Imperial College London

Microwave Imaging of the Sun with Machine Learning

Author

Louise Davis

CID: 01909253

Supervised by

Prof. Stepan Lucyszyn

Second Marker

Prof W. T. Pike

A Thesis submitted in fulfillment of requirements for the degree of
Master of Engineering in Electronic and Information Engineering

Department of Electrical and Electronic Engineering
Imperial College London

2024

Abstract

This thesis investigates the development and implementation of a low-fidelity radio astronomy system for imaging the sun using machine learning techniques to enhance resolution. The primary focus is on converting a standard TV satellite dish into a functional radio telescope capable of capturing solar images. The study includes detailed discussions on the setup and integration of key components such as the low noise block (LNB) converter, software defined radio (SDR), and Raspberry Pi for system control and data acquisition. Part of the research is dedicated to enhancing the captured images using super-resolution convolutional neural network (SRCNN). Various SRCNN models were trained and tested to improve image quality, addressing issues such as noise reduction and resolution enhancement. The results demonstrate the feasibility of using consumer-grade equipment for educational and preliminary research in radio astronomy and provide a replicable implementation. Additionally, the study explores the challenges and limitations of this approach, providing insights for future developments in low-cost radio astronomy systems.

Declaration of Originality

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.

I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.

I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me, but is represented as my own work.

I have used ChatGPT v4 as an aid in the preparation of my report. I have used it to improve the quality of my English occasionally, however all technical content and references comes from my original text.

Copyright Declaration

Acknowledgments

I would like to thank my supervisor, Prof. Stepan Lucyszyn, for his guidance and support throughout this project. I appreciate his assistance and the valuable insights he has provided.

Contents

List of Acronyms

EM electromagnetic

ALMA Atacama Large Millimeter/submillimeter Array

ARRMS Automated RF & Microwave Measurement Society

CMEs coronal mass ejections

LNB low noise block

RF radio frequency

IF intermediate frequency

SNR signal to noise ratio

SGD stochastic gradient descent

ADC Analogue to Digital Converter

Google Colab Google Colaboratory

ML machine learning

UROP Undergraduate Research Opportunities Programme

SR super-resolution

CNN convolutional neural network

SSIM structural similarity

STB set-top box

SDR software defined radio

DSP digital signal processing

HPBW half-power beamwidth

BPF bandpass filter

LNA low noise amplifier

FCD FUNCube dongle

LO local oscillator

DC direct current

NF noise figure

F noise factor

ADC analog-to-digital converter

USB Universal Serial Bus

VHF Very High Frequency

UHF Ultra High Frequency

LF Low Frequency

MF Medium Frequency

HF High Frequency

IQ In-phase Quadrature

FFT fast Fourier transform

AC alternating current

LPF low-pass filter

2D two-dimensional

PV Photovoltaic

CSP Concentrated Solar Power

CCD Charge-Coupled Device

GPS Global Positioning System

ReLU Rectified Linear Unit

GAN Generative Adversarial Network

SRCNN super-resolution convolutional neural network

LRS learning rate scheduler

MSE Mean Squared Error

PSNR Peak Signal-to-Noise Ratio

PC personal computer

SCART Syndicat des Constructeurs d'Appareils Radiorécepteurs et Téléviseurs

TV television

AV Audio Visual

HDMI High-Definition Multimedia Interface

TP transponder

RS232 Recommended Standard 232

SSH Secure Shell

RC resistor and capacitor

OOP Object Oriented Programming

GPU Graphics Processing Unit

RGB Red, Green, Blue

I In-phase

Q Quadrature

RMS root-mean-square

PPM parts per million

SRGAN super-resolution generative adversarial network

SRRes super-resolution residual network

FPGA Field Programmable Gate Array

Adam Adaptive Moment Estimation

Nomenclature

Constants and Variables

D	Antenna diameter (m)	m
λ	Wavelength	m
k	Factor of antenna dependent on the reflector shape and feed illumination pattern	$^\circ$
G_A	Antenna gain	
$HPBW$	Half Power Beam Width	$^\circ$
e_A	Angular diameter of observed object	$^\circ$
σ_T	Noise temperature	K
τ	Time constant	s
B	Bandwidth	MHz
c	Speed of light	ms^{-1}
F	Noise figure	db
f	Frequency	GHz
f_s	Sampling rate	MSPS
$f_{\max}$	Maximum observed frequency	GHz
I_f	Spectral brightness	$\frac{\mathrm{W}}{\mathrm{m}^2\mathrm{Hzsr}}$
k_B	Boltzmann Constant	J/K
T_b	Brightness temperature	K
T_s	System noise temperature	K

1 Introduction

Contents

1.1 Introduction

1.1.1 Radio Astronomy

Radio astronomy is the study of the emission of celestial bodies in the radio region of the electromagnetic (EM) spectrum, with the purpose of better understanding the structure and origin of our universe. Naturally occurring radio waves are emitted from stars, planets, galaxies, etc. The observed radio brightness temperature can be used to infer the physical temperature of the emitting gas. In a similar way to visual imaging, the radio imaging of the universe can create a picture of the objects that remain undetectable to the human eye. This is often used to study dying stars, emissions from gas giants, and parts of the centre of galaxies.

Astronomers capture images of the Sun to study phenomena such as sunspots, solar flares, prominences, and coronal mass ejections (CMEs), which help understand the Sun's magnetic field, energy transfer, and solar atmosphere dynamics. Analysing these images allows scientists to predict space weather events affecting Earth, improve solar activity models, and gain insights into processes occurring in other stars. High-resolution images provide valuable data for researching solar heating, plasma behaviour, and the Sun's impact on the solar system. The time required to capture an image significantly impacts the type of data observed, as phenomena range from brief events like solar radio bursts (seconds to minutes) and solar flares (minutes to hours) to longer-lasting activities such as coronal mass ejections (hours to days) and sunspots (days to months). Understanding these phenomena is crucial for comprehending solar activity and its effects on space weather.

