UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

Disambiguation of range-Doppler radar measurements

Andrea Marie Alvsaker

FYS-3931 Master's Thesis in Space Physics - 30SP
01 June 2022

The front page shows an optical image of the lunar nearside. I acknowledge the use of imagery from Lunar QuickMap, a collaboration between NASA, Arizona State University & Applied Coherent Technology Corp [6].

Abstract

Observing the Moon by a monostatic synthetic-aperture radar system has the inherent problem that any given combination of range and Doppler shift in a measurement will map to two different regions on the lunar surface. This ambiguity has previously been avoided using an interferometric radar configuration or selective illumination. The objective of this thesis was to write and validate a method for disambiguating monostatic inverse synthetic-aperture radar measurements of the Moon. The method implemented in this thesis was conducted by simulating multiple range-Doppler radar maps with differing apparent rotation axes, based on data from NASA's Horizon ephemeris [13] and optical reflectivity measurements from the Lunar Reconnaissance Orbiter Camera [6]. Included in the simulation were the Hagfors scattering law to allow scaling the amount of backscattered power based on the incidence angle of the electromagnetic wave, range dependent power reduction and noise due to random surface and subsurface undulations. The disambiguation, thus estimation of the true normalized scattering cross-sections, was conducted by solving an overdetermined linear least square system of the simulated maps. The disambiguation based on three range-Doppler maps resulted in an error standard deviation of 18.56% of the average reflectivity value, which decreased by including additional maps in the procedure. The method described in this thesis was found to be unbiased and can be used with EISCAT3D when it becomes operational in the future, and existing Arecibo measurements of Venus [3].

Acknowledgements

First and foremost, I would like to thank my supervisors Dr. Juha Vierinen and Torbjørn Tveito for introducing me to this interesting topic, and for developing and sharing with me your idea behind the method utilized in this thesis. Thanks to Juha for all interesting and informative discussions, and for your enthusiastic approach to whatever question I have asked. Thanks to Torbjørn for always being available, for your quick responses and willingness to help no matter what theoretical question or programming problem I have consulted with you. I would like to thank you both for such extraordinary assistance and support throughout this project. I truly appreciate all the help you have provided me and the countless hours you have spent on me.

I would also like to thank my fellow master's students, especially Ingeborg, for all academic and especially non-academic discussions, and for creating a good atmosphere in the office. Last, but not least, I would like to thank my family and friends for their support and for taking the time to help me proofread this thesis.

Contents

List of Figures

List of Tables

List of symbols

k_B	Boltzmann constant	[J/K]
d	Doppler shift	[Hz]
τ_p	Effective pulse length	[m]
ϕ	Incidence angle	[deg]
φ	Latitude	[deg]
ϑ	Longitude	[deg]
R_{Moon}	Moon radius	[m]
T_c	Observation time	[s]
λ_{radar}	Radar wavelength	[m]
r	Range	[m]
T_p	Rotation period	[s]
f_p	Rotation rate	[Hz]
c	Speed of light	[m/s]

1 Introduction

The first instrumental observations of the Moon were performed using a telescope by Thomas Harriot on the 26th of July 1609, closely followed by Galileo Galilei four months later. Their observations resulted in sketches of the lunar surface variations and calculations describing the lunar orbit [23]. Since then, we have gained more information about the Moon than any other planetary objects besides the Earth [16]. However, there still remains numerous mysteries regarding the evolution of the Moon.

One of these mysteries includes the lunar farside-nearside asymmetry, even visible to the naked eye. The asymmetry has been hypothesized to be the result of an ancient global redistribution of subsurface igneous rocks, but a recently published article by Jones et al. (2022) [11] provided an extended hypothesis. Their study revealed a potential correlation between the impact event creating the south pole-Aitken basin, and the global redistribution causing the asymmetry. By simulations, they demonstrated that the creation of this vast impact crater, about 4.3 billion years ago, caused sufficient heat to trigger a flow of igneous rocks, rich of Titanium and Thorium, to the lunar nearside. The large presence of Titanium and Thorium cause the low albedo in the nearside mare region, which results in the asymmetry [11].

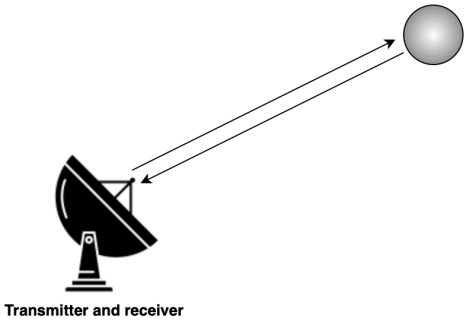
Exploration of the lunar structure and composition is a key source in revealing information about the history of our solar system. The moon has been exposed to 4.5 billion years of environmental changes and its internal structure may reveal the initial stages of terrestrial planetary evolution [16]. Thus, being able

to examine the different lunar layers has significant scientific value.

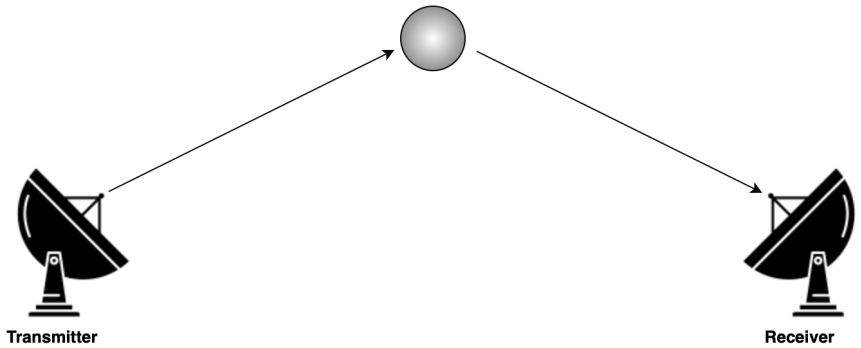
Remote sensing is a broad concept covering the numerous techniques of measuring and determining a target's physical properties from a distance. The common idea between the various techniques is measuring energy either emitted from, or reflected off, a target. Remote sensing techniques has developed through time from visually observing a target to technologically advanced sensors today. Remote sensing gained momentum as the photography was invented in the late 1830's. The French journalist Félix Nadar was the first to take pictures from the air when he photographed the city of Paris from a balloon in 1858 [8]. Some of the first actively used radars were developed prior to, and during, the second world war [20]. These devices were used in aircraft and marine vessels enabling mapping and monitoring of the surrounding terrain [9]. Radar technology has greatly improved since the 1930's and is today applied for numerous diverse purposes including atmospheric, environmental, and meteorological surveillance, space and planetary studies, and aircraft and marine traffic control to name a few [20].

Remote sensing may be divided into two categories, passive and active. Passive remote sensing involves measuring existing energy either emitted by or reflected off a target, while active remote sensing involves generating and transmitting energy before subsequently measuring the reflected energy. The energy used in remote sensing can originate from either mechanical waves, as used in sound navigation and ranging (SONAR) and seismology, or electromagnetic waves, used in radio detection and ranging (RADAR) and light detection and ranging (LIDAR). The wavelengths used in LIDAR systems lies in the ultraviolet, visible and near infrared part of the electromagnetic spectrum [12], while active RADAR systems most commonly utilize the microwave to radio wave part of the spectrum [2].

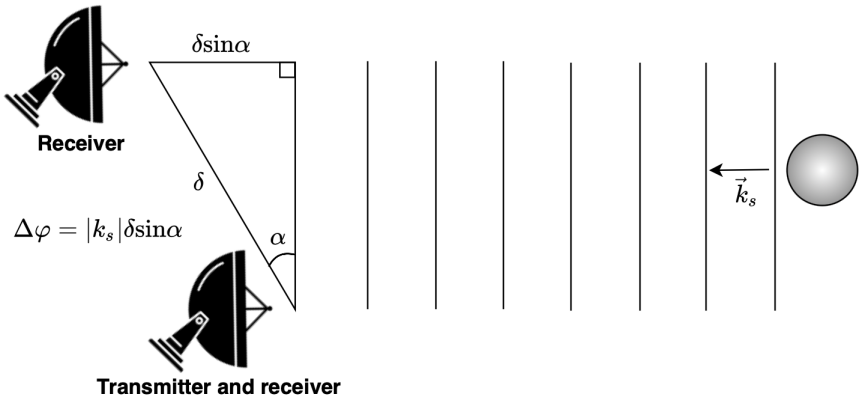
The long wavelengths obtainable by an active radar system have the property of ground penetration. This property is essential in order to investigate the different layers of planetary objects, such as the Moon [2]. The longer the wavelength, the deeper the penetration depth, thus ancient structures further back in evolutionary history can be detected. Such ground penetrating radars have been used to discover intact lunar lava tubes. [21] Due to these tubes being shielded from external impacts, like meteors and cosmic radiation, they are considered, from a scientific perspective, as potential locations for lunar based instrumental constructions and even human habitats in the future [21].



(a) Illustration of a monostatic radar configuration. The transmitting and receiving antennas are co-located.



(b) Illustration of a multistatic radar configuration. The transmitting and receiving antennas are separated in distance.



(c) Illustration of an interferometric radar configuration. One antenna is both a transmitting and a receiving antenna, while one or more are separated receiving antennas. Vector $\vec{k}_s$ is the wave vector of the scattered plane waves from the target. The relative phase, $\Delta\varphi$, of the receiving echoes, due to the separation, δ, is measured.

Figure 1.1: Illustration of a monostatic, multistatic and an interferometric radar configuration.

An active radar system consists of a transmitting antenna and one or more receiving antennas. The antennas may be co-located as shown in Figure 1.1a, termed a monostatic configuration, or separate as shown in Figure 1.1b, termed a multistatic configuration. In addition, there is an interferometric radar configuration consisting of one transmitter and two or more receiving antennas as shown in Figure 1.1c. In this configuration the relative phase, $\Delta\varphi$, of the received echoes is measured [2]. These radar configurations may be ground based, satellite based or a combination of both.

As a spherical target, like the Moon, is observed by a radar system it is essential to separate the measurements according to their origin on the target surface. A monostatic synthetic-aperture radar (SAR) measures the time delay of each signal, making it possible to separate the measurements in range-direction. In addition, SAR takes advantage of the relative motion between the radar and target by measuring the Doppler shift of the signal. SAR defines the relative motion as the motion of the emitter, while an inverse SAR (ISAR) defines the relative motion as the target motion. Due to the motion of the Moon relative to an Earth based antenna, as well as the Moon's rotation about its own axis, the Doppler shift of each measurement by a monostatic ISAR enables azimuthal separation.

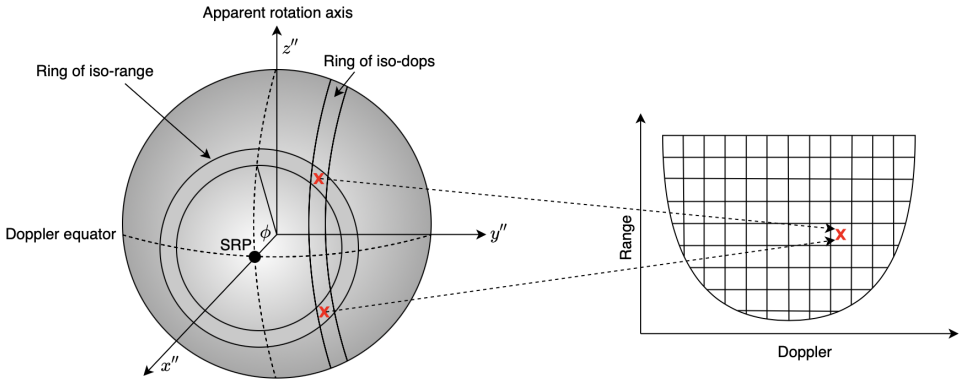


Figure 1.2: Illustration of the range-Doppler geometry during a Moon measurement by a monostatic ISAR system, as well as the Doppler north/south mapping ambiguity. The figure is adapted from Bruce Campbell et al. (2007) [4].

The geometry involved with a monostatic ISAR measurement of the Moon is illustrated in Figure 1.2. Due to relative motion at the time of the observation an apparent coordinate system is used. The apparent z''-axis corresponds to the apparent rotation axis, thus does not necessary points towards selenographic north. The apparent y''-axis is oriented parallel to the apparent Doppler equator, perpendicular to the apparent rotation axis. The apparent x''-axis points towards the radar, and the sub-radar point (SRP) is the intersection between

the target surface and the vector between the radar and the target center [19]. Positive Doppler shift corresponds to a surface region moving towards the radar, while negative Doppler shift corresponds to a region moving away from the radar. The magnitude of the Doppler shift indicates how far along the Doppler equator a surface region deviates from the sub-radar point.

As a result of measuring a spherical target by a monostatic ISAR system, there will be rings of equal range centered about the sub-radar point, as illustrated in Figure 1.2. All lunar regions within an iso-range ring have the exact same delay, thus the same range value. Because the Doppler shift enables separation in azimuthal direction, along the Doppler equator, all points located at the same apparent y''-coordinate have the same measured Doppler shift. Due to separation of resolution cells in range and Doppler coordinates, measurements retrieved by a monostatic ISAR configuration have the inherent problem that any given combination of range and Doppler shift in a measurement will map to two different points on the Moon [4]. This ambiguity is shown in Figure 1.2 causing symmetry about the Doppler equator as the range-Doppler map is converted to a selenographic coordinate system.

The north-south mapping ambiguity has previously been avoided by the use of an interferometric SAR (InSAR) [17] or selective illumination [4]. An InSAR system measuring a target object by two antennas, separated along the targets rotation axis, results in a phase difference between signals scattered from the two Doppler hemispheres. The phase difference enables separation along the rotation axis, but requires a multi-static radar configuration. Selective illumination involves narrowing the antenna beamwidth such that only one of the two Doppler hemispheres is illuminated at a time. For ground based lunar measurements, this technique is challenging. The lunar surface covers only half a degree on the sky. Illuminating one hemisphere at a time, covering only quarter a degree, would require a very large antenna. To my knowledge, there are no radar antennas capable of serving this purpose after the collapse of the Arecibo telescope in 2020 [1].

This thesis seeks to develop a program able to rectify the ambiguity associated with monostatic ISAR measurements of the Moon. The concept behind this method involves simulating multiple range-Doppler maps with different rotation axes in such a way that the true scattering cross-sections can be estimated by solving an overdetermined linear least square system of the different maps.

2 Background

The essential background theory will be presented in this chapter. The theory includes the range-Doppler geometry of a rotating sphere, which is essential in order to derive the resolution obtained by a monostatic ISAR configuration. In addition, the incidence angle dependent backscattering, range dependent power reduction and a selection of types of noise, and common geometric effects in radar mapping, are discussed.

2.1 Inverse synthetic aperture radar resolution

The resolution obtained by a monostatic inverse synthetic aperture radar (ISAR) system can be divided into two categories. Range resolution determines the resolution extending along the radar look direction, while the azimuthal resolution determines the resolution along the apparent Doppler equator.

Prior to the development of the ISAR system technique, a real-aperture radar (RAR) system limited the azimuth resolution, Δw, by the physical antenna's angular beamwidth, ψ, and range, R, as follows [2]:

$$\Delta w = R\psi \quad , \psi \propto \frac{\lambda_{\text{radar}}}{L_{\text{eff}}} \tag{2.1}$$

In equation 2.1, R is the range between the radar and the target, λ_{radar} is the radar wavelength and L_{eff} is the effective antenna length. The azimuthal

resolution would improve by reducing the distance, R, or by increasing the effective antenna length.

The frequency resolution associated with an ISAR system is neither limited by the range nor the angular beamwidth of the radar antenna, but by the observation period, and can be expressed as follows [2]:

$$f_{\text{res}} = \frac{1}{T_c} \tag{2.2}$$

In equation 2.2, T_c is the observation time. Longer observation time will result in higher frequency resolution. For a rotating target, the frequency shift at some radial distance Δy, from the Doppler axis can, for a monostatic configuration, be derived from the round-trip Doppler shift as follows:

$$\Delta f = \frac{2\Delta V}{c} f_0 = \frac{2\Delta V}{c} \frac{c}{\lambda_{\text{radar}}} = \frac{2\Delta V}{\lambda_{\text{radar}}} \tag{2.3}$$

In equation 2.3 Δf is the Doppler frequency shift, ΔV is the relative velocity between the radar and target, f_0 is the radar frequency, c is the speed of light and λ_{radar} is the radar wavelength. The relative velocity can be expressed as how rapidly a point on the spherical surface rotates over an azimuthal distance Δy as follows:

$$\Delta V = \frac{2\pi \Delta y}{T_p} = 2\pi f_p \Delta y \tag{2.4}$$

In equation 2.4, f_p is the apparent rotation rate in hertz and T_p is the rotation period. By inserting equation 2.4 into equation 2.3, an expression for the Doppler shift at some radial distance, $\Delta y = 0 - y''$, is obtain as follows [2]:

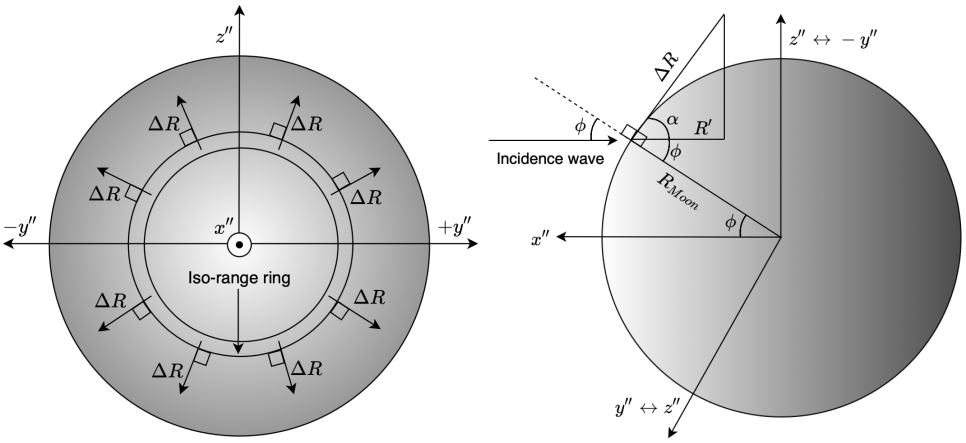
$$\Delta f = \frac{4\pi f_p \Delta y}{\lambda_{\text{radar}}} \tag{2.5}$$

The rotation rate of planetary objects varies over time. Higher rotation rate will result in improved resolution due to a larger Doppler spread. The Doppler shift is zero at the sub-radar point where Δy is zero and will increase in magnitude for increasing deviation from the sub-radar point.

Range resolution may be divided into slant range resolution, R', and ground range resolution, ΔR.The slant range , R', extends along the radar look direction and is defined as the distance the electromagnetic wave travels over half an effective pulse length, τ_p, as follows [2]:

$$R' = \frac{c\tau_p}{2} \tag{2.6}$$

Shorter effective pulse lengths will improve the slant range resolution.



(a) Ground range resolution. **(b)** Range resolution geometry.

Figure 2.1: Illustration of the range resolution geometry. The radar points along the negative x''-axis, into the paper, and the slant range extends along the radar look direction, thus along negative x''-axis. The ground range resolution, ΔR, is the projected slant range onto the surface radially outwards from the iso-range rings. The $y'' - z''$ -plane may be rotated about the apparent x''-axis depending upon at which arbitrary point along the iso-range ring the ground range resolution is calculated. The angle ϕ is the incidence angle, α is $\pi/2 - \phi$ and ϑ is the longitude of that arbitrary point. For the case of a non-tilted $y'' - z''$ -plane, the incidence angle becomes the apparent latitude.

The ground range resolution, ΔR, is a projection of the slant range, R', onto the surface, radially outwards from the iso-range rings as illustrated in Figure 2.1. Since the Moon is assumed to be a spherical target, the ground range resolution is difficult to calculate exactly. An estimate for ΔR is therefore derived from Figure 2.1b as a straight line along the surface as follows:

$$\cos \alpha = \frac{R'}{\Delta R} \tag{2.7}$$

Since $\alpha = \pi/2 - \phi$, equation 2.7 can be redefined as follows:

$$\cos\left(\frac{\pi}{2} - \phi\right) = \sin|\phi| = \frac{R'}{\Delta R} \tag{2.8}$$

Inserting the equation for slant range, R' into equation 2.8 an expression for ground range, ΔR, can be obtained as a function of incidence angle ϕ [2]:

$$\Delta R = \frac{c\tau_p}{2\sin|\phi|} \tag{2.9}$$

Because range resolution is equal for both positive and negative incidence angles, thus equal both above and below the apparent Doppler equator, equation

2.9 considers the absolute value of the incidence angle. For incidence angles equal to zero, at the sub-radar point, ΔR becomes infinitely large, causing significantly low resolution. The ground range resolution will increase for increasing incidence angles.

