pole, and at varying depths beneath the surface. Complementary models of water-ice distribution and stability allude to an active hydrological cycle that is not still well understood [35] [36]. The use of General Circulation Models has also been employed to compare against collected data of cloud observation and its seasonal variations [37]. A better understanding of the hydrological cycle and whether subterranean water does or has ever existed in liquid form will help to gain insight into historical processes and existences.

Gas Disequilibrium & Methane

Although the bulk atmospheric contributors are known relatively well, trace gas and more localized components are still of interest. Trace gases, including the previously mentioned methane, detected in localized regions will allow for study of possible sources and sinks. Inhomogeneities could possibly be due to past meteor impacts, geological activity, or even past or current biological activity [9]. In light of this, the Martian geography and geochemical processes are areas of significant interest. Disequilibrium in surface-level atmospheric constituents could indicate biological activity, but also could be caused by volcanic or other geological activity just beneath the surface [9] [38]. Regardless of the nature of the sources or sinks, the locations of the inhomogeneities would serve as good candidate locations for future study.

Martian Dust

The composition of the dust is still not well known, however, it is believed the dust is a product of weathering of the Martian surface. The largest barrier to our ability to properly characterize the dust is our inability to retrieve a sample from the planet. Despite this, many laboratory experiments are attempted to characterize the dust by comparing the similarities in optical properties to dust and other minerals present on Earth. A usual comparator is montmorillonite, a silicate-based clay found in volcanic ash [39]. In addition to montmorillonite, other proposed compositions include palagonite, basalt, and combinations thereof [29] [40].

One important quantity to help define the scattering properties of aerosols is the size parameter. The size parameter is another dimensionless quantity that describes the relative sizes of particles to the wavelength of the radiation which it is scattering. It is given as follows:

$$x = \frac{2\pi r}{\lambda}$$

where r is the radius of a spherical scattering particle and λ is the wavelength of light.

Table 2.4: A summary of published Martian dust distribution parameters.[2]

Publication	r_{eff} (μm)	v_{eff}	Notes
Toon et al. 1997	2.75	0.42	Viking Orbiter 5-40μm wavelength
Drossart et al. 1991	1.24	0.25	Phobos Mission 1-3μm wavelength
Korablev et al. 1993	0.8 at 25 km 1.6 at 15 km	0.2±0.1	Phobos Mission Solar occultations at 1.9 and 3.7 μm wavelengths
Pollack et al. 1995	1.85±0.3 1.52±0.3	0.5±0.3	Viking Lander Replacing earlier results from Pollack 1977, 1979 papers
Clancy et al. 1995	1.5	0.997	Ultraviolet to thermal IR
Tomasko et al. 1999	1.6±0.15	0.2-0.5 or more	IMP Observations

From the size parameter we can estimate the scattering behaviour of the light. Scattering occurring when the size parameter x << 1 is called Rayleigh scattering. Size parameters of 0.1 < x < 50 are referred to as being part of the Mie regime. When the size parameter is greater than 50 this is known as the geometric regime, where classical geometric optics dominate.

The size and shape of the dust are also important parameters when it comes to characterizing the dust through its optical properties. Two variables are used to parametrize the distribution of dust sizes: r_{eff} represents the geometric cross-section-weighted mean radius of the distribution while v_{eff} characterizes the dimensionless variance of the distribution [2]. Table 2.4, taken largely from Tomasko et al. (1999), shows reported radii and variances reported.

As described above, the optical depth may vary with wavelength. This is a result of the extinction efficiency having an implicit wavelength dependency. The extinction efficiency may be expressed in the following general form:

$$k_e \propto \lambda^{-\alpha}$$

where α represents a value called the Angstrom coefficient.

The Angstrom coefficient, and thus the relationship between the optical thickness and wavelength, is largely determined by the scattering regime in which the size parameter falls. For instance, it is known that for Rayleigh scattering the Angstrom coefficient is 4. This is normally seen as an upper limit for the Angstrom coefficient. The Angstrom coefficient may vary much more rapidly in the Mie regime than in other regimes due to the high dependence the scattering efficiency has on the shape and orientation of the scattering particles.

The dispersion of particle sizes is often described by a log-normal or gamma distribution, although numerous other distributions have been used [41]. As for the shape of the particulate dust, various forms have been proposed ranging from spherical [40] to disk particle shapes [42].

Chapter 3

The ExoMars Trace Gas Orbiter Mission

3.1 Announcement

In January of 2010 the European Space Agency (ESA) and the National Aeronautics and Space Administration (NASA) jointly issued an Announcement of Opportunity to solicit proposals for instruments to be mounted aboard the Trace Gas Orbiter (TGO), the first mission in the new Mars exploration programme, ExoMars [43]. The Trace Gas Orbiter mission was designed to lay the foundation for the first ever Martian sample return mission to take place in the next decade.

Designed with a 2016 launch date, the orbiter mission is led by ESA. The subsequent mission, set to follow in 2018, will be a surface mission consisting of two rovers, one designed by each ESA and NASA, delivered together as by a single launch vehicle. The rovers will host complementary instrumentation and search for evidence of past or present life while also demonstrating use of novel technologies for use in the yet to be announced return mission.

3.2 Mission Complications

Toward the end of 2011 it became obvious to all parties involved that there would be complications with the ExoMars mission as it was originally envisioned. The main concern was the budget cuts at NASA and the effect that would have on their ability to deliver their end of the bargain, namely the launch vehicle and the Trace Gas Orbiter. At the time one possible solution that presented itself to ESA was to forge a new partnership and seek out other collaborators.

In December of 2011, senior administrators from ESA, NASA, and the Russian Federal Space Agency, Roscosmos, met to discuss collaboration. At the time, two working groups were struck: one to redesign the 2016 payload to incorporate Russian-designed instruments and the other to examine the possibility of a launch aboard a

Russian Proton rocket [44]. Despite maintaining funding for the 2018 leg of the ExoMars campaign [45], it is uncertain what effect Russia's inclusion will have on the second phase. The exact details of the Russian-European partnership have yet to be ironed out.

3.3 Objectives and Goals

The Trace Gas Orbiter mission was constructed with primary objectives; a technological objective and a scientific objective. The technological objective is to successfully complete the "entry, descent, and landing (EDL) of a payload on the surface of Mars." This goal is to demonstrate capability for phase 2 of the exploration programme: the deployment of twin rover vehicles. More important to our work is the scientific objective "To study Martian atmospheric trace gases and their sources." [4] Further, the TGO will provide telecommunication and data relay services for the 2018 rover mission and other landed missions through 2022 [4].

A Joint Instrument Definition Team (JIDT) was struck in 2009 to "review the viability of these objectives within the constraints of such a 2016 joint mission". [46] Through this work the JIDT identified a set of prioritised scientific goals for the TGO. They are:

1. Detect a broad suite of atmospheric trace gases and key isotopes.

2. Characterise the spatial and temporal variability of methane and other key species.

3. Localise sources and derive the evolution of methane and other key specific, and their possible interactions.

4. Image surface features possible related to sources and sinks. [4]

In the next few sections I will briefly discuss some of the chosen TGO instruments and how their techniques and methods will help to accomplish the goals set out by ESA and NASA, as we understand them today.

Table 3.1: Instruments originally selected for the ExoMars Trace Gas Orbiter.

Mars Atmospheric Trace Molecule Occultation Spectrometer	MATMOS
Nadir and Occultation for Mars Discovery	NOMAD
ExoMars Climate Sounder	EMCS
High-Resolution Stereo Colour Imager	HiSCI
Mars Atmospheric Global Imaging Experiment	MAGIE

3.4 Mars Atmospheric Trace Molecule Occultation Spectrometer (MATMOS)

MATMOS is the Mars Atmospheric Trace Molecule Occultation Spectrometer headed by Paul Wennberg of the California Institute of Technology (Caltech). This instrument is in collaboration between Caltech, the Canadian Space Agency (CSA), and other Canadian and American partners.

MATMOS is a solar occultation Fourier Transform infrared spectrometer chosen to directly address the following goals of the ExoMars Trace Gas Orbiter mission:

1. Determine the origin of trace gasses diagnostic of active geological and biogenic activity.

2. Quantify the lifetimes of these diagnostic gases in the context of the atmospheric state.

3. Provide definitive detection and essential support for the TGO localization effort through identification of target gases and regions for focused mapping.

4. Solve the mystery of Mars methane. [9]

Occultation Technique

There are different types of occultation techniques that have been employed for planetary examination. There are three parties of importance during an occultation: the observer, the obstructing object, and the obstructed object. Often times these techniques are characterized by a combination of what they are looking past or through and what they are looking beyond to. For example, planetary stellar occultation would refer to looking beyond a planet that will be or was eclipsing a distant star. In many cases the terms “observer” and “obstructed object” may be reversible depending on the geometry.

MATMOS employs a technique for studying the Martian atmosphere called solar occultation. Once a satellite is put into orbit around Mars the event occurs during a sunrise or sunset from the perspective of the satellite. That is to say, as the satellite begins to see the sun appear from behind the planet, or once the sun begins to disappear behind the planet, this is an occultation event.

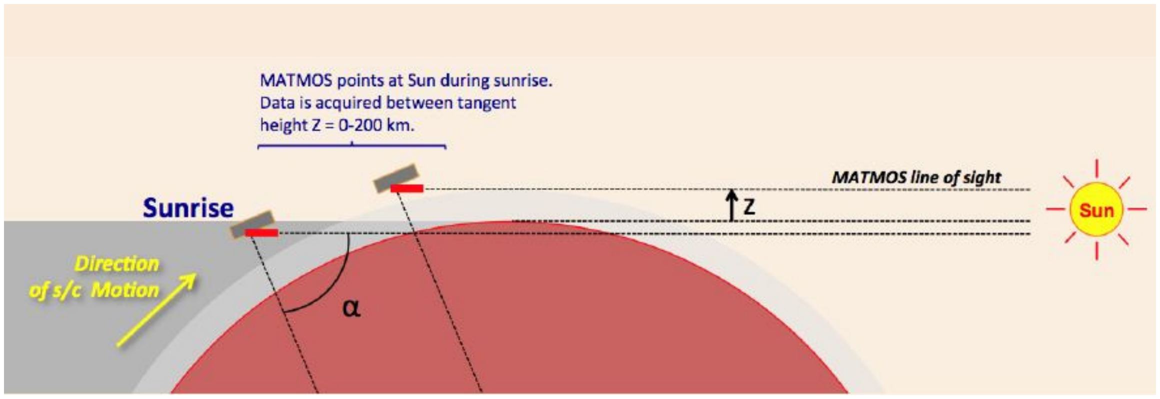


Figure 3.1: A cartoon diagram of the sunrise occultation geometry [3].

For a solar occultation of Mars, the key is collecting data at numerous instances while the occultation is occurring. As such, the light travelling from the sun into the satellite-mounted instrument will have each passed through different layers for varying path lengths. These independent measurements of different portions of atmosphere allow for the deducing of atmospheric components are differing altitudes.

Each occultation may begin or end with an exo-atmospheric measurement. This measurement, unobstructed by atmospheric gases or aerosols, provides the baseline for each occultation. As the satellite descends over the horizon, the ray-path travels through more and lower portions of the atmosphere. In a very basic sense, creating a model atmosphere consisting of concentric, homogeneous layers equal to the number of atmospheric measurements taken during the occultation a deconstruction algorithm known colloquially as “onion peeling” allows for each layer to be characterized.

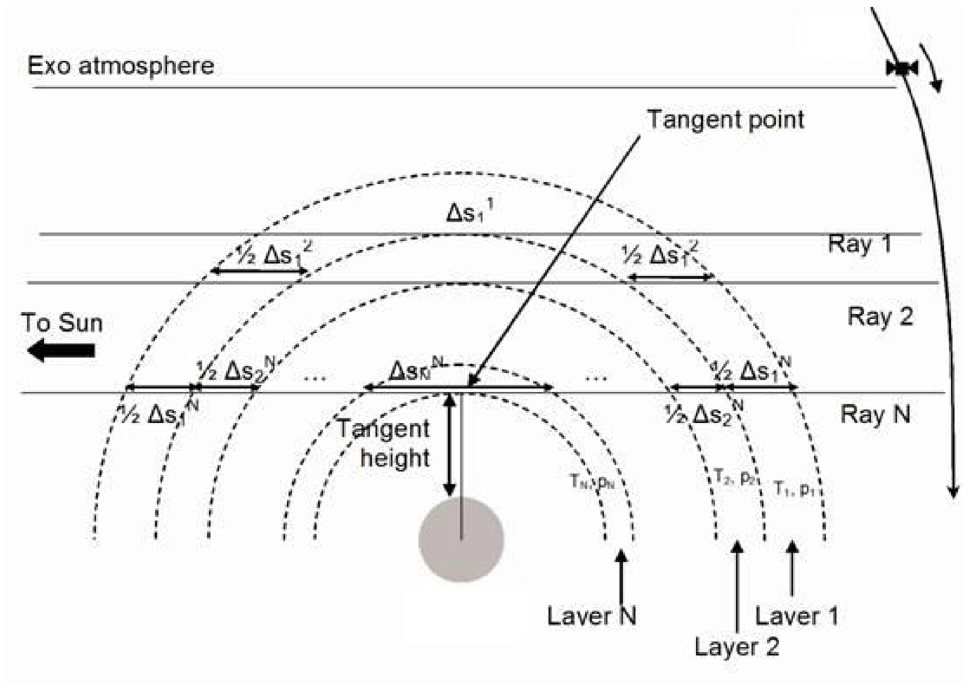


Figure 3.2: A visualization of the way light travels and is observed during an occultation event [6].

In practice, many sophisticated retrieval schemes are used (e.g. maximum likelihood) and compensations are made for specific obstructions, such as clouds or localized dust storms.

Solar Imager

The first part of the MATMOS spectrometer is a four-channel solar imager. The scientific goals of the imager are to image the full solar disk and resolve aerosol and thin cloud layers; however, the imager will provide a wealth of other useful scientific and technical information.

The imager will produce a series of time-stamped image sequences for each sunrise and sunset occultation. These four-channel image series will provide information of vertical profiles of atmospheric absorption and aerosol extinction, as well as global maps of seasonal atmospheric opacity and aerosol opacity [3].

Perhaps most importantly from a technical side, the imager will provide the science teams with information surrounding the pointing of the spacecraft. We will show that the imager is capable of determining the precise orientation of the spacecraft and thus

Table 3.2: Specifications for the MATMOS four-channel imager [3] [4].

Attribute	**Specification**
Solar Angle Subtended	5.56-6.79 mrad
Image Field of View	7 mrad
Instantaneous Field of View	0.115 mrad 0.2km on the limb
Profile Altitude Coverage	0-200 km
Spacecraft Pointing Accuracy	±0.5 mrad over occultation
Spectral Bands	320±20 nm, 512±10nm, 650±10nm, 1024±20nm.

the pointing of all instruments (to a higher degree of accuracy than the spacecraft).

Instruments mounted aboard the TGO spacecraft are to be static, meaning they will have no ability to point themselves and thus rely entirely on the spacecraft pointing. This presents significant challenges, especially as the spacecraft begins to "drift" and pointing accuracy can no longer be guaranteed to the same level of precision.

Table 3.2 outlines some of the specifications and capabilities for the MATMOS imager.

Figure 3.3 shows the relative sizes of the FTS field of view and solar image on a section of the detector.

Fourier Transform Spectrometer

The Fourier Transform spectrometer (FTS) is the second key component to the MATMOS system. It will be capable of sweeping a large spectral range within the infrared to detect the presence of a suite of gases. Specifically, the FTS will have 0.02 cm^{-1} sensitivity and a field of view allowing about 3 km vertical resolution at the limb [4].

Table 3.3 displays some of the attributes and specifications for the FTS.

3.5 ExoMars Configuration Changes

Disappointingly, the funding complications discussed in section 3.2 mean that the majority of the instruments originally set out for the TGO will not proceed; however,

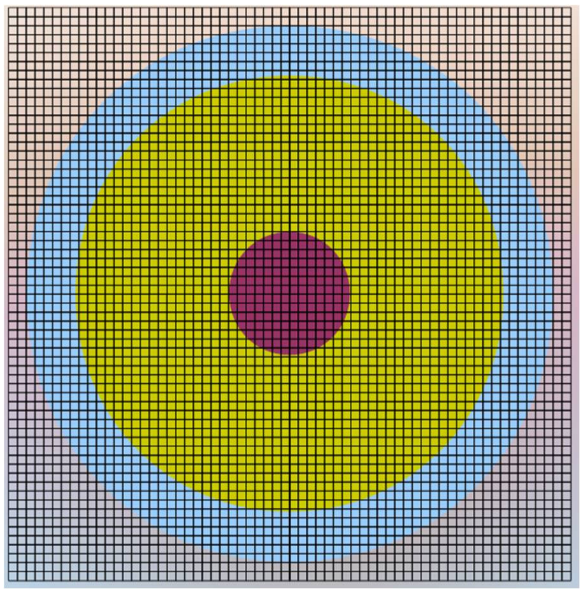


Figure 3.3: An image comparing the relative sizes of the imager field of view and solar disk [7]. The red circle is the size of the FTS field of view. The yellow and blue circles are the view of the solar disk at aphelion and perihelion respectively. The total size of the figure is a 64x64 pixel area.

Table 3.3: Specifications for the MATMOS Fourier Transform spectrometer [4].

Attribute	**Specification**
Spectral Range	850–4300 cm^{-1} 2.3–11.8 μm
Resolution	0.02 cm^{-1}
Field of View	1.56 mrad
Detectors	HgCdTe and InSb PV
Signal-to-Noise Ratio	>250:1
Sampling	2-6 sec/spectrum, 24 bits @ 215–645 kHz

recent news is that the NOMAD instrument (see below) will continue as part of the resigned payload. A new solar occultation instrument is expected to be part of the payload but with a significant reconfiguration to the FTS from the specifications set out for MATMOS; however, the imager component of MATMOS will also be maintained. Nonetheless, much of the work detailed in chapters 4 and 5 remains entirely applicable. As such, we retain MATMOS as the concept for a solar occultation spectrometer in this thesis.

3.6 The Nadir and Occultation for MArs Discovery (NOMAD)

NOMAD is a European instrument for ExoMars. The instrument PI is Ann C. Vandeale of the Belgium Institute for Space Aeronomy working in collaboration with scientists in Spain, Italy, the United Kingdom, Canada and the United States. NOMAD has a combination of capabilities in the infrared, visible and ultraviolet regions of the spectrum [47] [4].

NOMAD has three operational modes: Solar Occultation mode; Limb, Nadir and Occultation mode; and Ultraviolet and Visible mode.

The Solar Occultation (SO) mode operates by observing up to six small slices of the full spectral range each second. This allows observing several different target molecules that absorb at different wavelengths, whilst maximising the signal-to-noise ratio for each. During a solar occultation 300 spectra at each wavelength can be taken providing a profile of the atmospheric composition from the top of the atmosphere down to almost the surface, depending on dust levels.

The Limb, Nadir and Occultation (LNO) mode is sensitive to the lower light levels during nadir observations on Mars. The nadir coverage will facilitate the study of the atmospheric composition in addition to examining Martian surface features, such as ice and frost. This measurement will be carried out on average every 3 to 4 sols (a solar day on Mars, or sol, is 24 hours and 39 minutes) with varying local times across the planet.

The Ultraviolet and Visible (UVIS) mode will image the wavelength domain between 200 and 650 nm, every second, covering and providing more information about several interesting molecules, such as ozone, sulphuric acid and aerosols in the atmosphere.

NOMAD covers a visible wavelength range that MATMOS does not but has a lower spectral resolution. In the SO and LNO modes NOMAD spans 2.2-4.3 μm range and upwards of 0.15 cm$^{0.15}$ resolution, as compared to the 0.02 cm^{-1} MATMOS resolution. The UVIS mode ranged from 200–650 nm and a 1–2 nm resolution [4].

3.7 Russian Instrumentation and Continued Canadian Involvement

Following the reconfiguration of the spacecraft and withdrawal of the United States from the mission, a Russian instrument is being defined as a substitute for MATMOS. This instrument will also be a Fourier Transform Spectrometer and will cover a similar wavelength region in solar occultation, but at reduced spectral resolution. There is a strong similarity between the science objective of this (yet un-named) instrument and MATMOS and there are ongoing discussions between Canadian and Russian scientists on the inclusion of the Canadian imager experiment which is the subject of this thesis and other Canadian technology to the FTS section of the instrument.

Chapter 4

Occultation Instrument Imager and Pointing

4.1 Motivation

Occultation viewing is a powerful technique for deducing a vertical profile of gas or aerosol distributions but only in so far as we can determine exactly where we are looking. A large challenge is how precisely it is possible to determine the ray-path travelled by the light entering the occultation spectrometer.

Classically, the ray-path for an occultation spectrometer may be deduced in post-processing from the data itself. By focusing on a specific well-known absorption feature within the range of interest, say carbon dioxide for Earth, and by knowing how the concentrations of the absorber scale with height, it is possible to deduce the ray-path and thus the tangent height in the atmosphere. Given that the Martian atmosphere is primarily composed of carbon dioxide, CO_2 concentrations, in combination with thermal infrared measurements to deduce temperature, are also the commonly used proxy for atmospheric pressure. Specifically, looking at the peaks located at 4.3 and 15 μm.

There are drawbacks to this method. The necessity of determining the ray-path from the data, as opposed to determining it independently, diminishes the number of degrees of freedom and thus the total information we are able to retrieve. Thus, it is desirable to develop a technique that can simultaneously and autonomously determine the precise location of the field of view. For a solar occultation, this is often considered in terms of the "tangent height", the point of closest approach of the ray path to the planet.

The spacecraft itself can often provide some information about the direction in which it is facing, however, the degree of precision may be of concern. Recently, for the planned ExoMars Trace Gas Orbiter, Mark Allen of the Jet Propulsion Lab reported that the absolute pointing of the instruments could be guaranteed to a certainty within 1 mrad [48]. There is the possibility to do better with the imager.