Current methods to capture images of high spatial resolution and noise sensitivity of celestial objects involve radio telescopes of vast sizes. For instance, the Atacama Large Millimeter/submillimeter Array (ALMA) is an astronomical interferometer consisting of 66 radio telescopes located in the Atacama Desert in northern Chile. The 12 m diameter satellite dishes work in combination to provide data with higher angular precision (see Figure. 1.1).

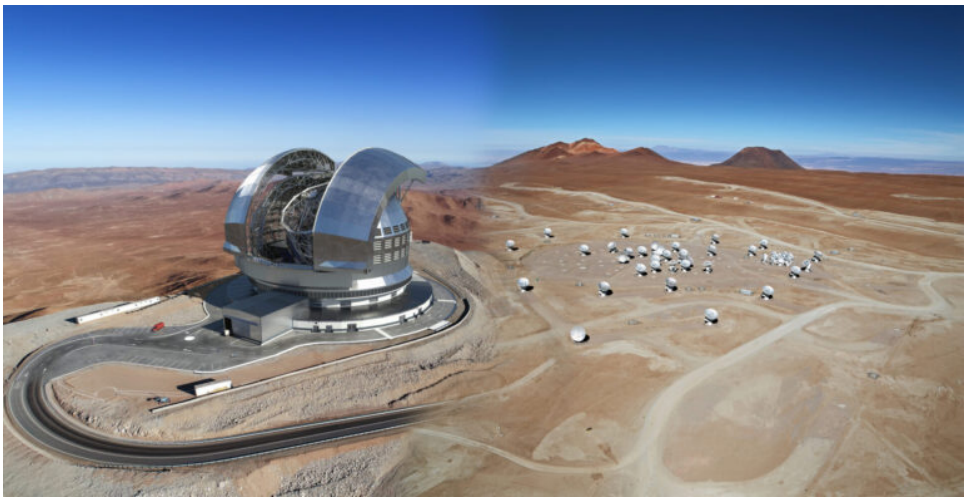

Figure 1.1: ALMA Telescope [1]

The configuration of ALMA's widely spaced satellite dishes enables the capture of high-resolution images. To increase the detail further, the data can be combined with data from the Hubble Space Telescope, creating images as illustrated in Figure 1.2.

The Hubble image is the sharpest view of this object ever taken and serves as the ultimate benchmark in terms of resolution. In this instance, ALMA captures radio emissions from molecular gas (shown in red/orange) in two galaxies, while the Hubble Space Telescope collects optical data of the stars (depicted in white/blue). ALMA observes at much longer wavelengths which makes it much harder to obtain comparably sharp images. However, when the full ALMA array is completed its vision will be up to ten times sharper than Hubble.

Figure 1.2: The Antennae Galaxies (**NGC!** 4038 and 4039) are colliding spiral galaxies about 70 Million light-years away in the constellation Corvus. This image combines ALMA observations in two wavelength ranges with visible light data from the Hubble Space Telescope. [2]

Although high spatial resolution and noise sensitivity are achieved, the installation and data-capture processes are highly complex. The array of antenna covers

6,600 square meters and can be positioned up to 16 kilometres apart. With a construction and running cost of approximately US$1.4Billion, it is the most expensive ground-based telescope currently in operation.

1.1.2 Low Fidelity Systems

Low-fidelity radio astronomy systems are simpler, less expensive, and typically less sensitive than high-fidelity systems. They are often used in educational settings, amateur astronomy, or preliminary research where advanced capabilities and high precision are not critical [3] [4].

Raster scanning and interferometry are two key techniques in both low and high fidelity radio astronomy. As explored by Mangum *et al.* [5], raster scanning uses a single dish to systematically move across the sky in a grid pattern, collecting data point-by-point, which is simpler but offers lower-resolution and is time-consuming. In contrast, interferometry uses multiple widely separated antennas to observe the same object simultaneously [6], combining their signals to produce high-resolution images. This method is more complex and expensive but provides superior resolution and efficiency. Both techniques are essential, with raster scanning suited for detailed regional studies and interferometry ideal for capturing fine details of distant celestial phenomena.

Multiple attempts have been made to achieve radio-wave images using smaller antennas, fewer antennas, or more cost-effective equipment overall. Fadul *et al.* [7] demonstrates a highly simplified system including a 1.2 m satellite dish and low-cost equipment to capture images of the sun. A single antenna possesses low angular resolution, but effectively demonstrates how a TV satellite dish can be converted into a small radio telescope, providing a cost-effective solution for preliminary radio astronomy research.

Alternatively, [6] has explored the possibility of obtaining two-dimensional radio images of the sun using an interferometry installation using two commercial dish TV antennas and a minimal budget. Although the work is still in progress to fine tune the pointing accuracy of the dishes and develop a Field Programmable Gate Array (FPGA) based spectrocorrelator to observe over the entire 10.7 to 11.7GHz frequency band, they demonstrate a successful system to produce meaningful solar data.

Several attempts have been explored to introduce tracking methods in radio astronomy to automate the capture of images and data collection. Notably, Cook [8] implemented a Moon tracking system as part of a Undergraduate Research Opportunities Programme (UROP) with Prof. Stepan Lucyszyn, which precisely aligns a rotating system to the Moon with refinement through image taking and a CNN to centre the moon in the image.

A significant step in defining a low-budget setup for radio astronomy was made by Rawlinson in 2013 through his project to implement a satellite dish radio receiver [9]. It was first published as an article in the first issue of the "RAGazine" [10] in September 2013 and then later presented at a Automated RF & Microwave Measurement Society (ARRMS) conference [11] in November 2013 lead by Professor Stepan Lucyszyn. Rawlinson uses off-the-shelf satellite equipment and a single axis rotation to perform a raster scan on the Sun. Much of this project is based on Rawlinson's work, which supplies a more detailed and replicable method to low fidelity radio astronomy, and explores developments to his implementation.