The area of each range-Doppler resolution cell can be calculated as the product of the range resolution along the apparent x''-axis and the Doppler shift along the apparent y''-axis as follows [2]:

$$\Delta A = \Delta x \Delta y \tag{2.10}$$

Δy can be found from equation 2.5 as follows:

$$\Delta y = \frac{\lambda_{\text{radar}}}{4\pi T_c f_p} \tag{2.11}$$

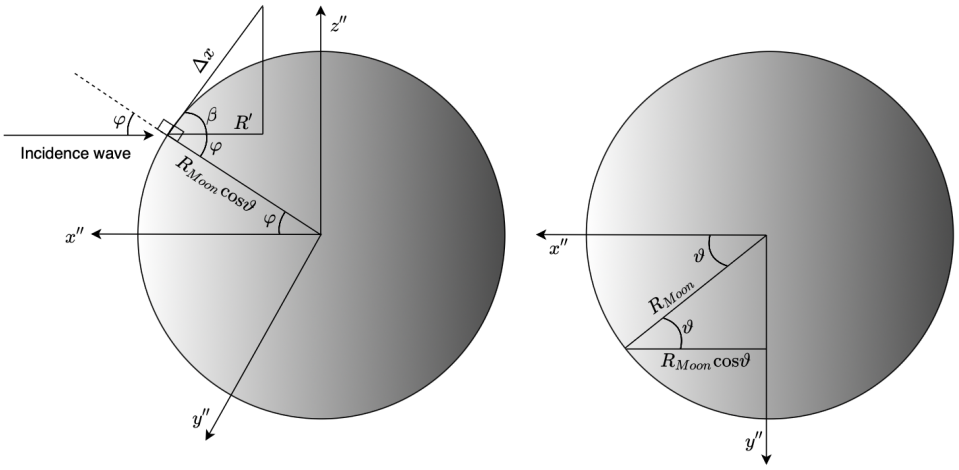


Figure 2.2: Δx is the projected ground range along the apparent x''-axis. The y''-and z''-axes are invariable, unlike in Figure 2.1, but the radial distance between the lunar center and the surface point can slide along the apparent y''-axis depending upon the longitude, ϑ, as shown in the rightmost part of the figure. The angle φ is the latitude and β is $\pi/2 - \varphi$.

Δx can be found as the ground range projected along the apparent x''-axis as illustrated in Figure 2.2, and can be derived as follows:

$$\begin{aligned} \cos\beta = \cos\left(\frac{\pi}{2} - \varphi\right) = \sin|\varphi| = \frac{R'}{\Delta x} \\ \Rightarrow \Delta x = \frac{R'}{\sin|\varphi|} \\ \Rightarrow \Delta x = \frac{c\tau_p}{2\sin|\varphi|} \end{aligned} \tag{2.12}$$

Since Δx is equal both above and below the apparent Doppler equator the equation 2.12 considers the absolute sign of the latitude.

The area of a resolution cell is found by inserting Δy and Δx from equation 2.11 and 2.12 into equation 2.10 as follows [2]:

$$\begin{aligned} \Delta A &= \Delta x \Delta y \\ \Rightarrow \Delta A &= \frac{c\tau_p}{2\sin|\varphi|} \frac{\lambda_{\text{radar}}}{4\pi T_c f_p} \\ \Rightarrow \Delta A &= \frac{c\tau_p \lambda_{\text{radar}}}{8\pi T_c \sin|\varphi| f_p} \end{aligned} \tag{2.13}$$

For apparent latitude equal to zero, the area of a resolution cell becomes infinitely large causing low resolution along the apparent Doppler equator. The resolution will increase for increasing apparent latitudes.

2.2 Effects in radar mapping

As a target is observed by a radar, the ratio of transmitted power relative to received power will depend upon numerous effects, including the distance between the radar and target, local incidence angle, roughness, and surface properties. The surface properties are usually divided into two groups, dielectric, and conductive properties. Highly dielectric materials, like terrestrial seawater with an average real dielectric constant of 80 [2], have the feature of causing strong power returns. This feature is due to a low loss tangent for such materials and thereby low signal attenuation. Conductive materials have the opposite property, causing low power returns due to a high loss tangent and thereby high signal attenuation. In this section, a selection of effects in radar mapping will be discussed.

2.2.1 Incidence angle dependent backscattering

As the electromagnetic wave interacts with the surface, the orientation of the main lobe of the angular scattering pattern depends upon the angle between the incidence wave and the local surface slope. Assuming that the target is a perfect sphere, the angle between the incoming wave and the surface is the incidence angle.

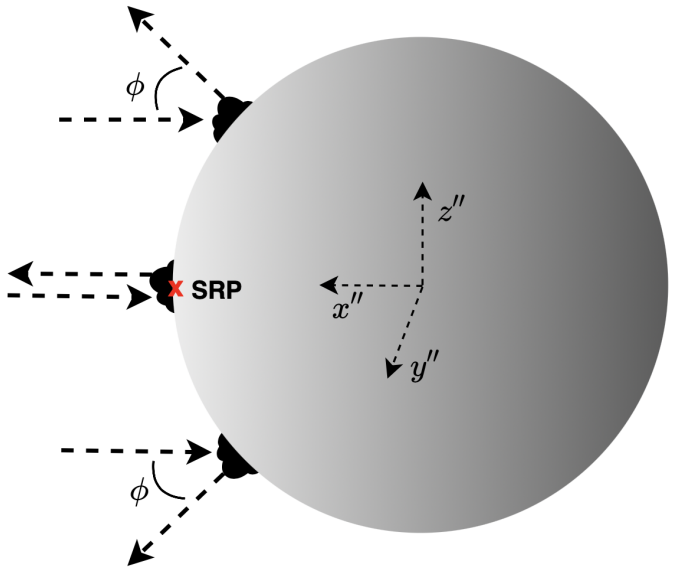


Figure 2.3: Illustration of how the backscattered signal is dependent upon the incidence angle. The arrows pointing inwards represents the incoming wave and the arrows pointing outwards represent the scattered wave pointing in the direction according to the main lobe of the angular scattering pattern. The red cross labeled SRP corresponds to the sub-radar point.

Assuming a gently undulating surface, where the horizontal scales are much longer than the radar wavelength, a model for the power reduction has been developed. Hagfors scattering law defines the reduction in backscattered power as a function of incidence angle, ϕ, as follows [2]:

$$\sigma^0(\phi) = \frac{C\rho_0}{2}\Big(\cos^4(\phi) + C\sin^2(\phi)\Big)^{-3/2} \tag{2.14}$$

In equation 2.14, σ^0 is the backscatter coefficient, ϕ is the incidence angle, C is the inverse square of the root mean square slope, and ρ_0 is the Fresnel normal reflectivity. Because the scattering law contains both ρ_0 and C, it also considers some material properties of the surface in addition to the incidence angle [2].

2.2.2 Range dependent power reduction

As an electromagnetic wave is transmitted by an isotropic antenna, the wave propagates radially outwards as a sphere with radius, R, equivalent to the distance of propagation. The signal power will be distributed over the sphere,

thus be reduced proportional to the spherical area as follows [14]:

$$P_{\text{at target}} \propto \frac{P_{\text{transmitted}}}{4\pi R^2} \tag{2.15}$$

This reduction in power as a function of distance will occur during the backscattering as well. Assuming a monostatic radar configuration, where the distance between the transmitting antenna and target equals the distance between the target and the receiving antenna, the round-trip signal power reduction can be expressed as follows [14]:

$$P_{\text{received}} \propto \frac{P_{\text{transmitted}}}{16\pi^2 R^4} \tag{2.16}$$

The case of an isotropic antenna is not feasible. The signal power is, in reality, not equally radiated in all directions. Despite the fact that an antenna is always somewhat directed, the reduction of signal power is still proportional to the area of a sphere as described in equation 2.15, but a gain factor, G, is introduced to scale the power in the main radiation direction. Antenna gain, G, is a measure of how much power an antenna transmits in the direction of most radiation relative to an isotropic antenna [15].

2.2.3 Noise

Random surface undulations

As a target is observed by an ISAR system, each range-Doppler measurement of a resolution cell is the sum of numerous individual scatters, ξ, from within that surface region [2]. These scatters are a result of random undulations on the target surface and subsurface. The measurements of these scatters can be modeled as proper complex normal random variables [18] as follows:

$$\xi \sim N_{\mathbb{C}}\left(0, \sigma^2_{\text{Moon}}\right) \tag{2.17}$$

By assuming a sinusoidal signal, the mean voltage is zero. The variance equals the variance of the lunar surface, σ^2_{Moon}. The received signal from a range-Doppler cell is in fact the absolute value squared of a proper complex normal random variable,

$$\left\langle \xi \xi^* \right\rangle = \sigma^2_{\text{Moon}} \tag{2.18}$$

A way of estimating σ^2_{Moon} can therefore be done by calculating the product of a complex normal random variable $\xi = x + iy$ and its complex conjugate $\xi^* = x - iy$, where x, y are both real normal random variables as follows:

$$x, y \sim N\left(0, \sigma^2\right), \quad \left\langle x^2 \right\rangle = \sigma^2, \quad \left\langle y^2 \right\rangle = \sigma^2 \tag{2.19}$$

The mean of these two distributions is zero and the variance, σ^2, can be found as follows:

$$\begin{aligned} \left\langle \xi\xi^* \right\rangle &= \left\langle x^2 + 2ixy + y^2 \right\rangle = \left\langle x^2 \right\rangle + \left\langle 2ixy \right\rangle + \left\langle y^2 \right\rangle \\ \left\langle 2ixy \right\rangle &= 0 \quad \text{, Since} \quad \left\langle \operatorname{Re}(\xi) \operatorname{Im}(\xi) \right\rangle = 0 \\ \sigma^2_{\text{Moon}} &= \left\langle x^2 \right\rangle + \left\langle y^2 \right\rangle = 2\sigma^2 \\ \Rightarrow \sigma^2 &= \frac{\sigma^2_{\text{Moon}}}{2} \end{aligned} \tag{2.20}$$

Thermal noise

Due to the electrical system in an antenna, heat is generated. This heat cause particle motion in the circuits. Electrically charged particles in motion induce currents and additional voltage in the signal resulting in noise. The noise power associated with thermal motion is given by Johnson-Nyquist as follows [7]:

$$N = k_B T B \tag{2.21}$$

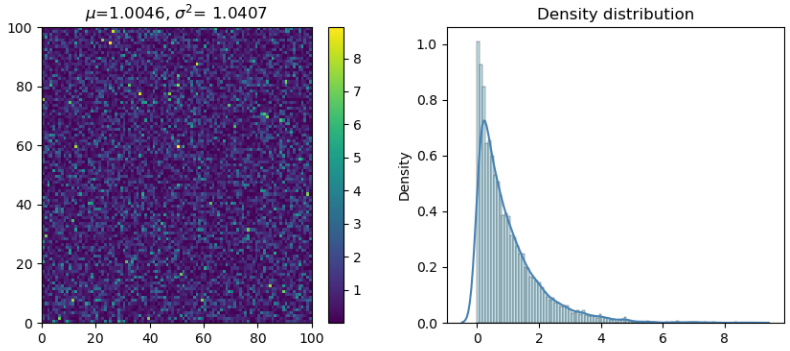
In equation 2.21, k_B is the Boltzmann constant, T is the temperature in Kelvin and B is the frequency bandwidth in hertz. The noise temperature present in an ISAR measurement of the Moon would include the Moon temperature, radar operating temperature as well as cosmic temperature. Typically, the input noise will dominate the noise caused by the receiver temperature [20]. Particle motion is unpredictable in nature and the thermal motion can therefore be assumed to be a random process. This random process can be modeled as a random Gaussian distribution with zero mean. The variance of the distribution can be found in the same way as in equation 2.20.

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad \text{where} \quad \sigma^2 = \frac{(\mathrm{k_B TB})^2}{2} \tag{2.22}$$

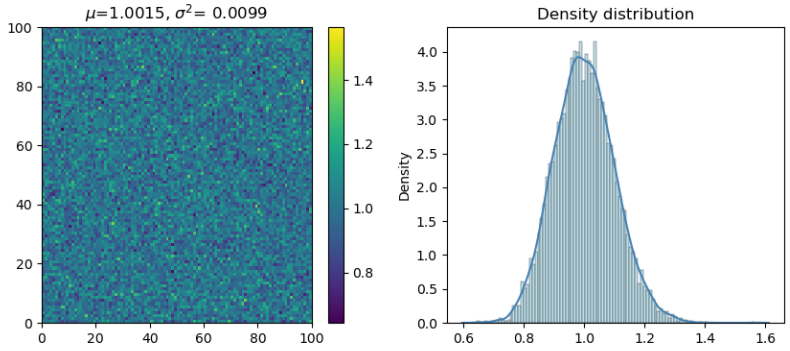
Speckle

As previously stated, each radar measurement is assumed to be the sum of complex random variables. The probability density function of measured power, including noise, follows approximately a chi-squared distribution as shown in the rightmost part of Figure 2.4a. Measurements in the far right of the distribution cause bright spots, and those in the far left part cause dark spots. This noise phenomenon is termed speckle and is illustrated in the leftmost part of Figure 2.4a. Because the chi-squared distribution has a heavy tail, the presence of bright spots are likely to be dominant in the radar image. In order to reduce the effect of speckle, several measurements must be averaged together. The effect of speckle is reduced proportionally to the square root of the number of independent measurements averaged together as $\sqrt{N}$ [2]. As the number

of measurements, N, increases, the variance decreases, and the chi squared distribution approaches a Gaussian distribution as shown in the rightmost part of Figure 2.4b. This reduction in variance results in less speckle as shown in the leftmost part of Figure 2.4b



(a) Illustration of one range-Doppler measurement containing 100×100 scatters. The variance, σ^2, is in this case 1.0407. The presence of speckle can be seen as the bright spots in the measurement. The density distribution follows approximately a chi-squared distribution.

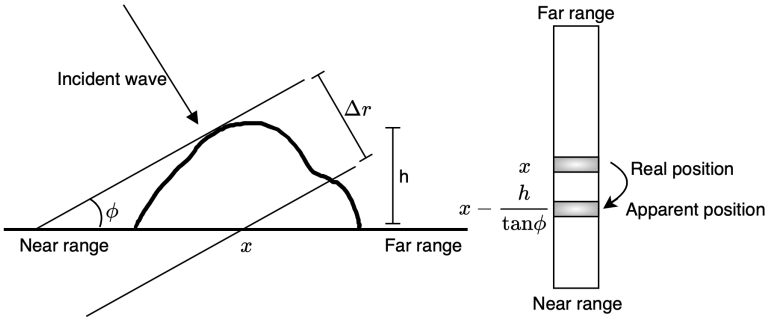


(b) Illustration of a range-Doppler measurement when 100 measurements are averaged together. The variance σ^2 is in this case 0.0099. The density distribution follows approximately a Gaussian distribution.

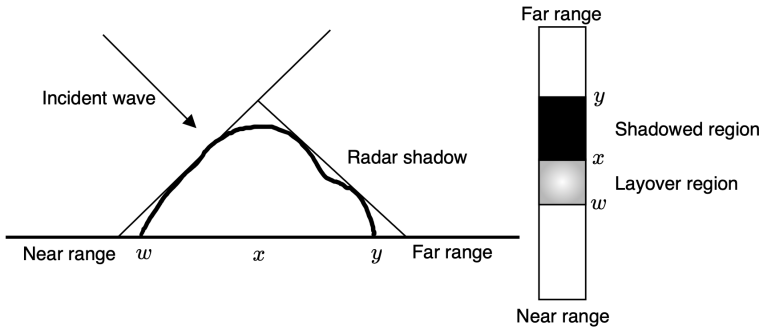
Figure 2.4: Illustration of speckle reduction by averaging 100 measurements together.

2.2.4 Geometric mapping effects

The mapping into spatial coordinates usually contains an assumption that the surface is either flat or spherical. Thus, real surface topography with varying elevation exceeding beyond the surface assumptions may cause errors in the radar mapping [2].



(a) Illustration of radar parallax. The point x appears to be more near in range than its real position.



(b) Illustration of radar shadowing and layover for an assumed to be plane surface with vertical topography variations [2].

Figure 2.5: Illustration of geometric effects in radar mapping. The figures are adapted from Bruce A. Campbell [2].

One possible error is radar parallax shown in Figure 4.3a. Since range is determined by time delay between the target and receiver, the echoes from the top of the illustrated surface structure appear to be closer to the receiver than echoes from point x. This will cause an error in topography estimation [2]. Another geometric effect is layover shown in Figure 4.3b. If the local incidence angle is comparable to the slope of the surface topography, the echoes from the surface slope will be received at approximately the same time. This will cause overlap of the radar returns, making it impossible to distinguish the different structures within this region. [2]. These effects are only noticeable if they affect areas exceeding the spatial resolution in such a way that surface features relocate into an adjacent resolution cell. Lastly, if a surface slope is oriented 90° or more to the incident wave, radar shadowing will occur. This phenomenon is shown in Figure 4.3b, causing no received echo from the region between x and y [2]. These effects are especially significant at regions dominated by craters where the elevation changes substantially for relatively small horizontal scales.

3 Method

In this chapter, the method of both simulating and disambiguating range-Doppler radar maps will be discussed. The data used in the simulation were retrieved from NASA's Horizon ephemeris [13] and include the selenographic sub-radar points, elevation, and range to the center of the Moon. The ephemeris data was generated from the 13^{th} to 15^{th} of February 2022 with a 10-minute timestep. The observation site was specified at Skibotn, Norway. In the simulation a radar wavelength of 1.6 meters was used. Due to extensive computational time, the observation period, T_c, was set to 50-seconds and the effective pulse length, τ_p, was set to 10-microseconds. Reflectivity measurements of the lunar surface at optical wavelengths, retrieved from the Lunar Reconnaissance Orbiter Camera [6], were used as a model for surface scattering cross-sections, and is shown in Figure 3.1. The method of simulating range-Doppler radar maps involved creating a range-Doppler grid based on the observation time and effective pulse length. This grid was converted to apparent Cartesian coordinates, and rotated and mapped to selenographic coordinates. The selenographic coordinate grid was used to sample reflectivity measurements from the optical lunar mosaic, symmetrically about the apparent Doppler equator at the time of the observation. In order to simulate realistic maps, noise due to random undulations was considered. In addition, effects such as the Hagfors scattering law, range dependent power reduction and the area of the resolution cells were taken into account. The importance of including these effects lies in the functionality of the program removing them. The thermal noise was, in this case, disregarded for simplicity.

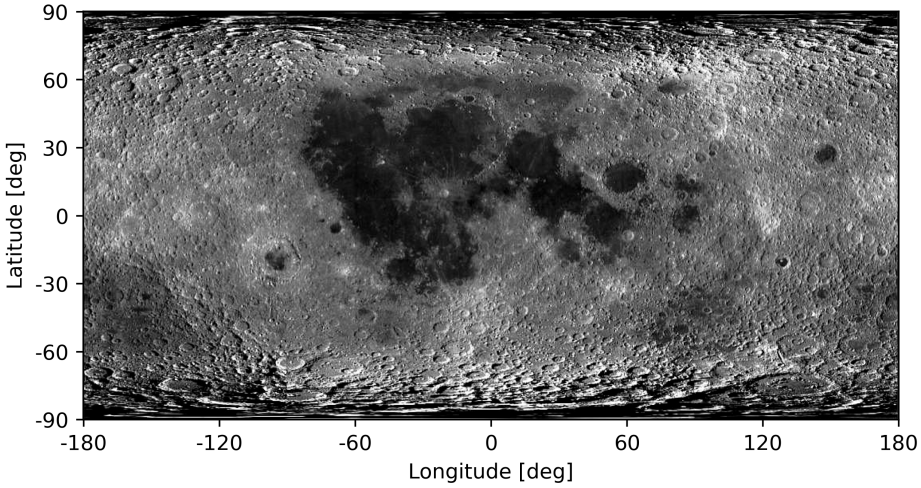


Figure 3.1: The optical lunar mosaic from the Lunar Reconnaissance Orbiter Camera [6]. The mosaic has been exposed to increased contrast to enlighten the different surface structures.

The disambiguation procedure was conducted as an overdetermined linear least squares system of the simulated range-Doppler maps. In order to establish an overdetermined system, a minimum of three maps at three different rotation axes had to be simulated.

3.1 Simulation

An essential part of the simulation is the mapping between range-Doppler coordinates and apparent selenographic coordinates. In order to avoid handling a continuously changing coordinate system, an assumption was made stating a constant rotation axis during the observation period. This assumption is within reason considering that the apparent rotation axis of the Moon change relatively slowly. As a result of this assumption, the apparent lunar rotation rate could be estimated from the ephemeris data as follows:

$$f_p = \frac{\delta}{2\pi R_{Moon} T} \tag{3.1}$$

I equation 3.1, f_p is the rotation rate in hertz, δ is the distance between two adjacent sub-radar points and T is the time between the measurements. The chosen sub-radar point intervals used in this simulation are shown in Figure 3.2 and their associated rotation rates, found by this approach, are given in table 3.1.