We utilize the equation for angular diameter for spherical objects:

$$\delta = 2 \arcsin \left(\frac{1}{2} \frac{d}{R} \right)$$

where d represents the object diameter and R is the distance at which the object is being viewed. For an occultation situation, this can be simplified by using small angle approximations.

The sun appears from Mars with a angular diameter in the range 5.56–6.79 mrad due to the high eccentricity of the orbit. Given the imager IFOV of 0.115 mrad, this sets the solar diameter at pixels 48–60 pixels on the imager detector.

TGO orbits at about 400 km above the planet. At a tangent height of 20 km the tangent point is approximately 1650 km away from the spacecraft and each pixel on the imager translates to 190 m vertical extent at the tangent point and the absolute pointing error (APE) of 1 mrad quoted by JPL would guarantee pointing to an accuracy of 1.65 km on the limb.

Given the hydrostatic equation for the atmosphere and a scale height of 11 km the 1 mrad error gives a potential pressure error or error in mass in the path of about 15%. This would significantly degrade the accuracy of the measurements of gases and so at a minimum the aforementioned retrieval techniques using the FTS data must be used to improve the pointing knowledge. However, the imager provides a potential additional source of information to increase our knowledge of the pointing.

Again assuming that the atmosphere is hydrostatic, a 1% pressure error is equivalent to 110m in altitude or 66 μrad or 0.5 pixels on the imager detector. Thus if it is possible to determine the location of the centre of the sun in the imager to an accuracy of <0.5 pixels on the detector, then with a knowledge of the Mars location, Sun location, and Spacecraft position to similar accuracy it is possible to determine the path through the atmosphere without using the FTS data to 1% accuracy. The first two items are a matter of knowledge of the solar system, the spacecraft position is known from navigational information, and the relative locations of the FOVs can be found by pre- and post-launch calibration. The remaining issue is whether the imager data can be processed to locate the solar centroid to the required accuracy. Later in this chapter we will consider whether this is feasible.

Although the imager will not be used to help steer the spacecraft, the data and images collected will provide information during the post-processing phase about where the TGO was pointing. Adapting active tracking techniques for centre-seeking and identification would be a good solution; however, there are many factors associated with limb viewing through the Martian atmosphere that must be examined and tested to ensure the robustness of the algorithm.

Dust presents a great challenge in that its extinction profile, especially at lower attitudes, will challenge the method. With only the information from a portion of the solar disk, the technique must continue to reliably find the centre of the image. Also, dust and aerosol layers that are inhomogeneous across the field of view of the imager will may affect the algorithm in different ways. For instance, a thick layer across the bottom portion of the image with an entirely transparent top layer could confuse the algorithm into believing there is a flattened image. Common to the Earth's atmosphere is a flattening of the solar disk near the horizon due to refraction; however, this is not a concern in the thin Martian atmosphere.

An additional factor to consider in implementing such a technique on a Martian-based mission is the reality that once the instrument is launched there will never be another opportunity to adjust, replace, or calibrate hardware. As such, our technique should be robust enough to identify and compensate for issues that are common or likely in such systems. Further, it is desirable to know the extent to which these malfunctions affect our ability to know the pointing and how much error they may contribute to the final limb-viewing uncertainty.

This chapter will examine the development and analysis of an adapted technique for pointing determination. The method will be summarized and results will be presented as to the health of the technique when pushed beyond the limits of the likely.

4.2 Method

Detector

The technique requires a few steps before we can begin the actual centre-seeking portion. First, we must develop a grid onto which the solar image will be projected.

This "hardware" portion of the image will allow us to examine how separate images of the sun will appear on similar layouts. Further, it will allow us to study hardware defects that could reasonably take place.

To do this we create a similar model template of a the detector being proposed for the MATMOS instrument, the STAR1000 CMOS detector shown in figure 4.1. Such a detector is a piece of hardware used for digital imaging whereby photons are incident on the surface during an exposure time and subsequently converted to an electrical charge in spatially separate bins, pixels, or "wells". Following this, the wells are then discharged into a converter that measured the charge and infers a value corresponding to the number of incident photons on a particular pixel.

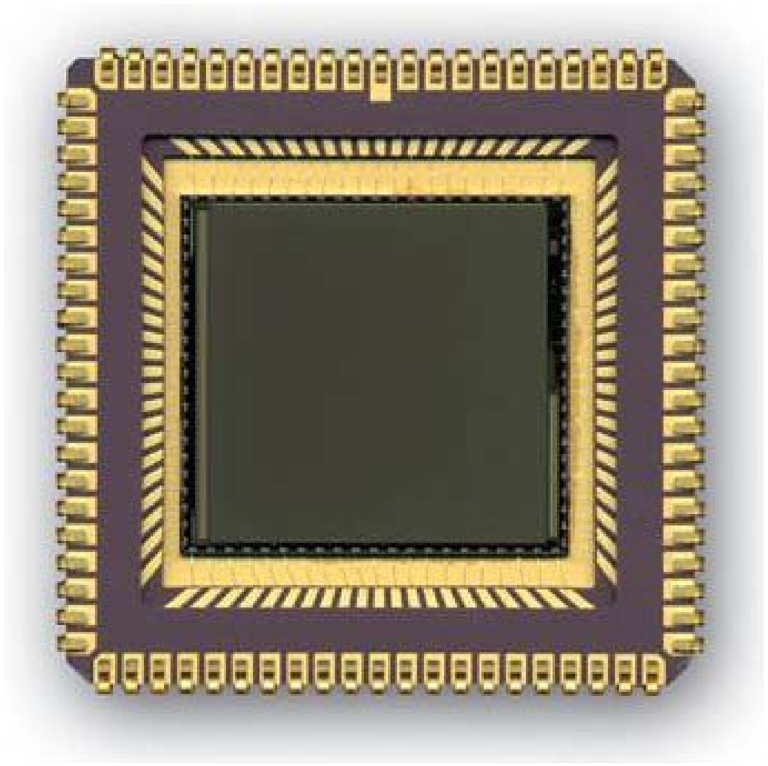


Figure 4.1: A STAR1000 CMOS image sensor [8].

There are several technical aspects of a detector that are important to be cognisant of. Firstly, the design of the detector must be considered. Ensuring that all of the bins are uniformly created and responsive is important. Should one bin have a unequal surface area over which incident photons will generate a signal, this will create artificially large signals over what might be thought to be a smaller area. We are assured that through the design process and rigorous testing of the detector prior to installation that this will not be a concern. As for responsiveness, there can be issues of how the photons trigger the electronic response. When bins are receiving particularly high signals, to the point of or near saturation, there can be "spillage" to adjacent bins. Further, malfunctioning bins can artificially create saturated or non-responsive pixels in the image.

The translation of measured current from the number of photons incident on a particular pixel is not entirely straight-forward. There are several factors that affect this conversion. Our group is working to study these factors with experimental tests on equipment similar to what is proposed for the imager; however, at this time these factors will not interplay with our study of the centre-seeking technique. More discussion of such factors will appear in chapter 5.

For the simulation of the detector we employ a computer program written in Python to create a two dimensional array of lists. The two dimensions represent the physical dimensions of the square detector. For each element of the array, which represents an individual pixel, there is a list of terms used to describe the attributes of the given pixel.

Defects are introduced when a set of detectors are created. Four main detector defects were simulated:

1. Random dead pixels (always dark, no signal)

2. Random bright pixels (always saturated signal)

3. Random dead columns (always dark, no signal for the entire column)

4. Brightness gradient (fading of image)

Dead and bright pixels are selected to simulate malfunctioning pixels on the CMOS detector. An example of a dark failure is a simple pixel failure in which charge is not accumulated for that pixel, an example of bright failure is a case where a cosmic ray is incident on the detector surface. In this latter case, it is possible that the high energy transfer will produce an overwhelming signal that may saturate the receiving pixel and likely others in its vicinity. This over-saturation may lead to temporary or permanent depending upon the exact location and energy of the cosmic ray incursion.

In either case a study of dark and bright failures would help us to determine their impact on our technique and also allow us to develop strategies to minimise their impact on the higher level data products from the instrument.

For the dead or bright pixels a percentage of pixels was selected to have malfunctioned. Then, using the Python “random” function, individual pixels across the face of the detector were chosen at random to be turned dead or bright. Should a pixel be selected that happened to already have been adjusted, the algorithm jumps to a new space so that there are always the same number of malfunctioning pixels in a set of detectors. This process is carried out for a large number of detectors to ensure a truly random sample set. For our purposes we chose to select the set size to be 1000 detectors.

For detectors with dead columns, the number of dead columns is selected for a given set. The selected number of columns are chosen at random and turned off entirely. Similar to with selecting the dead pixels, the columns are chosen at random and not duplicated.

Column failure is a real possibility arising from failure in the electronics. Since the signal is discharged over a column, any failure in its ability to properly measure the signal will result in an unreliable or non-responsive column. Further, in the case where a row may experience the same phenomenon, the column case may be used analogously.

The next question examined was how would the algorithm hold up when a darkness gradient was applied over the image. This is a significant problem for Mars as transparency may vary as the occultation descends through the atmosphere. Extinction due to dust or dust layers, as well as other aerosol clouds or layers, could very well create this realistic scenario.

For the vertical brightness gradient we apply a gradual darkening across the face of the sun. The gradient is characterized by the ratio of the brightness at the top of the solar disk to that at the bottom.

Once a set of detectors is made they are numbers, stored, and recorded in an inventory of all previous sets of detectors. Table 4.1 lists the sets of detectors created and used for our purposes.

Table 4.1: A list of detector sets simulated and their defect features..

Set Name	**Defect Type**	**Description**
Blank	None	A single detector without defects for comparison.
Dead1pc1k	Dead Pixels	detectors with 1% random dead pixels.
Dead2pc1k	Dead Pixels	detectors with 2% random dead pixels.
Dead4pc1k	Dead Pixels	detectors with 4% random dead pixels.
Dead8pc1k	Dead Pixels	detectors with 8% random dead pixels.
Dead16pc1k	Dead Pixels	detectors with 16% random dead pixels.
Dead32pc1k	Dead Pixels	detectors with 32% random dead pixels.
Dead40pc1k	Dead Pixels	detectors with 40% random dead pixels.
Dead42pc1k	Dead Pixels	detectors with 42% random dead pixels.
Dead45pc1k	Dead Pixels	detectors with 45% random dead pixels.
Dead50pc1k	Dead Pixels	detectors with 50% random dead pixels.
Bright1pc1k	Bright Pixels	detectors with 1% random bright pixels.
Bright50pc1k	Bright Pixels	detectors with 50% random bright pixels.
D1col	Dead Columns	detectors with 1 dead column.
D2col	Dead Columns	detectors with 2 dead column.
D3col	Dead Columns	detectors with 3 dead column.
D10col	Dead Columns	detectors with 10 dead column.
gaingrad2xface	Vertical Gradient	detectors with vertical fading of a factor of 2.
gaingrad3xface	Vertical Gradient	detectors with vertical fading of a factor of 3.
gaingrad10xface	Vertical Gradient	detectors with vertical fading of a factor of 10.

Solar Images

Our program then has solar images loaded into a directory. For our purposes we took an image from a webcam at Dalhousie University. The image was then shrunk to a diameter of 70 pixels on a square 256 pixel image to fit atop the simulated detectors. Figure 4.2 shows the original and altered images. Although the webcam employs a charge-coupled device-type detector and the MATMOS imager is proposed to use a CMOS detector, for our purposes at this stage the webcam image is a sufficient substitution.

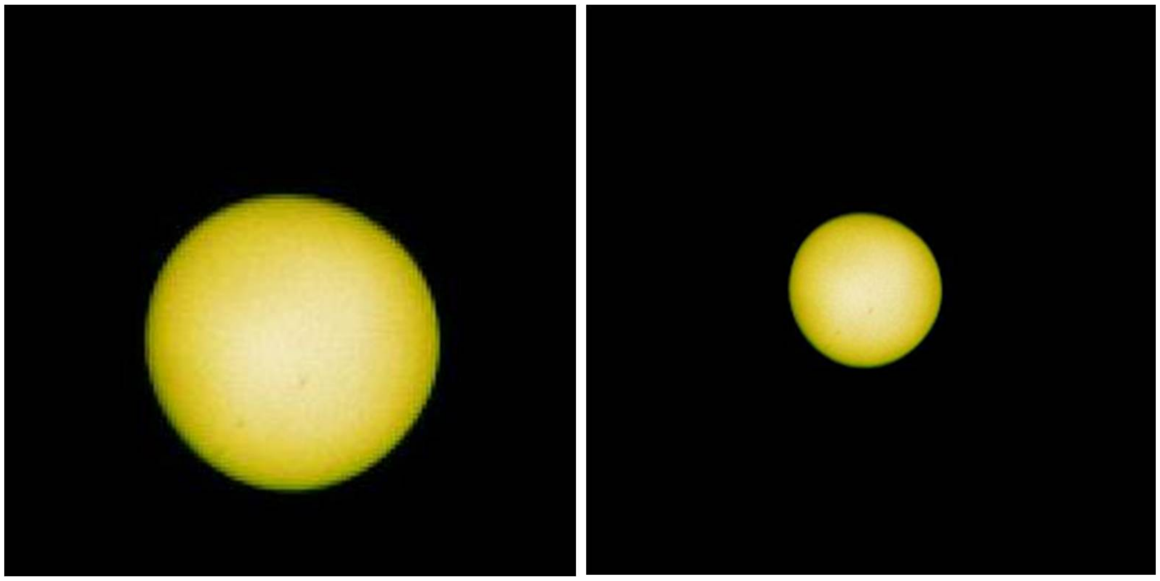


Figure 4.2: **Left**: An image taken from Earth (Provided by J. Franklin). The solar diameter is about 135 pixels. **Right**: An adapted and centred version of the image on the left. The solar diameter is calculated to be about 70 pixels.

Defect Corrections

It is desirable to compensate for known defects before the images on the detectors enter the centre-seeking analysis.

Without knowing the dimensions of the sun *a priori*, it is difficult to know exactly which pixels with zero signal are defective and which are simply outside of the solar disk. Similarly, pixels that are artificially saturated cannot easily be distinguished from potentially saturated pixels from the solar signal.

For detectors with dead columns, should a column appear across the face of the sun it would be entirely visible. Further, defective columns are stationary and may have a fix permanently applied. As a fix we apply a simple interpolation. Once the column is identified its values are replaced by averaging the values on either side.

Centre Seeking Analysis

Currently an active sun tracker is employed by our research group to perform ground-based FTS measurements above Dalhousie University in Halifax, Nova Scotia. This high-precision solar tracker allows for the FTS to follow the apparent motion of the sun through the sky throughout the day. Figure 4.3 shows a typical result from a day of sun tracking above Dalhousie. The success and precision of the tracker is, at least in part, reliant on the visibility of the solar disk. An adaptation of this active correction technology is the basis for our post-processing code and the determination of the instrument pointing.

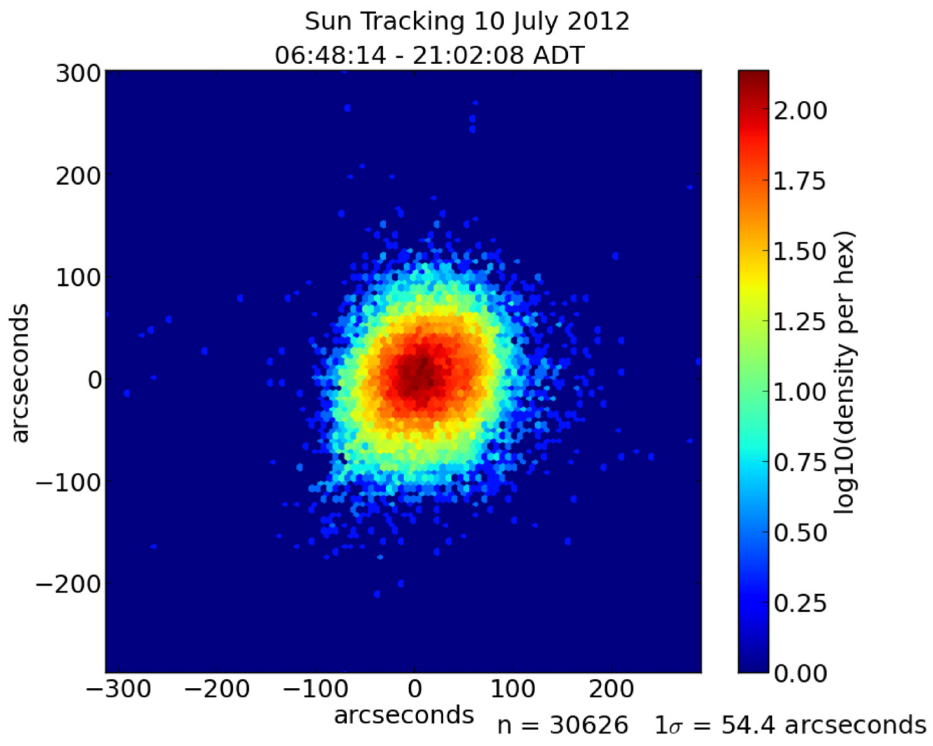


Figure 4.3: Deviation of the detected solar disk from the optimal position of the solar beam for the Dalhousie FTS ground station using the sun-tracking algorithm. (Figure supplied by J. Franklin.)

The technique employs a python module called OpenCV [49]. OpenCV is a "computer vision" package that was originally designed by Intel with functions to enable programmers an increased level of functionality using real images. Such facets include real-time image processing and shape identification.

The centre-seeking algorithm works through a series of steps to then combine each detector within a particular set with the image being process onto the detectors. I will list the steps and their function below. For a more full understanding, please see Appendix A for the full code.

Once the image is chosen, a threshold may be selected. The threshold scans the image and zeros all pixels below that value. This allows the image to be cleaner and lets the algorithm focus on the areas of strong signal instead of weaker background noise. Presently, the threshold may be determined numerically or visually. The advantage to numerical selection is consistency across many detectors for a similar image. The option to visually select the threshold is advantageous when working with a new image for the first time to roughly determine an appropriate threshold value.

The image is then passed into the main function called "process_image". The function then applies the threshold to a new copy of the chosen image. Then the OpenCV function "FindContours" identifies all contours of the image. Then, a specially designed function called "pickcontour" selects the correct contour for the edge of the solar disk by identifying that which has the greatest number of points. From there, the contour is plotted and an ellipse is fit to it. The properties of the ellipse are extracted and dumped to a series of files. From these files the results may be examined.

It is important to note that all results show the deviation between the algorithm finding the centre in the perfect case and the finding the centre of the defective image. Thus, the results are displaying only the deviation and loss of accuracy occurring as a result of the particular defects and/or obstructions.

4.3 Results

Dark & Bright Pixels

First we will examine how increasing the number of defective pixels affects the ability for the algorithm to detect the true centre of the solar disk. Figures 4.4 and 4.5 show the ranges of defects from 1% defective pixels to 50% defective pixels.

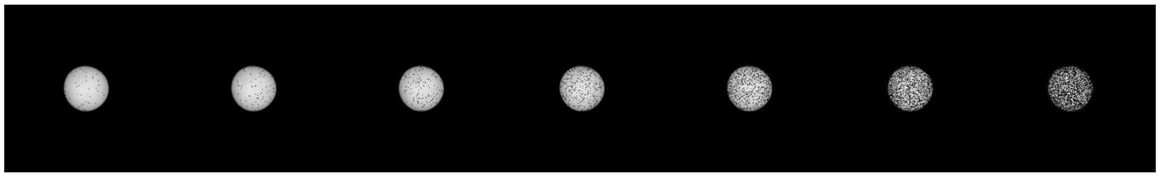


Figure 4.4: **Left to Right**: Examples of 1%, 2%, 4%, 8%, 16%, 32%, 50% dead pixels on a solar image.

Each set of 1000 detectors with randomly distributed dead and bright pixels was analysed. First we examine the detectors with dead pixels. Two sample scatter plots

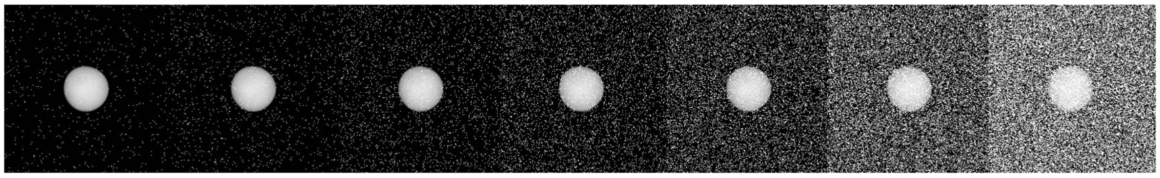


Figure 4.5: **Left to Right**: Examples of 1%, 2%, 4%, 8%, 16%, 32%, 50% bright pixels on a solar image.

of the deviation from centre are shown in figures 4.6 and 4.7.