Over the past 50 years, many advancements have been made in radio astronomy and celestial imaging. Innovations in data processing and big data analytics have allowed astronomers to handle the vast amounts of data generated by modern radio telescopes, leading to more precise and comprehensive observations [12]. Recently, the rapid development of machine learning methods and convolutional

neural network (CNN) has unlocked further methods in data processing and image enhancement.

super-resolution (SR) is a technique that enhances the resolution of images beyond the limitations of imaging systems, often using methods like interpolation, machine learning (ML), and reconstruction. SR has developed rapidly and is common in image enhancement, it also has a lot of potential in radio imaging. In [13], Schmidt *et al.* use a deep-learning CNN to reduce the "dirty noise" in the images taken through interferometry. In [14], Ledig *et al.* produce a super-resolution generative adversarial network (SRGAN) and super-resolution residual network (SRRes) networks which produce higher performance than the SRCNN in [15]. Honma uses methods based on sparse modelling for astronomical images, particularly those obtained through interferometric imaging to enhance images of black holes [16]. While these networks perform well, they require extensive resources to learn and time to train. However, simple CNN's are now common and once the computationally expensive training of a CNN is complete it can easily be loaded by uses and adopted by the wider public.

1.2 Project Specification

1.2.1 Objectives

The aim of this project is to design and develop a cost-effective system for capturing solar microwave intensity images of the Sun at microwave frequencies using commercial satellite dishes. This project builds upon the work of Rawlinson by incorporating a tracking system and machine learning methods to enhance automation and robustness. The project seeks to contribute to existing research by providing comprehensive implementation details to facilitate replicability. Furthermore, it in-

vestigates the feasibility of using a single, small-sized satellite dish as an alternative to interferometry methods. The project is carried out through four main objectives:

- Establish a data observation system and identify the necessary variables.
- Develop a control system to automatically track the Sun's position and perform raster scanning for data collection.
- Create an image from the data collected during the raster scanning.
- Train and apply a neural network to increase the resolution of the Sun's microwave image.

These objectives aim to enhance the use of cheaper and smaller radio telescopes while still formulating accurate and higher-resolution images, thus making the technology more readily accessible for research, education or low-cost applications.

1.2.2 Setup

The setup will receive microwave emission through a 0.35 m satellite dishes combined with LNB down-converters for radio frequency (RF) collection and converting the RF into an intermediate frequency (IF) transition through a coaxial cable to a set-top box (STB) (Comag digital satellite receiver (DVB-S SL 65/12). Utilising the IF output on the STB, the signal will be further relayed to a SDR to improve signal to noise ratio (SNR) by integrating over a calibrated time and digitise the received signal for computational processing. To maximise the quality of the image, an azimuth and elevation Sun tracking system combined with a vertical raster scanning pattern will be implemented which will also aid with reconstructing the image based on the intensity of each scan line. Finally, a SR using a CNN technique will be created and applied to the microwave intensity image to increase the resolution to visualise the Sun's phenomena.

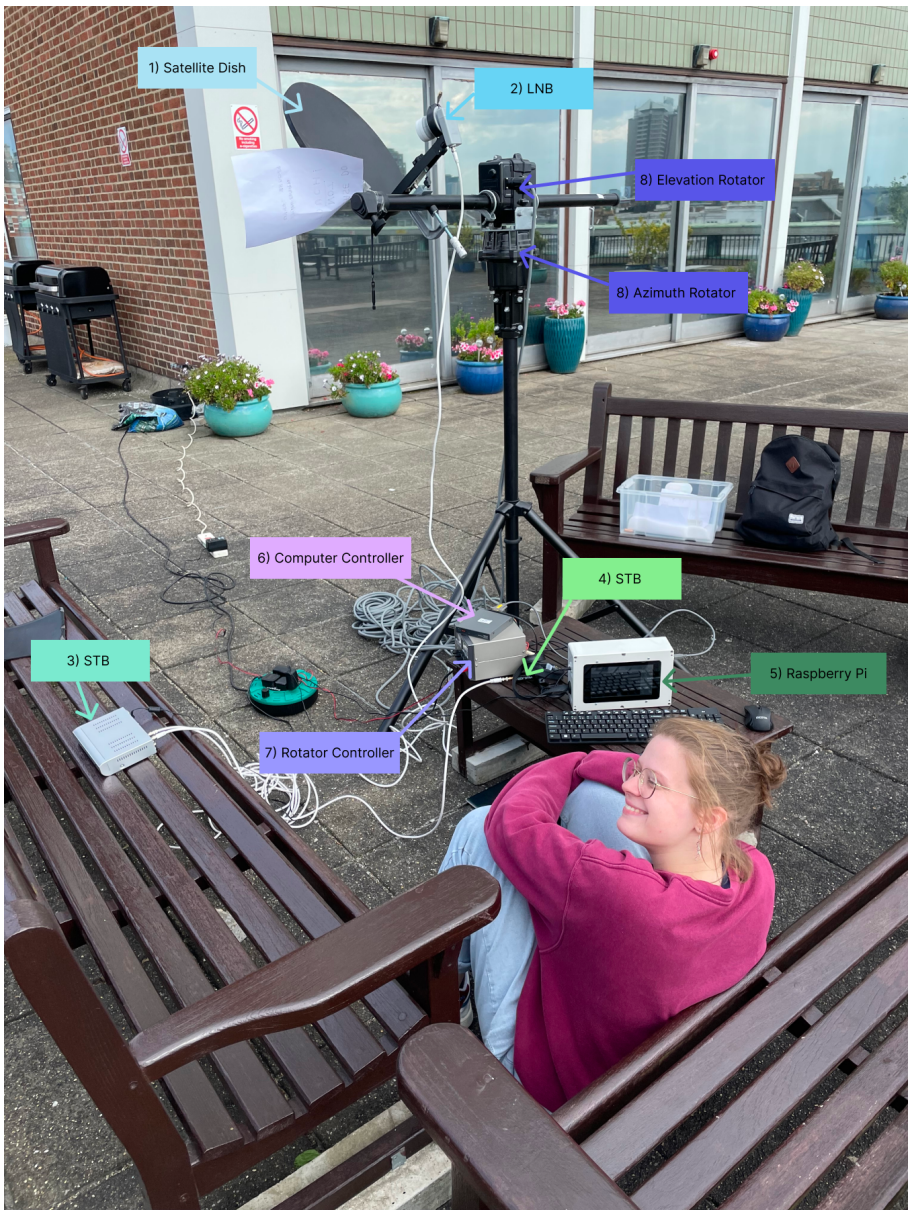

Figure 1.3: Setup of collection system.