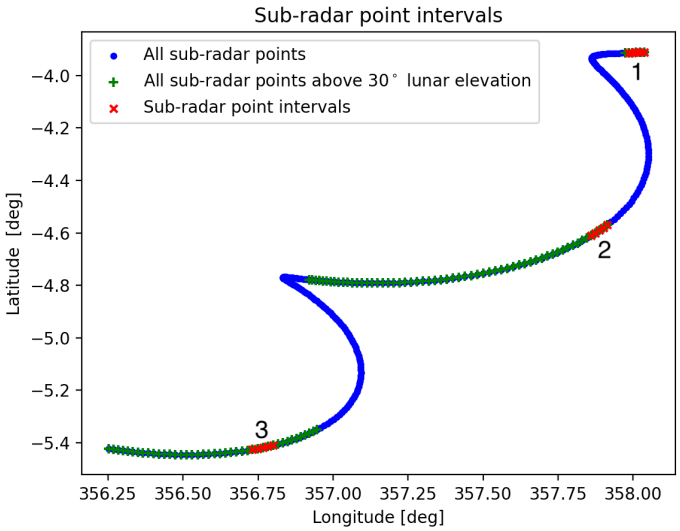


Figure 3.2: Plot of the selenographic sub-radar points from the ephemeris data. The blue dots correspond to all sub-radar points in the ephemeris data. The green plus-signs correspond to the sub-radar points when the lunar elevation exceeded 30 degrees, thus in this case assumed to be visible to an Earth based radar. The red crosses labeled 1,2, and 3 are the three chosen sub-radar point intervals used in the simulation.

Table 3.1: The rotation rates associated with the three sub-radar point intervals in Figure 3.2.

Interval 1	$2.43 \cdot 10^{-6}$ [Hz]
Interval 2	$3.04 \cdot 10^{-6}$ [Hz]
Interval 3	$3.17 \cdot 10^{-6}$ [Hz]

The transformation of range-Doppler coordinates to apparent Cartesian coordinates shown in Figure 1.2, was calculated as follows:

$$x'' = \frac{R_{Moon} - r}{R_{Moon}} \tag{3.2}$$

$$y'' = -\frac{d\lambda_{\text{radar}}}{4\pi R_{Moon} f_p} \tag{3.3}$$

$$z'' = \sqrt{|R_{Moon}^2 - x''^2 - y''^2|} \tag{3.4}$$

In equation 3.2 and 3.3, r is the range value from the sub-radar point and d is the Doppler value.

The apparent selenographic coordinates were found as follows:

$$\vartheta = \sin^{-1}\left[\frac{y''}{R_{Moon}}\right] \tag{3.5}$$

$$\varphi = \sin^{-1}\left[\frac{z''}{R_{Moon}}\right] \tag{3.6}$$

In equation 3.5 and 3.6, ϑ and φ are the selenographic longitude and latitude respectively.

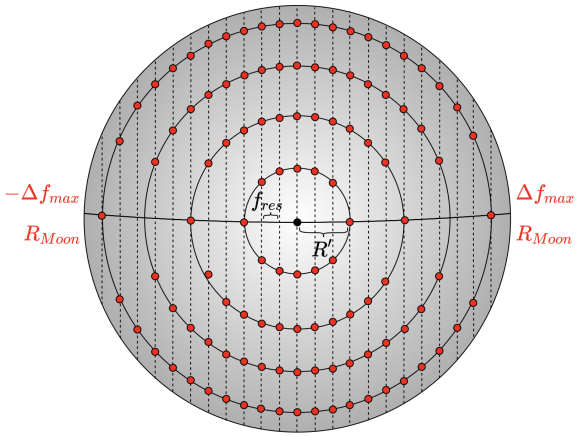


Figure 3.3: Illustration of the simulated range-Doppler coordinate grid. The distance between each vertical dashed line, corresponds to the frequency resolution step, f_{res}. The distance between each iso-range ring corresponds to the slant range resolution R'. The red dots correspond to the range-Doppler coordinates.

The range coordinates of the iso-range rings were estimated as equally spaced values between zero range at the sub-radar point, and the lunar radius at the limbs as shown in Figure 3.3. The spacing between each range coordinate corresponds to the slant range resolution given in equation 2.6 as follows:

$$r = 0, R', 2R', \ldots, R_{Moon} \tag{3.7}$$

The maximum Doppler frequency shift, Δf_{max}, is found at the limb and calculated from equation 2.5 with $\Delta y = R_{Moon}$. The Doppler frequencies will be evenly distributed along the Doppler equator from zero Doppler shift at the sub-radar point to $\pm\Delta f_{max}$ at the limbs as shown in Figure 3.3. The frequency step equals the frequency resolution, f_{res}, given in equation 2.2. The maximum Doppler shift at a given iso-range ring, r, was found as follows:

$$d_{\max}(r) = \Delta f_{\max} y''(r) \quad \text{where} \quad y''(r) = \sqrt{R^2_{\text{Moon}} - x''(r)^2} \tag{3.8}$$

Each iso-range ring, r, is symmetric around the sub-radar point such that all Doppler values for an iso-range ring, r, could be found as follows:

$$d(r) = -d_{max}(r), \ldots, -2f_{res}, -f_{res}, 0, f_{res}, 2f_{res}, \ldots, d_{max}(r) \tag{3.9}$$

As the range-Doppler coordinate grid was found, the sampling of reflectivity measurements from the optical lunar mosaic could be performed. To account for the noise due to random surface undulations, each range-Doppler coordinate was sampled as a $N \times N$ sub-grid shown in Figure 3.4.

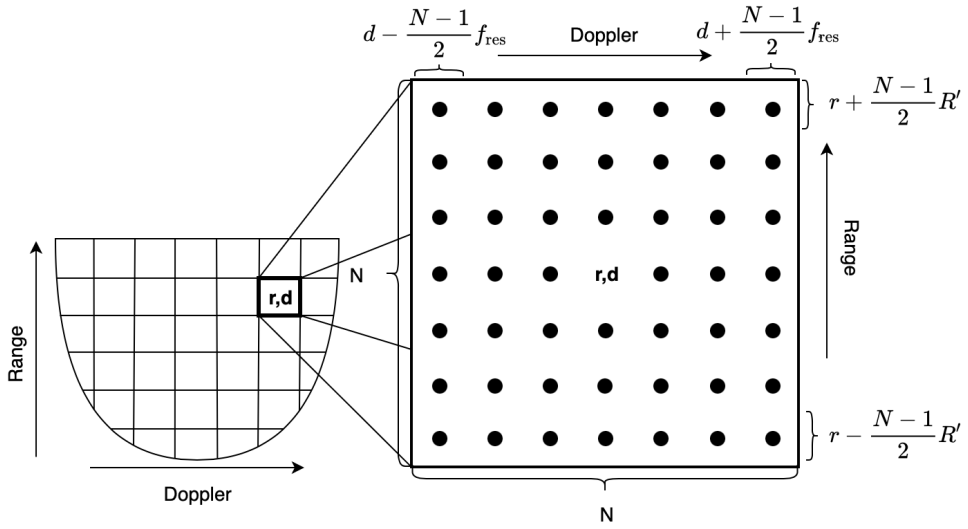


Figure 3.4: The generated sub-grid for each simulated range-Doppler coordinate. The sub-grid ranged from $r - \frac{N-1}{2}R'$ to $r + \frac{N-1}{2}R'$ in range and $d - \frac{N-1}{2}f_{\text{res}}$ to $d + \frac{N-1}{2}f_{\text{res}}$ in Doppler.

The sub-grid was generated with range values from $r - \frac{N-1}{2}R'$ to $r + \frac{N-1}{2}R'$ and Doppler values from $d - \frac{N-1}{2}f_{\text{res}}$ to $d + \frac{N-1}{2}f_{\text{res}}$ for each simulated range-Doppler coordinate (r, d).

In order to sample the optical mosaic, the coordinates within these grids were converted to selenographic coordinates. This conversion was done by three-dimensional rotation matrices with rotation angles corresponding to the opposite rotation of what is required to rotate the sub-radar point to zero selenographic longitude and latitude and the apparent rotation axis to the selenographic z-axis. The rotation matrices are given as follows:

$$\boldsymbol{R}_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{3.10}$$

$$\boldsymbol{R}_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{3.11}$$

$$\boldsymbol{R}_z(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.12}$$

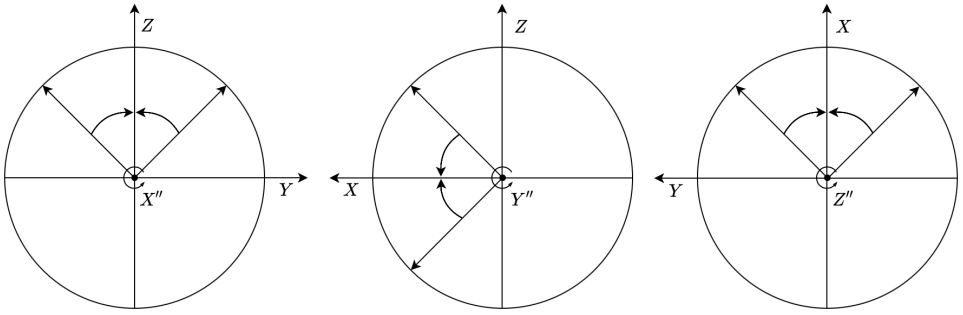


Figure 3.5: Rotation of the coordinates to true Cartesian coordinates. The rotation about the apparent x''-axis is shown in the leftmost part of the figure where the apparent x''-axis point out of the paper. The rotation about the apparent y''-axis is shown in the middle part of the figure where the apparent y''-axis point out of the paper. Lastly, the rotation about the apparent z''-axis is shown in the rightmost part of the figure where the apparent z''-axis point out of the paper. Positive rotation for all coordinate systems occurs counterclockwise.

In order to find the required rotation about the apparent x''-axis, the apparent rotation axis, $\vec{v}''_{\text{rot}}$, was found as the cross product between the first sub-radar point and the normalized distance to an adjacent sub-radar point in the sub-radar point interval as follows:

$$\vec{v}_{\text{rot}} = \vec{v}_{\text{SRP1}} \times \frac{\vec{v}_{\text{SRP1}} - \vec{v}_{\text{SRP2}}}{||\vec{v}_{\text{SRP1}} - \vec{v}_{\text{SRP2}}||} \tag{3.13}$$

The rotation angle about the apparent x''-axis was then found as the angle between the rotation axis, found by equation 3.13, and the selenographic z-axis in the $y-z$-plane as follows:

$$\theta_x = \text{sign}(\text{y}''_{\vec{v}_{\text{rot}}})\cos^{-1}\left(\frac{[0,1]\cdot[\text{y}''_{\vec{v}_{\text{rot}}}, \text{z}''_{\vec{v}_{\text{rot}}}]}{||[0,1]||\ ||[\text{y}''_{\vec{v}_{\text{rot}}}, \text{z}''_{\vec{v}_{\text{rot}}}]||}\right) \tag{3.14}$$

Multiplication with the sign of the y-coordinate of the apparent rotation axis results in positive rotation about the apparent x''-axis if the y-coordinate is

positive, and negative rotation if the y-coordinate is negative as shown in leftmost part of Figure 3.5. The rotation angle about the apparent y''-axis was found as the angle between the sub-radar point vector and the selenographic x-axis in the $x-z$ plane as follows:

$$\theta_y = \text{sign}(\text{z}''_{\vec{v}_{\text{SRP1}}})\cos^{-1}\left(\frac{[1,0]\cdot[\text{x}''_{\vec{v}_{\text{SRP1}}},\text{z}''_{\vec{v}_{\text{SRP1}}}]}{||[1,0]||\;||[\text{x}''_{\vec{v}_{\text{SRP1}}},\text{z}''_{\vec{v}_{\text{SRP1}}}]||}\right) \tag{3.15}$$

Multiplication with the sign of the z-coordinate of the sub-radar point cause positive rotation about the apparent y''-axis if the z-coordinate is positive, and negative rotation if the z-coordinate is negative as shown in middle part of figure 3.5. The rotation angle about the apparent z''-axis was found as the angle between the sub-radar point vector and the selenographic x-axis in the $x-y$ plane as follows:

$$\theta_z = -\text{sign}(\text{y}''_{\vec{v}_{\text{SRP1}}})\cos^{-1}\left(\frac{[1,0]\cdot[\text{x}''_{\vec{v}_{\text{SRP1}}},\text{y}''_{\vec{v}_{\text{SRP1}}}]}{||[1,0]||\;||[\text{x}''_{\vec{v}_{\text{SRP1}}},\text{y}''_{\vec{v}_{\text{SRP1}}}]||}\right) \tag{3.16}$$

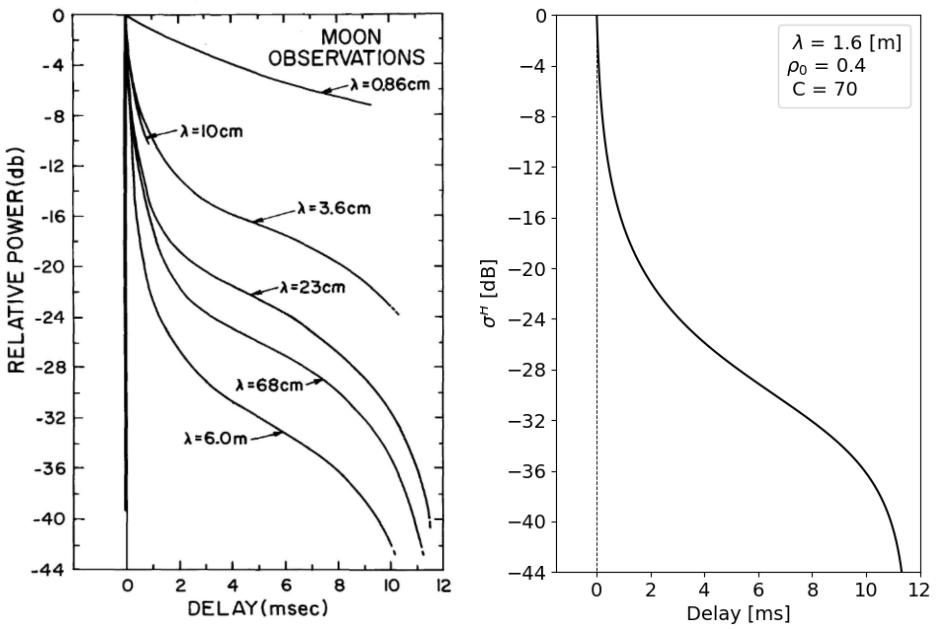
Multiplication with the negative sign of the y-coordinate of the sub-radar point cause positive rotation about the apparent z''-axis if the y-coordinate is negative, and negative rotation if the y-coordinate is positive as shown in rightmost part of Figure 3.5. As the rotation angles required to rotate the sub-radar point and the apparent rotation axis to true Cartesian coordinates were found, the coordinates, $\vec{v}''$ from the sub-grid in Figure 3.4, were rotated by the rotation matrices with the opposite angles as follows:

$$\vec{v} = \boldsymbol{R}_x(-\theta_x)\boldsymbol{R}_y(-\theta_y)\boldsymbol{R}_z(-\theta_z)\vec{v}'' \tag{3.17}$$

In equation 3.17, $\vec{v}''$ is a Cartesian coordinate from the sub-grid in Figure 3.4 and $\vec{v}$ is the true Cartesian coordinate. The selenographic coordinates were found by converting $\vec{v}$ to longitude and latitude by equation 3.5 and 3.6.

In order to simulate a realistic range-Doppler map, the radar range equation, area of each resolution cell, and Hagfors scattering law were considered. To be able to remove these effects later in the simulation, they first had to be included. The power reduction due to distance of propagation was included by dividing the sampled reflectivity by the traveled distance to the power of four as given in equation 2.16. The inverse scaling of power due to the area of each resolution cell was included by multiplying the sampled reflectivity with its associated area given in equation 2.13.

In order to include the incident angle dependent backscattering, the sampled reflectivity was multiplied with the Hagfors backscattering coefficient as given in equation 2.14. The parameters C and ρ_0 were estimated to fit real Moon measurements done by Hagfors and Evans [10] as shown in Figure 3.6.



(a) Real Moon measurements of power reduction as a function of time delay retrieved from Hagfors and Evans (1968) [10].

(b) Power reduction as a function of time delay with $C = 70$ and $\rho_0 = 0.4$ for $\lambda_{\text{radar}} = 1.6$ meters.

Figure 3.6: Estimation of the parameters C and ρ_0 in Hagfors scattering law from real Moon measurements retrieved by Hagfors and Evans [10]. For a radar wavelength equal to 1.6 meters, $C = 70$ and $\rho_0 = 0.4$ appears to be an adequate fit compared to real Moon measurements.

Each selenographic coordinate within the sub-grid represents a random surface or subsurface undulation. A real and imaginary random Gaussian distribution were therefore created for each coordinate. The variance of these Gaussian distributions was calculated as given in equation 2.20, where σ^2_{Moon} equals the sampled reflectivity. These real random variables were multiplied with their associated complex conjugates before averaged together. The resulting average value was used as the true scattering cross-section for the range-Doppler coordinate (r, d) in Figure 3.4.

As the range-Doppler map with noise was simulated, the range, scattering, and area effects were removed. This was performed by dividing the noisy scattering cross-sections with their associated Hagfors backscattering coefficient and resolution cell area, and multiplied by the distance of propagation to the power of four.

3.2 Disambiguation

A range-Doppler radar map consists of the combined power from resolution cells symmetrically located about the apparent Doppler-equator. Since the Doppler equator is dependent upon the apparent lunar rotation axis at the time of the observation, the apparent location of resolution cells with equal range and Doppler value will change over time. The result of this causes the power from each range-Doppler pixel to differ between range-Doppler maps with different apparent rotation axes as shown in Figure 3.7. Observing the Moon at the three apparent rotation axes shown in Figure 3.7, results in the range-Doppler measurements shown in Figure 3.8.

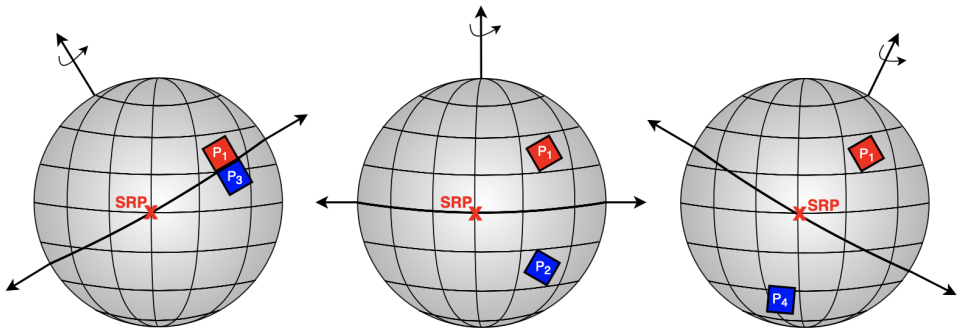


Figure 3.7: Illustration of how the north-south ambiguity affects the resolution cells at different rotation axes. A selenographic region P_1 is shown together with its range-Doppler symmetric region P_2, P_3 and P_4 at three different apparent rotation axes. The longitude and latitude grid at each sphere corresponds to the selenographic coordinate grid.

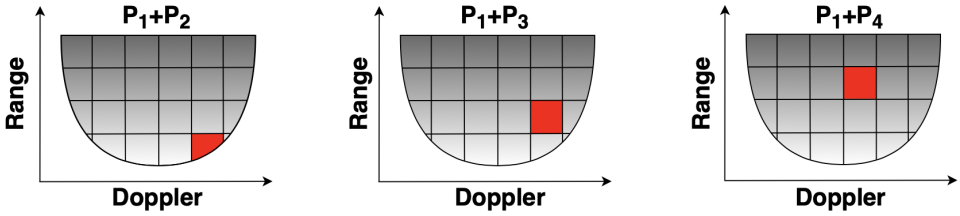


Figure 3.8: Range-Doppler maps for the different rotation axes given in Figure 3.7. The red squares correspond to the combined measurement from region P_1 and P_2 in the left map, P_1+P_3 in the middle map and P_1+P_4 in the right map, where P_1,P_2,P_3 and P_4 correspond to the resolution cells shown in Figure 3.7

The measurements from the three apparent rotation axes in Figure 3.8 can be expressed as follows:

$$\begin{aligned} \mathrm{m}_1 &= \mathrm{P}_1\sigma_1 + \mathrm{P}_2\sigma_2 \,, \\ \mathrm{m}_2 &= \mathrm{P}_1\sigma_1 + \mathrm{P}_3\sigma_3 \,, \\ \mathrm{m}_3 &= \mathrm{P}_1\sigma_1 + \mathrm{P}_4\sigma_4 \end{aligned} \tag{3.18}$$

In equation 3.18, m_1, m_2, m_3 are the three measurements marked in red in Figure 3.8. P_1,P_2,P_3,P_4 correspond to the resolution cells in Figure 3.7 and σ_1, σ_2, σ_3 and σ_4 correspond to the true scattering cross-sections within the regions P_1,P_2,P_3 and P_4.