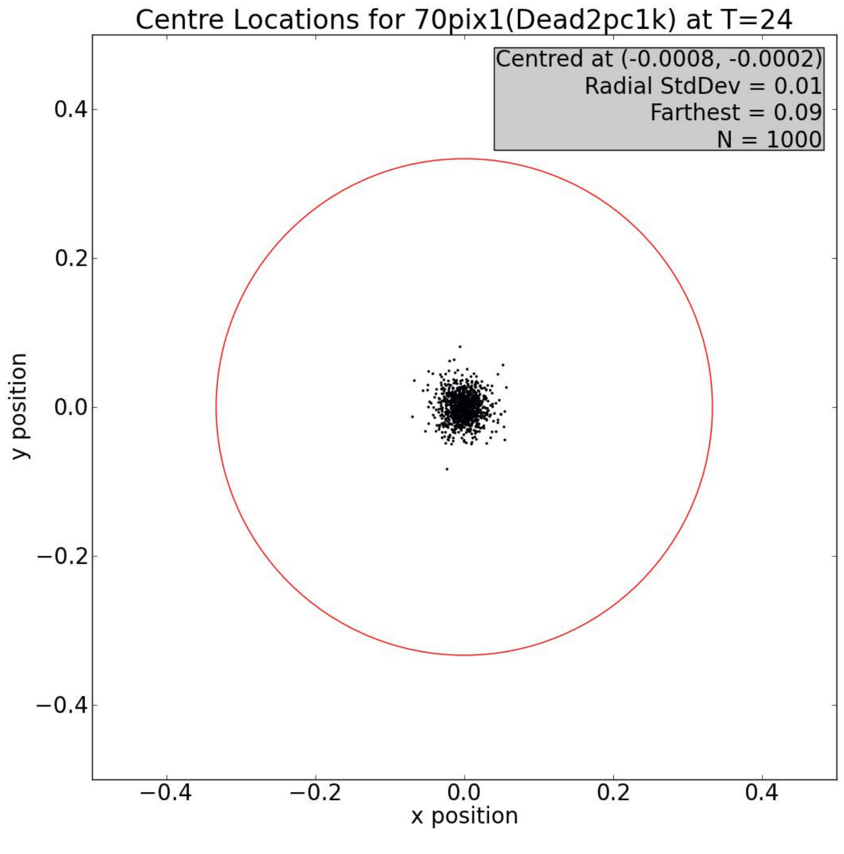


Figure 4.6: A 2D scatter plot of the deviation in the found centroids for a solar disk applied to a CMOS detector with 2% dead pixels. This was conducted for 1000 detectors with random dead pixel placement. The circle is set at a radius of 1/3 of a pixel which is representative of the accuracy in pointing that is required (we use 1/3 rather than 1/2 to provide a tighter criterion for success). This same circle is also shown in following figures.

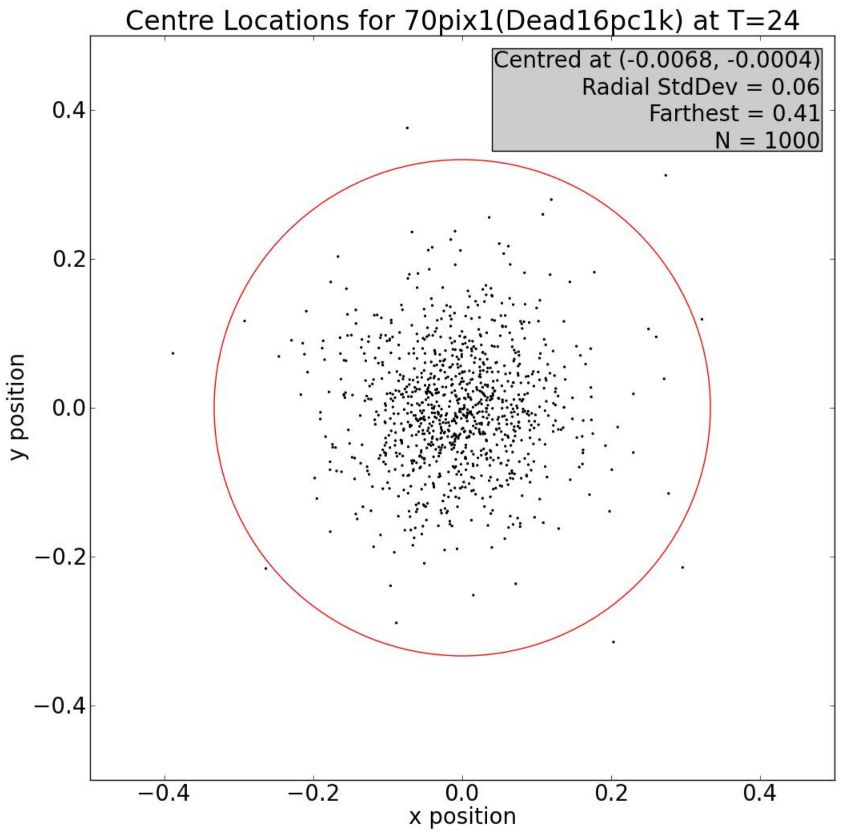


Figure 4.7: A 2D scatter plot of the deviation in the found centroids for a solar disk applied to a CMOS detector with 16% dead pixels. This was conducted for 1000 detectors with random dead pixel placement.

One important observation is how symmetrical the distribution is, with each scatter plot centred with little deviation from (0,0). Figure 4.8 shows how the standard deviation increases with more pixel defects.

Figures 4.9 and 4.10 show the deviation from centre for detectors with bright pixel defects. Figure 4.11 shows the standard deviation of the bright pixel defective detectors.

Dead & Interpolated Columns

The centre-seeking analysis is run on detectors with dead, dark columns. Figure 4.12 shows an example of 1 and 2 dead columns across the solar face. An example of a

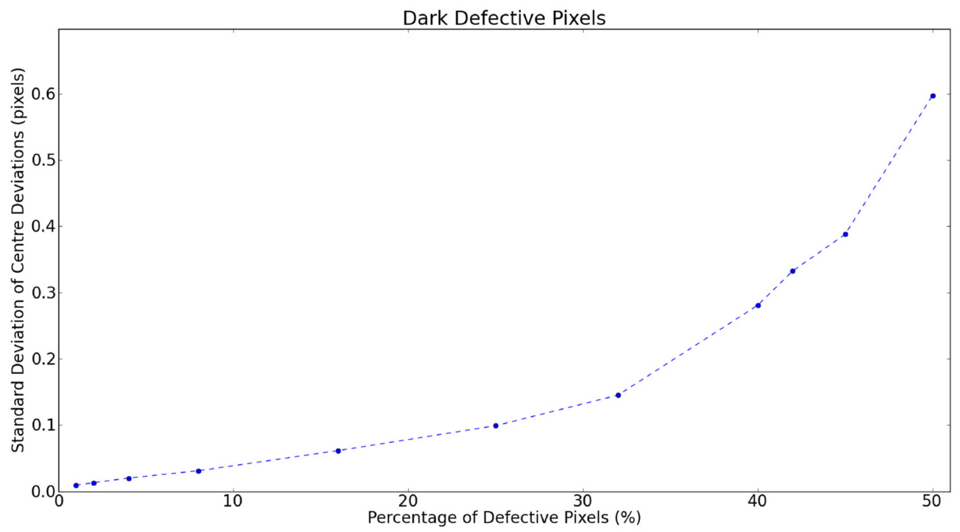


Figure 4.8: A plot of the standard deviation of the deviation in the found centroids for a solar image against the percentage of dead pixels on the CMOS detectors to which the image was applied.

scatter plot of centres is shown in figure 4.13.

One initial observation is the impact that the placement of the dead column has on the tracking. If the dead column falls outside of the detector face, it is nearly undetectable and has no impact on the method. Alternately, a dead column across the solar disk without any repair algorithm has a significant impact on the ability of the algorithm to find the correct centre. Also, the location of the defective column has a strong impact on where the false centre is found.

The probability that for a single randomly placed dead column to appear over the solar face is roughly the solar diameter (70 pixels) over the width of the detector (256 pixels), about 27%. As the number of randomly placed dead columns increases, the probability of any column appearing over the solar disk.

Figure 4.13 shows a large range of possible values but a high concentration of points are located at (0,0) as they were unaffected by a defective column outside of the solar disk. Given this, we do not see a normal distribution and thus the standard deviation would be an inappropriate way to characterize the data. Instead we will use the range, specifically the farthest distance the centre has been found from the true centre. Figure 4.14 shows a plot of how the range changes with increasing column malfunctions, from 1 to 5 dead columns.

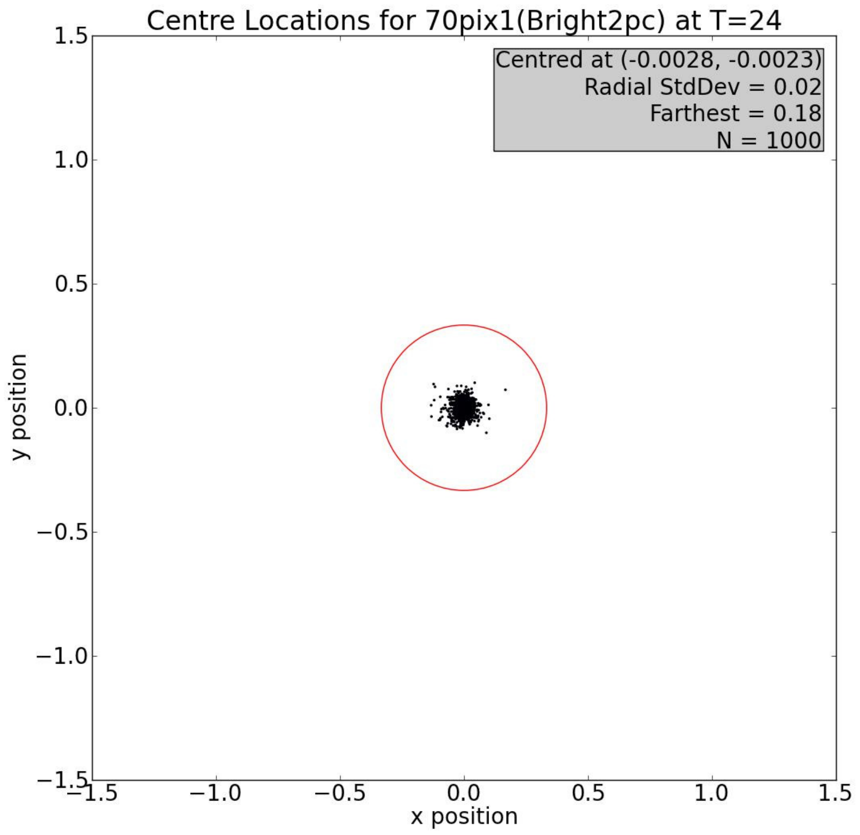


Figure 4.9: A 2D scatter plot of the deviation in the found centroids for a solar disk applied to a CMOS detector with 2% saturated pixels. This was conducted for 1000 detectors with random saturated pixel placement.

We now look at the results for the same sets of detectors with the simple interpolation applied. Figure 4.15 shows the scatter plot for one interpolated column, in contrast to figure 4.13. A comparison of the standard deviation for 1-5 fixed and defective columns is shown in figure 4.16.

Vertical Brightness Gradients

The vertical brightness gradients across the face of the solar disk are next analysed. Figure 4.17 shows examples of the gradual effect of the gradient. Figure 4.18 show the results for brightness gradients ranging from 2x - 10x.

As the fading becomes stronger the method seems to break down. Specifically,

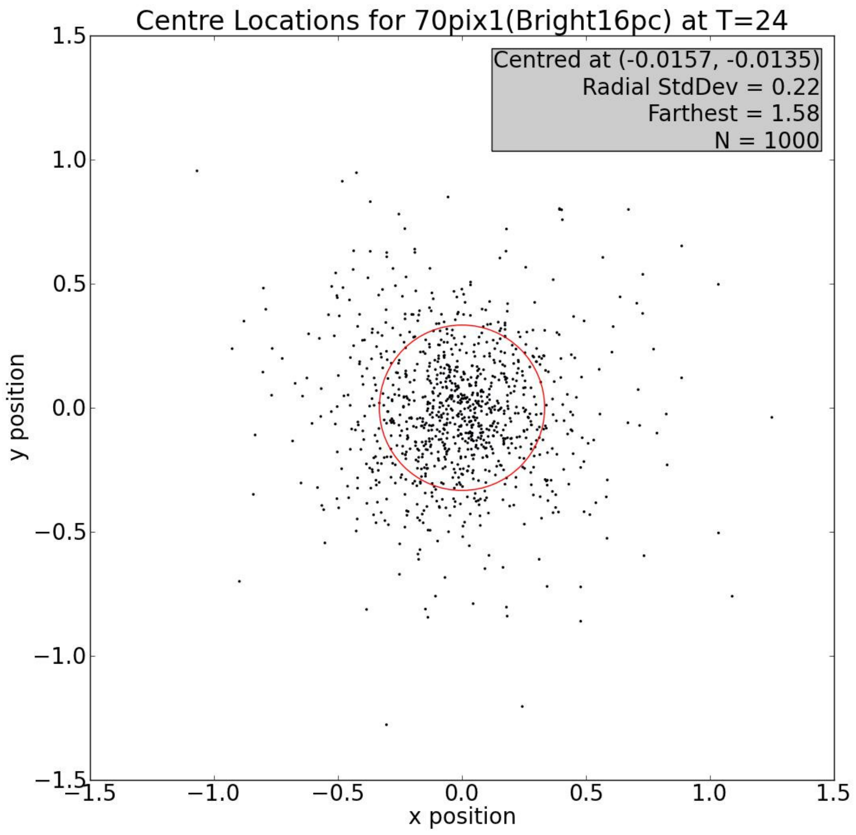


Figure 4.10: A 2D scatter plot of the deviation in the found centroids for a solar disk applied to a CMOS detector with 16% saturated pixels. This was conducted for 1000 detectors with random saturated pixel placement.

with less information at the bottom of the solar disk the algorithm cannot properly detect the edge and thus shifts the centre upward. Figure 4.19 shows where the disk is detected in 5x gradient image.

4.4 Conclusions

From figure 4.8 we can see the trend of how the standard deviation changes with increasing defective dark pixels. At a case with 1% defective pixels the algorithm is reliable, only deviating less than 0.01 pixels. The trend grows very slowly at first keeping under 0.1 pixels for cases with up to 16% defective pixels. Even with 50% of pixels malfunctioning the algorithm can still find the correct centre of the solar disk

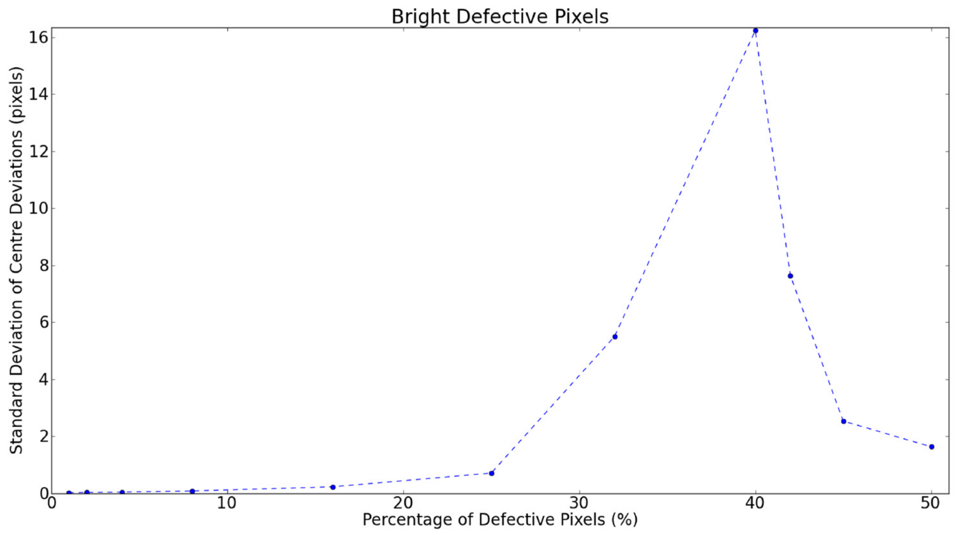


Figure 4.11: A plot of the standard deviation of the deviation in the found centroids for a solar image against the percentage of saturated pixels on the CMOS detectors to which the image was applied.

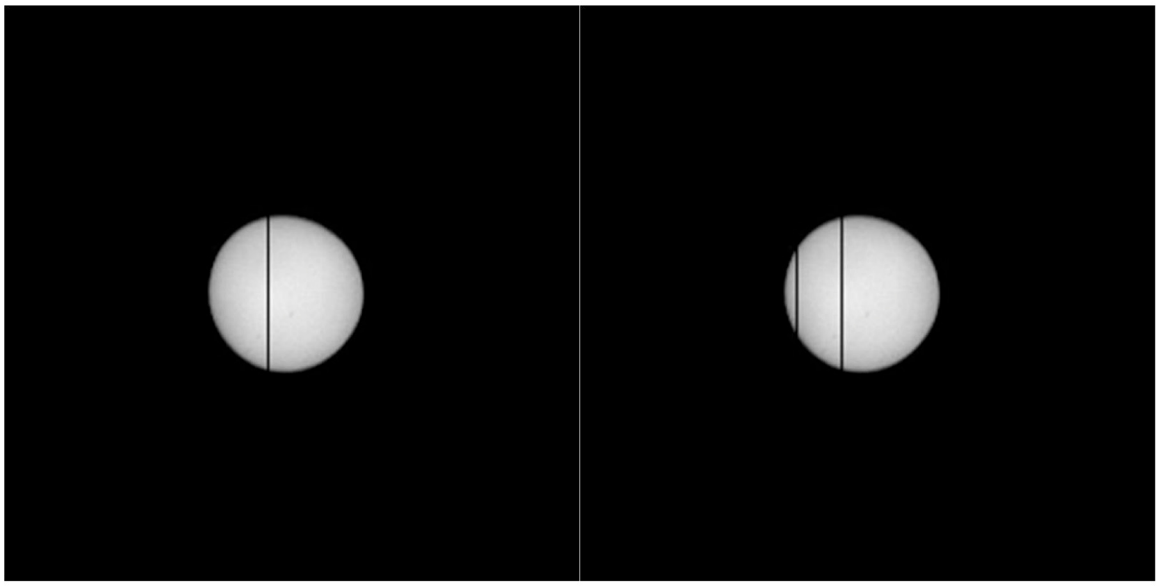


Figure 4.12: Two examples of detectors with dead columns appearing across the solar disk.

to within 0.6 pixels, an uncertainty of about 114 meters on the limb.

The algorithm seems extremely robust in the case of missing or malfunctioning pixels. Although it is extremely unlikely that such a large portion of the detector would die, this case study shows that the information to precisely determine the centre of the solar disk is contained within much fewer pixels than we may have

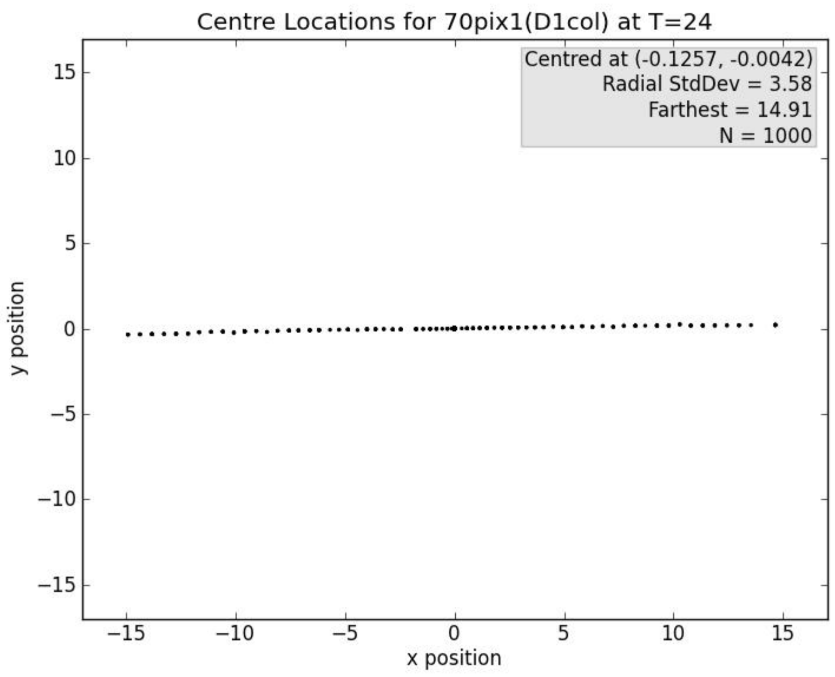


Figure 4.13: A 2D scatter plot of the deviation in the found centroids for a solar disk applied to a CMOS detector with one dead column. This was conducted for 1000 detectors with random dead column placement. **Note**: Many points are overlapping at the (0,0) position

available. This allows for a huge amount of flexibility. For example, should questions arise as to the integrity or proper function of a number of pixels, even across the face of the sun, it appears an appropriate course of action could be to ignore the signal for these pixels entirely. Further, interpolation by using adjacent pixel information may provide even stronger results.

The bright pixels show a slightly different story. The standard deviation grows quicker than was the case for the dark pixels. Also, there is a significant deviation from the trend seen at 32% bright pixels whereby the standard deviation of the distribution jumps above 5 pixels. This does not continue as 50% bright pixels settles back to below a 2 pixel standard deviation. This about-face the deviation takes in the bright pixels case as we extend towards very large percentages of defective pixels. Although this may seem counter-intuitive, the answer is an artifact of the centre-finding routine.

As the entire image tends towards white, the technique may detect many contours with the wide distribution of adjacent dark and bright pixels. The selection criteria

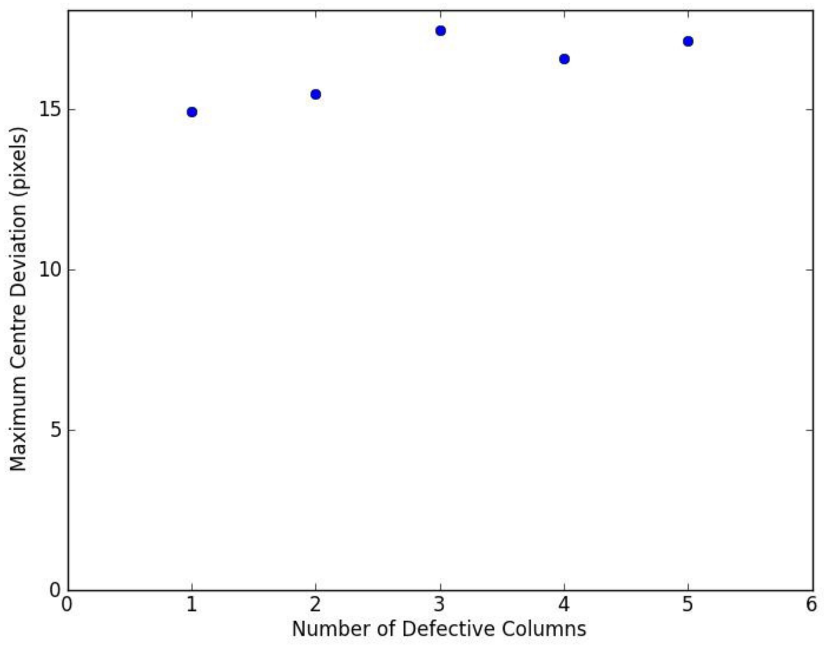


Figure 4.14: A plot of the maximum deviation of the found centre on detectors with dead columns.

in the algorithm relies on picking the contour with the most points along the fit edge. Thus, as the image tends to white the algorithm would tend to pick the largest possible ellipse, which happens to be a circle inscribed by the entire detector array. Consequently, this would find the centre to be at the exact centre of the detector, where the unaltered image is centred.