The control tracking algorithm and image processing will be implemented on a Raspberry Pi 3 Model B [17]. The satellite dish will be mounted on a pole controlled by rotators which will be controlled by a Python script running on the Raspberry Pi. The Raspberry Pi also communicates with a computer controller [18] [19], that converts the angle of rotation from the script to the azimuth and elevation dual controller unit [20] language. This angle of rotation is then received by the dual

controller and operates the rotators movement. The rotators a fixed atop a tripod, with a metal rod attached through the elevation rotator. The digitised signal from the SDR will be looped back to the Raspberry Pi to coordinate with the raster scan lines to formulate a microwave intensity image.

The formulated image will then be transferred to a separate personal computer (PC), due to the high complexity and computation resources that the CNN will require. This follows a similar method made in [paper], where a SRCNN will perform SR on the microwave intensity image. This is written in Python using the resources of Google Colaboratory (Google Colab) [21].

1.3 Report Structure

The structure of this report is organised as follows: Chapter 2 reviews the relevant background and foundational knowledge in radio astronomy and image processing. Chapter 3 outlines the project's methodology, including details on the hardware setup and software development. Chapter 4 presents the project's results, focusing on the outcomes of the image processing. Chapter 5 continues with the presentation of results, emphasising data analysis of the system. Chapter 6 discusses various aspects of the project, its implications, and potential improvements. Finally, Chapter 7 concludes the report, summarising the key findings and suggesting directions for future work.

2

Background

Contents

There are four major parts to this project's background. The first section covers the conceptual and functional knowledge about the satellite dish, LNB and STB. Following this is the digital signal processing (DSP) of the input IF signal to the SDR and image creation, with the controlling of the rotators and scanning pattern for

the noise capture. The final section details the various structures and architectures used in ML.

2.1 Solar Observation Equipment

The solar observation equipment consists of the satellite dish, LNB (from a digital mini satellite system box) and SDR. Each component affects the capability of the system and the background theory for each component is now discussed.

2.1.1 Satellite Antenna Theory

Satellite dish systems have been used ubiquitously since the 1970s for recreational television and scientific purposes. They are designed to receive signals from orbiting satellites and consequently have formerly been typically used for receiving television broadcasts. The advent of online streaming has rendered satellite dishes almost redundant, making them an affordable option for the purpose of radio astronomy [22]. Satellite dishes have sufficient sensitivity to pick up celestial radiation including radio waves from the Big Bang, as such they are suitable for scientific observation. In order to determine how many raster scanning movements are required in the system, the half-power beamwidth (HPBW) must be calculated from standard satellite antenna theory.

In [23], Jabbar *et al.* conducted a study to investigate the dependence of a parabolic antenna's spatial resolution and power intensity on parameters including the parabolic shape, gain, beamwidth, and resolving power based on the diameter and depth of the antenna. Their analysis and findings are now summarised here.

The design of a satellite dish consists of a parabolic reflecting surface which focuses a single point where it is transduced by an LNB.

Figure 2.1 shows the power pattern of the antenna, showing how the radiation intensity varies as a function of angle incidence.

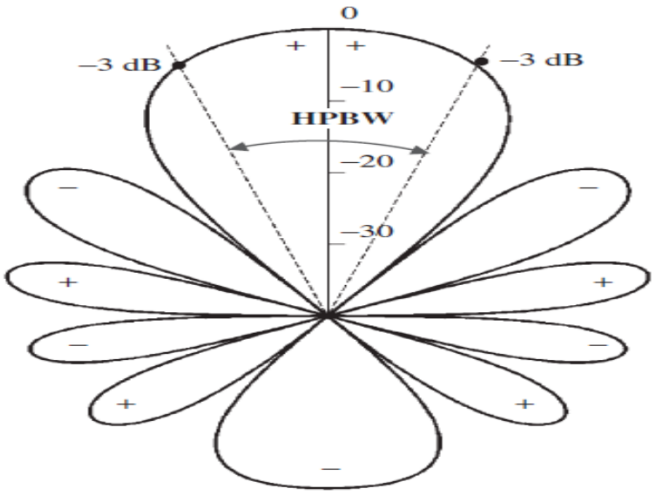

Figure 2.1: Power Pattern of the antenna [24].

The main lobe beamwidth characterises the width of the peak in radiation intensity as a function of angle incidence. One way in quantifying beamwidths is the HPBW, defined by measuring the main lobe at the half power points (−3 dB) and commonly used to define the resolution of the antenna.

The HPBW of a parabolic antenna can be calculated using the wavelength (λ) of incident radiation and the antenna diameter D as,

$$\text{HPBW} = k\frac{\lambda}{D} \tag{2.1}$$

where k is a factor that is dependent on the reflector shape and feed illumination pattern but is commonly taken to be 70°.

An alternative calculation of the HPBW based on of the gain of the antenna, G_A, [9].

$$\mathrm{G_A} = \frac{\pi^2 D^2}{\lambda^2} e_A \tag{2.2}$$

$$\mathrm{HPBW} = \sqrt{\frac{\pi^2 k^2 e_A}{\mathrm{G_A}}} \tag{2.3}$$

where e_A is the angular diameter of the observed object which describes the size of appearance of a sphere from a point of view. For the Sun, the angular diameter is approximately 0.6° [25].