Collecting range-Doppler measurements from multiple different maps covering mainly the same lunar area results in an overdetermined system, thus more measurements, m, than unknown true scattering cross-sections, σ. This system can be organized in a matrix format. An example of such a matrix system is shown in equation 3.19. The numbering of the pixels in equation 3.19 is randomly generated for illustration purposes.

$$\begin{bmatrix} \mathrm{m}_1 \\ \mathrm{m}_2 \\ \mathrm{m}_3 \\ \vdots \\ \mathrm{m}_M \end{bmatrix} = \begin{bmatrix} \mathrm{P}_1 & \dots & \mathrm{P}_{10} & \dots & \dots \\ \dots & \mathrm{P}_5 & \dots & \dots & \mathrm{P}_N \\ \mathrm{P}_1 & \dots & \ddots & \mathrm{P}_{17} & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \dots & \mathrm{P}_5 & \dots & \mathrm{P}_{17} & \dots \end{bmatrix} \begin{bmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \vdots \\ \sigma_N \end{bmatrix} \tag{3.19}$$

$\boldsymbol{M}$: $\mathrm{M} \times 1$ $\quad$ $\boldsymbol{A}$: $\mathrm{M} \times \mathrm{N}$ $\quad$ $\boldsymbol{X}$: $\mathrm{N} \times 1$

The first vector, $\boldsymbol{M}$, contains the M ambiguous measurements. The matrix, $\boldsymbol{A}$, contains the corresponding resolution cells which the measurements originate from. By assuming that a measurement is entirely contained in one resolution cell at each Doppler hemisphere, thus is not distributed over adjacent cells, P_1 to P_N in the $\boldsymbol{A}$ -matrix could be set to 1. Each row does therefore only contains zeros, except two values of one, representing the north and south ambiguous resolution cells. The $\boldsymbol{A}$ -matrix is organized in such a way that the column positions of the non-zero values indicate which resolution cells the measurement originates from, and correspond to the rows of the associated true scattering cross-sections, σ, in the $\boldsymbol{X}$ -vector.

The purpose of the $\boldsymbol{A}$-matrix is to estimate the true normalized scattering cross-sections, σ_1 to σ_N, in the $\boldsymbol{X}$ -vector. This estimation can be performed as an overdetermined linear least squares system follows:

$$\boldsymbol{X}_{\mathrm{LS}} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{M} \tag{3.20}$$

An issue arises because the inversion of the $\boldsymbol{A}$ -matrix is too computational heavy due to its size. By creating the $\boldsymbol{A}$ -matrix based on three ambiguous lunar maps, at the sub-radar point intervals given in Figure 3.2, with an observation period of 50s and an effective pulse length of 10μs, the $\boldsymbol{A}$ -matrix will be of size $3.42 \cdot 10^6 \times 2.97 \cdot 10^5$. An approximate function to equation 3.20 was therefore used to estimate the true scattering cross-sections as follows:

$$\boldsymbol{X}_{\text{LS}} = \text{argmin}_{\boldsymbol{X}} \, ||\boldsymbol{AX} - \boldsymbol{M}||^2 \tag{3.21}$$

The true normalized scattering cross-sections are estimated by minimizing equation 3.21 with respect to σ_1 to σ_N in the $\boldsymbol{X}$-vector.

An example will be presented to illustrate the procedure of solving the overdetermined linear least squares system. The image used in this example is shown in Figure 3.9. This is a 64x64 pixel grid of grayscale values ranging gradually from black to white.

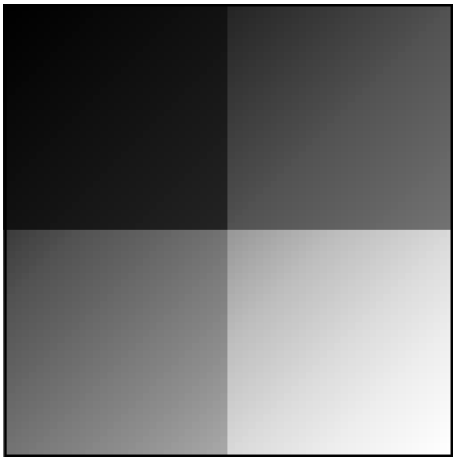
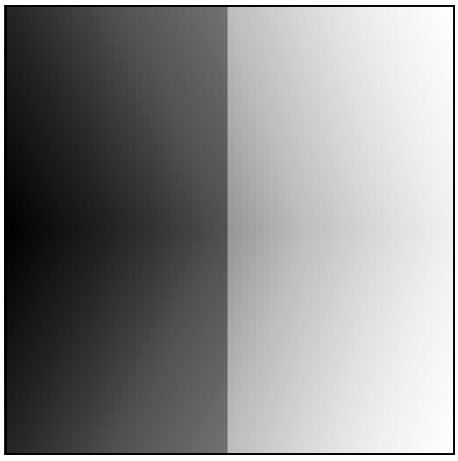
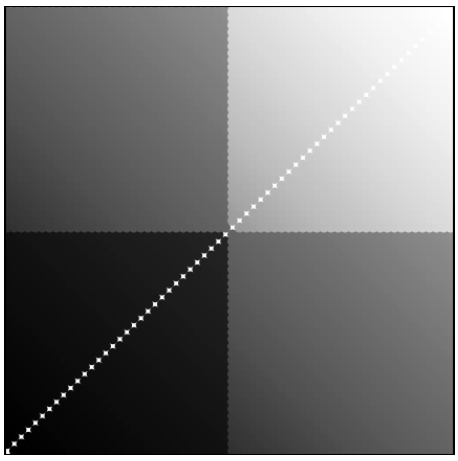


Figure 3.9: A 64x64 pixel grayscale image. The image values range gradually from black to white. The black frame surrounding the image was included after the processing to make the image visible on a white pdf-page.

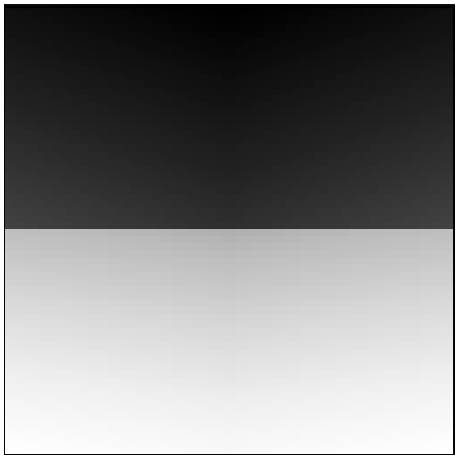
In order to simulate the ambiguity, the pixel values were averaged about three different rotation axes as shown in Figure 3.10. In Figure 3.10a the rotation axis is oriented vertically causing symmetry about the horizontal axis. In Figure 3.10b the rotation axis is oriented diagonally between the upper right corner and the lower left corner, resulting in symmetry about the opposite diagonal. Lastly, in Figure 3.10c the rotation axis is oriented horizontally causing symmetry about the vertical axis.



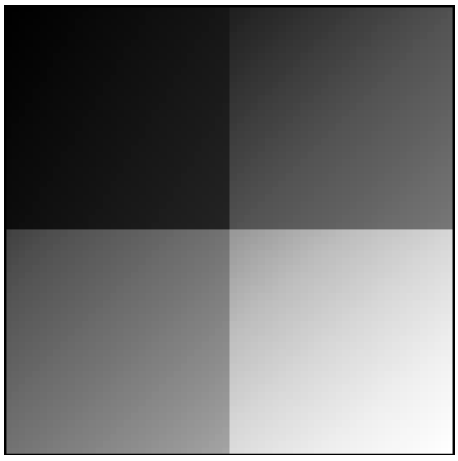
(a) Vertical rotation axis.



(b) Oblique axis of rotation.



(c) Horizontal rotation axis.



(d) The disambiguated image.

Figure 3.10: Three ambiguous test images for three different rotation axes along with the resulting disambiguated image. The black frames surrounding the images were included after the processing to make them visible on a white pdf-page.

From these three ambiguous images the measurements, in this case the greyscale values, and pixel indexes, were organized according to the $\boldsymbol{A}$-matrix illustrated in equation 3.19. As the $\boldsymbol{A}$-matrix was created, the linear least squares system in equation 3.21 was solved. The resulting disambiguated image from this example is shown in Figure 3.10d. The method was found to be unbiased and the error standard deviation of the disambiguated image relative to the original image was calculated to be $\sigma = 6.48 \cdot 10^{-3}$, indicating a successful estimation of pixel values in this case.

In this simplified example, the measurements were evenly distributed over the image. This will not be the case for real Moon observations, as the measurements are distributed depending upon the resolution and the apparent rotation at the time of the observation. In order to perform the linear least squares on ambiguous lunar measurements, the lunar surface must be divided into a fixed coordinate grid. This was accomplished by a quadtree algorithm developed by Christian Hill [5]. The quadtree algorithm divides the lunar surface into rectangular pixels. The size of each rectangular pixel depends upon the number of measurements within that pixel region. The algorithm works in such a way that each rectangle has limited capacity, meaning a limited number of measurements in each pixel. If the capacity is exceeded as the measurements are inserted into the quadtree, the pixel is divided into four sub-pixels. This leads to small pixels, thus higher resolution, at lunar regions containing a large number of measurements, and large pixels, thus low resolution, at lunar regions containing few measurements.

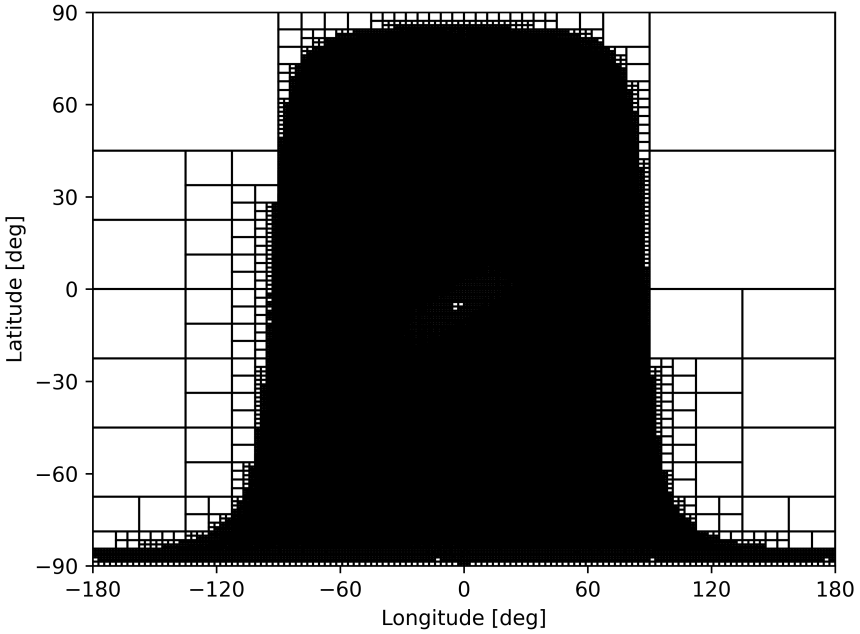


Figure 3.11: The rectangular pixel grid generated by the quadtree algorithm [5] for three selenographic maps at the three sub-radar point intervals shown in Figure 3.2.

The coordinate grid generated by this algorithm, for three simulated selenographic maps at the sub-radar point intervals in Figure 3.2, is shown in Figure 3.11. The edges surrounding the densest part of the quadtree contain large

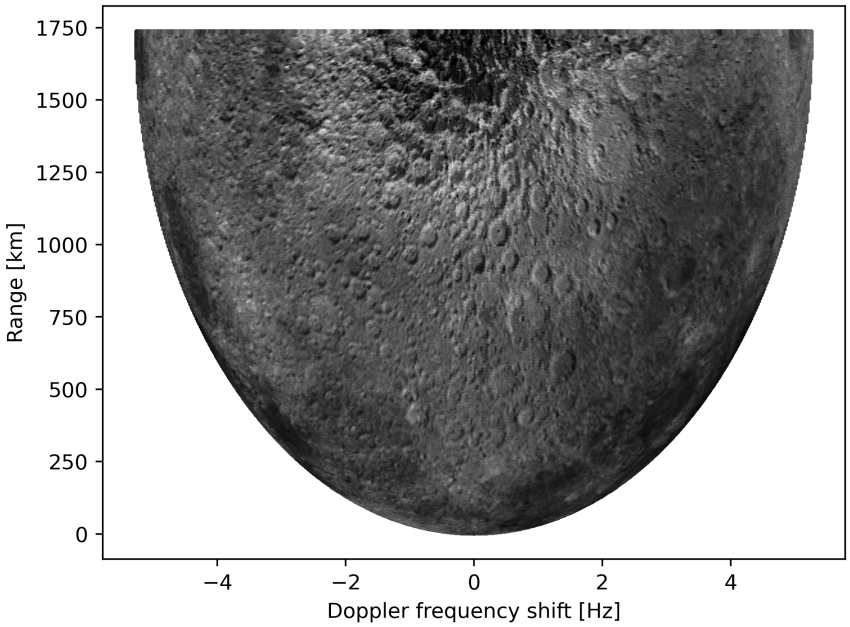
rectangles, thus suffers from low resolution. This is expected as the three maps are located about close, but different, sub-radar points, thus do not cover the exact same lunar region. The edges do only consist of measurements from a single map, making these pixels larger than the pixel containing measurements from all three maps. The sub-radar points, all a few degrees south of zero selenographic longitude and latitude, suffers from low resolution as expected.

4

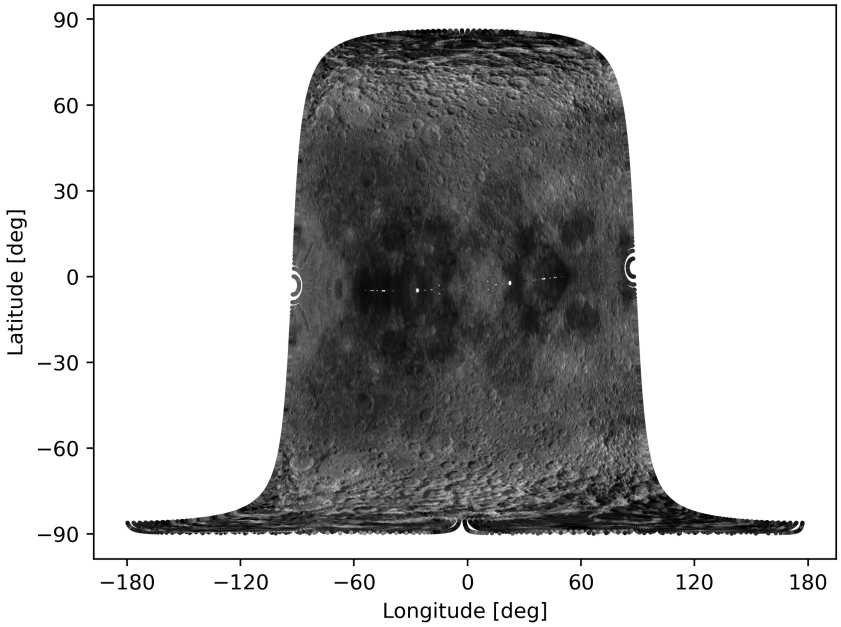
Results

The results of both the simulation and disambiguation method are presented in this chapter. The first part of the results include the three simulated range-Doppler maps as well as their associated ambiguous selenographic maps. The first set of maps, shown in Figure 4.1, correspond to the rotation axis at sub-radar point interval 1 in Figure 3.2. The apparent rotation rate at this interval is given in table 3.1 as $2.43 \cdot 10^{-6}$ Hz. The second set of maps, shown in Figure 4.2, correspond to the rotation axis at sub-radar point interval 2 in Figure 3.2. The apparent rotation rate at this interval is given in table 3.1 as $3.04 \cdot 10^{-6}$ Hz. Lastly, the third set of maps shown in Figure 4.3, correspond to the rotation axis at sub-radar point interval 3 in Figure 3.2. The apparent rotation rate at this interval is given in table 3.1 as $3.17 \cdot 10^{-6}$ Hz.

Because the sub-radar point intervals of the maps are located slightly south of zero selenographic longitude and latitude, the maps cover more of the south pole region than of the north pole. The regions on the maps exceeding $\pm 90°$ longitude for negative latitudes are thus on the dark side of the Moon. The line of low resolution across the selenographic maps correspond to the Doppler equator at the time of the observation.

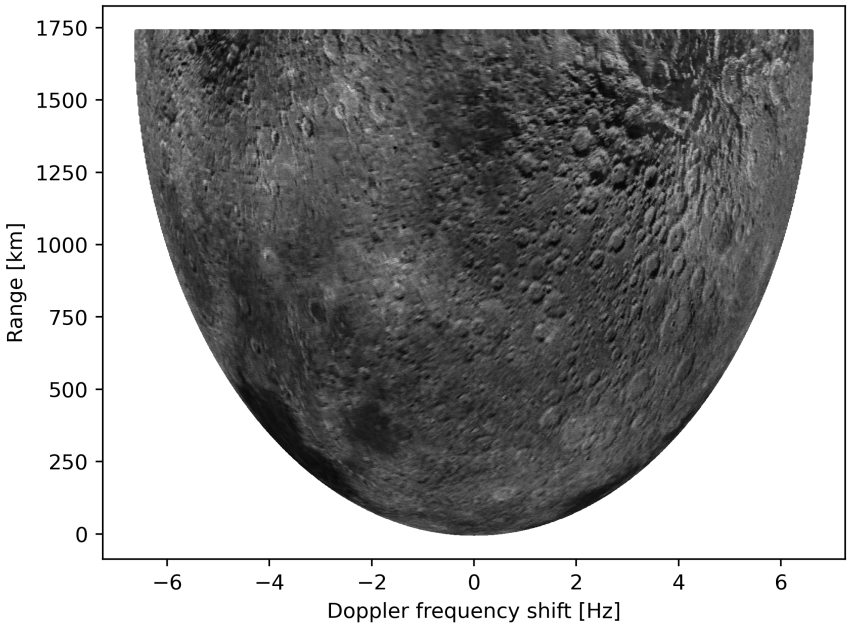


(a) Simulated range-Doppler map at sub-radar point interval 1 given in Figure 3.2

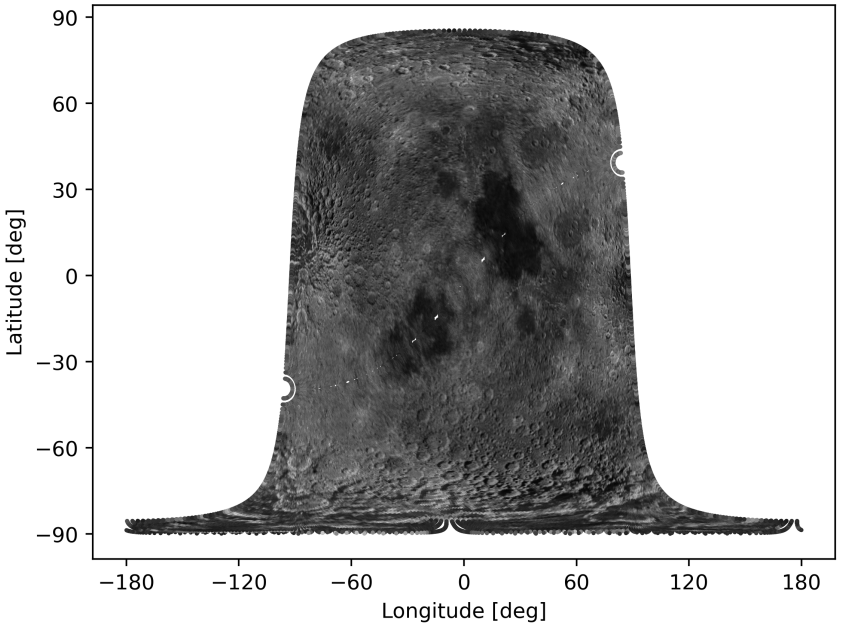


(b) Ambiguous selenographic radar map at sub-radar point interval 1 given in Figure 3.2. The area of no pixels across the middle of the map corresponds to the Doppler equator at the time of the observation.

Figure 4.1: The simulated range-Dopper map and the ambiguous selenographic map at sub-radar point interval 1 given in Figure 3.2. The area of no pixels across the middle of the selenographic map corresponds to the Doppler equator at the time of the observation.