Nevertheless, comparing the affect of the bright pixels to the dark pixels tells a very obvious tale - the dark pixels have a dramatically lower impact on the ability of the technique to locate the centre of the solar disk. This is likely because dark pixels off the solar disk are nearly untraceable among the background and do not affect the ability for the algorithm to find the edge of the sun. Moreover, dark pixels located within the solar disk place have no affect on the gradient edge of the sun and thus have no bearing on the method. Although little impact is seen by bright pixels being located within the solar disk, those near the edge and outside can influence the algorithm to chose false gradients as its guess for the solar disk based on the criteria used in the "pickcontours" portion of the algorithm.

One possible, although not necessarily ideal, way of dealing with bright pixels,

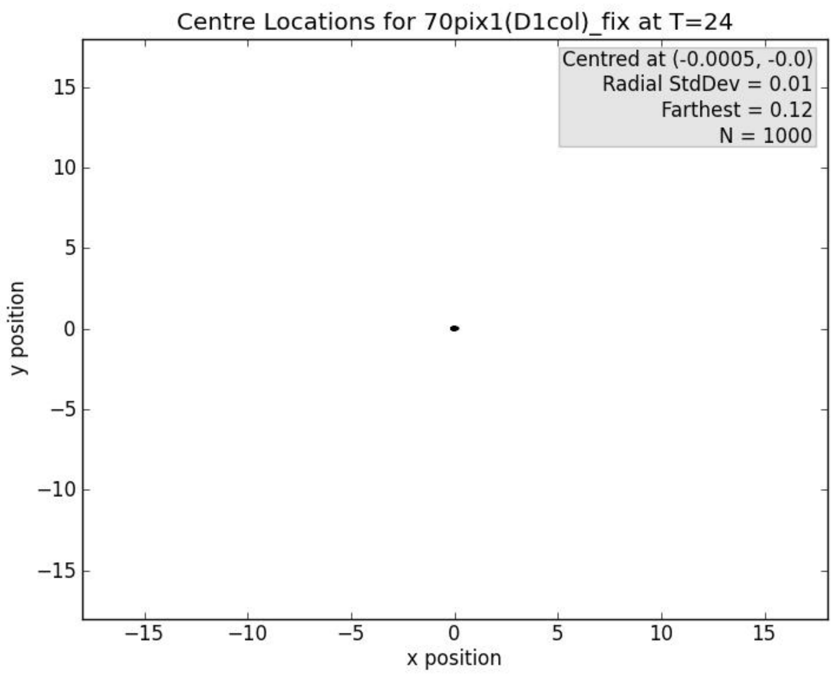


Figure 4.15: A 2D scatter plot of the deviation in the found centroids for a solar disk applied to a CMOS detector with one interpolated column. This was conducted for 1000 detectors with random interpolated column placement.

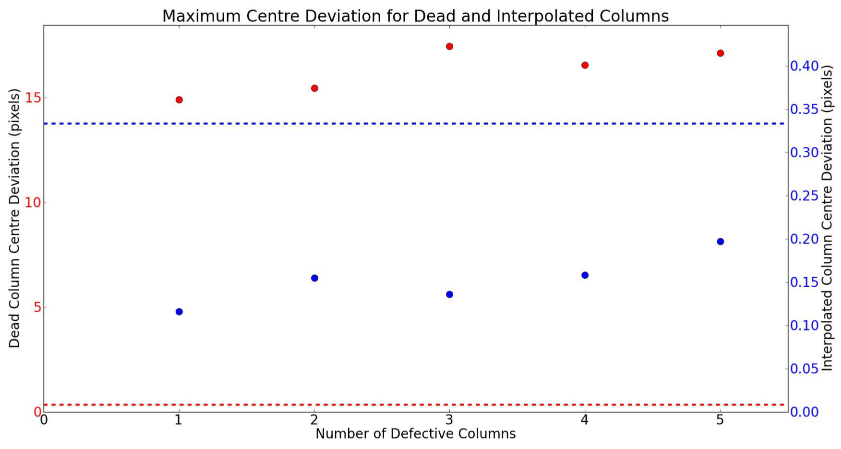


Figure 4.16: A plot of the maximum deviation of the found centre on detectors contrasting dead columns (top) and interpolated columns (bottom). The dashed lines show the 1/3 pixel target for reference in each scale.

whether permanent defects or transients, would be to automatically zero out the signal on any pixels with a saturated signal. This would allow us to reduce the

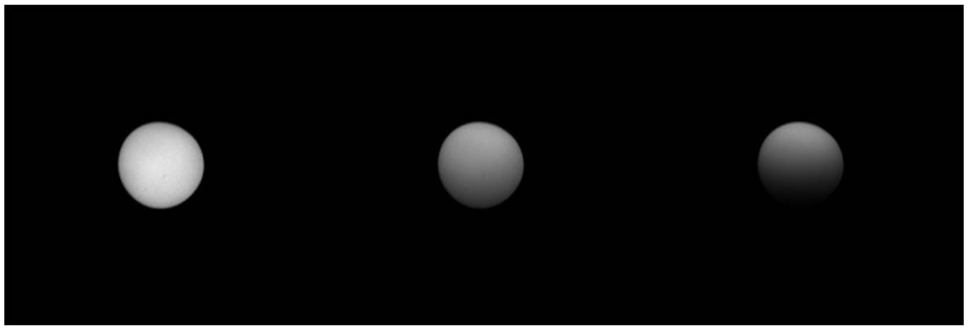


Figure 4.17: **Left**: Unaltered solar image. **Centre**: Solar image with a 2x gradient. **Right**: Solar image with a 10x gradient.

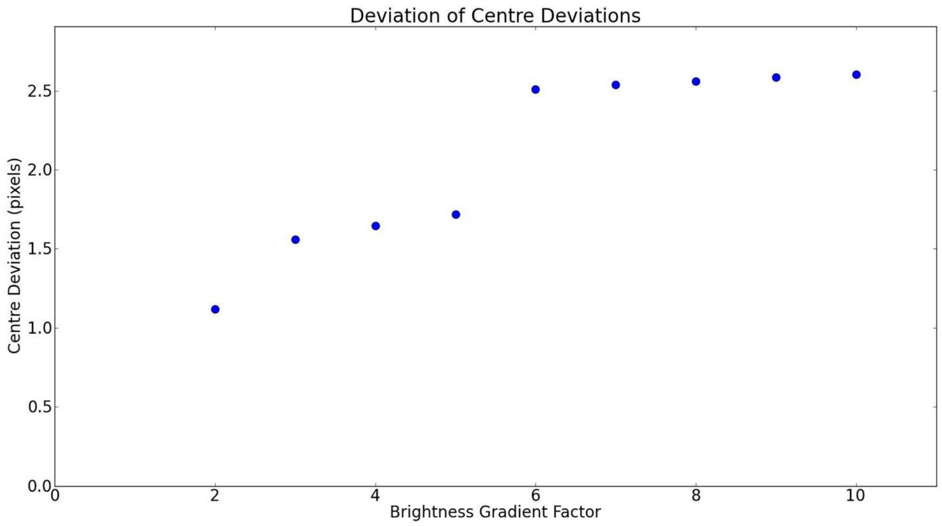


Figure 4.18: A plot of the centre deviation of the found centroids for the solar disk against the brightness gradient applied to the image on the CMOS detector.

deviation close to that seen in the dark pixel scenarios.

Looking at the results for the dead columns, we see a couple of starkly different scenarios playing out. First, given the strong likelihood that the dead column will fall outside of the solar disk, there is a very large concentration of "deviations" located at the (0,0) point in figure 4.13. For the cases where one or more columns do fall over the solar disk the ability to find the centre of the sun is severely compromised. Again, figure 4.13 shows a range in certain cases that extends as far as 14.9 pixels from the true centre. We compare the results from the dead columns to the same set once the interpolation is applied. As we can see from figure 4.16, the correction improves the accuracy of the method dramatically. For one dead column, once corrected the drops

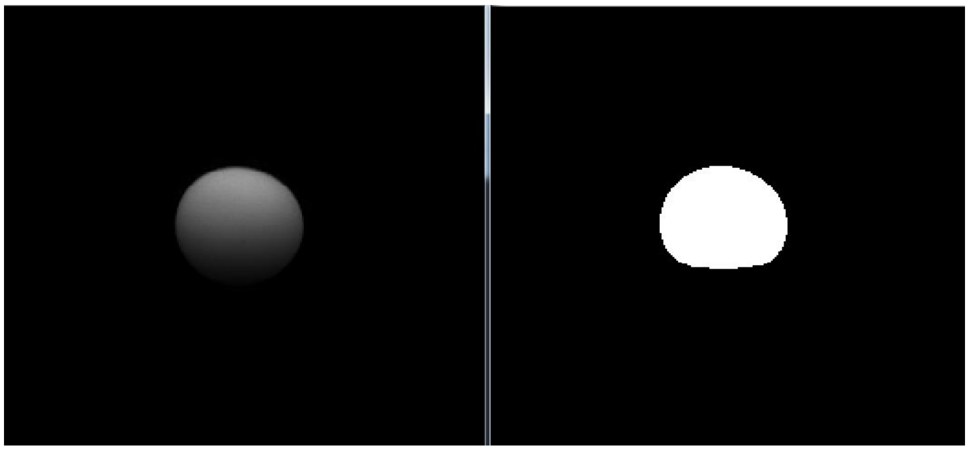


Figure 4.19: **Left**: A solar image with a 5x brightness gradient applied. **Right**: The solar disk still detected by the algorithm with the 5x brightness gradient image input.

from the aforementioned 14.9 pixel range to a 0.12 pixel maximum deviation. The result continues to work with great efficiency when interpolating for multiple dead columns across the face of the solar disk.

This result is encouraging. We clearly see that an interpolation technique for a permanent column defect will resolve any issue the centre-seeking algorithm may have in trying to locate the centroid. There is the possibility to further improve this result with a higher-order correction.

Figure 4.18 shows an unsurprising result. As the face of the solar disk is faded from the bottom upward, the centre is found increasingly higher on the detector. This extends from a deviation of 1.12 pixels at a 2x fade and up to 2.61 pixels at a 10x fade. Considering no additional constraints or adjustments are applied at this stage makes this result encouraging.

What is not taken into account here is the fact that the algorithm will not be dealing with individual, independent images but rather with a succession of images taken sequentially. This means that an temporary obstructions, such as a detached CO_2 crystal cloud, would occur before and/or after other unobstructed images and thus information from other views will help to overcome these gaps.

One leading possibility to help further constrain this technique would be to use an exo-atmospheric image of the sun to fit and determine the radius. This radius could then become a constraint for other fitting during the same occultation. This would discourage the algorithm from tending to flatten the fitted solar disk as to increase the number of intersected contour points, as is part of the selection criteria for the best contour. Some preliminary studies confirm this as an encouraging area for further studies.

Overall, we have studied the method and its robustness against defective pixels, dead columns, and dimming across the solar disk. The algorithm appears robust against defective pixels or small transient defects. Interpolation has been shown to be an effective method for dealing with detector columns. Lastly, significant fading across the solar disk will lead the algorithm away from the true centre; however, information from a sequence of images, as opposed to a single image, may allow us to properly constrain the technique to avoid this.

Chapter 5

Requirements of Studying Martian Aerosols

5.1 Motivation

The ExoMars announcement of the Trace Gas Orbiter called for instrumentation that could detect a broad suite of gases and characterize spacial and temporal evolution of said gases. Although the highest level objectives focus on the gas content of the atmosphere, we should also study the other principal component of the Martian atmosphere – the suspended dust aerosol. Given the aerosol's prevalence at low altitudes and its broad extinction features, it will have significant impacts on our ability to study any atmospheric feature and thus is of tremendous interest and importance.

Both dust and ice crystals are found suspended in the lower atmosphere. To determine the level of precision needed to study these aerosols we look to previous studies conducted by Montmessin et al. (2006). Montmessin's work used stellar occultation to observe detached CO_2 layers typically located around 100 km altitudes [21]. They report slant opacities between 0.01 and 1.00; however, this is being observed at 200 nm. Additionally, Chassefiere et al. (1992) performed solar occultations in the range of 280 nm to 3700 nm and observed opacities ranging from 0.1-0.3 [50].

We make some assumptions to approximate the required level of precision for an imager with similar specifications as the MATMOS model. First, assuming Rayleigh scattering, we look at a worst case scenario with an angstrom coefficient of 4, knowing the true angstrom coefficient should be lower than this. Knowing slant opacities around 0.1 are observed with a 200 nm channel and a MATMOS imager channels are up to an order of magnitude higher, around 1000 nm, we can use these proportions to deduce that MATMOS may see opacities as low as 0.05. In order to progress beyond this level of knowledge, we need to exceed this level of measurement and a target of 0.001 would enable the detection of clouds an order of magnitude thinner than the previous studies.

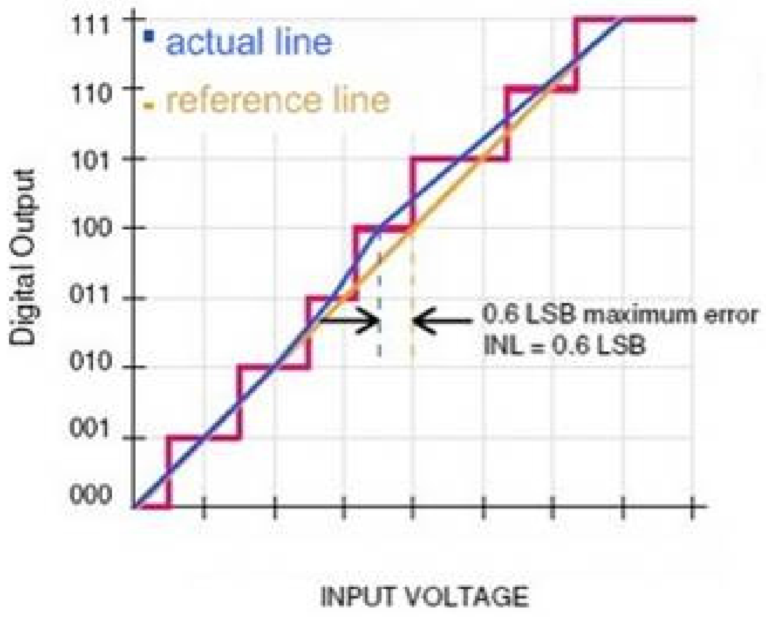
There are two issues that will be addressed over the course of this chapter. We will first look at the performance of the analog-to-digital converter (ADC) and its adherence to design specifications. We will then examine the spectral characteristics in the infrared region of the Martian atmosphere for regions of interest that would make possible study of the aerosol.

5.2 Signal & Hardware Analysis

We must minimize the impact the hardware will have on the data collected. This can be achieved in two ways: design the hardware to extremely high precision specifications or characterize and calibrate the known hardware deviations and correct for deficiencies in the data processing. Through this chapter we will, in part, examine an imager prototype similar to that proposed for the TGO mission in Chapter 3 and consider the possibility of characterizing the on-board hardware, namely the detector and the analog-to-digital converter.

First, we examine the non-linearity of the ADC. The ADC takes the signal received by the detector, in the form of a voltage, and converts it to a digital count. The output is categorized based on the bit-depth of the converter. For instance, a 10-bit converter will return values between 0 and 1023. In theory each of the bins spans an equal voltage range; however, even carefully designed ADCs have a degree of non-linearity to them. It is important to characterize this to understand the expected deviation and the cumulative affect it will have on the data. At each possible value a Differential Non-Linearity (DNL) is measured as the deviation from the ideal step function.

The overall characterization is often reported as the Integral Non-Linearity (INL) of the ADC, which is the maximum deviation from the reference line of the ideal step function, as a result of a cumulative DNL deviation [51]. Figure 5.1 shows an example of the INL analysis on a 3-bit converter. The INL is typically reported in units of the voltage range of the least significant bit (LSB). One feature worth noting is that, by design, the ADC is constrained at its limits, values of 0 and 1023. As a result, observing transmissions near 0 and near 1 should have minimal contributed error and a stronger likelihood of achieving the desired precision. Consequently, transmissions near the mid-range of the ADC domain are at risk of larger cumulative DNL, thus meaning degradation to possibly 0.2 or 0.3% error.



s

Figure 5.1: An example of a real 3-bit analog-to-digital converter. The actual line shows the deviation from an ideal converter, demonstrating how the INL is observed.

Due to the high volume of data being collected by all instruments aboard an orbiter, some of the data processing must be done on-board to appropriately ration the available bandwidth. Given this, the imager must rely on an on-board integration of consecutive data points to limit the data transmitted while simultaneously minimizing random error in the measurements. Integration times cannot be too long, however, as this can introduce deviations due to the evolving conditions of the environment or the detector itself. Specifically, the smear time (the time for the atmosphere to move 1 pixel relative to the spacecraft) of the imager is about 0.3 seconds, thus placing an upper limit on the integration time. Additionally, instabilities in the detector that emerge over time, such as those produced by changing local temperatures, will contribute unwanted noise to a broader integration swath.

Allan Variance is an important concept introduced to examine the stability of the imager detector system. It is defined as:

$$\sigma_y^2(\tau) = \frac{1}{2} < (y_{n+1}(\tau) - y_n(\tau))^2 >$$

where $y_n(\tau)$ is the average of the function over a time τ and the subscript n refers to successive averaging periods. The angle brackets denote the average value of the quantity within the brackets. The Allan Deviation is the square root of the Allan Variance.

For sufficiently short averaging periods, the noise within any period can be considered to be random and therefore both the average and the Allan Variance/Deviation obey the expectations of random noise and in particular the Allan Deviation declines as the square root of the observation time.

However for larger values of τ there will be drift and other effects within the averaging interval and the Allan Deviation will be larger than would be expected from purely random noise. Therefore by looking at the Allen Deviation as a function of τ we can see a square root dependence for small τ, but this will break down at longer times as drift and other instability effects become significant. The maximum value of τ for which the square root regime applies gives an estimate of the maximum integration time for which drift and other effects can be ignored in assessing the accuracy of the measurement.

Sources of instability for the MATMOS detector are any time-evolving systematic errors, probably most significant among them for our instrument is the temperature of the hardware. As the satellite comes over the horizon during a sunrise occultation, having been eclipsed by the planet for a time, the detector receives a significant thermal shock that, as the detector heats up, may heat the electronics to a degree that could introduce a new error into the measurements.

The question posed in this early phase of the project is two-fold: a) does the detector demonstrate an instability that can be seen and characterized?; and b) at what integration time does the error introduced by the instability begin to become significant? Specifically with regard to the latter question, if the point at which the instability begins to dominate is before the smear time (the time during which the atmosphere moves one pixel relative to the spacecraft) then we know the stability will be the limiting factor in our ability to integrate the measurements.

Method

As discussed, two aspects of the signal processing may be characterized to help us more precisely understand the data collected: the Integral Non-Linearity and the Allan Variance.

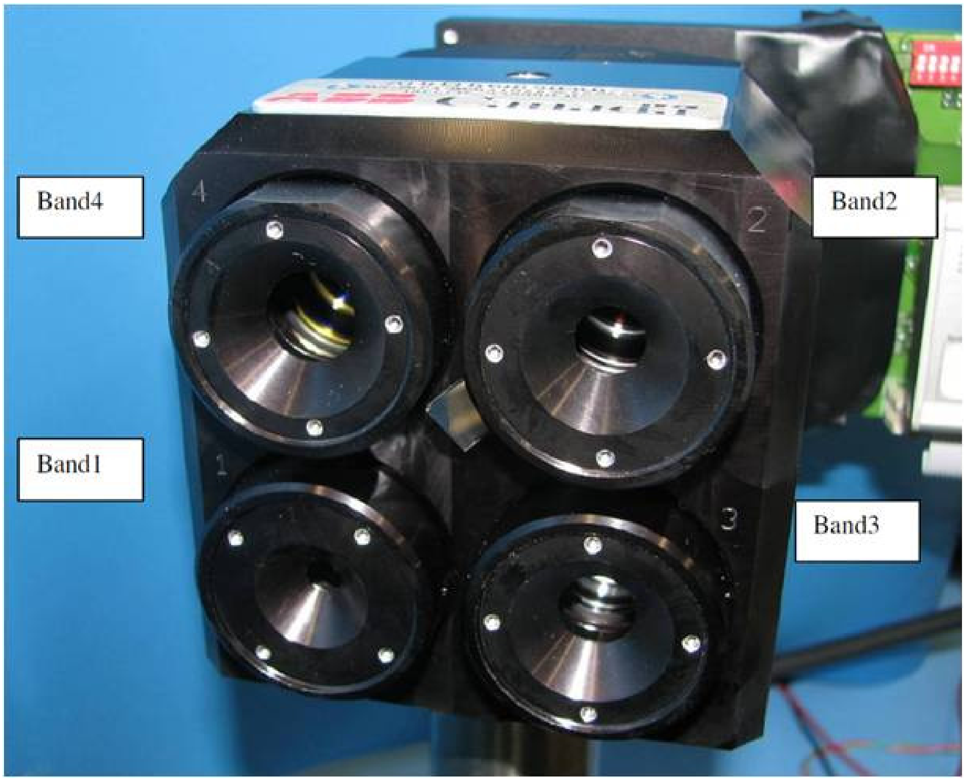


Figure 5.2: A front view of the 4-channel imager prototype. (Image provided by A. Tikhomirov.)