Equations (2.2) and (2.3) imply that the HPBW scales inversely with antenna diameter. A smaller HPBW results in a higher angular resolution and thus increases the capability of distinguishing the fine details.

In [9], Rawlinson conducted tests with a 0.6 m and 1.2 m diameter dishes and obtained a decrease in HPBW of 2.3° to 1.5° at 12 GHz. Rawlinson went on to show that an even larger diameter dish would even further improve their system and produced higher resolution images but did not continue this investigation due to cost considerations.

In [26], Morgan conducted tests with a 3 m dish at $1,420$ MHz. The measured HPBW was 8° due to the increase in wavelength. The focus of the experiment was measuring the hydrogen emissions of the galactic plane, at this given frequency, which necessitated a larger dish.

2.1.2 Low Noise Block Converter

LNB converters are the receiving devices in the satellite television (TV) system that collect the incoming radio waves and convert them into an analogue signal to be transmitted through a coaxial cable.

The system is comprised of multiple bandpass filter (BPF)s, a low noise amplifier (LNA), a frequency mixer, a local oscillator (LO) and an IF amplifier, as shown in Figure.2.2.

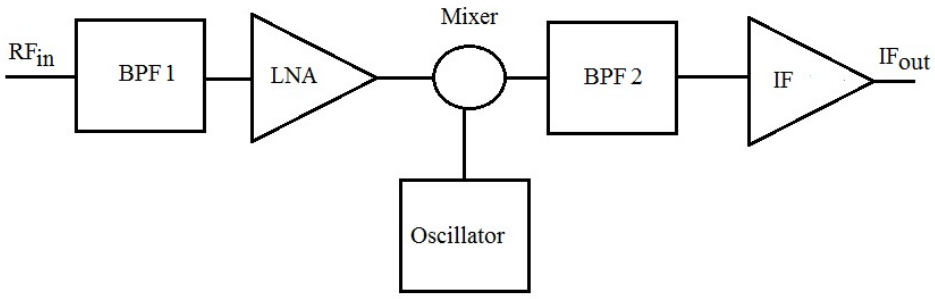

Figure 2.2: The architecture inside the LNB [27].

Each component serves a specific purpose in down-converting the RF to an IF based on the LO frequency, Eq. 2.4.

$$IF = RF - LO. \tag{2.4}$$

The incoming signal is weak and needs amplification by the LNA before any down-converting can proceed. The frequency mixers combine the amplified signal with the LNB specified LO to produce an IF by generating new frequencies, including the difference and sum of the original signal [28]. The sum terms are removed by the second BPF. Finally, the IF amplifier amplifies the output of the frequency mixer to produce the final desired IF output. LNBs will have BPFs due to the specific frequency ranges they're designed to observe.

LNBs are designed for different frequency bands (Table. 2.1). Universal LNBs are adapted to have a switchable LO, which changes depending on the whether the band of reception is low or high. Universal LNBs can receive both horizontal and vertical polarisation and the full frequency range of K_u and C band [28].

Band	Frequency Range (GHz)
L	1 to 2
S	2 to 4
C	4 to 8
X	8 to 12
K_u	12 to 18
K_a	26 to 40

Table 2.1: Different frequency bands and their ranges [29].

A direct current (DC) voltage to the LNB is required through the coaxial cable, either 13 V or 18 V, which is used to power the LNB and control which polarisation direction the LNB has, vertical or horizontal. Another element that the LNB receives through the coaxial cable is an extra tone of either 0 Hz or 22 kHz in which it indicates to the LNB which band to use, 9.75 GHz or 10.6 GHz. For the LO to be set at 10.6 GHz (high), the 22 kHz needs to be fed in otherwise the default is 9.75 GHz (low), Table 2.2.

Voltage (V)	Tone (kHz)	Local Oscillator (GHz)
13	0	9.75
18	0	9.75
13	22	10.60
18	22	10.60

Table 2.2: Different LNB settings that trigger different LO values.

The quality of celestial observations is heavily determined by the quality of the LNB due the augmentation of the gain drift. This can be seen in [9], in which Rawlinson determined that the major factor in gain instability was due to the poor quality of the LNB following which he adopted a generic K_u band LNB prime focus.

Rawlinson also suggested adding another LNA after the LNB to further amplify the down-converted signal.

Each LNB has a given noise figure (NF), equivalent to noise factor (F) in dB, which quantifies its noise performance as the ratio of SNR of the input to output, (2.5). If there were a perfect amplifier, the NF would equal 0 dB as no noise would be accrued between the input and the output.

$$F = \frac{\mathrm{SNR}_i}{\mathrm{SNR}_o} \tag{2.5}$$

2.1.3 Set-Top Box

A STB is a receiver that decodes or converts signals inputs to a display on a television set. This piece of hardware is capable to display satellite television from converting an analogue signal into a digital signal.

The STB used in this project is the digital satellite receiver (DVB-S SL 65/12), that has pre-coded satellites and programs which impacts the LNB's LO band selection, and polarisation.

The box has nine connection on the rear side corresponding to different functions, as seen in Appendix B. These are the possible output from the STB to observe the incoming IF from the LNB.

2.2 Digital Signal Processing

DSP mathematically analyses and processes a digital input, through mathematical functions. This can include voice, audio, temperature, pressure. Before DSP can be

applied, the analogue input must be digitised using an analog-to-digital converter (ADC).

2.2.1 Software Defined Radio

SDR are radios in which some or all the physical components are software-defined. They are typically designed to receive signals over the frequency range 50 Hz to 2,500 MHz without needing to change hardware components [26].

Rawlinson [9] used a SDR called a FUNCube dongle (FCD) [30] which is capable of covering a wide range of frequencies from 150 kHz to 1,900 MHz and can be combined with compatible software to visualise and analyse the radio frequency spectrum, SpectraVue [31]. It is capable of receiving radio signals so long they are within an 80 kHz bandwidth. As the FCD is costly, another version of a SDR was used.