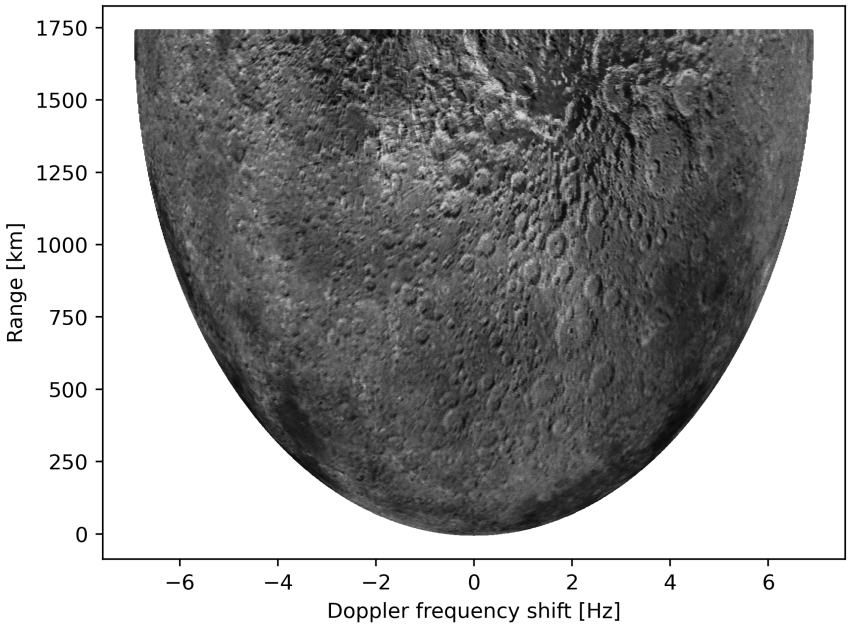


(a) Simulated range-Doppler map at sub-radar point interval 2 given in Figure 3.2

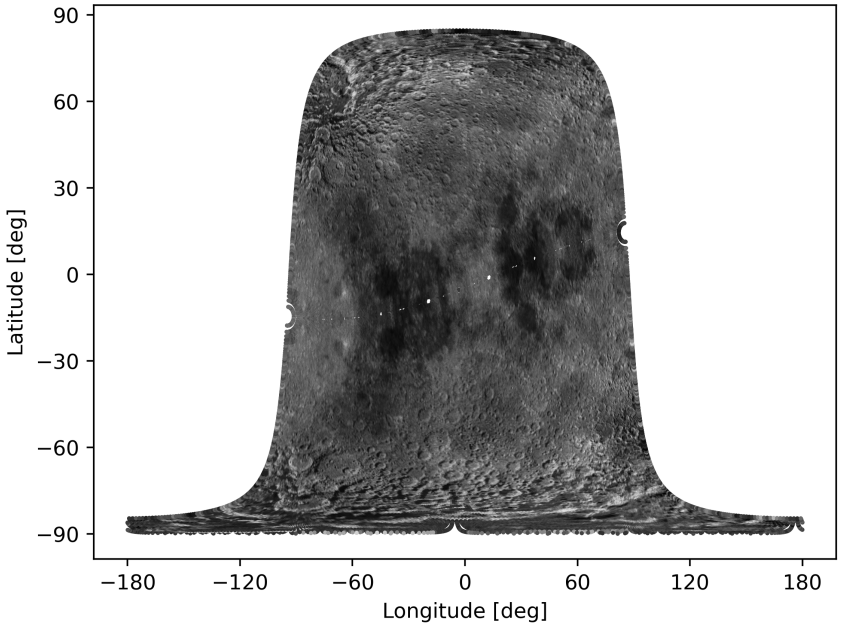


(b) Ambiguous selenographic map at sub-radar point interval 2 given in Figure 3.2. The area of no pixels across the middle of the map corresponds to the Doppler equator at the time of the observation.

Figure 4.2: The simulated range-Dopper map and the ambiguous selenographic map at sub-radar point interval 2 given in Figure 3.2. The area of no pixels across the middle of the selenographic map corresponds to the Doppler equator at the time of the observation



(a) Simulated range-Doppler map at sub-radar point interval 3 given in Figure 3.2



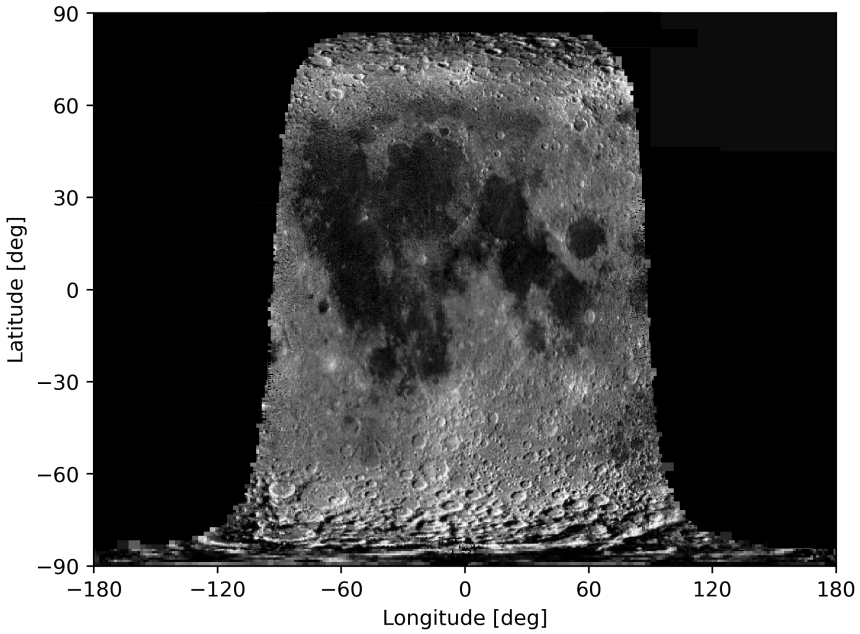
(b) Ambiguous selenographic map at sub-radar point interval 3 given in Figure 3.2. The area of no pixels across the middle of the map corresponds to the Doppler equator at the time of the observation.

Figure 4.3: The simulated range-Dopper map and ambiguous selenographic map at sub-radar point interval 3 given in Figure 3.2. The area of no pixels across the middle of the selenographic map corresponds to the Doppler equator at the time of the observation

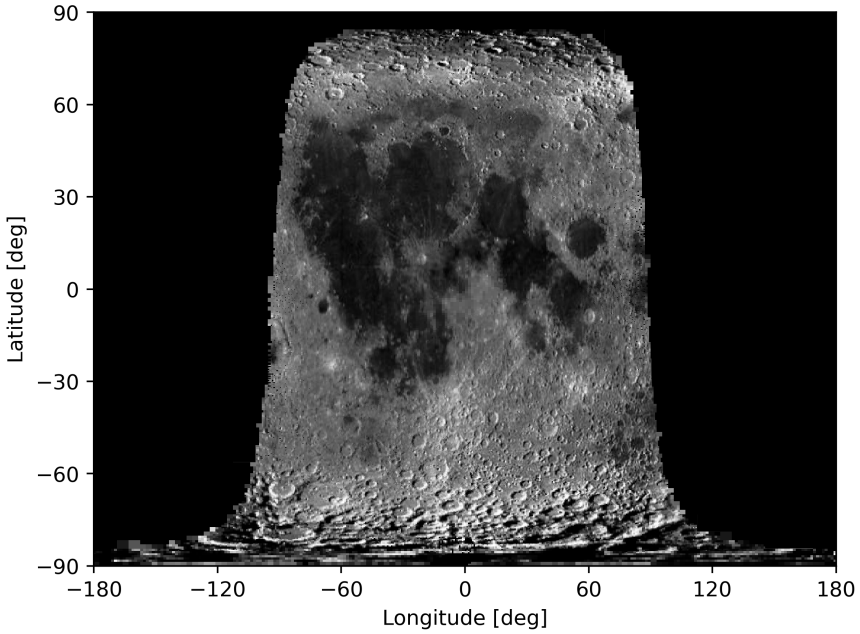
The second and most essential part of the results is the disambiguated map. The map is shown in Figure 4.4a along with the optical lunar mosaic in Figure 4.4b in the same quadtree covering the same selenographic range for comparison reasons. As given by equation 2.13, the pixels yielding the highest resolution is expected to occur at high apparent latitude. The apparent Doppler equators at the three ambiguous maps in Figure 4.1, 4.2 and 4.3 are mainly tilted down towards negative longitudes. The pixels yielding the highest resolution in the disambiguated image are therefore expected to occur at the upper left part of the map as well as lower right part of the map. This can be observed to be the case in Figure 4.4a. The pixels covering the sub-radar point suffers from low resolution as expected, even though it is difficult to see in print.

Comparing the two images in Figure 4.4 indicate significant similarities. The majority of the different surface structures in the optical lunar mosaic in Figure 4.4b are visible in the disambiguated image in Figure 4.4a as well, although slightly grainy. A contributing factor to this grainy effect is the random noise due to surface and subsurface undulation which was included in the simulation but not present in the optical mosaic. Comparing the edges surrounding the two maps demonstrate lower resolution for the disambiguated map compared to the optical mosaic. This is due to few measurements in these regions causing poorly estimates for the true scattering cross-sections. It is also worth noting that the optical lunar mosaic is of higher resolution than what was feasible to achieve by the simulation considering computation time and memory, such that the disambiguated map overall will suffer from lower resolution.

The estimated scattering cross-sections obtained by the disambiguation technique are, in Figure 4.5, plotted with the reflectivity measurements sampled from the optical lunar mosaic. The relative error distribution is displayed in Figure 4.6. The error plot in Figure 4.5 shows a linear correlation between the estimated and real values indicating an unbiased estimate. The error standard deviation of the disambiguated image relative to the optical reflectivity measurements were calculated to be $\sigma = 13.39$, which corresponds to 18.56% of the average reflectivity value from the optical mosaic. As can be observed from the error plot, the optical reflectivity measurements range in value from 0 to 256, while some estimated scattering cross-sections exceed this range. Negative scattering cross-sections have no physical meaning and are considered as erroneous estimates, but indicate that these values, in reality, are close to zero. The same reasoning applies to the estimated scatting cross-sections exceeding 256, which are erroneous estimates as well, but indicate that in reality, these values are close to 256.



(a) Disambiguated image based on the three range-Doppler maps in Figure 4.1, 4.2 and 4.3.



(b) The optical lunar mosaic plotted by the quadtree algorithm in the same coordinate grid as used in Figure 4.4a.

Figure 4.4: The disambiguated map based on three simulated range-Doppler maps with different apparent rotation axes, along with the optical lunar image plotted by the equivalent quadtree algorithm for comparison reasons.

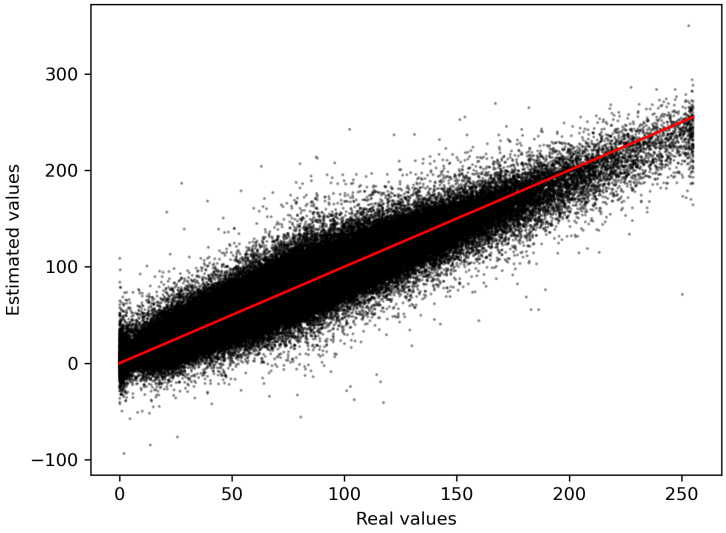


Figure 4.5: The error related to the disambiguated image based on three simulated range-Doppler maps relative to the optical lunar mosaic. The red line corresponds to a linear function with a y-intercept equal to zero and a slope equal to one. The error standard deviation was calculated to be $\sigma = 13.39$, which corresponds to 18.56% of the average reflectivity value.

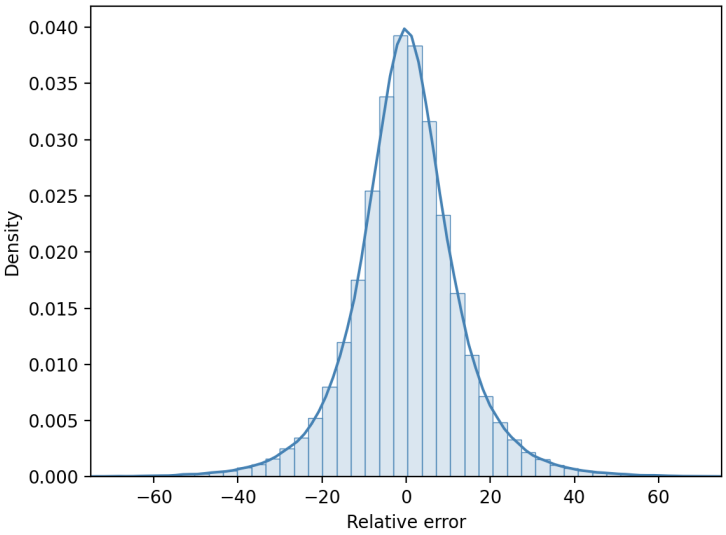


Figure 4.6: The density distribution of the relative error between the disambiguated image in Figure 4.4a and the optical mosaic in Figure 4.4b. The error density follows approximately a Gaussian distribution.

As previously stated, the disambiguated map in Figure 4.4a is based on the three ambiguous maps is Figure 4.1, 4.2 and 4.3. By increasing the number of maps included in the disambiguation, thus increase the number of measurements relative to unknown scattering cross-sections, the accuracy is expected to increase. In order to validate that assumption, three additional ambiguous maps were simulated. The sub-radar point intervals corresponding to the three additional maps are given in Figure 4.7. The intervals labeled 1,2 and 3 correspond to the three sub-radar point intervals from Figure 3.2, while the intervals labeled 4,5 and 6 correspond to the intervals used in simulation of the additional maps. The rotation rates associated with the sub-radar point intervals 4, 5 and 6 are given in 4.1 below. The resulting disambiguated maps based on the four, five and six range-Doppler maps, along with their associated error plots, are displayed in Figure 4.8, 4.9 and 4.10.

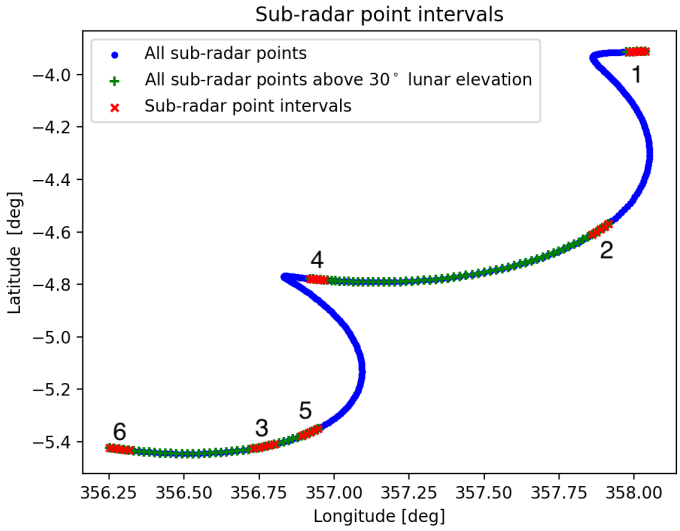
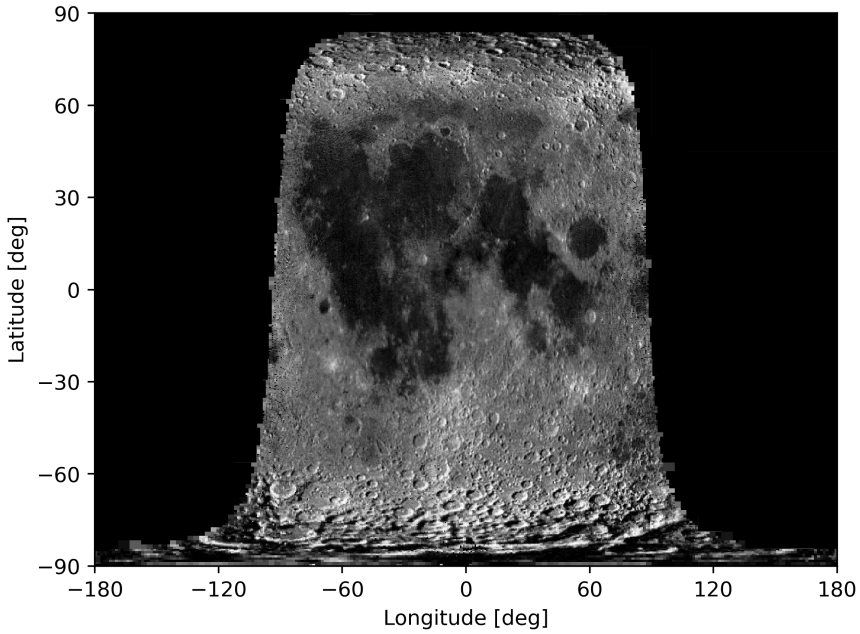


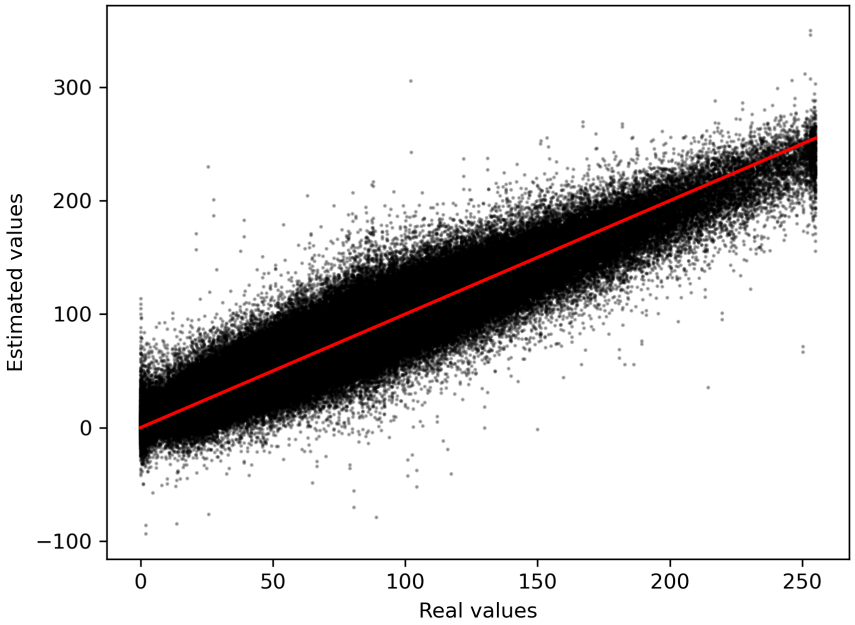
Figure 4.7: Plot of the selenographic sub-radar point intervals retrieved from the ephemeris data. The intervals labeled 1,2 and 3 correspond to the intervals used to simulate the maps in Figure 4.1, 4.2 and 4.3. The intervals labeled 4,5 and 6 correspond to the intervals used to simulate the three additional maps.

Table 4.1: The rotation rates associated with the three sub-radar point intervals 4, 5 and 6 in Figure 4.7.

Interval 4	$2.10 \cdot 10^{-6}$ [Hz]
Interval 5	$2.71 \cdot 10^{-6}$ [Hz]
Interval 6	$2.94 \cdot 10^{-6}$ [Hz]

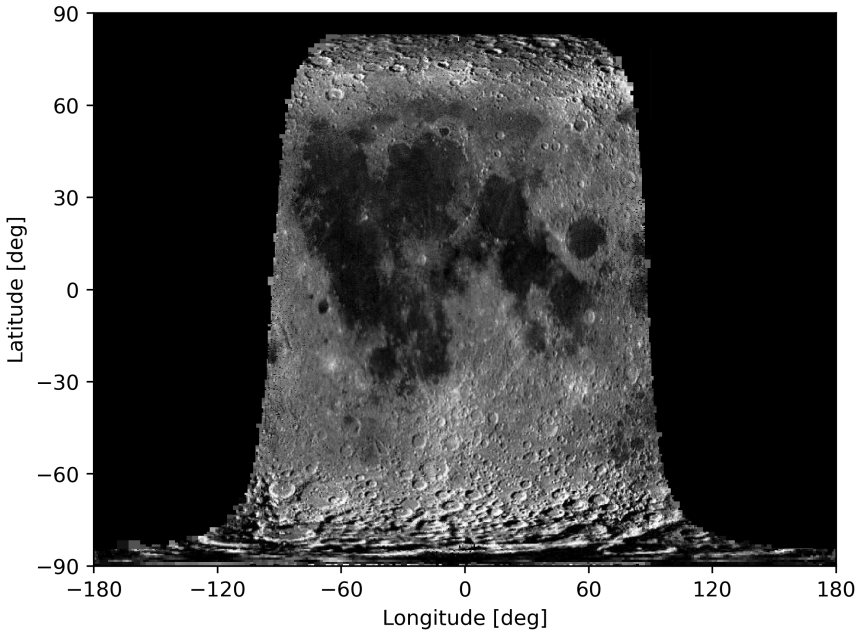


(a) Disambiguated map based on four range-Doppler maps corresponding to the sub-radar point intervals 1, 2, 3 and 4 given in Figure 4.7.