Some preliminary lab-based feasibility test results were produced by our research. Specifically, a prototype detector was received from ABB of Quebec City and tested to show, in principle, the feasibility of the aforementioned signal characterizations. Figure 5.2 shows the imager prototype and figure 5.3 shows the lab setup configuration.

To study the integral non-linearity of the system a known signal is applied to the detector. In our case a 66 kHz sine wave is used. The expected response of the ADC is then mapped against the digital results to examine deviations made during the conversion. Each test consisted of 100000 data points which were integrated over periods of 0.03 to 1000 seconds. (Tests made by A. Tikhomirov.)

Results

Figure 5.4 shows a plot of the measured DNL and INL for the applied sine wave as a function of the expected digital output.

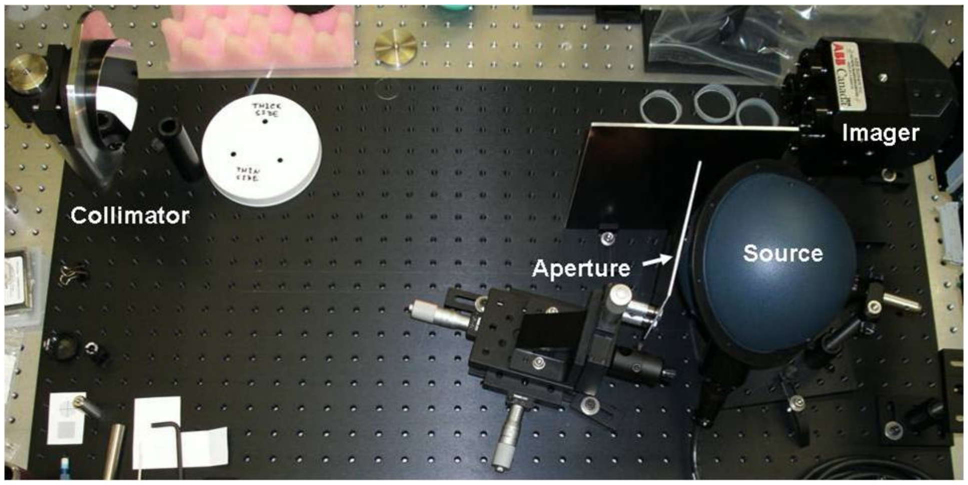


Figure 5.3: An overhead view of the imager test setup. A collimated beam is directed into the imager from a nearby halogen lamp source. (Image provided by A. Tikhomirov.)

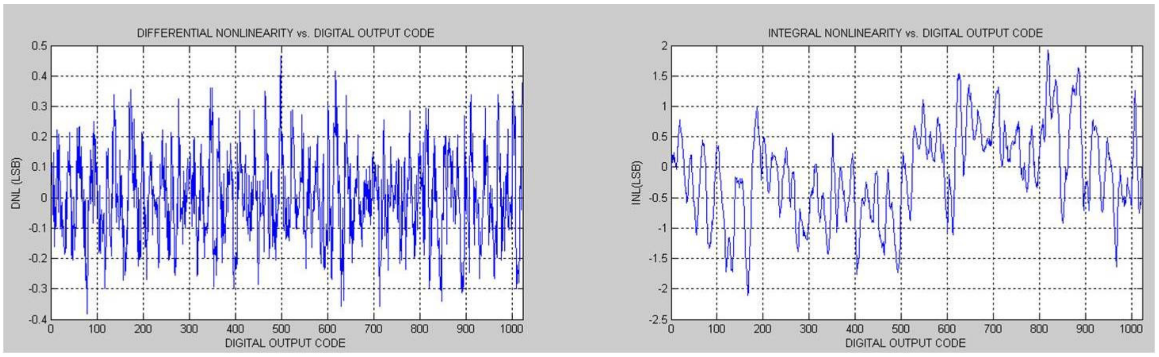


Figure 5.4: **Left**: A plot of the DNL of the imager using the prototype 10-bit ADC. **Right**: A plot of the INL cumulative effect of the imager using the prototype 10-bit ADC. (Plots provided by A. Tikhomirov.)

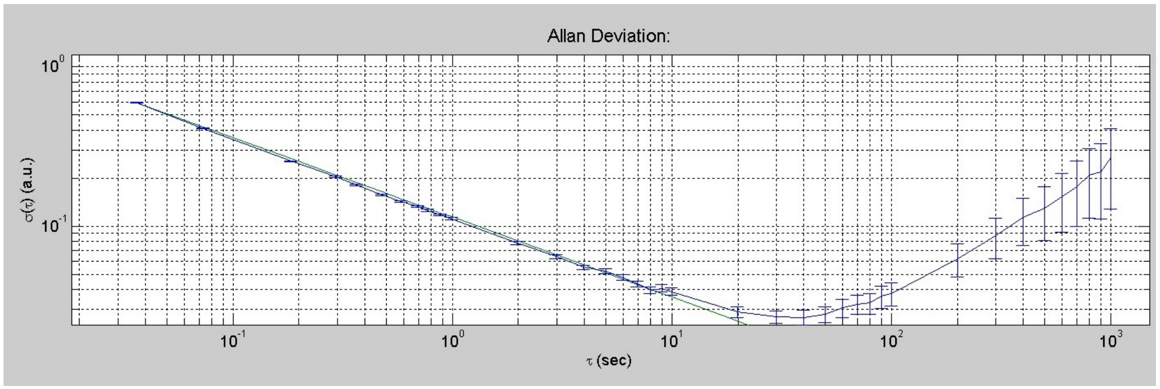


Figure 5.5: The Allan Deviation for pixel (359,124) is plotted against integration time. A smooth trendline is drawn. A local minimum is clearly observed between 3 and 4 seconds. (Plot provided by A. Tikhomirov.)

Within the four bands of the imager, a number of pixels are selected to examine the Allan deviation. The Allan Deviation for a typical pixel (359,124) from imager band #4 is shown in figure 5.5 plotted against the integration time. The expected random error, which reduces as $\frac{1}{\sqrt{\tau}}$, is also shown for comparison.

5.3 Spectral Analysis

To study the dust aerosol using the on-board FTS system we require a good knowledge of the expected gas and dust spectra. Toon et al. (1977) show derived optical properties for the Martian dust that match well with the known optical spectrum for montmorillonite spectra at a number of altitudes with broad dust extinction features. Based on observations made by IRIS during the 1971-1972 global dust storm, Toon observes extinction features in the range of 800–1400 cm^{-1}, peaking at about 1050 cm^{-1}, and relatively flat extinction at increasing wavenumbers.

Figure 5.6 is a plot taken from Wennberg et al. (2010) of the simulated limb transmittance spectra that incorporates the dust described by Toon [9].

MATMOS had proposed an FTS with a spectral range from 800–4300 cm^{-1} (as discussed in chapter 3.) It is known that the dust is primarily concentrated near the surface of Mars; however, this is also the region of highest gas content. The question that must be posed is: given the gaseous absorbers in the proposed spectral region, will we have the ability to study dust and other aerosols simultaneously? Presented another way, are there areas within the 800–4300 cm^{-1} where absorption "windows"

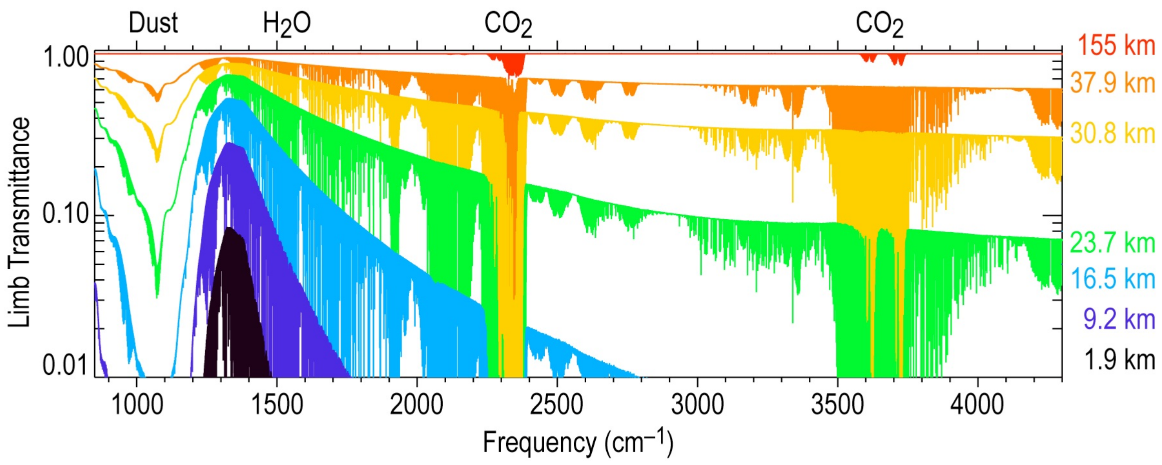


Figure 5.6: A plot of the simulated limb-viewed transmittance spectra for the Martian atmosphere taken from Wennberg (2010)[9].

may exist to examine the extinction of Martian aerosols or their fine structure? This question must be examined at varying altitudes observed during the occultation.If we can demonstrate that windows exist within the proposed region at low altitudes in sufficient abundance, this will suffice to indicate the possibility of studying the dust aerosol using the FTS model proposed.

Method

MATMOS has proposed a wavenumber range of 800 cm^{-1} to 4300 cm^{-1} for the Fourier Transform spectrometer. To study the expected spectrum seen through the limb-viewing geometry we will employ the use of high-resolution spectral modelling simulator, SpectralCalc [52].

SpectralCalc is a web-based spectral modeller developed by GATS, Inc., a US-based private aerospace company and NASA collaborator. SpectralCalc is a subscription-based program that provides numerous remote-sensing resources including solar and blackbody calculators, custom gas-cell simulations, and, importantly for our work, simulated atmospheric path spectra.

The atmospheric path simulator has two important features: it includes a creator-built Martian atmosphere option and has a built-in limb-viewing orientation.

The pre-built Martian atmosphere includes vertical mixing ratios for the bulk atmospheric gases (see figure 5.7 for a graph of mixing ratios at 20 km), and also

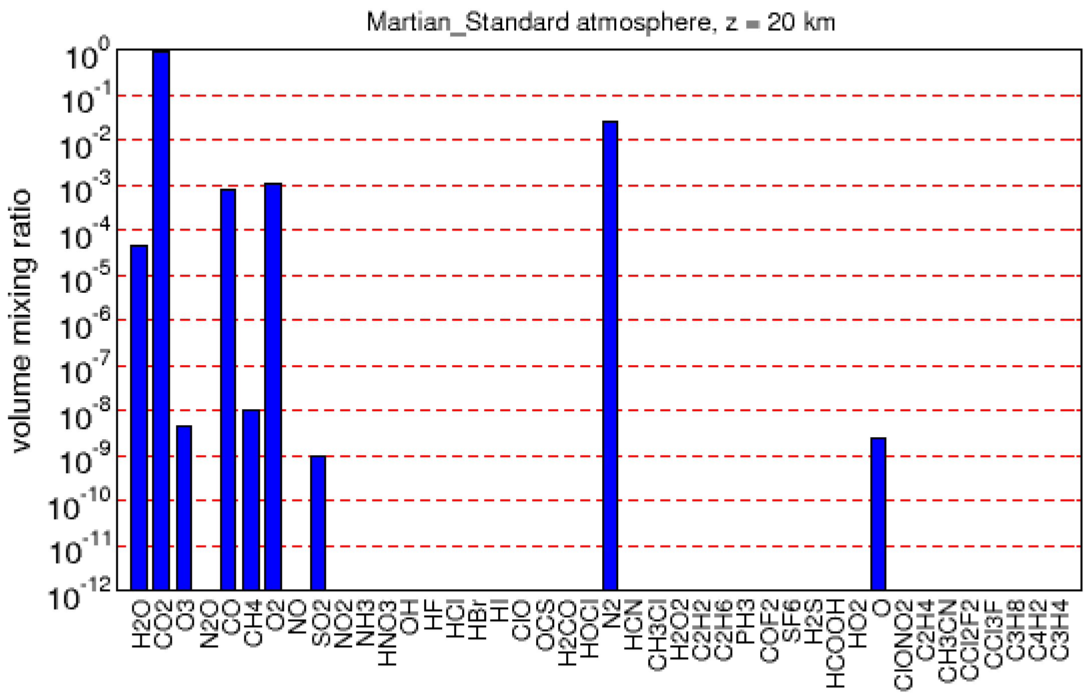


Figure 5.7: The vertical mixing ratio of gases present at 20 km in the SpectralCalc simulated atmosphere.

temperature and pressure profiles up to 300 km (see figure 5.8).

The simulated atmospheres are free of dust. This will allow us to find areas within the transmission spectrum where dust and aerosol extinction may be studied without the contamination from gas spectra.

Limb-viewing spectra were simulated for tangent heights ranging from 2 km to 100 km. A Gaussian instrument line shape function of 0.03 cm^{-1} is chosen. Each spectrum is simulated for the full FTS range in 14 separate portions, given the high resolution and amount of data. The spectra are then re-combined locally.

To search the spectra for “windows” through which the dust can be studied we run each spectrum file through a python program entitled “windowsearch.py”. The threshold under which a region may be considered a window is taken as an input. A transmission threshold of 0.999 was selected to be the condition for a clear window.

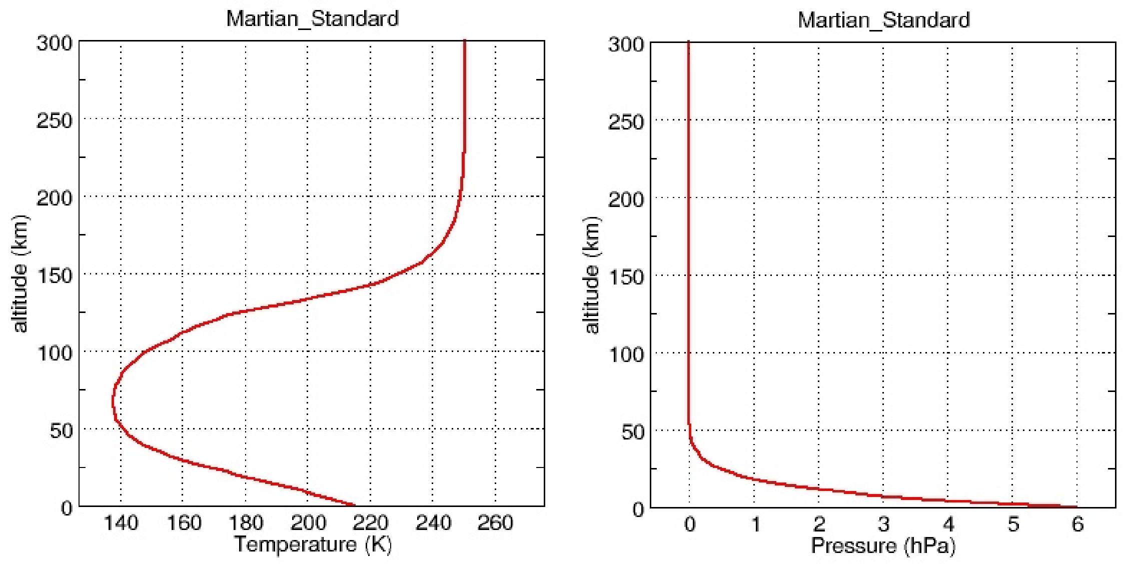


Figure 5.8: The temperature and pressure profiles used in the SpectralCalc simulated atmosphere.

Results

Figures 5.9 and 5.10 shows the spectrum taken from a SpectraCalc limb-viewing simulation at a 20 km tangent height. A computer algorithm named "windowssearch" was created for to extract spectral windows. The technique was run on the retrieved spectra. The region where windows exist are overlaid. Windows are shaded dark green if they are larger than 10 cm^{-1}, light green if smaller than 10 but larger than 5 cm^{-1}, and grey smaller than 5 but larger than 2 cm^{-1}.

Table 5.1 lists windows of $2cm^{-1}$ or greater found at the various altitudes examined.

Figure 5.11 shows the spectral distribution of the windows greater than $2cm^{-1}$ plotted at the simulated altitudes.

Windows in the 800–$1400cm^{-1}$ range are of particular interest to observe the dust extinction features. Figure 5.12 shows windows in that portion of the spectrum overlaid with the relevant portion from figure 5.6.

If the criteria for windows at least $2cm^{-1}$ is relaxed, we can look at the "micro" windows available in the region. Given the FTS has a resolution of $0.02cm^{-1}$, windows an order of magnitude larger ($0.2cm^{-1}$) would be desirable. Figure 5.13 shows the same plot as figure 5.12 with the new relaxed criterion.

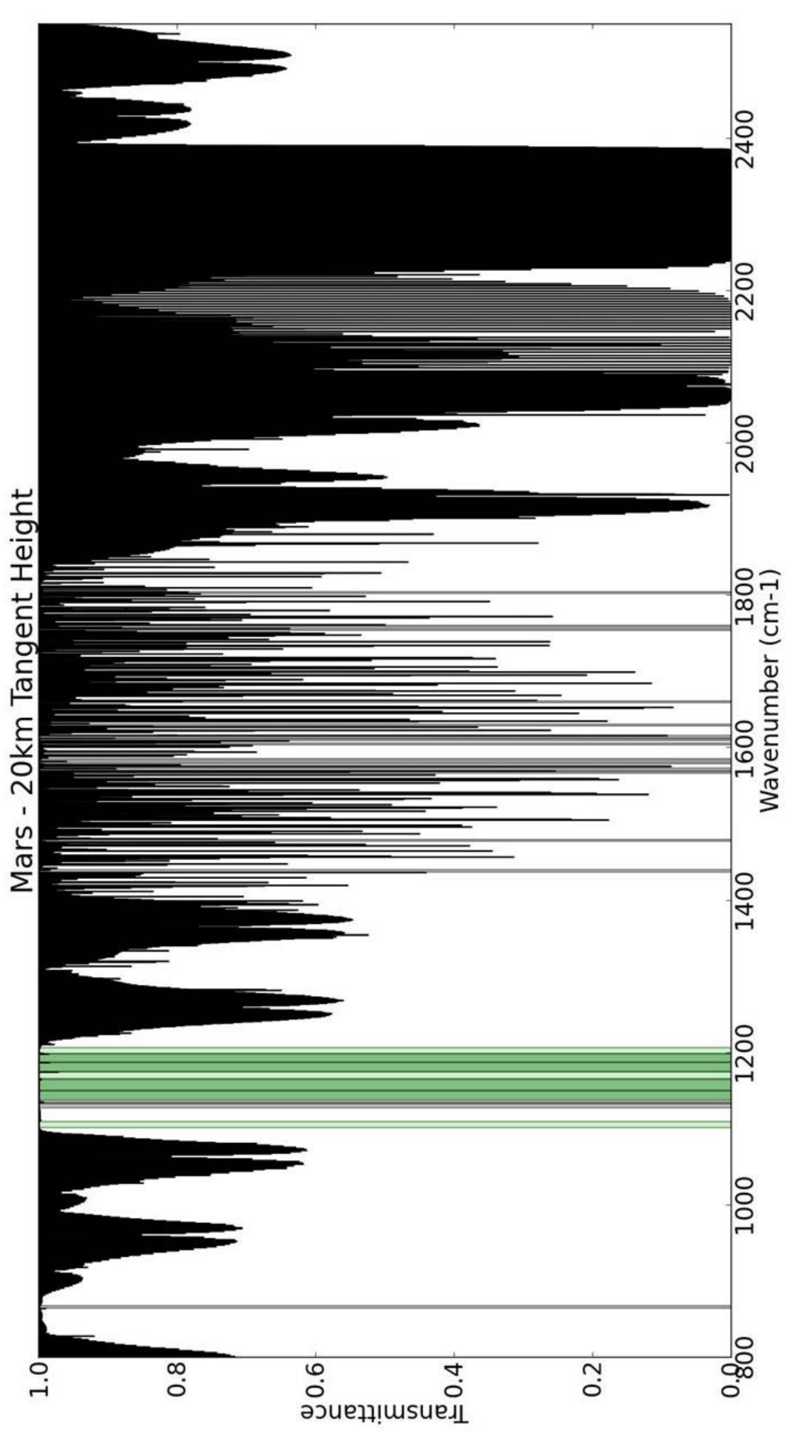


Figure 5.9: A simulated Martian transmission spectrum generated by SpectraCalc at a 20 km limb-viewing geometry showing 1800 cm^{-1} to 2550 cm^{-1}. Windows are shaded dark green if larger than 10 cm^{-1}, light green if 5-10 cm^{-1}, and grey if 2-5 cm^{-1}.