A Nooelec v5 RTL-SDR [32] is a SDR receiver built as a Universal Serial Bus (USB) stick that can cover a range of frequencies from 100 kHz to 1,750 MHz. The RTL-SDR is a specific SDR that uses the RTL2832 ADC unit by Realtek [33]. The design of this SDR is Fig.2.3.

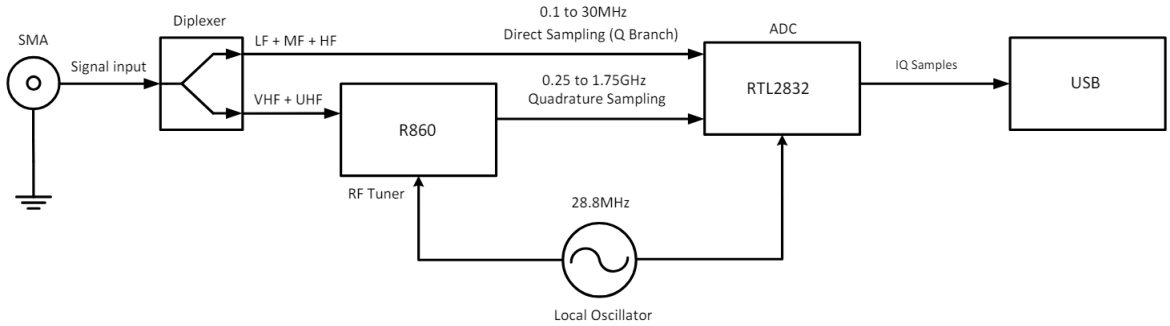

Figure 2.3: The architecture inside the Nooelec RTL-SDR.

The input signal is split by a diplexer into two branches that operate over the

different frequency ranges, shown Table. 2.3.

SDR Frequency Name	**Range**
Low Frequency (LF)	30 kHz to 300 kHz
Medium Frequency (MF)	300 kHz to 3 MHz
High Frequency (HF)	3 MHz to 30 MHz
Very High Frequency (VHF)	30 MHz to 300 MHz
Ultra High Frequency (UHF)	300 MHz to 3 GHz
Direct Sampling	0.1 MHz to 30 MHz
Quadrature Sampling	0.25 GHz to 1.75 GHz

Table 2.3: Frequency Bands in SDR [34].

The higher frequency VHF and UHF branch is fed into a RF Tuner (R860) with a 28.8 MHz LO, to produce an IF which is called the quadrature sampling branch. The other lower frequency LF, MF and HF branch has no processing before feeding into the ADC, and is called the direct sampling branch. Both of these branches are inputs to the ADC. The digitalised ADC samples are transferred to the USB connection, prepared for DSP.

The internal implementation details of the RTL2832 are proprietary and so cannot be described in detail here. Nevertheless, the ADC converts a real signal input from either branch into a complex form for each sample read. The In-phase (I) and Quadrature (Q) components are combined to form a complex signal, Eq. 2.6, which represents the RF signal in baseband, still retaining both the amplitude and phase information of the RF signal.

$$z(t) = I(t) + jQ(t) \tag{2.6}$$

Through mixing the signal with the two orthogonal LO signals, the RF sig-

nals are down-converted to baseband, which is the band centred around DC (0 Hz). Down-converting to baseband is achieved by low-pass filtering out the high-frequency components produced through mixing, only leaving the baseband frequencies, Figure. 2.4.

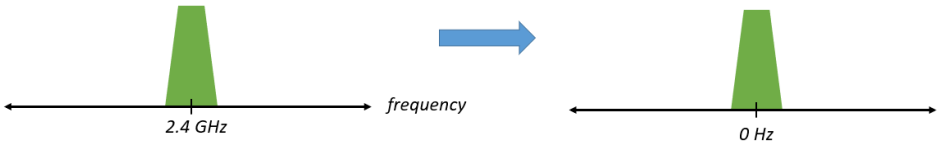

Figure 2.4: Down-conversion of signal to baseband.

By down-converting to baseband, there's no change in bandwidth, B, but the maximum frequency, $f_{\max}$, is now equal to half the bandwidth, with positive and negative frequencies present. The ADC can process the signal at a sample rate greater than or equal to the highest frequency present in the baseband signal, which satisfies Nyquist's rate, Eq. 2.7.

$$f_S > 2(\frac{B}{2}) \tag{2.7}$$

As the bandwidth is equal to the maximum frequency $f_{\max}$, Nyquist's theorem, which states the minimum sampling rate (f_s) must be greater than twice the highest input frequency, is also satisfied. This ensures preservation of all signal information and avoid aliasing. Therefore, this restricts the ADC sampling rate.

The SDR has a maximum sample rate of 3.2 MSPS and has a 7-bit ADC. Therefore it can be digitise an input IF bandwidth of 3.2 MHz.

DC offset

With SDRs, a sizeable spike in the centre of the fast Fourier transform (FFT) can occur called a DC offset (DC spike, LO leakage), Figure. 2.5, where through the process of down-converting the signal to baseband, leakage from the LO appears at the centre of the perceived bandwidth. The combination of multiple frequencies causes the creation of additional energy in the SDR. One solution to avoiding the DC offset is to over-sample and off-tune by changing the carrier and increasing the sample rate so that the wanted signal is undisturbed by the spike with the original carrier and sample rate. Another solution is to use a DC blocker after the ADC to remove the DC component.