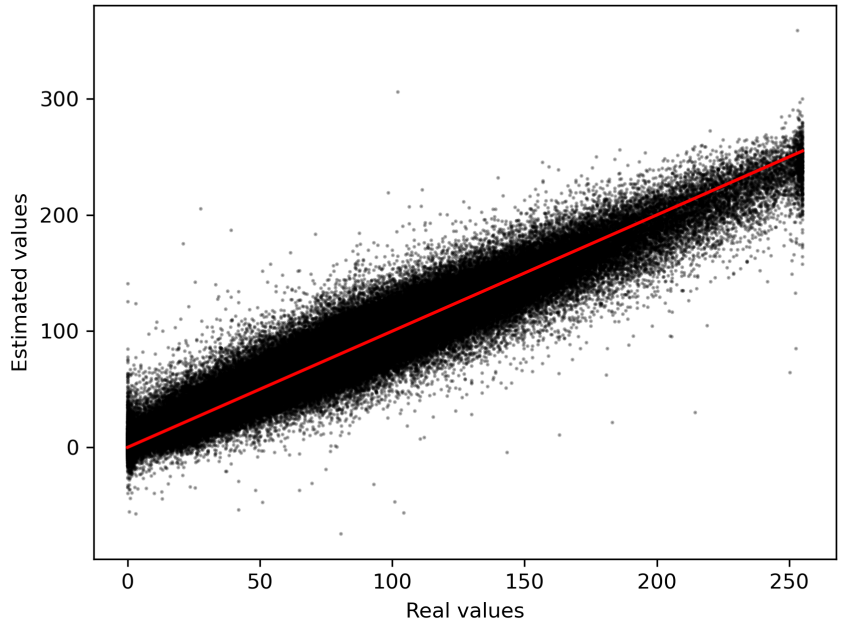


(b) The error of the disambiguated image, based on four simulated range-Doppler maps, relative to the optical lunar mosaic. The red line corresponds to a linear function with a y-intercept equal to zero and a slope equal to one. The error standard deviation was calculated to be $\sigma = 12.28$, which corresponds to 16.39% of the average reflectivity value.

Figure 4.8: Disambiguated map based on four simulated range-Doppler maps with different apparent rotation axes, and the associated relative error

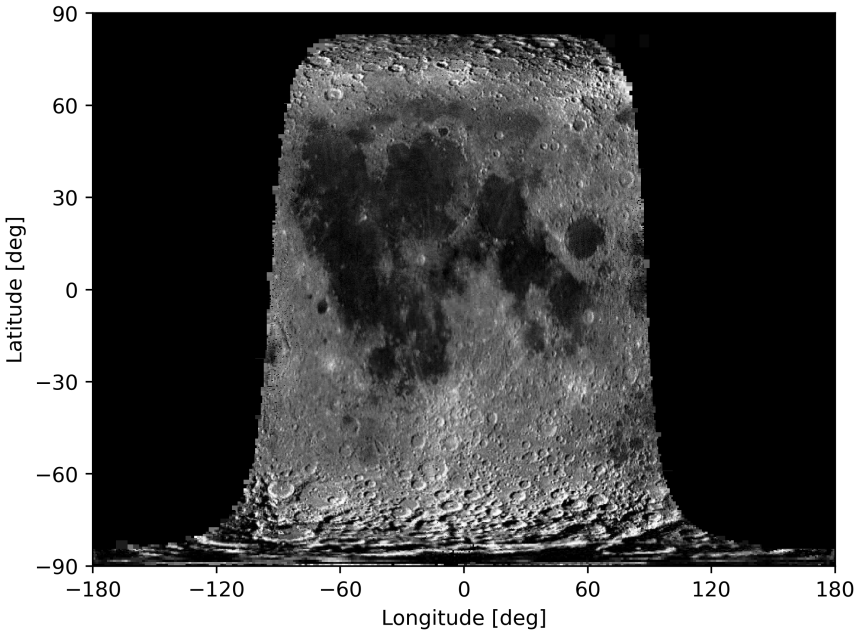


(a) Disambiguated map based on five range-Doppler maps corresponding to the sub-radar point intervals 1, 2, 3, 4 and 5 given in Figure 4.7.

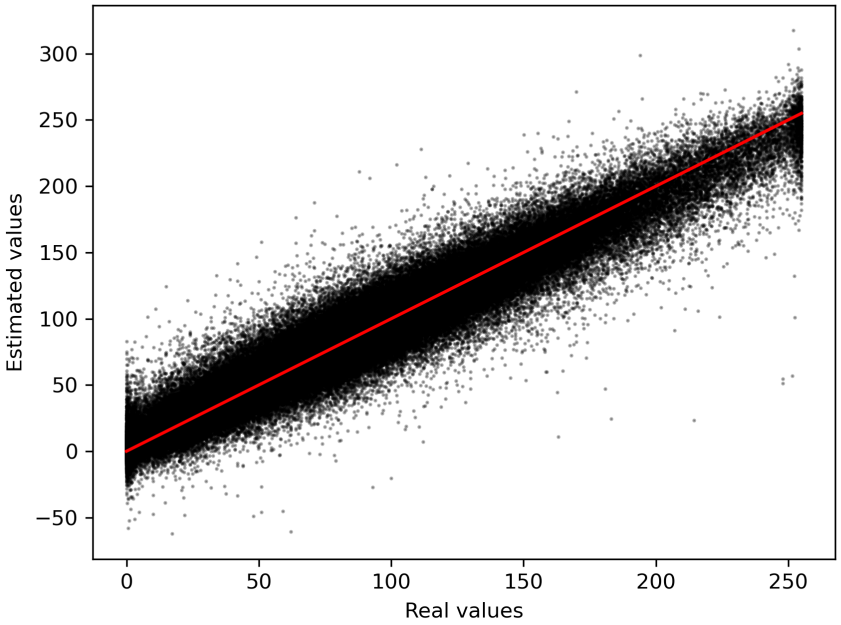


(b) The error of the disambiguated map, based on five simulated range-Doppler maps, relative to the optical lunar mosaic. The red line corresponds to a linear function with a y-intercept equal to zero and a slope equal to one. The error standard deviation was calculated to be $\sigma = 11.21$, which corresponds to 14.97% of the average reflectivity value.

Figure 4.9: Disambiguated map based on five ambiguous range-Doppler maps with different apparent rotation axes, and the associated relative error.



(a) Disambiguated map based on six range-Doppler maps corresponding to the six sub-radar point intervals given in Figure 4.7.



(b) Error related to the disambiguated image based on six ambiguous simulated range-Doppler maps relative to the optical lunar mosaic. The red line corresponds to a linear function with a y-intercept equal to zero and a slope equal to one. The error standard deviation was calculated to be $\sigma = 10.60$, which corresponds to 14.39% of the average reflectivity value.

Figure 4.10: Disambiguated map based on six ambiguous range-Doppler maps with different apparent rotation axes, and the associated relative error.

By visually observing the changes between all four disambiguated maps, it appears that the maps become less grainy and more similar to the optical measurements for each additional map included in the disambiguation procedure. This statement is also supported by the reduction in relative standard deviation as given in table 4.2 and the error density distributions displayed in Figure 4.11.

Table 4.2: The error standard deviation of four disambiguated maps relative to the optical lunar image with reflectivity measurements ranging from 0 to 256.

Number of maps	Standard deviation σ
3	13.39
4	12.28
5	11.21
6	10.60

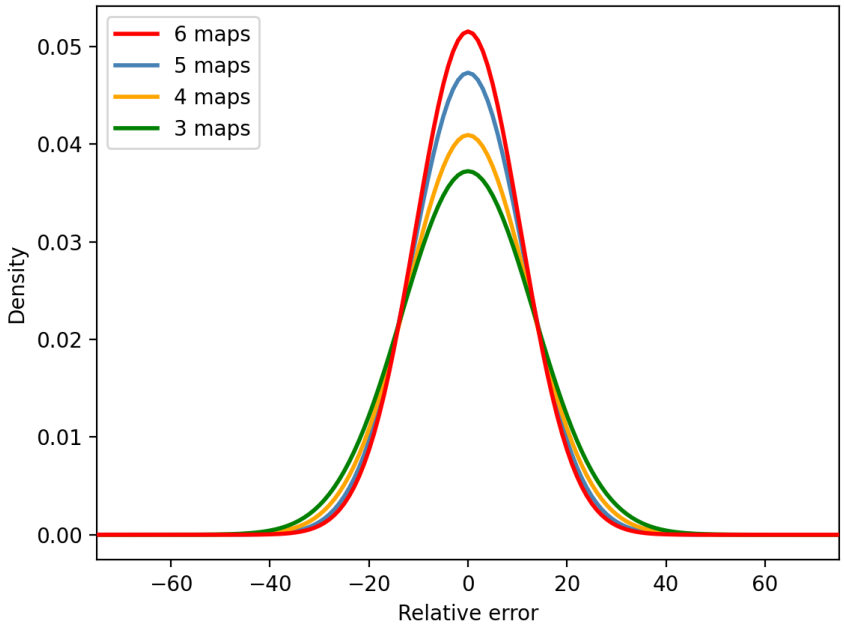


Figure 4.11: The relative error density distributions of the disambiguated maps from Figure 4.4a, 4.8a, 4.9a, and 4.10a.

5

Summary and conclusion

The objective of this thesis was to write and validate a method able to disambiguate simulated range-Doppler maps captured by a monostatic ISAR system. The objective was conducted by firstly simulating three range-Doppler maps at three different apparent rotations. The data used in the simulation were retrieved from NASA's Horizon ephemeris [13]. The simulated maps were converted to selenographic coordinates and used to sample reflectivity measurements of the lunar surface from the Lunar Reconnaissance Orbiter Camera [6]. Included in the simulation were the Hagfors scattering law, range dependent power reduction and noise due to surface and subsurface undulations. The disambiguation, thus estimation of the true normalized scattering cross-sections, was conducted as an overdetermined linear least square system of the ambiguous maps. Included in the disambiguation procedure was the use of a quadtree algorithm developed by Christian Hill [5] to define a fixed rectangular lunar surface pixel grid.

The program developed in this thesis is clearly a prototype. Due to the main objective of this thesis being writing and validation of the disambiguation technique, several effects associated with planetary radar measurements were disregarded in the simulation. A way of improving the program would be to include effects such as antenna gain pattern effects, thermal noise, range-Doppler migration [22], and improved rotation effects. The inclusion of such effects will provide a realistic simulation to a greater extent than what was achieved in this thesis.

A disadvantage of this program is that it is computationally heavy both in terms of time and space complexity. In order to enable simulation and disambiguation of lunar range-Doppler maps with high range and azimuthal resolution it would be beneficial to explore the possibility of increasing the computational efficiency of the program. Further work on this program may also include improving the algorithm generating the fixed pixel grid. Defining the pixels as rectangles do not represent a spherical surface accurately. Exploring other algorithms using different pixel shapes may yield a different and more accurate result.

The resulting disambiguated map, based on three simulated range-Doppler maps, shared significant visual similarities with the optical lunar mosaic. The method presented in this thesis was found to be unbiased, and the resulting error standard deviation was calculated to be 18.56% of the average reflectivity from the lunar mosaic. The error showed a decreasing tendency by including additional maps in the disambiguation procedure. A natural next step would be to use this technique with EISCAT 3D when it becomes operational in the future. The technique may also be used to disambiguate years of available Arecibo measurements of Venus [3].

A

Source code

The Python code used in this thesis is listed in this appendix. In section A.1 the code used to read the Horizon ephemeris is listed. The data retrieved from this script were later imported to the simulation script listed in section A.2. The simulation script produce "pickle" files containing the simulated data which were used in the disambiguation algorithm listed in section A.3. The disambiguation code also includes the application of the QuadTree algorithm developed by Christian Hill [5].

A.1 Read ephemeris - readtxt.py

```
import numpy as np
startkey = "$$SOE"
endkey =  "$$EOE"
filename = 'horizons_results.txt'
'''
Function to read ephemeris data from a given startkey to a given endkey.
-------------------------------------------------------------------------------
INPUT:  - Filename: The downloaded epheremris text file name
        - Startkey: A keyword that indicates where to start reading the file
        - Endkey:   A keyword that indicates where to end reading the file

OUTPUT: - A dictionary containing:
                - Elevation of the target
                - Selenographic longitude of the sub-radar points
                - Selenoographic latitude  of the sub-radar points
                - Range between the observer  and target center [km]
-------------------------------------------------------------------------------
'''
def ImportData(filename, startkey, endkey):
    file = open(filename, "rt")
    content = file.readlines()
    fulldata = []
    for lines in content:
        newline = lines.split(',')[:]
        fulldata.append(' '.join(newline).split())
    file.close()
    startidx = fulldata.index([startkey])
    endidx = fulldata.index([endkey])
    rawdata = fulldata[startidx+1:endidx]
    variables = fulldata[startidx-2:startidx-1][0]
    var = list(np.copy(variables))
    var[0] = 'Date'
    var.insert(1,'HR:MN:SS')
    cleandata = []
    for line in rawdata:
        line = [elem for elem in line if len(elem) != 1]
        cleandata.append(line)
    data = np.array(cleandata).T
    completedata = dict(zip(var, data))
    return completedata

data = ImportData(filename,startkey, endkey)
elevation = np.float64(data['Elev_(a-app)'])
ObsSubLong = np.float64(data['ObsSub-LON'])
ObsSubLat = np.float64(data['ObsSub-LAT'])
delta = np.float64(data['delta'])
```