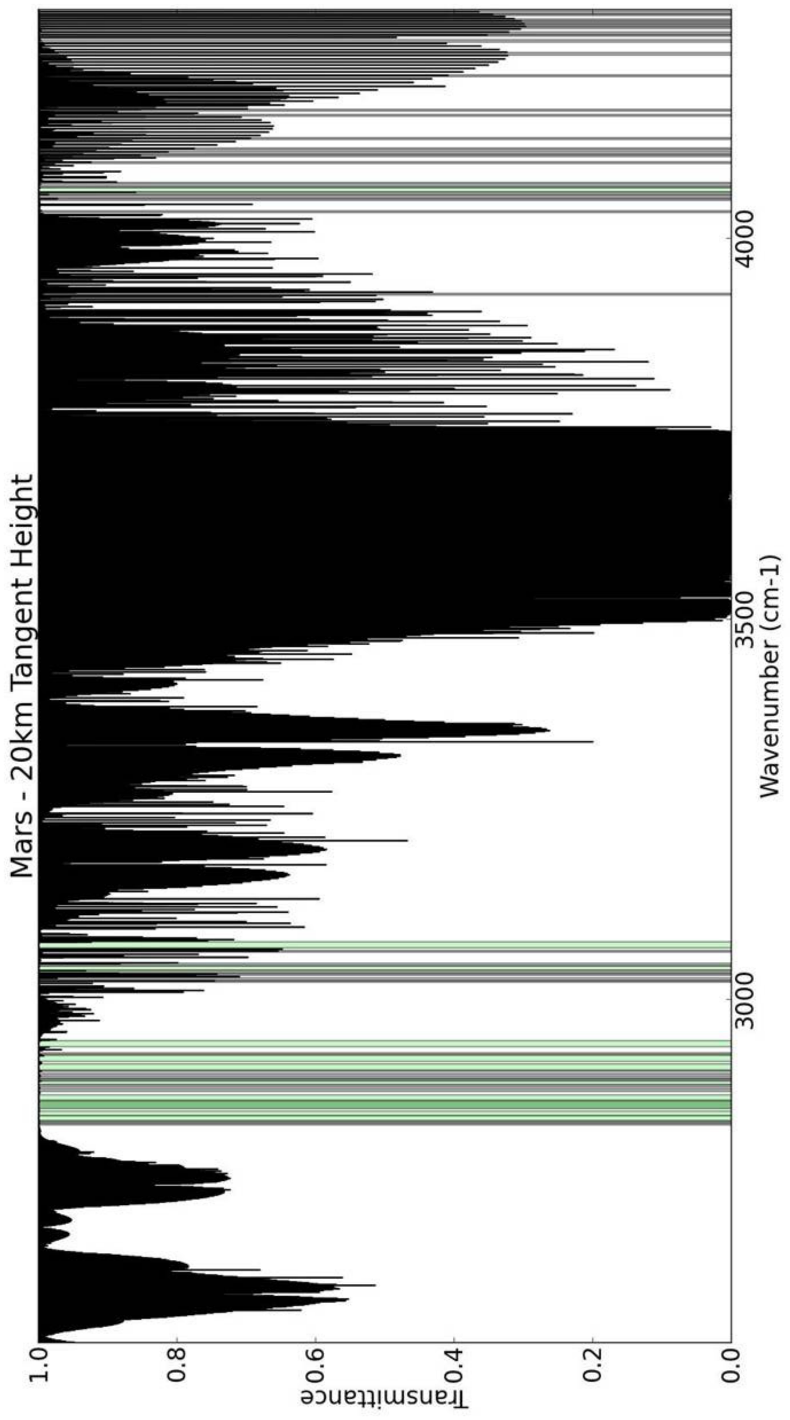


Figure 5.10: A simulated Martian transmission spectrum generated by SpectraCalc at a 20 km limb-viewing geometry showing 2550 cm^{-1} to 4300 cm^{-1}. Windows are shaded dark green if larger than 10 cm^{-1}, light green if 5-10 cm^{-1}, and grey if 2-5 cm^{-1}.

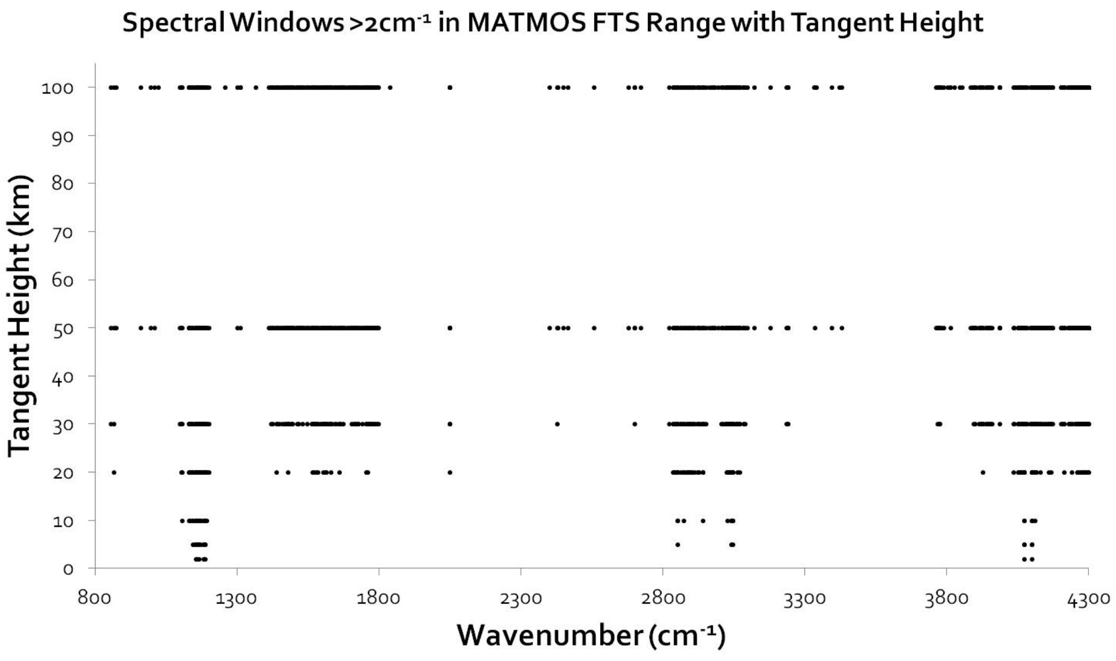


Figure 5.11: A plot of the distribution of windows over the full FTS range at simulated altitudes between 2 and 100 km.

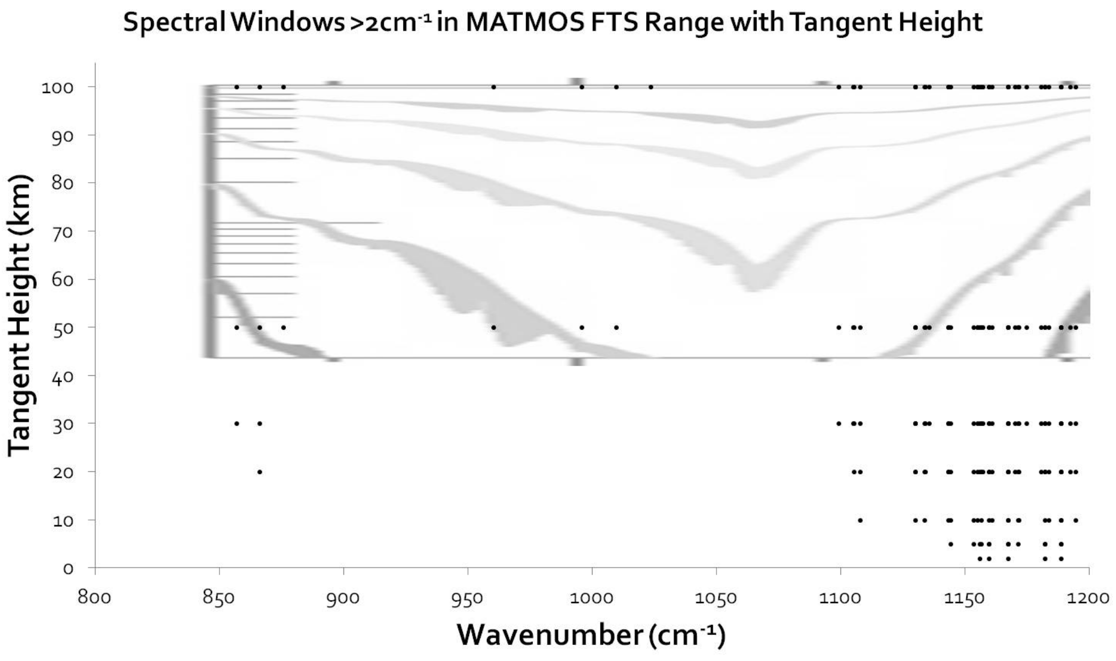


Figure 5.12: Windows of at least $2cm^{-1}$ width plotted for the various altitudes simulated, overlaid with a simulated dust spectrum for the same range.

Table 5.1: A table of the number and size of windows, here defined as transmissions exceeding 0.999, found between 800–4300 cm^{-1} at each simulated tangent height in the Martian atmosphere.

Tangent Height (km)	**Number of Windows** $> 10cm^{-1}$	**Number of Windows** $5 - 10cm^{-1}$	**Number of Windows** $2 - 5cm^{-1}$
2	0	0	7
5	0	1	13
10	1	4	20
20	6	19	75
30	9	24	145
50	25	69	481
100	12	16	26

5.4 Conclusions

As shown in figure 5.4, the INL of the ADC has been characterized. The INL appears to vary between ±2 LSB, which on the 10-bit converter amounts to an accuracy within ±0.2%. This is less than the manufacturer guarantee of ±3.5 LSB.

The Allan Deviation plotted in figure 5.5 clearly demonstrates the presence of instability of the detector. For integration times up 10 seconds the data agrees well with the overlaid random error decrease. At about 30-second integration the deviation of the data set betweens to resurge. This is a clear indication that the detector may be heating up or otherwise seeing instability which becomes significant on this time scale.

At the transmission threshold of 0.999 we identify numerous areas of interest where it may be possible to study the Martian dust aerosol. At very low tangent heights there are very few windows, most no larger than 2 cm^{-1}; however, at tangent heights where a strong signal would be observed, beginning at 10 km, there are numerous windows of varying sizes that persist as the measurement ascends through the atmosphere.

In order to identify a more adequate number of windows in the dust's unique extinction region of 800-1200 cm^{-1} the criterion of minimum window width may be relaxed from 2 cm^{-1} to 0.2 cm^{-1}, still well above the FTS resolution limit.

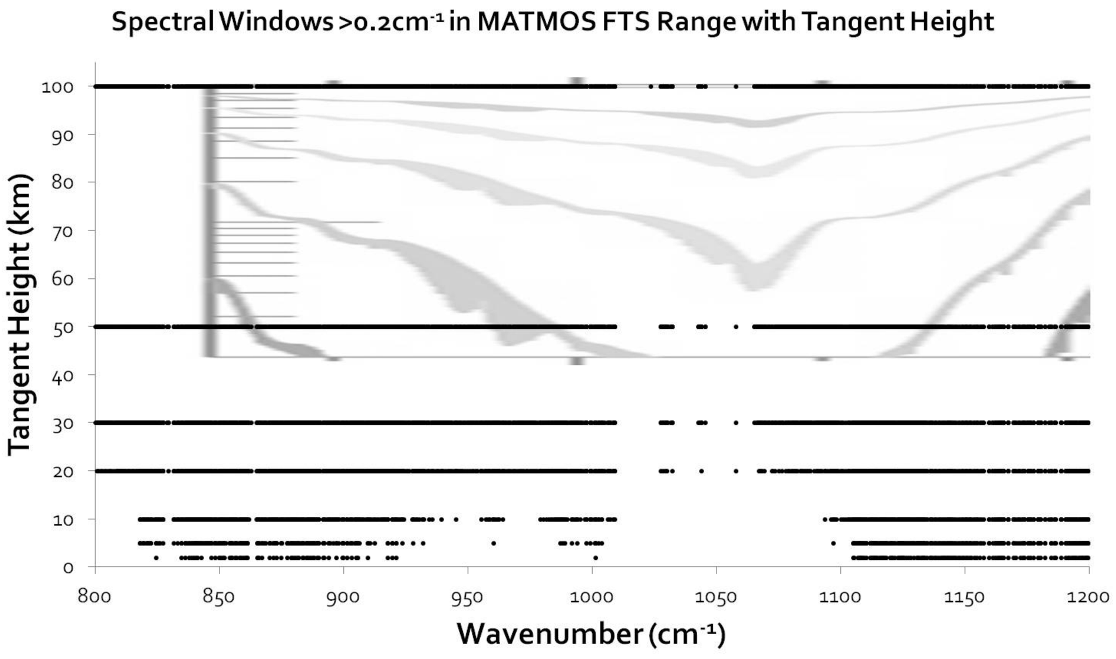


Figure 5.13: Windows of at least $0.2cm^{-1}$ width plotted for the various altitudes simulated, overlaid with a simulated dust spectrum for the same range.

One result worth noting is the decreasing number of windows at the highest tangent heights. This is because at these low pressures the windows begin to merge creating fewer, larger windows.

The method developed clearly identifies numerous windows in the transmission spectrum of a limb-viewing instrument over Mars. This method employs a simulated spectrum, which can be tailored to varying atmospheric conditions and gas abundances, and shows it is possible to identify spectral regions of interest to study the Martian dust aerosol.

Chapter 6

Conclusions and Future Work

Through the work undertaken over the course of this thesis we have helped us to further understand the precision achievable with the MATMOS imager.

With simulations of the CMOS detector we were able to create realistic-size images of the sun as seen by the MATMOS imager. Then, modifications were applied to the image to simulate both possible hardware defects and potential ray-path obstructions. Pixels were set to malfunction, columns were turned off, and dimming across the face of the solar disk were all examined.

It was concluded that individual unresponsive pixels had a minimal impact on the ability of the algorithm to detect the centroid. Defective pixels that artificially saturate had a larger impact, especially when appearing outside of the face of the solar disk, although known pixel defects can be turned off. Defective columns were shown to present a deviation of the detected centre of up to a maximum of between 14.9 pixels for one defective column and upwards of 20 pixels for multiple defective columns; however, by applying a simple integration using adjacent columns this deviation to below 0.2 pixel centre deviations. Lastly, fading across the face of the solar disk, meant to simulate fading from dust presence or spatial variations in hardware responsiveness, showed deviations amounting to 2.61 pixels at a 10x uniform fading.

It is postulated that constraining parameters, such as the apparent radius of the sun, over a sequence of images could help to improve the case where fading across the face of the image occurs. Additionally, it would be valuable to examine cases where detached layers, other non-uniform fading, or abstract shaped clouds would obscure the solar disk. Although partial distortions should, in theory, not inhibit the algorithm from determining the centre, the limit to which this is true and the degree to which the accuracy of the found centroid falls off would be valuable to quantify.

The result of this work is to conclude that the spacecraft pointing can be determined to a sufficient accuracy to permit an a priori estimate of the mass of atmosphere in the path to 1% or better.

Work was also conducted to characterize how MATMOS would be capable of studying Martian aerosols. Tests conducted by A. Tikhomirov on an imager prototype were presented. It was shown that it is possible to determine the INL of the ADC. Analysis of these tests allowed DNL and INL of the ADC to be extracted and it was concluded that the INL for the prototype was ± 2 LSB, less than the manufacturer quoted ± 3.5. Further, the instability is shown through the Allan Deviation plotted. It demonstrates an optimal integration time of less than 3 seconds, which is sufficient given the smear time of about 0.3 seconds. The conclusion is that although the accuracy of the digitiser is close to what is required (0.1%) further investigation is necessary.

The next steps in characterizing the imager would be to characterize the INL of the detector itself. Once all of the hardware characterized it will be possible to fully determine its relative contribution to the error in the measurements.

Lastly, simulations were performed using the SpectraCalc software package to simulate limb-viewing of the Martian atmosphere. Window regions in which to study the Martian aerosols were identified within the wavelengths of the FTS and catalogued for various altitudes. It was observed that many of the windows are clustered, many between 2800–3100 cm^{-1}.

SpectraCalc does allow for users to create and modify custom atmospheres with varying temperature profiles and mixing ratios. As part of the MATMOS objectives were to look for parts per trillion atmospheric constituents, introducing these smaller components would allow us to see where they appear in the FTS range and their potential impact on our windows used to study aerosols. Additionally, using the seasonal and spatial information available, one could create occultation simulations for varying regions or L_s seasonal times. Lastly, it may be worth examining other regions near the MATMOS-proposed range for other areas that, if added to the FTS range, would strengthen the ability to study aerosols (or trace gaseous) in the Martian atmosphere.

Bibliography

[1] David Williams. Mars Fact Sheet, 2010.

[2] M.G. Tomasko et al. Properties of dust in the Martian atmosphere from the Imager on Mars Pathfinder. *Journal of Geophysical Research*, 104:8987, 1999.

[3] P. Wennberg et al. MATMOS Investigation: Baseline Design Review. Technical report, 2012.

[4] Jet Propulsion Laboratory. *2016 ExoMars/Trace Gas Orbiter Instrument Kick-off Meeting*, Pasadena, California, October 2010.

[5] Swinburne University. The SAO Encyclopedia of Astronomy - Sublimation. http://astronomy.swin.edu.au/cosmos/s/sublimation.

[6] Belgian Instutite for Space Aeronomy. Retrieval of atmospheric information. http://venus.aeronomie.be/en/soir/retrievalofatminformation.htm.

[7] V.J. Hipkin & J.R. Drummond. Mars Atmosphere Trace Molecule Occultation Spectrometer (MATMOS) Solar Imager, Basic Performance and Description Document. Technical report, 2011.

[8] Cypress Semiconductor Corporation. 1M Pixel Radiation Hard CMOS Image Sensor. Technical report, January 2011.

[9] Paul Wennberg. MATMOS: Mars Atmospheric Trace Molecule Occultation Spectrometre. Technical report, California Institute of Technology, 2010.

[10] NASA Jet Propulsion Laboratory. All about mars - history. http://mars.jpl.nasa.gov/allaboutmars/mystique/history/1700.

[11] V. Krasnopolsky et al. Detection of methane in the martian atmosphere: evidence for life? *Icarus*, 172:537, 2004.

[12] V. Formisano et al. Detection of Methane in the Atmosphere of Mars. *Science*, 306:1758, 2004.

[13] F.W. Taylor. *Planetary Atmospheres.* Oxford University Press, 2010.

[14] R. Hanel et al. Mariner 9 Michelson Interferometer. *Applied Optics*, 11:2625, 1972.

[15] Michael Smith. Space Observations of the Martian Atmosphere. *Annu. Rev. Earth Planet. Sci.*, 36:191, 2008.

[16] J.R. Drummond. Refraction Effects in the Martian Atmosphere for Occultation Measurements. Technical report, Dalhousie University, 2011.

[17] John M. Wallace and Peter V. Hobbs. *Atmospheric Science - An Introductory Survey.* University of Washington, 2006.

[18] Gorn H. Pettengill. Winter Clouds over the North Martian Polar Cap. *Geophysical Research Letters*, 27:609, 2000.

[19] G. Neumann et al. Two Mars years of clouds detected by the Mars Orbiter Laster Altimeter. *Journal of Geophysical Research*, 108:5023, 2003.

[20] T.H. McConnochie et al. THEMIS-VIS observations of clouds in the martian mesosphere: Altitudes, wind speeds, and decameter-scale morphology. *Icarus*, 210:545, 2010.

[21] Franck Montmessin. Subvisible CO2 ice clouds detected in the mesosphere of Mars. *Icarus*, 183:403, 2006.

[22] L.K. Tamppari. Water-ice clouds and dust in the north polar region of Mars using MGS TES data. *Planetary and Space Science*, 56:227, 2008.

[23] L.K. Tamppari and R.W. Zurek. Viking era water-ice clouds. *Journal of Geophysical Research*, page 4087, 2000.

[24] R.T. Clancy et al. Water Vapor Saturation at Low Altitudes around Mars Aphelion: A Key to Mars Climate? *Icarus*, 122:36, 1996.

[25] R. Zurek et al. Dynamics of the atmosphere of Mars", year = 1992. *Mars*, page 835, 1992.

[26] J.A. Whiteway et al. Mars Water-Ice clouds and Precipitation. *Science*, 325:68, 2009.

[27] O.I. Korablev et al. Vertical Structure of Martian Dust Measured by Solar Infrared Occultations from the Phobos Spacecraft. *Icarus*, 102:76, 1993.

[28] N. Thomas et al. Observations of Phobos, Deimos, and bright stars with the Imager for Mars Pathfinder. *Journal of Geophysical Research*, 104:9055, 1999.

[29] O.B. Toon et al. Physical Properties of the Particles Composing the Martian Dust Storm of 1971-1972. *Icarus*, 30:663, 1977.

[30] T.Z. Martin et al. New dust opacity mapping from Viking infrared thermal mapper data. *Journal of Geophysical Research*, 98:10941, 1993.

[31] E. Anderson and C. Leovy. Mariner 9 Television Limb Observations of Dust and Ice Hazes on Mars. *Journal of the Atmospheric Sciences*, 35:723, 1978.

[32] F. Jaquin et al. The Vertical Structure of Limb Hazes in the Martian Atmosphere. *Icarus*, 68:442, 1986.

[33] R.T. Clancy et al. Extension of atmospheric dust loading to high altitudes during the 2001 Mars dust storm: MGS TES limb observations. *Icarus*, 207:98, 2010.

[34] Joshua Bandfield. High-resolution subsurface water-ice distributions on Mars. *Nature Letters*, 447:64, 2007.

[35] F. Fanale et al. Global Distribution and Migration of Subsurface Ice on Mars. *Icarus*, 67:1, 1986.

[36] A. Zent et al. Distribution and State of H2O in the High-Latitude Shallow Subsurface of Mars. *Icarus*, 67:19, 1986.

[37] Franck Montmessin. Origin and role of water ice clouds in the Martian water cycle as inferred from a general circulation model. *Journal of Geophysical Research*, 109, 2004.

[38] Vladimir Krasnopolsky. Some problems related to the origin of methane on Mars. *Icarus*, 180:359, 2006.

[39] G.R. Hunt. Mars: Components of Infrared Spectra and Composition of the Dust Cloud. *Icarus*, 18:459, 1973.

[40] R.T. Clancy et al. A new model for Mars atmospheric dust based upon analysis of ultraviolet through infrared observations from Mariner 9, Viking, and Phobos. *Journal of Geophysical Research*, 100:5251, 1995.

[41] Z.M. Dlugach et al. Physical properties of dust in the martian atmosphere: Analysis of contradictions and possible ways of their resolution. *Solar System Research*, 37:1, 2003.

[42] R.T. Clancy et al. Mars aerosol studies with the MGS TES emission phase function observations: Optical depths, particle sizes, and ice cloud types versus latitude and solar longitude. *Journal of Geophysical Research*, 108:5098, 2003.

[43] European Space Agency. Announcement of Opportunity for ExoMars Trace Gas Orbiter Instruments. http://exploration.esa.int/science-e/www/object/index.cfm?fobjectid=46297.

[44] Amy Svitak. ESA, NASA Redesigning ExoMars with Russia in Mind. *Aviation Week*, December 2011.

[45] Dan Thisdell. ESA: Russia stepping in to save ExoMars plan. *Flightglobal*, April 2012.