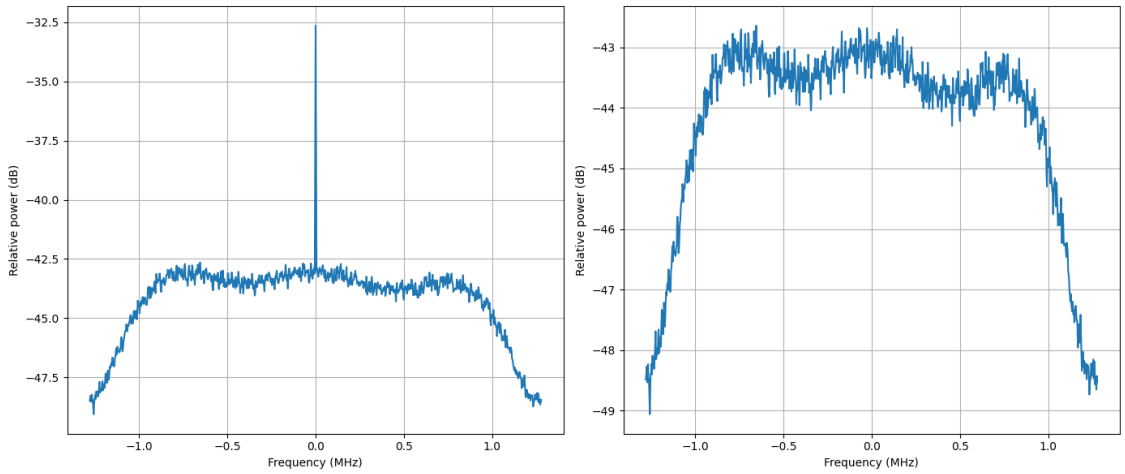

Figure 2.5: Power spectrum density with (right plot) and without (left plot) implemented DC blocker.

Bias tee

Some SDRs have a bias tee that provides power to the LNB but does not harm the SDR input with too much voltage. The bias tee allows alternating current (AC) through a capacitor but not the DC and an inductor that allows DC and not AC, Figure. 2.6.

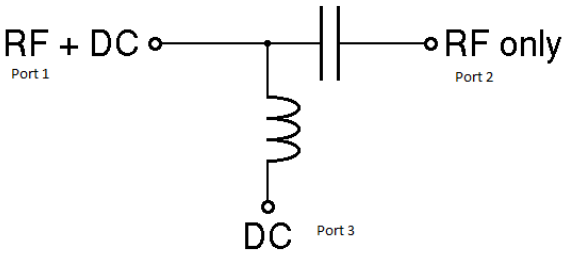

Figure 2.6: Bias tee schematic [35].

Rawlinson [9] adapted a version of a bias tee with a line power inserter. The line power inserter allowed 12 V to be fed into the LNB but protects the FCD.

2.2.2 Radiometer

Radiometry is the field of measuring any region of the EM spectrum. It is essential to understanding the properties of different sources by measuring the radiation intensity and spectrum. Since a SDR is used in this project, it must reflect the processes used in radiometry.

Radiometers measure the time average power of the incoming noise at a particular tuning frequency. The total noise power of a radiometer sums the independent contributions to antenna temperature and the radiometer temperature. Most radiometers include a RF amplifier followed by a frequency mixer, which multiplies the RF with the LO frequency to produce an IF output. These superheterodyne receivers are advantageous over the simple radiometer as lower frequency are easier to filter and amplify. LNBs contain this process with set a set LO and RF range. [36]

The explanation of the theory of radiometers taken from *Essential Radio Astronomy* [36] is now reproduced here.

Temperature in radio astronomy is referred to as the brightness temperature of

celestial object, corresponding to the measure of radio emission intensity. The Sun emits intense radio emission with a brightness temperature approximately 10^4 K. The brightness temperature is defined as,

$$T_b(f) \equiv \frac{I_f c^2}{2k_B f^2}, \tag{2.8}$$

where I_f is the spectral brightness, c is the speed of light, k_B is Boltzmann's constant and f is the frequency (Hz).

The first element to a radiometer is a bandpass low-pass filter (LPF) to discard any frequency outside the chosen range. A LPF is not required in the radiometer when it is integrated with an LNB, due to the existence of one inside the LNB.

The next stage is a square-law detector of which the output is proportional to the square of the input voltage.

An integrator takes the arithmetic mean of the envelope over a timescale (time constant τ), Eq.(2.9), by integrating over rapid variations.

$$\tau >> (B)^{-1}, \tag{2.9}$$

This smoothing can be done by using an resistor and capacitor (RC) filter or calculated numerically.

The number of independent samples (N) of total noise power in the time interval is given by,

$$N = 2\Delta\tau B. \tag{2.10}$$

A radiometer has a RC integrator, a resistor and a capacitor in a circuit, Figure. 2.7, to integrate the input signal and produce a voltage output.

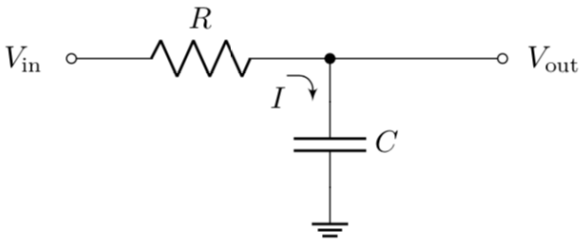

Figure 2.7: RC Integrator circuit [37].

The integrator can also be represented digitally as a difference equation (2.11),

$$y[n] = y[n-1] + m(x[n-1] - y[n-1]) \tag{2.11}$$

where $y[n]$ is the current output voltage, $y[n-1]$ is the previous output voltage and, $x[n-1]$ is the previous input voltage. The constant m represents $\frac{1}{RC}dt$, where RC is the time constant τ and dt is time for each sample (the reciprocal of the sampling rate) [38].

In terms of bandwidth and integration time, the average root-mean-square (RMS) voltage is equivalent to the ideal radiometer equation,

$$\sigma_T \approx \frac{T_s}{\sqrt{\tau B}} \tag{2.12}$$

where σ_T is the noise temperature and T_s is the system noise temperature.

2.2.3 Image Formation

Converting from the digitised IF signals into a spatially accurate image allows for images of the observed celestial object in the radio spectrum. As satellite dishes have no planar observational capabilities, they rely on techniques like raster scanning or interferometry to integrate two-dimensional (2D) images.

Many telescope setups have multiple antennae and use interferometric techniques which combine multiple signals to increase the resolution of the radio telescope. This includes an extra level of complexity as it involves computing time differences between signals from different antennae and was not used here, although it may be a viable technique to consider in future implementations.