A.2 Simulation - Simulation.py

```
import numpy as np
from tqdm import tqdm
from PIL import Image
import pickle
import matplotlib.pyplot as plt
import importlib
import readtxt as read
importlib.reload(read)

global imfilename
global savedata
global runsim
global plotsim
global plotHagfors
global plotSRPS
global wp

'''Programming switches'''
#Run simulation
runsim = True
#Save simulated data in pickle files
savedata = True
#Plot simulated data
plotsim = True
#Plot Hagfors backscatter coefficient estimate
plotHagfors = True
#Plot sub-radar point interval
plotSRPS = True

'''Importing ephemeris data'''
ObsSubLong = read.ObsSubLong
ObsSubLat = read.ObsSubLat
elev = read.elevation
delta = read.delta
'''Variables'''

radar_wave = 1.6/1000 #[km]
R_moon =  1737.4 #[km]
c = 2.99792458e5 #[km/s]
Tc = 50 #[s]
tau_p = 1e-5 #[s]
dt = 60*10 #[s]
C_const = 70
rho = 0.4
samplesize = 8
cutoff = 30 #[deg]
srpinterval = 6
srp_start_interval=[0,6,75,60,65,102]
srp_end_interval = [5,11,80,65,70,107]
#Filename of the optical lunar mosaic
imfilename = 'lroc3.jpg'

'''
Function that cut the data at times where the lunar elevation do not exceed a
given cutoff value.
-------------------------------------------------------------------------------
INPUT:  - data: List of data
OUTPUT: - New list of data where the elevation indexes below the cutoff value
```

```
          are removed.
-------------------------------------------------------------------------------
'''
def cutforelevation(data):
    index = []
    for idx, elem in list(enumerate(elev)):
        if elem >= cutoff:
            index.append(idx)
    cutlist = [data[i] for i in index]
    return cutlist
'''
Function that converts selenographic coordinates to apparent Cartesian
coordinates.
-------------------------------------------------------------------------
INPUT:  - long: Selenographic longitude value (float/int)
        - lat:  Selenographic latitude value (float/int)
OUTPUT: - list of apparent Cartesian coordinates (list of floats/ints)
-------------------------------------------------------------------------
'''
def geotocartesian(long,lat):
    long_rad = np.deg2rad(long)
    lat_rad = np.deg2rad(lat)
    x = np.cos(lat_rad)*np.cos(long_rad)
    y = np.cos(lat_rad)*np.sin(long_rad)
    z = np.sin(lat_rad)
    return [x,y,z]

'''Find the sub-radar points in both selenographic and Cartesian coordinates
given the sub-radar point interval'''
startidx = srp_start_interval[srpinterval-1]
endidx = srp_end_interval[srpinterval-1]
SubLong = cutforelevation(ObsSubLong)[startidx:endidx]
SubLat = cutforelevation(ObsSubLat)[startidx:endidx]
srpcarts = [geotocartesian(long, lat) for long,lat in zip(SubLong,SubLat)]
'''
Function to convert selenographic coordinates to range coordinates.
---------------------------------------------------------------------
INPUT:  - longitude: Selenographic longitude (float/int)
        - latitude:  Selenographic latitude (float/int)
OUTPUT: - Range value in km(float/int)
---------------------------------------------------------------------
'''
def Range(longitude,latitude):
    x,y,z = geotocartesian(longitude,latitude)
    r = R_moon-R_moon*x
    return r
'''
Function that finds the radial distance between two adjacent sub-radar points
in a given sub-radar point interval list.
-------------------------------------------------------------------------------
INPUT : - Sublong: Sub-radar point longitude values (list of floats/ints)
        - Sublat:  Sub-radar point latitude values (list of floats/ints)
OUTPUT: - The radial distance between the two middle adjacent sub-radar points
          in the sub-radar point interval (float)
-------------------------------------------------------------------------------
'''
def FindPointsForWp(SubLong,SubLat):
    x,y,z = [],[],[]
    for i,j in list(zip(SubLong,SubLat)):
        x1,y1,z1=geotocartesian(i,j)
        x.append(x1)
```

```
        y.append(y1)
        z.append(z1)

    idx1 = int(((len(SubLong)-1)/2)-1)
    idx2 = int(((len(SubLong)-1)/2))
    point1 = np.array((x[idx1],y[idx1],z[idx1]))
    point2 = np.array((x[idx2],y[idx2],z[idx2]))
    dist = np.linalg.norm(point1-point2)
    return dist
'''
Function that calculates the apparent rotation rate [rad/s] between two
sub-radar points in a given sub-radar point interval.
-------------------------------------------------------------------------
INPUT : - Sublong: Sub-radar point longitude values (list of floats/ints)
        - Sublat:  sub-radar point latitude values (list of floats/ints)
OUTPUT: - The rotational rate in rad/s (float)
-------------------------------------------------------------------------
'''
def RotationalFreq(SubLong,SubLat):
    dist = FindPointsForWp(SubLong,SubLat)
    wp = dist/dt
    return wp

wp = RotationalFreq(SubLong,SubLat)
'''
Function that calculates the Doppler frequency shift [Hz] for a given
y-coordinate.
-------------------------------------------------------------------------
INPUT : - y: y-coordinate (float/int)
OUTPUT: - Doppler frequency shift in Hz (float)
-------------------------------------------------------------------------
'''
def Doppler(y):
    D = -(4*np.pi*R_moon*wp*y/radar_wave)
    return D
'''
Function that converts selenographic coordinates to range-Doppler coordinates.
-------------------------------------------------------------------------
INPUT : - Sublong: Sub-radar point longitude values (list of floats/ints)
        - Sublat:  Sub-radar point latitude values (list of floats/ints)
        - longitude: Selenographic longitude (float/int)
        - latitude:  Selenographic latitude (float/int)
OUTPUT: - Range value in km (float)
        - Doppler frequency shift in Hz (float)
-------------------------------------------------------------------------
'''
def RunLLtoRD(SubLong,SubLat,longitude,latitude):
    x,y,z = geotocartesian(longitude, latitude)
    r = Range(longitude,latitude)
    dop = Doppler(y)
    return r,dop
'''
Function that calculates the x-coordinate for a given range value [km].
-------------------------------------------------------------------
INPUT : - R: Range value in km (float/int)
OUTPUT: - x-coordinate (float)
-------------------------------------------------------------------
'''
def x_coor(R):
    x = -(R-R_moon)/R_moon
    if x < 0:
```

```
        x = 0
    elif x > 1:
        x = 1
    return x
'''
Function that calculates the y-coordinate for a given Doppler frequency [Hz].
-------------------------------------------------------------------------
INPUT : - D: Doppler value in Hz (float/int)
OUTPUT: - y-coordinate (float)
-------------------------------------------------------------------------
'''
def y_coor(D):
    y = -((D*radar_wave)/(4*np.pi*R_moon*wp))
    if y > 1:
        y = 1
    elif y < -1:
        y = -1
    return y
'''
Function that calculates the z-coordinate for a given x and y coordinate.
-------------------------------------------------------------------------
INPUT : - x: x-coordinate (float/int)
        - y: y-coordinate (float/int)
OUTPUT: - z-coordinate (float)
-------------------------------------------------------------------------
'''
def z_coor(x,y):
    z = np.sqrt(np.abs(1-x**2-y**2))
    if z > 1:
        z = 1
    elif z < -1:
        z = -1
    return z
'''
Function that converts Cartesian coordinates to selenographic coordinates.
-------------------------------------------------------------------------
INPUT : - x: x-coordinate (float/int)
        - y: y-coordinate (float/int)
        - z: z-coordinate (float/int)
OUTPUT: - long: Selenographic longitude (float)
        - lat:  Selenographic latitude (float)
-------------------------------------------------------------------------
'''
def cartesiantogeo(x,y,z):
    lat = np.rad2deg(np.arcsin(z))
    if x>0:
        long = np.rad2deg(np.arctan(y/x))
    elif y>0:
        long = np.rad2deg(np.arctan(y/x)) + 180
    else:
        long = np.rad2deg(np.arctan(y/x)) - 180
    return long,lat
'''
Function that converts a range-Doppler coordinate to Cartesian coordinates.
-------------------------------------------------------------------------
INPUT : - r:   Range coordinate [km] (float/int)
        - dop: Doppler coordinate [Hz] (float/int)
OUTPUT: - poscart: Cartesian coordinate on the north Doppler hemisphere (float)
        - negcart: Cartesian coordinate on the south Doppler hemisphere (float)
-------------------------------------------------------------------------
'''
```

```
234 def RunRDtoCart(r,d):
235     x = x_coor(r)
236     y = y_coor(d)
237     z = z_coor(x,y)
238     poscarts = [x,y,z]
239     negcarts = [x,y,-z]
240     return poscarts, negcarts
241 '''
242 Function that converts a range-Doppler coordinate to both positive
243 latitude and negative latitude selenographic coordinates.
244 ----------------------------------------------------------------
245 INPUT : - r:   Range coordinate [km] (float/int)
246         - dop: Doppler coordinate [Hz] (float/int)
247 OUTPUT: - poslong: Selenographic longitude (float)
248         - poslat: Positive selenographic latitude (float)
249         - neglong: Selenographic longitude (float)
250         - neglat: Negative selenographic latitude (float)
251 ----------------------------------------------------------------
252 '''
253 def RunRDtoLL(r,dop):
254     poscoor,negcoor = RunRDtoCart(r,dop)
255     poslong,poslat = cartesiantogeo(poscoor[0], poscoor[1], poscoor[2])
256     neglong,neglat = cartesiantogeo(negcoor[0],negcoor[1],negcoor[2])
257     return poslong,poslat, neglong,neglat
258 '''
259 Function that calculates the Doppler bandwidth [Hz] of the Moon.
260 ----------------------------------------------------------------
261 OUTPUT: - Bandwidth of the Moon [Hz] (float)
262 ----------------------------------------------------------------
263 '''
264 def FindBW():
265     bw = 8*np.pi*R_moon*wp/radar_wave
266     return bw
267 '''
268 Function that calculates the range values of the iso-range rings.
269 ------------------------------------------------------------------------
270 OUTPUT: - r_res_array: The iso-range ring range values [km] (list of floats)
271         - dr: Slant range resolution [km] (float)
272 ------------------------------------------------------------------------
273 '''
274 def RangeRes():
275     dr = (c*tau_p/2)
276     r_res_array =  np.arange(0,R_moon+1,dr)
277     return r_res_array, dr
278 '''
279 Function that calculates all Doppler values of a iso-range ring.
280 ------------------------------------------------------------------------
281 INPUT : - R: Iso-range ring range value [km] (float/int)
282 OUTPUT: - d_list: List of all Doppler values within that iso-range ring [Hz]
283           (list of floats)
284 ------------------------------------------------------------------------
285 '''
286 def Max_Dopper(R):
287     BW = FindBW()
288     dmax = np.sqrt(1-x_coor(R)**2)*(BW/2)
289     k = 0
290     dlist = []
291     d = 0
292     while d<=dmax:
293         d = k/Tc
294         if d<=dmax:
```

```
            dlist.append(k/Tc)
            d = k/Tc
            k += 1
    for i in range(1,k):
        dlist.append(-i/Tc)
    return dlist
'''
Function that calculates all Doppler values to all iso-range rings in such a
way that both lists are equal in length.
-------------------------------------------------------------------------------
INPUT : - rangelist:     Iso-range ring range values [km] (list of floats/ints)
OUTPUT: - dopplervalues: List of all Doppler coordinates [Hz] (list of floats)
        - newrange:      List of all range coordinates [km] (list of floats)
-------------------------------------------------------------------------------
'''
def RunMaxDoppler(rangelist):
    dopplervalues = []
    for r in rangelist:
        dopplervalues.append(Max_Dopper(r))
    newrange = []
    for i in range(len(dopplervalues)):
        newrange.append([rangelist[i]]*len(dopplervalues[i]))
    return dopplervalues, newrange
'''
Function that finds the rotation matrix for rotations about the x-axis
-------------------------------------------------------------------------------
INPUT:  - x_ang: The angle of rotation about the x-axis in radians (float/int)
OUTPUT: - 3x3 Rotation matrix (array of floats)
-------------------------------------------------------------------------------
'''
def xrot(x_ang):
    return np.array(([1,0,0], [0,np.cos(x_ang),-np.sin(x_ang)],
    ↪  [0,np.sin(x_ang),np.cos(x_ang)]))
'''
Function that finds the rotation matrix for rotations about the y-axis
-------------------------------------------------------------------------------
INPUT:  - y_ang: The angle of rotation about the y-axis in radians (float/int)
OUTPUT: - 3x3 Rotation matrix (array of floats)
-------------------------------------------------------------------------------
'''
def yrot(y_ang):
    return np.array(([np.cos(y_ang),0,np.sin(y_ang)], [0,1,0],
    ↪  [-np.sin(y_ang),0,np.cos(y_ang)]))
'''
Function that finds the rotation matrix for rotations about the z-axis
-------------------------------------------------------------------------------
INPUT:  - z_ang: The angle of rotation about the z-axis in radians (float/int)
OUTPUT: - 3x3 Rotation matrix (array of floats)
-------------------------------------------------------------------------------
'''
def zrot(z_ang):
    return np.array(([np.cos(z_ang),-np.sin(z_ang),0], \
          [np.sin(z_ang),np.cos(z_ang),0], [0,0,1]))
'''
Function that calculates the angle in radians between two vectors
-------------------------------------------------------------------
INPUT:  - vec1:  A vector (array of floats/ints)
        - vec2:  A vector (array of floats/ints)
OUTPUT: - angle: Angle between the two vectors in radians (float)
-------------------------------------------------------------------
'''
```

```
354 def vectorangle(vec1, vec2):
355     angle = np.arccos(np.dot(vec2, vec1)/(np.linalg.norm(vec1))*\
356             np.linalg.norm(vec2))
357     return angle
358 '''
359 Function that calculates the y and z angle in radians between a vector and
360 the x axis.
361 ----------------------------------------------------------------------
362 INPUT:  - vec:  A vector (array of floats/ints)
363         - x_ax: X axis vector [1,0,0] (array of floats/ints)
364 OUTPUT: - y and z rotation angles in radians (list of floats)
365 ----------------------------------------------------------------------
366 '''
367 def find_yz_ang(vec, x_ax):
368     #This should be wrong due to the signs, but results in  correct rotation
369     y_ang = -np.sign(vec[2])*(vectorangle([vec[0], vec[2]], \
370     [x_ax[0], x_ax[2]]))
371     z_ang = np.sign(vec[1])*vectorangle([vec[0], vec[1]], [x_ax[0], x_ax[1]])
372     return [y_ang, z_ang]
373 '''
374 Function that rotates a vector about the y and z axes by given angles
375 ----------------------------------------------------------------------
376 INPUT:  - vec:     Vector to rotate (array of floats/ints)
377         - ang:     y and z rotation angles (list of floats/ints)
378 OUTPUT: - full_rot: the rotated vector (array of floats)
379 ----------------------------------------------------------------------
380 '''
381 def rotate_to_SRP(vec,ang):
382     partial_rot = np.dot(vec,yrot(ang[0]))
383     full_rot = np.dot(partial_rot,zrot(ang[1]))
384     return full_rot
385 '''
386 Function that calculates the rotation axis at a given sub-radar point interval
387 ----------------------------------------------------------------------------
388 INPUT:  - srp_1: First sub-radar point vector (array of floats/ints)
389         - srp_2: Second sub-radar point vector (array of floats/ints)
390 OUTPUT: - rotax: Rotation axis vector (array of floats)
391 ----------------------------------------------------------------------------
392 '''
393 def findrotax(srp_1,srp_2):
394     srp1,srp2 = np.array(srp_1),np.array(srp_2)
395     diff = srp1-srp2
396     length= np.linalg.norm(diff)
397     rotax = np.cross(srp1,diff/length)
398     return rotax
399 '''
400 Function that rotates Cartesian coordinates in the opposite way a sub-radar
401 point is rotated to [1,0,0].
402 ----------------------------------------------------------------------------
403 INPUT:  - carts:   Cartesian coordinates (nested list of floats/ints)
404 OUTPUT: - rotlong: Rotated point in selenographic longitude (list of floats)
405         - rotlat:  Rotated point in selenographic latitude (list of floats)
406 ----------------------------------------------------------------------------
407 '''
408 def rotatepoint(carts):
409     x_ax,z_ax = [1,0,0],[0,0,1]
410     rot_yz = find_yz_ang(srpcarts[0],x_ax)
411     rot_ax = findrotax(srpcarts[0],srpcarts[2])
412     rot_ax_ang = np.sign(rot_ax[1])*vectorangle(rot_ax[1:],z_ax[1:])
413     roty = np.array(rot_yz[0])*-1
414     rotz = np.array(rot_yz[1])*-1
```

```
    pointrotyz = [roty,rotz]
    rotatedbyrotax = [np.dot(xrot(-rot_ax_ang),i) for i in carts]
    rotatedtoSRP = [rotate_to_SRP(i,pointrotyz) for i in rotatedbyrotax]
    rotlong,rotlat = list(zip(*[cartesiantogeo(c[0], c[1], c[2]) for c in \
    rotatedtoSRP]))
    return rotlong,rotlat
'''
Function that calculates all Doppler values and all iso-range ring values
and convert them to positive and negative Cartesian coordinates.
--------------------------------------------------------------------------
OUTPUT: - flatd:   Doppler coordinates [Hz] (list of floats)
        - flatr:   Range coordinates [km] (list of floats)
        - poscart: Cartesian coordinate on the north Doppler hemisphere
                   (list of floats)
        - negcart: Cartesian coordinate on the north Doppler hemisphere
                   (list of floats)
--------------------------------------------------------------------------
'''
def RunResolution():
    rangelist,dr = RangeRes()
    dlist,rlist = RunMaxDoppler(rangelist)
    flatd = [item for sublist in dlist for item in sublist]
    flatr = [item for sublist in rlist for item in sublist]
    poscarts,negcarts = list(zip(*[RunRDtoCart(r, d)for r,d in \
    zip(flatr,flatd)]))
    return flatd,flatr,poscarts, negcarts
'''
Function that calculates the Hagfors backscatter coefficient and incidence
angle for each iso-range ring.
--------------------------------------------------------------------------
INPUT:  - r:       Range value [km] (float/int)
OUTPUT: - sigma_H: Hagfors backscatter coefficient (float)
        - phi:     Incidence angle (float)
--------------------------------------------------------------------------
'''
def HagforsRange(r):
    srp = np.array([1,0,0])
    xyz = np.array([1-(r/R_moon), 0, np.sqrt(1**2-(1-(r/R_moon))**2)])
    phi = np.arccos(np.dot(xyz,srp))
    sigma_H = (C_const*rho/2)*((np.cos(phi))**4+C_const*\
              (np.sin(phi))**2)**(-3/2)
    return sigma_H,np.rad2deg(phi)
'''
Function that calculates the pixel area for each positive latitude coordinate.
--------------------------------------------------------------------------
INPUT:  - latitude: Positice selenographic latitude (float/int)
OUTPUT: - A: The pixel area [1/km^2] (float)
--------------------------------------------------------------------------
'''
def DeltaA(latitude):
    if np.sin(np.deg2rad(latitude)) == 0:
        A = ((c*radar_wave*tau_p)/(4*Tc*wp*np.abs(np.sin(np.deg2rad(0.01)))))\
            /(R_moon**2)
    else:
        A = ((c*radar_wave*tau_p)/(4*Tc*wp*np.abs(np.sin(np.deg2rad(latitude))\
            )))/(R_moon**2)
    return A
'''
Function that calculates the area of each iso-range ring. This is used to plot
the Hagfors scattering law for estimation of C and rho.
--------------------------------------------------------------------------
```

```
INPUT:  - rlist: Range coordinates [km] (list of floats/ints)
        - dr:    Slant range resolution [km] (float)
OUTPUT: - A:     The iso-range ring area [1/km^2] (float)
-------------------------------------------------------------------------
'''
def RingArea(rlist,dr):
    A = []
    for r in range(len(rlist)):
        if r == 0:
            H = np.sqrt(1**2-(1-(dr/R_moon))**2)
            A.append(np.pi*H**2)
        else:
            H = np.sqrt(1**2-(1-(rlist[r]/R_moon))**2)
            H_prev = np.sqrt(1**2-(1-(rlist[r-1]/R_moon))**2)
            A.append(np.pi*(H**2-H_prev**2))
    return np.array(A)
'''
Function that finds the pixel coordinates of the optical image from
selenographic coordinates.
-------------------------------------------------------------------
INPUT:  - long:    Selenographic longitude value (float/int)
        - lat:     Selenographic latitude value (float/int)
        - imarray: The optical lunar image (2D-array of floats/ints)
OUTPUT: - alpha:   The horizontal/longitude image coordinate (int)
        - beta:    The vertical/latitude image coordinate (int)
-------------------------------------------------------------------
'''
def ImageCoordinates(long,lat,imarray):
    height,length = imarray.shape
    alpha = int(np.round((1/2)*length*(1+long/180)))
    beta = int(np.round((1/2)*height*(1-lat/90)))
    if alpha >= length:
        alpha = length-1
    if beta >= height:
        beta = height-1
    return alpha, beta
'''
Function that finds the brightness value of the optical image from
selenographic coordinates.
-------------------------------------------------------------------
INPUT:  - long:       Selenographic longitude value (float/int)
        - lat:        Selenographic latitude value (float/int)
        - imarray:    The optical lunar image (2D-array of floats/ints)
OUTPUT: - brightness: Brightness value (float)
-------------------------------------------------------------------
'''
def RUNImageBrightness(long,lat,imarray):
    long_idx,lat_idx = ImageCoordinates(long, lat,imarray)
    brightness = float(imarray[lat_idx,long_idx])
    return brightness
'''
Function that estimates the noise due to random surface and subsurface
undulations by creating a random normal distribution for the sampled brightness
values from the range-Dopper grid. Isserlis therorem is then applied on the
random variables which results in the estimated brightness value for the main
range-Doppler coordinate in the sampling grid.
-------------------------------------------------------------------------
INPUT:  - var1: Positive latitude brightness values (list of floats/ints)
        - var2: Negative latitude brightness values (list of floats/ints)
OUTPUT: - varP: The estimated lunar surface variance (float)
-------------------------------------------------------------------------
```

```
'''
def AddRandomNoise(var1,var2):
    P_real_1 = np.array([np.random.normal(0, np.sqrt(v/2)) for v in var1])
    P_im_1 = 1j*np.array([np.random.normal(0,np.sqrt(v/2)) for v in var1])
    P_real_2 = np.array([np.random.normal(0, np.sqrt(v/2)) for v in var2])
    P_im_2 = 1j*np.array([np.random.normal(0,np.sqrt(v/2)) for v in var2])
    P_real = P_real_1+P_real_2
    P_im = P_im_1+P_im_2
    varP = np.mean((P_real + P_im)*(P_real-P_im)).real
    return varP
'''
Function that reduces the signal power by the distance traveled.
-------------------------------------------------------------------------
INPUT:  - r: range value [km] (float/int)
OUTPUT: - reduction_coeff: The power reduction coefficient [1/km^2](float)
-------------------------------------------------------------------------
'''
def AddRangeEffect(r):
    d = delta[0]+r
    reduction_coeff = 1/(((4*np.pi)**2)*d**4)
    return reduction_coeff
'''
Function that increase the signal power by the area traveled.
-------------------------------------------------------------------------
INPUT:  - r: range value [km] (float/int)
OUTPUT: - increasing_coeff: The increasing power coefficient [km^2](float)
-------------------------------------------------------------------------
'''
def RemoveRangeEffect(r):
    d = delta[0]+r
    increasing_coeff = ((4*np.pi)**2)*d**4
    return increasing_coeff
'''
Function that rotates the coordinate to true selenographic coordinate and
samples the optical image for a range-Doppler coordinate including noise, area,
range and scattering effects.
-----------------------------------------------------------------------------
INPUT:  - r:       range value [km] (float/int)
        - d:       Doppler value [Hz] (float/int)
        - imarray: The optical lunar image (2D-array of floats/ints)
        - N:       Samplegrid size (N+1 x N+1)
OUTPUT: - posrotlong:   True selenographic longitude value (float)
        - posrotlat:    True positive selenographic latitude value (float)
        - negrotlong:   True selenographic longitude value (float)
        - negrotlat:    True negative selenographic latitude value (float)
        - scaledbright: Brightness value with noise and removed range,
                        scattering and area effects.
-----------------------------------------------------------------------------
'''
def SuperSampling(r,d,imarray,N=samplesize):
    rpix = np.linspace(r-(int(N/2)*(tau_p*c/2))/N,r+(int(N/2)*\
    (tau_p*c/2))/N,N+1)
    dpix = np.linspace(d-(int(N/2)*(1/Tc))/N,d+(int(N/2)*(1/Tc))/N,N+1)
    poscart,negcart = [],[]
    for r in rpix:
        for d in dpix:
            pcart,ncart = RunRDtoCart(r,d)
            poscart.append(pcart)
            negcart.append(ncart)
    posrotlong,posrotlat = rotatepoint(poscart)
    negrotlong,negrotlat = rotatepoint(negcart)
```

```
    sigma = HagforsRange(r)[0]
    dA = DeltaA(posrotlat[int(N/2)])
    pb = [RUNImageBrightness(lo, la, imarray) for lo,la in zip(posrotlong,\
         posrotlat)]
    nb = [RUNImageBrightness(lo, la, imarray) for lo,la in zip(negrotlong,\
         negrotlat)]
    rawposbright = sigma*np.array(pb)*dA*AddRangeEffect(r)
    rawnegbright = sigma*np.array(nb)*dA*AddRangeEffect(r)
    add_noise = AddRandomNoise(rawposbright,rawnegbright)
    scaledbright = (add_noise/(sigma*dA)) * RemoveRangeEffect(r)
    return posrotlong[int(N/2)],posrotlat[int(N/2)],negrotlong[int(N/2)]\
    ,negrotlat[int(N/2)],scaledbright
'''
-------------------------------------------------------------------------
Function that runs the simulation with the ability of saving the simulated
data as "pickle" files and plot the simulated data as well as the sub-radar
point interval and the Hagfors backscatter coefficients estimation.
-------------------------------------------------------------------------
'''
def RunSimulation():
    print('Run simulation')
    image = Image.open(imfilename)
    imarray = np.array(image)[:,:,1]
    height,length = imarray.shape
    dlist, rlist, poscart,negcart = RunResolution()
    poslong,poslat,neglong,neglat,bright=list(zip(*[SuperSampling(r,d,imarray)\
    for r,d in tqdm(zip(rlist,dlist),desc='SuperSampling',total=len(rlist),\
    position=0, leave=True)]))
    if savedata == True:
        print('Saving data')
        rd = [rlist,dlist]
        pickle.dump(rd, open(f"RD {startidx}-{endidx}", "wb"))
        poscoor = [(poslong[i],poslat[i]) for i in range(len(poslong))]
        negcoor = [(neglong[i],neglat[i]) for i in range(len(neglong))]
        coor = [None]*(len(poscoor)+len(negcoor))
        coor[::2] = poscoor
        coor[1::2] = negcoor
        pickle.dump(coor,open(f"Coordinates {startidx}-{endidx}", "wb"))
        pickle.dump(bright,open(f"Brightness-value {startidx}-{endidx}", "wb"))
    if plotsim == True:
        plt.figure()
        plt.title(f'Rotated coordinates. $T_c$ = {Tc} and $\\tau_p$ ={tau_p} \
        for idx {startidx} to {endidx}')
        plt.scatter(poslong,poslat,c='r',marker = '+')
        plt.scatter(neglong,neglat,c='b',marker = '+')
        plt.scatter(poslong[0],poslat[0],c='g',marker = '+')
        plt.xlabel('Longitude')
        plt.ylabel('Latitude')
        plt.show()
        plt.figure()
        plt.title(f'$T_c$ = {Tc} and $\\tau_p$ ={tau_p} for idx \
        {startidx} to {endidx} ')
        plt.scatter(poslong,poslat,c = bright,cmap='Greys_r',linewidths=0.001)
        plt.scatter(neglong,neglat,c = bright,cmap='Greys_r',linewidths=0.001)
        plt.colorbar()
        plt.xlabel('Longitude')
        plt.ylabel('Latitude')
        plt.show()
        plt.figure()
        plt.title(f'$T_c$ = {Tc} and $\\tau_p$ ={tau_p} for idx \
        {startidx} to {endidx} ')
```

```
        plt.scatter(rlist,dlist,c = bright,cmap='Greys_r',linewidths=0.001)
        plt.scatter(rlist,dlist,c = bright,cmap='Greys_r',linewidths=0.001)
        plt.colorbar()
        plt.xlabel('Longitude')
        plt.ylabel('Latitude')
        plt.show()
if __name__=='__main__':
    print(f'tau_p = {tau_p}[s]\nTc = {Tc}[s]\nfp = {wp*(2*np.pi)}[Hz]\
          \nC,rho = {C_const},{rho}\nSRP interval = {startidx}-{endidx}\
          \nRun sim = {runsim}\nSave sim = {savedata}\nPlot sim = {plotsim}\
          \nPlot Hagfors = {plotHagfors}\nPlot SRPs = {plotSRPS}')
    if runsim == True:
        RunSimulation()
    if plotHagfors == True:
        ran,dr = RangeRes()
        deltatime = np.array([(2*r)/c for r in ran])*1e3
        sigma_H = [HagforsRange(r)[0] for r in ran]
        area = RingArea(ran,dr)
        sigma_a_srp = sigma_H[0]*area[0]
        sigma_a = sigma_H*area
        plt.figure(figsize=(6,7))
        plt.plot(deltatime,10*np.log10(sigma_a/sigma_a_srp),c='black', \
        label = (f'$\\lambda$ = {radar_wave*1000} [m]\n$\\rho_0$ = {rho}\
        \nC = {C_const}'))
        plt.legend(fontsize=14,handlelength=0, handletextpad=0)
        plt.xticks(fontsize = 14)
        plt.yticks(np.arange(-44,1,4)[::-1],fontsize = 14)
        plt.ylabel(r'$\sigma^{H}$ [dB]',fontsize=14)
        plt.xlabel('Delay [ms]',fontsize=14)
        plt.vlines(0, -43.9, 0, colors='black',linestyle ='--',linewidth = 0.7)
        plt.ylim(-44,0)
        plt.xlim(-1.5,12)
        plt.tight_layout()
        plt.savefig('HagforsImage')
        plt.show()
    if plotSRPS == True:
        plt.figure()
        plt.title('Sub-radar point intervals')
        plt.scatter(ObsSubLong,ObsSubLat,marker ='o',c='b',\
        label='All sub-radar points',s=10)
        plt.scatter(cutforelevation(ObsSubLong),cutforelevation(ObsSubLat),\
        marker = '+',c='g',label='All sub-radar points above cutoff',s=30)
        plt.scatter(SubLong,SubLat,marker ='x',c='r',\
        label=f'Sub-radar point interval {startidx}-{endidx}',s=20)
        plt.xlabel('Longitude')
        plt.ylabel('Latitude')
        plt.legend()
        plt.show()
```