[46] R. Zurek et al. Final Report from the 2016 Mars Orbiter Bus Joint Instrument Definition Team, 2009.

[47] European Space Agency. NOMAD - Nadir and Occultation for MArs Discovery. http://exploration.esa.int/science-e/www/object/index.cfm?fobjectid=48530.

[48] Mark Allan. Private Communication, 2011.

[49] Willow Garage. Open Source Computer Vision. http://opencv.willowgarage.com.

[50] E. Chassefiere et al. Vertical Struecture and Size Distribution of Martian Aerosols from Solar Occultation Measurements. *Icarus*, 97:46, 1992.

[51] J. Doernberg et al. Full-Speed Testing of A/D converters. *Journal of Solid-State Circuits*, 19:820, 1984.

[52] GATS Inc. Spectral Calculator-Hi-resolution gas spectra. http://www.spectracalc.com.

Appendix A

Python Code

A.1 Interface.py

```
import sys, os, shutil
import cv
import cPickle as pickle
import time
import pdb, gc # pdb.set_trace()
import numpy as np

# Written by me
import CCD, process, centre, plotter

# Fixed Variables
bits = 8 # Digitizer Bits
digi = 2**bits # Based on bit number of digitizer (8-bit -> 255) and (12-bit -> 4095)

# Default directory
defaultdir = os.getcwd()

# Useful Functions:

def readdir(addr, delim="; "):
    dir = open(addr, "r")
    list = dir.readlines()
    i = 0
    list2 = []
    for line in list:
        line=line.split(delim)
        list2.append(line)
        end = len(line) - 1
        list2[i][end]=list2[i][end].rstrip("\n")
        i += 1
    dir.close()
    total = i - 1 # -1 for less the title
    return list2, total

def deleteline(fileaddr, linenum):
    list1 = open(fileaddr, 'r')
    lines = list1.readlines()
    list1.close()

    end = len(lines)
    if linenum >= end:
        return
    list2 = open(fileaddr, 'w')
    list2.writelines([item for item in lines[0:linenum]])
    list2.writelines([item for item in lines[linenum+1:end]])
    list2.close()
    return

def progress(counter, total, t_elapsed, t_next, t_step=30): #times in seconds
    if t_elapsed > t_next:
        print "-->", int(float(counter)/float(total)*100), "% Complete --", round(t_elapsed/60,1),
        "minute(s) elapsed."
        t_next += t_step
        return t_next
    else:
        return t_next

##################################################

def viewCCDs():
    print "\nRead from the CCDlist directory:"
    list1, total = readdir("CCDdir/ccdlist.txt")
    i = 0
    print "\tName \t\tCopies \tComments \t\t\t\t\tWidth \tHeight"
    print "\t--\t\t--\t--\t\t\t\t\t\t--\t--"
    while i < total:
        if len(list1[i+1][0]) < 16:
            name = list1[i+1][0] + (" ") * (16 - len(list1[i+1][0]))
        elif len(list1[i+1][0]) > 16:
            name = list1[i+1][0][:16]

        if len(list1[i+1][2]) < 48:
            comments = list1[i+1][2] + (" ") * (48 - len(list1[i+1][2]))
        elif len(list1[i+1][2]) > 48:
            comments = list1[i+1][0][:48]

        print "%(index)d\t%(name)s%(copies)s\t%(comments)s%(width)s\t%(height)s" %
        {'index':i+1, 'name':name, 'copies':list1[i+1][1], 'comments':comments,
        'width':list1[i+1][3], 'height':list1[i+1][4]}
        i += 1
    return

##################################################
```

```
def CCDdefects():
    print "\nRead from the defects directory:"
    list2, total = readdir("defects.txt")
    i = 0
    print "\tName \t\tDescription"
    print "\t--\t\t--"
    while i < total:
        print str(i+1) + "\t" + list2[i+1][0] + "\t\t" + list2[i+1][1]
        i += 1
    return

##################################################

def createCCD(auto=False):

    if not auto:
        cycle = 1
        # Uses CCD.selection to produce list of desired defects
        while True:
            defectlist = CCD.selection()
            if len(defectlist) > 0:
                break
            elif len(defectlist) == 0:
                choice1 = raw_input("\nNo defects were selected! Is this ok? Y/N ")
                if choice1 in ('Y', 'y', 'Yes', 'yes'):
                    break
                elif choice1 in ('N', 'n', 'No', 'no'):
                    pass
                else:
                    print "\n\t Not a valid input. Try selection again."
                    pass
            else:
                print "Error. Please reselect defects you wish to use."

        # Converts functions in defectlist to executables
        a = 0
        while a < len(defectlist):
            func = getattr(CCD, defectlist[a][0])
            defectlist[a][0] = func
            a += 1

        # Naming CCD
        list2, total = readdir("CCDdir\\ccdlist.txt")

        while True:
            newname = raw_input("\nWhat would you like to name this CCD? ") # Name
            j = total
            while j > 0: # Toggles through list2 looking for duplicate name
                if newname == list2[j-1][0]:
                    print "\n\tThis name is already taken! Try again."
                    j = -1
                else:
                    j -= 1
            if j == 0: # If toggled through and it was not found
                break


        # Select how many CCDs to make
        while True:
            try:
                numberCCD = input("\nHow many CCDs of this type would you like to generate? ") #Copies
                if (type(numberCCD)==int) and (numberCCD > 0):
                    break
            except(NameError):
                print "\n\tThat is not a valid number of CCDs to create."


        # Add comments in
        while True:
            comment = raw_input("\nComments: ")
            comment = str(comment)
            break

        # Creating and saving CCDs as pickle files
        newdir = defaultdir + "\\CCDdir\\" + newname
        while True:
            if not os.path.exists(newdir):
                os.makedirs(newdir)
                break
            else:
                try:
                    os.rmdir(newdir) #Removes directory created if empty
                except(WindowsError):
                    print "\n\tNon-empty directory already exists."
                    askremove = raw_input("Remove this directory? ")
                    if askremove in ['y','yes','Y','Yes']:
                        shutil.rmtree(newdir)
                    else:
                        raise KeyboardInterrupt

    while cycle > 0:
        l = 0
        m = 30 # Progress bar time steps in seconds
        dimX = getattr(CCD, 'dimX')
        dimY = getattr(CCD, 'dimY')
```

```
        while True:
            try:
                t1 = time.clock()
                while l < numberCCD:
                    picklename = newname + str(l+1)
                    CCD.generate(picklename, defectlist, newdir)
                    l += 1
                    t2 = time.clock()
                    m = progress(l, numberCCD, t2-t1, m)
                t2 = time.clock()
                print "\n\tAll", numberCCD, "CCDs were successfully created! -- Total time was",
                int((t2-t1)/60), "minutes."
                break
            except(KeyboardInterrupt):
                print "\n\tOnly", l, "CCDs were successfully created before termination."
                break

        # Saves info in CCDdir.txt
        CCDdir = open("CCDdir/ccdlist.txt", "a")
        CCDdir.write("\n" + newname + "; " + str(l) + "; " + comment + "; " + str(dimX) + "; " + str(dimY))
        CCDdir.close()

        if l == 0:
            shutil.rmtree(newdir) #Removes directory created if empty

        cycle -= 1

    if auto:
        script = open("scripts//script_CCD.txt", wb)
        ### Change READY to DONE
        script.close()

    return #End. Returns to main menu.

##################################################

def delCCDs(auto=False):
    CCDlist, total = readdir("CCDdir/CCDlist.txt")

    while True:
        i = 1
        print "\nWhat CCD set would you like to delete?"
        while i <= total:
            print "\t%(index)d. %(name)s -%(descrpt)s" % {"index": i, "name": CCDlist[i][0],
            "descrpt": CCDlist[i][2]}
            i += 1
        CCDinput = input()
        if type(CCDinput) == int and CCDinput in range(1,total+1):
            CCDfolder = "CCDdir//" + CCDlist[CCDinput][0]
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    prompt = raw_input("Are you sure you want to remove " + CCDlist[CCDinput][0] + "? ")
    if prompt in ('Yes', 'Y', 'yes', 'y'):
        shutil.rmtree(CCDfolder)
        deleteline("CCDdir/CCDlist.txt", CCDinput)
        print "\n\tCCDs sucessfully deleted."
    else:
        print "\n\tCCDs not deleted."
    return

##################################################

def fiximages(auto=False):

    if not auto:
        cycles = 1
        loglist, total = readdir("ProcessedImages/logs.txt")
        while True:
            i = 1
            print "\nWhich set of images would you like to fix?"
            while i <= total:
                print "\t%(index)d. %(name)s" % {"index": i, "name": loglist[i][0]}
                i += 1
            processedinput = input()
            if type(processedinput) == int and processedinput in range(1,total+1):
                CCDnumber = int(loglist[processedinput][1])
                T = int(loglist[processedinput][3])
                imagename = loglist[processedinput][4]
                CCDset = loglist[processedinput][5]
                break
            else:
                print "\n\tThat is not a valid selection. Try again"

        fixes = []
        fixlist, total2 = readdir("fixes.txt")
        while True:
            j = 1
            print "\nWhat fix would you like to apply?"
            while j <= total2:
                print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": j, "name": fixlist[j][0],
                "descrpt": fixlist[j][1]}
                j += 1
            fixinput = input()
            if type(fixinput) == int and fixinput in range(1,total2+1):
                fixes.append(fixlist[fixinput][0])
                break
            else:
                print "\n\tThat is not a valid selection. Try again"

        fixedname = raw_input("\nWhat would you like to name the fixed CCD set? ")

        while True:
            fixeddir = "ProcessedImages\\" + fixedname
            if not os.path.exists(fixeddir):
                os.makedirs(fixeddir)
                break
            else:
                try:
                    os.rmdir(fixeddir) #Removes directory created if empty
                except(WindowsError):
                    print "\n\tNon-empty directory already exists."
                    askremove = raw_input("Remove this directory? ")
                    if askremove in ['y','yes','Y','Yes']:
                        shutil.rmtree(fixeddir)
                    else:
                        raise KeyboardInterrupt

    if auto:
        autolist, lines = readdir("scripts//script_fix.txt")
        cycles = lines - 2

    while cycles > 0:
        if auto:
            select = lines - cycles
            CCDnumber = int(autolist[select][0])
            T = int(autolist[select][1])
            imagename = autolist[select][2]
            CCDset = autolist[select][3]
            fixes = []
            fixes.append(autolist[select][4])
            fixedname = autolist[select][5]

        k = 0
        while k < len(fixes):
            func = getattr(process, fixes[k])
            fixes[k] = func
            k += 1

        logfile = open(fixeddir + "//logfile(" + str(T) + ").pkl", 'wb')

        output = {'file': fixedname, 'threshold': T}
        baseimage = cv.LoadImage("Imagedir//" + imagename + ".tif", 0)
        (cent,size,angle)=centre.findCentre(T, baseimage)
        output['true_cent'] = cent
        output['true_size'] = size
        output['true_angle'] = angle


       # Loop through CCD pickles, deform images, "fix" images, and saves info in logfile
        k = 1
        t_next = 30 # Progress bar time steps in seconds
        result = ''
        while True:
            try:
                t1 = time.clock()
                while k <= CCDnumber:
                    pick = defaultdir + "\\CCDdir\\" + CCDset + "\\" + CCDset + str(k) + ".pkl"
                    defimage = process.deform("Imagedir//" + imagename, pick)
                    repairedimage = process.fix(defimage,fixes)
                    (cent,size,angle) = centre.findCentre(T,repairedimage)
                    output[str(k)+'_cent'] = cent
                    output[str(k)+'_size'] = size
                    output[str(k)+'_angle'] = angle
                    t2 = time.clock()
                    t_next = progress(k, CCDnumber, t2-t1, t_next)
                    k += 1
                if k == CCDnumber + 1:
                    print "\n\tImage was processed on all", CCDnumber, "CCDs."
                    break
            except(KeyboardInterrupt):
                print "\n\t"+ str(k-1) + " images were fixed and processed before termination."
                break

        pickle.dump(output, logfile)
        logfile.close()

        logs = open(defaultdir + "\\ProcessedImages\\logs.txt", 'a')
        logs.write("\n" +  fixedname + "; " + str(CCDnumber) + "; logfile(" + str(T) + ").pkl;
        " + str(T) + "; " + imagename + "; " + CCDset)
        logs.close()

        cycles -= 1

    return

##################################################

def viewImages(auto=False):
    print "\nRead from the imagelist directory:"
    list2, total = readdir("Imagedir/imagelist.txt")
    i = 0
    print "\tName \tFormat \t\tComments \t\t\t\tWidth \tHeight"
    print "\t--\t--\t\t--\t\t\t\t\t--\t--"
    while i < total:
        print str(i+1) + "\t" + list2[i+1][0] + "\t" + list2[i+1][1] + "\t\t" + list2[i+1][2] + "\t" +
        list2[i+1][3] + "\t" + list2[i+1][4]
        i += 1
    return

##################################################

def loadImage(auto=False):

    imagefold = defaultdir + "\\Imagedir\\" #Default image directory

    while True:
        where = raw_input("\nWhat is the full address of the image? ")
        if os.path.isfile(where): #Checks to see if there is a file at that address
            break
        else:
            print "\n\tCould not find the file requested. Try again." #No file

    while True:
        name = raw_input("\nWhat would you like to name this image? ")
        dest = imagefold + name + ".tif" #Full address of the new file once copied into directory
        if os.path.isfile(dest): #Checks that it does not already exist
            print "\n\tFile name already exists. Try again."
        else:
            comments = raw_input("\nComments: ") #Add comments for imagelist.txt
            break

    shutil.copyfile(where, dest) #Copy
    im = Image.open(dest)
    dimX, dimY = im.size #Gets X and Y dims from file
    imagelist = open(imagefold + "imagelist.txt", 'a')
    imagelist.write("\n" + name + "; .tif; " + comments + "; " + str(dimX) + "; " + str(dimY))
    imagelist.close()

    print "\n\tSuccess!"

    return

##################################################

def deleteImage(auto=False):
    imglist, total = readdir("Imagedir//imagelist.txt")

    while True:
        i = 1
        print "\nWhat image would you like to delete?"
        while i <= total:
            print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": i, "name": imglist[i][0],
            "descrpt": imglist[i][2]}
            i += 1
        imginput = input()
        if type(imginput) == int and imginput in range(1,total+1):
            img = "Imagedir/" + imglist[imginput][0] + imglist[imginput][1]
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    prompt = raw_input("Are you sure you want to remove " + imglist[imginput][0] + imglist[imginput][1] + "? ")
    if prompt in ('Yes', 'Y', 'yes', 'y'):
        os.remove(img)
        deleteline("Imagedir//imagelist.txt", imginput)
        print "\n\tImage sucessfully deleted."
    else:
        print "\n\tImage not deleted."
    return

##################################################

def deformImage():

    imglist, imgtotal = readdir("Imagedir/imagelist.txt")
    while True:
        i = 1
        print "\nWhat image would you like to process?"
        while i <= imgtotal:
            print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": i, "name": imglist[i][0],
            "descrpt": imglist[i][2]}
            i += 1
        imginput = input()
        if type(imginput) == int and imginput in range(1,imgtotal+1):
            img = "Imagedir/" + imglist[imginput][0] + imglist[imginput][1]
            br eak
        else:
            print "\n\tThat is not a valid selection. Try again"

    ccdlist, ccdtotal = readdir("CCDdir/CCDlist.txt")
    while True:
        j = 1
        print "\nOn which CCD(s) would you like to process this image?"
        while j <= ccdtotal:
            print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": j, "name": ccdlist[j][0],
            "descrpt": ccdlist[j][2]}
            j += 1
        CCDinput = input()
        if type(CCDinput) == int and CCDinput in range(1,ccdtotal+1):
            CCDset = ccdlist[CCDinput][0]
            CCDnumber = int(ccdlist[CCDinput][1])
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    while True:
        print "\nWhich image in the set? (Between 1 and %(num)d)" % {"num": CCDnumber}
        numpick = input()
        if type(numpick) == int and numpick in range(1,CCDnumber+1):
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    pick = defaultdir + "\\CCDdir\\" + CCDset + "\\" + CCDset + str(numpick) + ".pkl"

    imagename = raw_input("What would you like to name this image? ")

    defimage = process.deform(img, pick)
    cv.SaveImage("Imagedir\\" + imagename + ".tif",defimage)

    return


##################################################

def processImage(auto=False):

    # Selects the image to use
    imglist, imgtotal = readdir("Imagedir/imagelist.txt")
    while True:
        i = 1
        print "\nWhat image would you like to process?"
        while i <= imgtotal:
            print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": i, "name": imglist[i][0],
            "descrpt": imglist[i][2]}
            i += 1
        imginput = input()
        if type(imginput) == int and imginput in range(1,imgtotal+1):
            img = "Imagedir/" + imglist[imginput][0] + imglist[imginput][1]
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    # Selects threshold determination
```

```
while True:
    method = ''
    print "\nHow would you like the threshold set?"
    print "\t1. User input threshold value"
    print "\t2. Use interactive threshold determination"
    print "\t3. **Run a threshold determination algorithm **Not setup yet"
    method = raw_input()
    if method in ['1','2','3']:
        if method == '1':
            T = input("What is the threshold? (input an integer value) ")
            if T in range(0,digi):
                break
            else:
                print "Threshold must be an integer between 0 and", digi
        elif method == '2':
            T = centre.Tinteractive(img)
            break
        elif method == '3':
            T = centre.Talgorithm(img)
            break
    else:
        print "\n\t" + x + " is an invalid option. Try again."

# Selects the CCDs to process image onto
ccdlist, ccdtotal = readdir("CCDdir/CCDlist.txt")
while True:
    j = 1
    print "\nOn which CCD(s) would you like to process this image?"
    while j <= ccdtotal:
        print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": j, "name": ccdlist[j][0],
        "descrpt": ccdlist[j][2]}
        j += 1
    CCDinput = input()
    if type(CCDinput) == int and CCDinput in range(1,ccdtotal+1):
        CCDchoice = ccdlist[CCDinput][0]
        CCDnumber = int(ccdlist[CCDinput][1])
        break
    else:
        print "\n\tThat is not a valid selection. Try again"


# Creates folder for image in CCD dir
newdir = defaultdir + "\\ProcessedImages\\" + imglist[imginput][0] + "(" + CCDchoice + ")\\"
if not os.path.exists(newdir):
    os.makedirs(newdir)

# Creates output logfile and adds "true" centre info
nameroot = imglist[imginput][0] + "(" + CCDchoice + ")"
```

```
    logfile = open(newdir + "logfile(" + str(T) + ").pkl", 'wb')

    output = {'file': nameroot, 'threshold': T}
    baseimage = cv.LoadImage(img, 0)
    (cent,size,angle)=centre.findCentre(T, baseimage)
    output['true_cent'] = cent
    output['true_size'] = size
    output['true_angle'] = angle

    # Loop through CCD pickles, deform images, and saves info in logfile
    k = 1
    l = 30 # Progress bar time steps in seconds
    result = ''
    while True:
        try:
            t1 = time.clock()
            while k <= CCDnumber:
                pick = defaultdir + "\\CCDdir\\" + CCDchoice + "\\" + CCDchoice + str(k) + ".pkl"
                defimage = process.deform(img, pick) #MatImage output
                (cent,size,angle) = centre.findCentre(T,defimage)
                output[str(k)+'_cent'] = cent
                output[str(k)+'_size'] = size
                output[str(k)+'_angle'] = angle
                t2 = time.clock()
                l = progress(k, CCDnumber, t2-t1, l)
                k += 1
            if k == CCDnumber + 1:
                print "\n\tImage was processed on all", CCDnumber, "CCDs."
                break
        except(KeyboardInterrupt):
            print "\n\t", k-1, "images were processed onto CCDs before termination"
            break

    pickle.dump(output, logfile)
    logfile.close()

    logdir = open(defaultdir + "\\ProcessedImages\\logs.txt", 'a')
    logdir.write("\n" + nameroot + "; " + str(CCDnumber) + "; logfile(" + str(T) + ").pkl; "
    + str(T) + "; " + imglist[imginput][0] + "; " + CCDchoice)
    logdir.close()
    print "\n\t Complete"

    return # Returns to main menu

##################################################
def seefit():
    loglist, total = readdir("ProcessedImages/logs.txt")
    while True:
        i = 1
        print "\nFor which image would you like to see the fit? "
        while i <= total:
            print "\t%(index)d. %(name)s" % {"index": i, "name": loglist[i][0]}
            i += 1
        processedinput = input()
        if type(processedinput) == int and processedinput in range(1,total+1):
            CCDnumber = int(loglist[processedinput][1])
            img = "Imagedir\\" + loglist[processedinput][4] + ".tif"
            CCDset = loglist[processedinput][5]
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    while True:
        print "\nWhich image in the set? (Between 1 and %(num)d)" % {"num": CCDnumber}
        numpick = input()
        if type(numpick) == int and numpick in range(1,CCDnumber+1):
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    pick = defaultdir + "\\CCDdir\\" + CCDset + "\\" + CCDset + str(numpick) + ".pkl"
    defimage = process.deform(img, pick) #MatImage output
    cv.SaveImage('temp.tif',defimage)
    T = centre.Tinteractive('temp.tif')
    return


##################################################
def sequenceFit():
    # Selects the image to use
    imglist, imgtotal = readdir("Imagedir/imagelist.txt")
    while True:
        i = 1
        print "\nWhat image would you like to process?"
        while i <= imgtotal:
            print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": i, "name": imglist[i][0],
            "descrpt": imglist[i][2]}
            i += 1
        imginput = input()
        if type(imginput) == int and imginput in range(1,imgtotal+1):
            img = "Imagedir/" + imglist[imginput][0] + imglist[imginput][1]
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    # Select the CCDs to be used in the series
```

```
while True:
    sizeofseq = input("How many CCDs would you like as part of the sequence? ")
    if type(sizeofseq) == int and sizeofseq > 1:
        n = sizeofseq
        break
    else:
        print "\n\tThat is not a valid selection. Try again"

sequence = []

ccdlist, ccdtotal = readdir("CCDdir/CCDlist.txt")
while n > 0:
    j = 1
    print "\nOn which CCD would you like to process this image?"
    while j <= ccdtotal:
        print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": j, "name": ccdlist[j][0],
        "descrpt": ccdlist[j][2]}
        j += 1
    CCDinput = input()
    if type(CCDinput) == int and CCDinput in range(1,ccdtotal+1) and int(ccdlist[CCDinput][1]) == 1:
        sequence.append(ccdlist[CCDinput][0])
        n -= 1
    else:
        print "\n\tThat is not a valid selection. Try again"

#Set threshold to zero.
T = 0

#Sets up directory
seriesname = raw_input("What would you like to name this occultation series?")
newdir = defaultdir + "\\ProcessedImages\\" + imglist[imginput][0] + "(" + seriesname + ")seq\\"
if not os.path.exists(newdir):
    os.makedirs(newdir)

nameroot = imglist[imginput][0] + "(" + seriesname + ")seq"

k = 0
while k < len(sequence):
    if k == 0:
       logfile = open(newdir + str(sequence[k]) + "(baseline)\\logfile(" + str(T) + ").pkl", 'wb')
    else:
       logfile = open(newdir + str(sequence[k]) + "\\logfile(" + str(T) + ").pkl", 'wb')

    if k == 0:
        R = 0
    output = {'file': nameroot, 'threshold': T}
    baseimage = cv.LoadImage(img, 0)
    (cent,size,angle)=centre.findCentre(T, baseimage)
```

```
        output['true_cent'] = cent
        output['true_size'] = size
        output['true_angle'] = angle


    #retrieve R


    #fit others


##################################################
def plot():

    # Select Data
    datalist, ttl = readdir("ProcessedImages//logs.txt")
    while True:
        i = 1
        print "\nWhich data set would you like to view?"
        while i <= ttl:
            print "\t%(index)d. %(name)s - T=%(T)s" % {"index": i, "name": datalist[i][0],
            "T": datalist[i][3]}
            i += 1
        print "\t%(index)d. Compare multiple sets of data." % {"index": i}
        datainput = input()
        if type(datainput) == int and datainput in range(1,ttl+1):
            n_plots = 1
            log = "ProcessedImages\\" + datalist[datainput][0] + "\\" + datalist[datainput][2]
            break
        elif type(datainput) == int and datainput == ttl+1:
            n_plots = 3
            while i <= ttl:
                print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": i, "name": datalist[i][0],
                "descrpt": datalist[i][1]}
                i += 1
            datainput1 = input("Set 1: ")
            datainput2 = input("Set 2: ")
            datainput3 = input("Set 3: ")
            if type(datainput1) == int and datainput1 in range (1,ttl+1):
                log1 = "ProcessedImages\\" + datalist[datainput1][0] + "\\" + datalist[datainput1][1]
            if type(datainput2) == int and datainput2 in range(1,ttl+1):
                log2 = "ProcessedImages\\" + datalist[datainput2][0] + "\\" + datalist[datainput2][1]
            if type(datainput3) == int and datainput3 in range(1,ttl+1):
                log3 = "ProcessedImages\\" + datalist[datainput3][0] + "\\" + datalist[datainput3][1]
            break
        else:
            print "\n\tThat is not a valid selection. Try again"
```

```
# Select Plot Type (1)

while n_plots == 1:
    print "\nHow would you like to view the data?"
    print "\n\t1. Scatter Plot of Centre Deviation"
    print "\t2. Radial Distribution Plot"
    print "\t3. Scatter Centre Dev + Radial Dist Plots"
    plotinput = input()
    if type(plotinput) == int and plotinput in range(1,4):
        print "\n\t1. Auto Scale" # Select scaling
        print "\t2. Manually Scale"
        selscale = input()
        if type(selscale) == int and selscale in range(1,3):
            if selscale == 1:
                scale = 'Auto'
            elif selscale == 2:
                scale = 'Manual'
            else:
                print "\n\tInvalid input --> Defaulting to auto scale."
                scale = 'Auto'
        if plotinput == 1:
            plotter.scatter(log, scale)
            break
        elif plotinput == 2:
            plotter.radialdist(log, scale)
            break
        elif plotinput == 3:
            plotter.scat_raddist(log, scale)
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

# Select Plot Type (3)

while n_plots == 3:
    print "\nHow would you like to view the data?"
    print "\n\t1. 3 Scatter Plots Compare"
    plotinput = input()
    if type(plotinput) == int and plotinput in range(1,2):
        # Select scaling
        print "\n\t1. Auto Scale"
        print "\t2. Manually Scale"
        selscale = input()
        if type(selscale) == int and selscale in range(1,3):
            if selscale == 1:
                scale = 'Auto'
            elif selscale == 2:
                scale = 'Manual'
                else:
                    print "\n\tInvalid input --> Defaulting to auto scale."
                    scale = 'Auto'
            if plotinput == 1:
                plotter.scatter3(log1,log2,log3,scale)
                break
    return
##################################################
def vertslice():
    image = raw_input("What is the image address? ")
    imagearray = np.array(cv.LoadImageM(image,0))

    dimX = getattr(CCD, 'dimX')
    dimY = getattr(CCD, 'dimY')

    while True:
        colnum = input("What column number? ")
        if type(colnum) == int and colnum in range(0,dimX):
            break
        else:
            print "\n\tThat is not a valid selection. Try again"

    rownum = []
    colvalue = []
    i = 0
    while i < dimY:
        rownum.append(i)
        colvalue.append(imagearray[i][colnum])
        i += 1

    rownum = np.array(rownum)
    colvalue = np.array(colvalue)

    plotter.vertslice(rownum,colvalue,dimY)

    return

##################################################
def auto():
    scriptlist, total = readdir("scripts//scripts.txt")
    while True:
        i = 0
        print "\nWhat script would you like to run?"
        while i <= total:
            print "\t%(index)d. %(name)s" % {"index": i, "name": scriptlist[i][0]}
            i += 1
        scriptinput = input()
        if type(scriptinput) == int and scriptinput in range(1,total+1):
            selectedscript = "scripts//" + scriptlist[scriptinput][0] + ".txt"
            break
        else:
            print "\n\tThat is not a valid selection. Try again"
    script = open(selectedscript, wb)
    script.readlines()

##################################################


##################################################

if __name__ == '__main__':
    print "\n\tHello"
    ### MAIN MENU ###
    x = 0
    try:
        while True:
            print "\n-- What would you like to do? --"

            print "\nCCDs:"
            print "\t1. View full CCD inventory"
            print "\t2. View CCD defect inventory"
            print "\t3. Create a new CCD set"
            print "\t4. Delete an existing CCD set"
            print "\t5. Repair a set of images"

            print "\nImages:"
            print "\t6. View full image library"
            print "\t7. Load a new image"
            print "\t8. Delete a loaded image"
            print "\t9. Produce a deformed image"

            print "\nProcess:"
            print "\t10. Process image on a CCD set"
            print "\t11. See circle fitting"
            print "\t12. Sequence of images to set"

            print "\nResults:"
            print "\t13. View results"

            print "\nOther:"
            print "\t14. Exit"
            x = raw_input()
            if x in ['1','2','3','4','5','6','7','8','9','10','11','12','13','14']:
                if x == '1':
                    viewCCDs()
                elif x == '2':
                    CCDdefects()
                elif x == ’3’:
                    createCCD()
                elif x == ’4’:
                    delCCDs()
                elif x == ’5’:
                    fiximages()
                elif x == ’6’:
                    viewImages()
                elif x == ’7’:
                    loadImage()
                elif x == ’8’:
                    deleteImage()
                elif x == ’9’:
                    deformImage()
                elif x == ’10’:
                    processImage()
                elif x == ’11’:
                    seefit()
                elif x == ’12’:
                    sequenceFit()
                elif x == ’13’:
                    plot()
                elif x == ’14’:
                    raise KeyboardInterrupt
                gc.collect() # Collects garbage
            else:
                print "\n\t" + x + " is an invalid option. Try again."
    except(KeyboardInterrupt):
        # Exit
        print "\n\tGoodbye!\n"

##################################################
```