A.3 Disambiguation - Disambiguation.py

```
import pickle
import importlib
import Simulation as func
importlib.reload(func)
import matplotlib.pyplot as plt
import scipy.sparse as sparse
import scipy.sparse.linalg as linalg
import numpy as np
from PIL import Image
from tqdm import tqdm
from quadtreecurrent import QuadTree, Point, Rect
import matplotlib.patches as ptc
from matplotlib.collections import PatchCollection

"""Load all pickle files"""
coor0_5 = pickle.load(open("Coordinates 0-5", "rb"))
bright0_5 = pickle.load(open("Brightness-value 0-5", "rb"))
rd05 = pickle.load(open("RD 0-5","rb"))
coor6_11 = pickle.load(open("Coordinates 6-11", "rb"))
bright6_11 = pickle.load(open("Brightness-value 6-11", "rb"))
rd611 = pickle.load(open("RD 6-11","rb"))
coor75_80 = pickle.load(open("Coordinates 75-80", "rb"))
bright75_80 = pickle.load(open("Brightness-value 75-80", "rb"))
rd7580 = pickle.load(open("RD 75-80","rb"))
coor60_65 = pickle.load(open("Coordinates 60-65", "rb"))
bright60_65 = pickle.load(open("Brightness-value 60-65", "rb"))
rd6065 = pickle.load(open("RD 60-65","rb"))
coor65_70 = pickle.load(open("Coordinates 65-70", "rb"))
bright65_70 = pickle.load(open("Brightness-value 65-70", "rb"))
rd6570 = pickle.load(open("RD 65-70","rb"))
coor102_107 = pickle.load(open("Coordinates 102-107", "rb"))
bright102_107 = pickle.load(open("Brightness-value 102-107", "rb"))
rd102107 = pickle.load(open("RD 102-107","rb"))

'''The number of ambiguous maps to include in the disambiguation'''
nr = 2

'''Programming switches'''
#Plot range-Doppler plot
plotrd = True
#Save range-Dopper plot (can only be saved if it is plotted)
saverd = True
#Plot selenographic plot
plotseleno = True
#Save selenographic plot (can only be saved if it is plotted)
saveseleno = True
#Run disambiguation
rundisamb = True
#Plot disambiguated image
plotdisamb = True
#Save disambiguated image (can only be saved if it is plotted)
savedisamb = True
#Plot the quadtree structure
plotquadtree = True
#Save quadtree plot (can only be saved if it is plotted)
savequadtree = True
#Find error variance
runerror = True
```

```
#Plot error
ploterror = True
#Save error plot (can only be saved if it is plotted)
saveerror = True
'''
Function to plot range-Doppler maps.
---------------------------------------------------------------------
INPUT: - rd: Range-Doppler coordinates (nested list of floats/ints)
       - bright: Brightness values (list of floats/ints)
       - filename: Filename to save the image (str)
       - save: if save = True the image is saved. if False it is not
---------------------------------------------------------------------
'''
def plotrangedoppler(rd, bright,filename,save):
    r = rd[0]
    d = rd[1]
    plt.figure()
    plt.scatter(d,r,c=bright,s=1, cmap = 'Greys_r')
    plt.xlabel("Doppler frequency shift [Hz]")
    plt.ylabel('Range [km]')
    if save == True:
        plt.savefig(filename,dpi=400)
'''
Function to plot selenographic maps.
---------------------------------------------------------------------
INPUT: - coor: Selenographic coordinates where every second element
               is the positive and negative symmetric coordinate.
               (list of floats/ints)
       - bright: Brightness values (list of floats/ints)
       - filename: Filename to save the image (str)
       - save: if save = True the image is saved. if False it is not
---------------------------------------------------------------------
'''
def plotselenographic(coor, bright, filename, save):
    poscoor = coor[::2]
    negcoor = coor[1::2]
    poslong,poslat = zip(*poscoor)
    neglong,neglat = zip(*negcoor)
    plt.figure()
    plt.scatter(poslong,poslat,s=1,c= bright,cmap = 'Greys_r')
    plt.scatter(neglong,neglat,s=1,c=bright,cmap = 'Greys_r')
    plt.yticks(np.arange(-90,91,30)[::-1])
    plt.xticks(np.arange(-180,181,60))
    plt.xlabel('Longitude [deg]')
    plt.ylabel('Latitude [deg]')
    if save == True:
        plt.savefig(filename,dpi = 400)
if plotrd == True:
    print('Running range-Doppler plots')
    plotrangedoppler(rd05,bright0_5,'RD 0-5',saverd)
    plotrangedoppler(rd611,bright6_11,'RD 6-11',saverd)
    plotrangedoppler(rd7580,bright75_80,'RD 75-80',saverd)
    plotrangedoppler(rd6065,bright60_65,'RD 60-65',saverd)
    plotrangedoppler(rd6570,bright65_70,'RD 65-70',saverd)
    plotrangedoppler(rd102107,bright102_107,'RD 102-107',saverd)
    print('Range-Doppler plot finished')
if plotseleno == True:
    print('Running Selenographic plot')
    plotselenographic(coor0_5,bright0_5,'seleno05',saveseleno)
    plotselenographic(coor6_11,bright6_11,'seleno611',saveseleno)
    plotselenographic(coor75_80,bright75_80,'seleno7580',saveseleno)
```

```
    plotselenographic(coor60_65,bright60_65,'seleno6065',saveseleno)
    plotselenographic(coor65_70,bright65_70,'seleno6570',saveseleno)
    plotselenographic(coor102_107,bright102_107,'seleno102107',saveseleno)
    print('Selenographic plot finished')
"""
Recursive function that finds all the "children nodes" by looking through the
quadtree. If the tree is divisible, it finds the "children" pixels until it is
at the "bottom". An empty input bondary list gets filled with "children" pixels
through the recursive function.
-------------------------------------------------------------------------------
INPUT:  - tree:       QuadTree
        - boundaries: An empty list

OUTPUT: A full boundary list
-------------------------------------------------------------------------------
"""
def FindLowestBoundaries(tree, boundaries):
    if tree.divided:
        FindLowestBoundaries(tree.ne, boundaries)
        FindLowestBoundaries(tree.nw, boundaries)
        FindLowestBoundaries(tree.sw, boundaries)
        FindLowestBoundaries(tree.se, boundaries)
    else:
        boundaries.append(tree.boundary)
"""
Function that runs the quadtree algorithm, creates the disambiguation matrix
and solve the overdetermined linear least square system.
-------------------------------------------------------------------------------
INPUT:  - coors:        List of all the coordinate values on the form with
                        every second element being the positive and negative
                        symmetric coordinate. (Nested list of floats/ints)
        - measurements: List of all the measurements corresponding to the
                        coordinate list. (list of floats/ints)
OUTPUT: - boundaries:   List of all the pixel boundaries (list)
        - sigma_est:    Estimated true brightness value for each pixel solved
                        by linear least squares.
-------------------------------------------------------------------------------
"""
def DoQtree(coors, measurements, plotqtree, save):
    # Define the image boundaries
    grid_domain = Rect(0, 0, 360.1, 180.1)
    # Inizialize quadtree for the domain defined above where each rectangle can
    # contain maximum 20 points.
    quadtree = QuadTree(grid_domain, 20)
    # Inizialize the points with their index
    points = [Point(*coors[i], i) for i in range(len(coors))]
    # Insert these points into the quadtree.
    for point in points:
        quadtree.insert(point)
    # Find all "boundary nodes", thus the "lowest" rectangles by  going through
    # each node in the quadtree
    boundaries = []
    FindLowestBoundaries(quadtree, boundaries)
    print(f'Number of rectangles: {len(boundaries)}')
    # list of measurement indexes
    pointsin = []
    m_idx = []
    # list of pixel indexes
    pix_idx = []
    # Loop through each child rectangle
    for i, pix in tqdm(
```

```
        enumerate(boundaries),
        total=len(boundaries),
        desc="Find matrix indexes",
        position=0,
        leave=True,):
        # Find the points within each rectangle/pixel
        points_in_pixel = []
        quadtree.query(pix, points_in_pixel)
        pointsin.append(points_in_pixel)
        # For each point in each pixel: The m index is the every second payload
        # of each point and the pixel index is the number of the pixel
        for point in points_in_pixel:
            m_idx.append(point.payload // 2)
            pix_idx.append(i)
    # Create pixel list
    Pix = [1] * len(pix_idx)
    # Create matrix
    Amat = sparse.coo_matrix((Pix, (m_idx, pix_idx)))
    # Convert matrix to csr to compress the matrix
    amatcsr = Amat.tocsr()
    # Solve matrix for the list of measurements
    sigma_est = sparse.linalg.lsqr(amatcsr, measurements)[0]
    print(f'Pixels = {len(Pix)}, Measurements = {len(m_idx)}, \
    sigma = {len(sigma_est)}')
    # Plot the quadtree grid
    if plotqtree == True:
        fig, ax = plt.subplots()
        for rectangles in boundaries:
            rectangles.draw(ax)
        ax.set_xlim(-180,180)
        ax.set_ylim(-90,90)
        ax.set_yticks(np.arange(-90,91,30)[::-1])
        ax.set_xticks(np.arange(-180,181,60))
        ax.set_xlabel("Longitude [deg]")
        ax.set_ylabel("Latitude [deg]")
        plt.show()
        if save == True:
            fig.savefig(f"Quadtree{nr}stk", dpi=600)
    return boundaries, sigma_est, pointsin
"""
Function to plot the true normalized scattering cross section in the quadtree
pixel grid.
-------------------------------------------------------------------------------
INPUT:  - boundaries: Pixel boundary list (list)
        - measurements: List of all the true normalized scattering cross
                        sections (list of floats)
        - save: if True: save the image, if False: only plot
-------------------------------------------------------------------------------
"""
def plot_disambiguous_im(boundaries, sigma, save):
    fig, ax = plt.subplots()
    patches = []
    #define  each pixel and insert the estimated sigma value
    for pixel in tqdm(
        boundaries,
        total=len(boundaries),
        desc="Plot disambiguated image",
        position=0,
        leave=True):
        xy = (pixel.west_edge, pixel.north_edge)
        width = pixel.east_edge - pixel.west_edge
```

```
        height = pixel.south_edge - pixel.north_edge
        patches.append(ptc.Rectangle(xy, width, height))
    collection = PatchCollection(patches,cmap='Greys_r')
    collection.set_array(sigma)
    collection.set_clim([0, 256])
    ax.add_collection(collection)
    ax.set_xlim(-180,180)
    ax.set_ylim(-90,90)
    ax.set_yticks(np.arange(-90,91,30)[::-1])
    ax.set_xticks(np.arange(-180,181,60))
    ax.set_ylabel("Latitude [deg]")
    ax.set_xlabel("Longitude [deg]")
    plt.show()
    if save == True:
        fig.savefig(f'Disambiguated{nr}stk',dpi = 400)
"""
Function to caluclate the error variance between the disambiguated image and
the optical image.
--------------------------------------------------------------------------------
INPUT:  - sigma: estimated values from the disambiguation (list of floats)
        - points: the coordinates in the quadtree (nested list)
        - number: number of maps in the disambiguation (int)
        - saveerror: if True: save the image,if False: do not save (bool)
--------------------------------------------------------------------------------
"""
def SampleError(sigma,points,number,saveerror):
    imfilename = 'lroc3.jpg'
    image = Image.open(imfilename)
    imarray = np.array(image)[:,:,1]
    diff = len(points)-len(sigma)
    print(f'Number of maps: {number}\nLength of sigma list: {len(sigma)}\
    \nNumber of points: {len(points)}\nDiff: {diff}')
    pix = [c for c in points[:-diff]]
    #Finds the average brightness from the optical image in each quadtree pixel
    findavbright = []
    for coor in tqdm(pix,total=len(pix),desc='Loop through pixels',\
    position = 0,leave=True):
        bright = []
        for point in coor:
            loidx,laidx = func.ImageCoordinates(point.x, point.y, imarray)
            b = float(imarray[laidx,loidx])
            bright.append(b)
        findavbright.append(np.mean(bright))
    #Calculate the error variance and standard deviation
    var =  np.round(np.nanvar(findavbright-sigma),2)
    std = np.round(np.nanstd(findavbright-sigma),2)
    print(f'{number}stk: var: {var}\n std: {std} \nstd in percentage: \
    {np.round((std/np.nanmean(findavbright))*100),2}% of the average reflectivity
↪   value')
    #Plot error together with a linear function.
    if ploterror == True:
        def linfunc(x,a,b):
            return a+b*x
        fit = [linfunc(x,0,1) for x in np.arange(0,256,1)]
        plt.figure()
        plt.plot(fit,c='r')
        plt.scatter(findavbright,sigma,s=0.7,c='black',alpha=0.3)
        plt.xlabel('Real values')
        plt.ylabel('Estimated values')
        if saveerror == True:
            plt.savefig(f'Errorplot {number}stk',dpi=300)
```

```
        plt.show()
    return findavbright,sigma
if rundisamb == True:
    if nr ==2:
        coors = coor0_5 + coor6_11
        measurements = bright0_5 + bright6_11
    if nr == 3:
        coors = coor0_5 + coor6_11 + coor75_80
        measurements = bright0_5 + bright6_11 + bright75_80
    elif nr == 4:
        coors = coor0_5 + coor6_11 + coor75_80 + coor60_65
        measurements = bright0_5 + bright6_11 + bright75_80 + bright60_65
    elif nr == 5:
        coors = coor0_5 + coor6_11 + coor75_80 + coor60_65 + coor65_70
        measurements = bright0_5 + bright6_11 + bright75_80 + bright60_65 + \
        bright65_70
    elif nr == 6:
        coors = coor0_5 + coor6_11 + coor75_80 + coor60_65 + coor65_70 + \
        coor102_107
        measurements = bright0_5 + bright6_11 + bright75_80 + bright60_65 + \
        bright65_70  + bright102_107
    print('start')
    boundaries, sigma_est,pointsin = DoQtree(coors, measurements, \
    plotquadtree, savequadtree)
    if plotdisamb == True:
        plot_disambiguous_im(boundaries, sigma_est, savedisamb)
    if savedisamb == True:
        pickle.dump(boundaries,open(f'boundaries{nr}stk','wb'))
        pickle.dump(sigma_est,open(f'sigma{nr}stk','wb'))
        pickle.dump(pointsin, open(f'pointsin{nr}stk', 'wb'))
    if runerror == True:
        findavbright,sigmaest = SampleError(sigma_est,pointsin,nr,saveerror)
```

Bibliography

[1] James K. Breakall. *History of Antenna Technology at the Arecibo Observatory in Arecibo, Puerto Rico [Historically Speaking]*. *IEEE Antennas and Propagation Magazine*, 63(2):103–105, 2021.

[2] Bruce A. Campbell. *Radar Remote Sensing of Planetary Surfaces*. Cambridge University Press, 2002. [ISBN: 9780521189651].

[3] Bruce A. Campbell and Donald B. Campbell. *Arecibo Radar Maps of Venus from 1988 to 2020*. *The Planetary Science Journal*, 3(3):55, March 2022.

[4] Bruce A. Campbell, Donald B. Campbell, J. L. Margot, Rebecca R. Ghent, Michael Nolan, John Chandler, Lynn M. Carter, and Nicholas J. S. Stacy. *Focused 70-cm Wavelength Radar Mapping of the Moon*. `https://doi.org/10.1109/TGRS.2007.906582`, 2007. [Online; ISSN: 1558-0644].

[5] Christian Hill. *Quadtrees 2: Implementation in Python*. `https://scipython.com/blog/quadtrees-2-implementation-in-python/`, 2020. [Online; accessed 28-Febuary-2022].

[6] E.Speyerer et. al. *A New Lunar Atlas: Mapping the Moon with the Wide Angle Camera*. `https://quickmap.lroc.asu.edu`, 2012. [Online; accessed 28-February-2022].

[7] Vijay Kumar Garg. *Wireless communications and networking*. Elsevier Morgan Kaufmann, 2007. [First edition. ISBN : 9786611119058].

[8] Humboldt State University. *History of Remote Sensing — Aerial Photography*. `http://gsp.humboldt.edu/olm/Courses/GSP_216/online/lesson1/history.html`, 2020. [Online; accessed 20-April-2022].

[9] R.J. James. *A history of radar*. *IEE Review*, 35(9):343–349, 1989. [ISSN: 0953-5683].

[10] John V Evans and Tor Hagfors. *Radar Astronomy*. 1968. [Page 237. Online: https://archive.org].

[11] Matt J. Jones, Alexander J. Evans, Brandon C. Johnson, Matthew B. Weller, Jeffrey C. Andrews-Hanna, Sonia M. Tikoo, and James T. Keane. *A South Pole-Aitken impact origin of the lunar compositional asymmetry*. *Science Advances*, 8(14), 2022.

[12] B. Kaifler and N. Kaifler. *A Compact Rayleigh Autonomous Lidar (CORAL) for the middle atmosphere*. *Atmospheric Measurement Techniques*, 14(2):1715–1732, 2021.

[13] NASA-Jet Propulsion Laboratory. *Horizon system*. https://ssd.jpl.nasa.gov/orbits.html. [Online; accessed 20-January-2022].

[14] Shawn R. German Mervin C. Budge. *Basic Radar Analysis*. Artech House, 2020. [Second edition. ISBN: 9781630815554].

[15] Reinaldo Perez. *Wireless Communications Design Handbook, Chapter 7 - Satellite Antennas*. https://doi.org/10.1016/S1874-6101(99)80020-9, 1998. [Online; ISSN: 1874-6101].

[16] National research council of the national academies. *The scientific context for exploration of the moon*. https://doi.org/10.17226/11954, 2007. [Online; ISBN: 978-0-309-18576-9].

[17] A. E. E. Rogers and R. P. Ingalls. *Venus: Mapping the Surface Reflectivity by Radar Interferometry*. *Science*, 165(3895):797–799, 1969.

[18] Peter J Schreier and Louis L Scharf. *Statistical signal processing of complex-valued data: the theory of improper and noncircular signals*. Cambridge university press, 2010.

[19] Shapiro, I. I. *Theory of the radar determination of planetary rotations*. https://adsabs.harvard.edu/full/1967AJ.....72.1309S, 1967. [Online; accessed 09-May-2022].

[20] Merrill Skolnik. *Radar handbook*. McGraw-Hill Professional, 2008. [Third edition. ISBN: 9780071485470].

[21] T. Kaku, J. Haruyama, W. Miyake, A. Kumamoto, K. Ishiyama, T. Nishibori, K. Yamamoto, Sarah T. Crites, T. Michikami, Y. Yokota, R. Sood, H. J. Melosh, L. Chappaz, K. C. Howell. *Detection of Intact Lava Tubes at Marius Hills on the Moon by SELENE (Kaguya) Lunar Radar Sounder*.

Geophysical research letters, 2017. [ISSN: 0094-8276 , 1944-8007].

[22] Junfeng Wang and Xingzhao Liu. *Automatic Correction of Range Migration in SAR Imaging. IEEE Geoscience and remote sensing letters*, 4, 2008. [ISSN:2153-7003].

[23] Jarosław Włodarczyk. *The pre-telescopic observations of the Moon in early seventeenth-century London: The case of Edward Gresham (1565–1613)*. `https://doi.org/10.1098/rsnr.2019.0009`, 2020. [Online; ISSN: 1743-0178].