A.2 CCD.py

```
from random import randint
from random import normalvariate
from random import choice
from random import shuffle
import numpy as np
import pdb
import cPickle as pickle

import interface
```

```
# CCD & Image Dimensions
dimX = 256 #width in pixels, or number of columns
dimY = 256 #height in pixels, or number of rows
npixels = dimX * dimY

digi = getattr(interface, 'digi')

def getcoords():
    coords = []
    x=y=0
    while x < dimX:
        while y < dimY:
            coords.append((x,y))
            y += 1
        y = 0
        x += 1
    shuffle(coords)
    return coords

##################################################

def reset(notify=False): #Reset defects to be put into detector
    onoff = gain = np.ones((dimY,dimX))
    offset = np.zeros((dimY,dimX))

    if notify == True:
        print "Defects Reset."
    return gain, offset, onoff

##################################################

def selection(): #Select which defects will be chosen for a set of new detectors

    defectlist = []

    # Presents defects for selection

    list2, total = interface.readdir("defects.txt")
    total

    while True:

        while True: # Select Defect (Loop 1 of 2)
            defect = []
            j = 1
            print "\nWhat defects would you like to include?"
            while j <= total: #List all defects
                print "\t%(index)d. %(name)s - %(descrpt)s" % {"index": j, "name": list2[j][0],
                "descrpt": list2[j][1]}
                j += 1
            print "\t%(index)d. Done Selection" % {"index": total+1}
            print "\t%(index)d. Start Over" % {"index": total+2}
            definput = input()
            if type(definput) == int and definput in range(1, total+3):
                if definput <= total:
                    defect.append(list2[definput][0])
                    break
                elif definput == total+1: # Done Selection
                    return defectlist
                elif definput == total+2: # Start Over
                    defectlist = []
                    print "\n\tSelections reset!"
            else:
                print "\n\tThat is not a valid selection. Try again"

        while True: # Select variables (Loop 2 of 2)
            i = 2
            while len(list2[definput]) - i > 0:
                n = input(" %(variable)s = " % {"variable": list2[definput][i]})
                if type(n)==int and n > 0:
                    defect.append(n)
                    i += 1
                else:
                    print "\n\tThat is not a valid selection. Try again"
            break
        defectlist.append(defect)

##################################################

def generate(picklename, defectlist, newdir):

    gain,offset,onoff = reset()
    A = {}
    name = picklename + '.pkl'
    pick = open(newdir + "\\" + name, 'wb')

    i = 0
    while i < len(defectlist):
        defect = defectlist[i][0]
        if len(defectlist[i]) == 2:
            n1 = defectlist[i][1]
            gain, offset, onoff = defect(n1, gain, offset, onoff)
        elif len(defectlist[i]) == 4:
            n1 = defectlist[i][1]
            n2 = defectlist[i][2]
            n3 = defectlist[i][3]
            gain, offset, onoff = defect(n1, gain, offset, onoff, n2, n3)
        i += 1

    A = {'gain': gain, 'offset': offset, 'onoff': onoff, 'dimX': dimX, 'dimY': dimY}
    pickle.dump(A, pick)

    pick.close()
    return

##################################################
##################################################

def deadpix(n, gain, offset, onoff, killtype='Dark'): #Kills 'n' random pixels entirely
    if (n < dimX * dimY):
        coords = getcoords()
        killtemp = ''
        i=j=k=0
        while (i < n):
            j = coords[i][0]
            k = coords[i][1]
            if killtype == 'Mixed':
                killtemp = choice(['Dark','Bright'])
            if killtype == 'Dark' or killtemp == 'Dark':
                onoff[k,j] = 0 #Sets signal modifier to x0.
            elif killtype == 'Bright' or killtemp =='Bright':
                onoff[k,j] = digi #Set signal modifier to max.
                offset[k,j] = 1 #In case of no signal, sets to 1 so onoff multiplier works.
            i += 1
    else:
        print "Number of pixels requested to kill is larger than pixels in the CCD"
    return gain, offset, onoff

def darkpix(n, gain, offset, onoff):
    gain,offset,onoff = deadpix(n, gain, offset, onoff, killtype='Dark')
    return gain,offset,onoff

def brightpix(n, gain, offset, onoff):
    gain,offset,onoff = deadpix(n, gain, offset, onoff, killtype='Bright')
    return gain,offset,onoff

def mixedpix(n, gain, offset, onoff):
    gain,offset,onoff = deadpix(n, gain, offset, onoff, killtype='Mixed')
    return gain,offset,onoff

##################################################

def deadrow(n, gain, offset, onoff, killtype='Dark'): #Kills 'n' random rows of pixels
    if (n < dimY):
        i=j=k=0
        while (i < n):
            k = randint(1, dimY) - 1 #Chooses a random row
            if killtype == 'Mixed':
                killtemp = choice(['Dark','Bright'])
            if killtype == 'Dark' or killtemp == 'Dark':
                while (j < dimX):
                    if (onoff[k,j] == 0): #Toggles through entire row to see if it is already dead.
                        j +=1
                    else:
                        j = 0
                        break
                while (j < dimX):
                    onoff[k,j] = 0
                    j += 1
                j = 0
                i += 1
            elif killtype == 'Bright' or killtemp == 'Bright':
                while (j < dimX):
                    if (onoff[k,j] == digi): #Toggles through entire row to see if it is already dead.
                        j +=1
                    else:
                        j = 0
                        break
                while (j < dimX):
                    onoff[k,j] = digi
                    offset[k,j] = 1
                    j += 1
                j = 0
                i += 1
    else:
        print "Number of rows requested to kill is larger than rows in CCD"
    return gain, offset, onoff

def darkrow(n, gain, offset, onoff):
    gain,offset,onoff = deadrow(n, gain, offset, onoff, killtype='Dark')
    return gain,offset,onoff

def brightrow(n, gain, offset, onoff):
    gain,offset,onoff = deadrow(n, gain, offset, onoff, killtype='Bright')
    return gain,offset,onoff

def mixedrow(n, gain, offset, onoff):
    gain,offset,onoff = deadrow(n, gain, offset, onoff, killtype='Mixed')
    return gain,offset,onoff

##################################################
```

```
def deadcolumn(n, gain, offset, onoff, killtype='Dark'): #Kills 'n' random column of pixels
    if (n < dimX):
        i=j=k=0
        while (i < n):
            j = randint(1, dimX) - 1 #Chooses a random column
            if killtype == 'Mixed':
                killtemp = choice(['Bright','Dark'])
            if killtype == 'Dark' or killtemp == 'Dark':
                while (k < dimY):
                    if (onoff[k,j] == 0): #Toggles through entire column to see if it is already dead.
                        k +=1
                    else:
                        k = 0
                        break
                while (k < dimY): #Toggles through a column (if not dead) and turns off pixels
                    onoff[k,j] = 0
                    k += 1
                k = 0
                i += 1
            elif killtype == 'Bright' or killtemp == 'Bright':
                while (k < dimY):
                    if (onoff[k,j] == digi): #Toggles through entire column to see if it is already dead.
                        k +=1
                    else:
                        k = 0
                        break
                while (k < dimY):
                    onoff[k,j] = digi
                    offset[k,j] = 1
                    k += 1
                k = 0
                i += 1
    else:
        print "Number of columns requested to kill is larger than columns in CCD"
    return gain, offset, onoff

def mixedcol(n, gain, offset, onoff):
    gain,offset,onoff = deadcolumn(n, gain, offset, onoff, killtype='Dark')
    return gain,offset,onoff

def brightcol(n, gain, offset, onoff):
    gain,offset,onoff = deadcolumn(n, gain, offset, onoff, killtype='Bright')
    return gain,offset,onoff

def mixedcol(n, gain, offset, onoff):
    gain,offset,onoff = deadcolumn(n, gain, offset, onoff, killtype='Mixed')
    return gain,offset,onoff
```

```
##################################################

def vertgaingrad(factor, gain, offset, onoff, start=0, end=dimY):
    if factor > 1:
        delta = (1-1/float(factor))/(float(end-start)-1) # Gain factor decreases linearly towards 1/factor
        i=start
        while i < end:
            gain[i] = 1 - delta * (i - start)
            i += 1
    else:
        print "Cannot have gain factor less than 1"
    return gain,offset,onoff

##################################################
```

A.3 Deform.py

```
import numpy as np
import cv
import cPickle as pickle
import pdb

import interface

##################################################

def deform(img, pick, newdir, name):
    i=j=0
    gain=offset=onoff=np.array((1,1))
    dimX=dimY=0

    cvmatimg = cv.LoadImageM(img, 0) #Imports the image in grayscale
    imgarray = np.array(cvmatimg) #Converts image to an array
    dtype = imgarray.dtype

    A = pickle.load(open(pick))
    gain = A['gain']
    offset = A['offset']
    onoff = A['onoff']
    #leakage = A['leakage']
    dimX = A['dimX']
    dimY = A['dimY']

    # Check dimX, dimY to image cols, rows
    if dimX != cvmatimg.cols or dimY != cvmatimg.rows:
        return 'sizeerror'

    # Deform
    defimage = (imgarray * gain + offset) * onoff

    # Make sure no pixel value is above limit
    #np.where(defimage > digi, digi, defimage)

    # Save Image
    cv.SaveImage(newdir + "\\" + name + ".tif", defimage)

    return 'good'
```

A.4 Centre.py

```
 import sys
 import os
 import cv
 import datetime
 import numpy
 from scipy import *

 import interface

 digi = 255

#########################################################


 def pickcontour(contour, goodfit=50):

         if contour == None: #Case where no contours (good or bad) are found
                 return 0,contour
  elif contour.h_next == None: #Case where 1 contour (good or bad) is found
                 if len(contour) >= goodfit:
                         return 1, contour
                 else:
                         return 0,contour
  else: #Multiple contours are found
  count = []
  count.append(len(contour))
  tempcont = contour.h_next()
  while tempcont:
  count.append(len(tempcont))
  tempcont = tempcont.h_next()
  #print count
  if max(count) < goodfit:
  return 0,contour
  else:
  a = count.index(max(count))
  for i in range(0,a):
  contour = contour.h_next()
  return 1,contour


 def process_image(slider_pos, image01, interactive=True):

         # Create copies of image for display windows
         image02 = cv.CloneImage(image01) #image02 is threshold applied to image01
         cv.Zero(image02)
         if interactive:
                 image03 = cv.CloneImage(image01)

  # All pixels in image01 below threshold (slider_pos) are set to zero in image02.
  cv.Threshold(image01, image02, slider_pos, digi, cv.CV_THRESH_BINARY)
         if interactive:
                 cv.ShowImage("Binary Output", image02)

  # Find all contours in image02.
  stor = cv.CreateMemStorage(0)
  cont = cv.FindContours(image02, stor, cv.CV_RETR_LIST, cv.CV_CHAIN_APPROX_NONE, (0,0))

  # Find contour with maximum number of points.
  result,cont = pickcontour(cont)
  if result == 0:
  # If no good contours, return.
  cv.Zero(image02)
                 if interactive:
                         cv.ShowImage("Result", image02)
  return

  else:
  # Alloc memory for contour point set and pull data from active contour.
  PointArray2D32f = cv.CreateMat(1, len(cont), cv.CV_32FC2)
  for (i,(x,y)) in enumerate(cont):
  PointArray2D32f[0,i] = (x,y)
                 if interactive:
                         cv.DrawContours(image03, cont, cv.RGB(255,255,255), cv.RGB(255,255,255),0,1,8,(0,0))

  # Fits ellipse to current contour.
  (cent,size,angle) = cv.FitEllipse2(PointArray2D32f)
  #print "cent =", cent, "size =", size, "angle =", angle
  centerint = (cv.Round(cent[0]),cv.Round(cent[1]))
  sizeint = (cv.Round(size[0]*0.5), cv.Round(size[1]*0.5))
  angle2 = -angle

  # Draw ellipse to image.
                 if interactive:
                         cv.Ellipse(image03, centerint, sizeint, angle2, 0, 360, cv.RGB(80,200,130),
                         1, cv.CV_AA, 0)
                         cv.ShowImage("Result", image03)
         if not interactive:
                 return (cent,size,angle)


 def fitCircle(array, x_m=128, y_m=128): #Largely from http://www.scipy.org/Cookbook/Least_Squares_Circle
         center_estimate = x_m, y_m
         center, ier = optimize.leastsq(centre.f, center_estimate)

         xc, yc = center
         Ri = centre.calc_R(*center)
         R = Ri_2.mean()
         residu = sum((Ri - R)**2)

         return center,size,angle


 def calc_R(xc, yc):
         """ calculate the distance of each 2D points from the center (xc, yc) """
         return sqrt((x-xc)**2 + (y-yc)**2)

 def f(c):
         Ri = calc_R(*c)
         return Ri - Ri.mean()

  def Tinteractive(baseimage):

  # Set default threshold to apply
  slider_pos = 0

  # Create windows.
  cv.NamedWindow("Source", 0)
  cv.NamedWindow("Binary Output",0)
  cv.NamedWindow("Result", 0)
  cv.CreateTrackbar("Threshold", "Binary Output", slider_pos, digi, null)

         # Load image
         image01 = cv.LoadImage(baseimage, 0)
  if not image01:
  print '\n\tERROR: Could not load image'
                 return -1
  cv.ShowImage('Source', image01)
  # Loop through main code
  while True:
                 try:
                         slider_pos = cv.GetTrackbarPos("Threshold", "Binary Output")

                         # Perform Ellipse Fitting
                         process_image(slider_pos, image01, 'True')

                         cv.WaitKey(500)

                 except(KeyboardInterrupt):
                         verify = raw_input("\nSet " + str(slider_pos) + " as threshold? ")
                         if verify in ('Y', 'y', 'Yes', 'yes'):
                                 T = slider_pos
                                 break
                         else:
                                 pass
  cv.DestroyAllWindows()
  return T



 def findCentre(Threshold, image):
         cv.SaveImage('temp1.tif',image)
         image01=cv.LoadImage('temp1.tif',0)
         (cent,size,angle) = process_image(Threshold, image01, interactive=False)
         os.remove('temp1.tif')
         return (cent,size,angle)


 def null(a):
         pass
```