Alternatively, raster scanning takes periodic samples and maps a corresponding colour based on the noise power received, as described in section 2.3.2.

Rawlinson [9] and Mangum *et al.* [5] implemented a raster scanning technique to map the inherited voltages to images as detailed in section 2.3.2. Once the continuum data is collected, the images can be synthesised by plotting the celestial noise power to pixel space, displaying intensity as colours.

Rawlinson [9], visualised the continuum radiation noise with gnuplot [39] (a program that can plot 2D plots) and Microsoft Excel's matrix function. Though Morgan [26] and Bhatia *et al.* [24] did use Microsoft Excel to store data, they do not explicitly explain how they mapped the image with the data, it is implied that the raster scan method is used and maps the collected data based on the specific scan line. Rawlinson's gnuplot script takes the matrix values from a text file and generates data files using Office software "LibreOffice Calc.".

2.3 Control and Positioning

Celestial object tracking is a technical method with a diverse set of approaches. Improving the tracking accuracy optimises the data collection with higher resolution and sensitivity.

2.3.1 Tracking System

Different tracking methods can involve single or dual axes and whether the mechanism is open-loop, closed-loop or hybrid-loop.

Number of axes

Active solar trackers can align with celestial objects using one or two axes. In [40], Nsengiyumva *et al.* review the advancements of solar tracking systems and how they affect the performance of both Photovoltaic (PV) cells and Concentrated Solar Power (CSP) systems and discuss the two main groups of solar tracking movement: single axis and dual axis tracking.

With a single-axis system, in [40], Nsengiyumva et al. state that the one axis of rotation can be aligned in any direction with an advanced control system and found that the preferred orientation is along the north meridian axis. Single-axis systems are considerably cheaper and easier to build, but they face a loss of efficiency compared to dual-axis systems. Dual-axis systems can maximise their solar radiation absorption. The two parameters of a dual-axis system are azimuth and elevation angles.

Tracking Strategies

There are typically three strategies for movement control with varying degrees of dependence on feedback: open-loop, closed-loop, and hybrid-loop.

The open-loop strategy is where the mechanism is independent of any feedback and does not require any external sensors to provide data on the position of the Sun. The system operates purely on an algorithm to calculate the position of the Sun based on date, time, and geological location, as used in [41].

The closed-loop strategy involves receiving feedback from a sensor (e.g., camera, light sensors, etc.) and updating the control mechanism accordingly. It follows a predetermined path (not real-time), calculated based on location and date, and uses the sensors to perform fine adjustments like [27] used. In practice, closed-loop systems are popular due to their accuracy. In [42], Chong *et al.* used either a Charge-Coupled Device (CCD) sensor or a photodiode sensor for solar tracking. Though closed-loop systems work poorly in bad weather, the higher alignment accuracy (range of a few milliradians) during good weather outweighs the fallback in poor weather conditions.

The hybrid-loop strategy is a combination of the open-loop and closed-loop strategies. A Sun tracking algorithm and a feedback loop for fine adjustments when needed in the hybrid-loop system provide a robust, accurate system. In [43], Bularka & Gontean use a hybrid-loop system for solar energy harvesting. Bularka & Gontean calculate the elevation and azimuth angles of the Sun, based on the Global Positioning System (GPS) location, and update the Sun's location each minute. For each update, the dish rotates backwards and forwards by a small change in angle for fine-tuning. After each day of tracking, the parabolic dish is manually directed to point towards the east horizon for the next day. The hybrid-loop system improved energy production by 28% from its previously fixed installation.

This project used Yaesu G-5500 rotators in dual axis for rotation in the azimuth and elevation directions. In [24], Bhatia *et al.* highlighted the Yaesu G-5500 as essential for amateur radio astronomers due to the limited accuracy of low-cost positioning systems. These positioning systems lack high movement speed, slack in the gears, and low wind sensitivity and return current position data with low accuracy (± 1 with 10 resolutions).

2.3.2 Raster Scanning

Raster scanning is a rectangular pattern technique used for image reconstruction on televisions. The particular pattern is uni-directional and resembles a rake-like formation, where straight horizontal lines (scan lines) are swept across the screen and rapidly returned for the sweeping to recur in an approximate saw-tooth form.

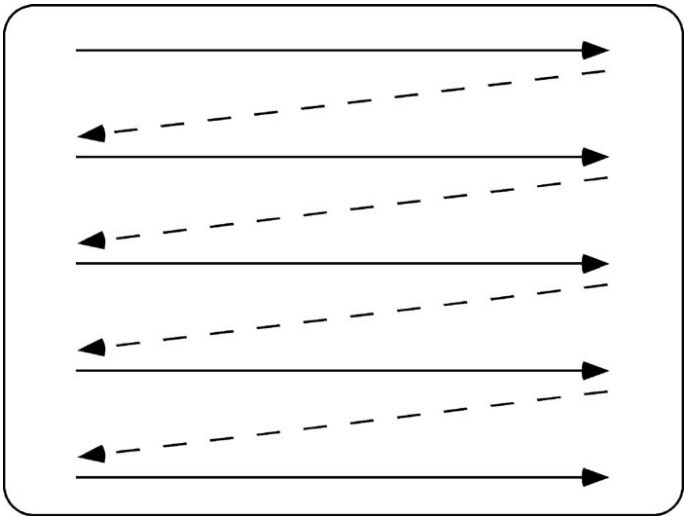

Figure 2.8: Raster Scanning Pattern [44].

In radio astronomy, this pattern can map a larger area and a broader view for data collection. Raster scanning also contributes to producing higher-resolution radio images.

In [9], Rawlinson utilises the Earth's rotation to perform the azimuth angle rotation and controls the elevation angle through a rotator. The speed of the elevation rotator was used to calculate the number of degrees rotated for each sample due to its continuous movement. The scanning process produced 19 vertical scan lines to form the image.

2.4 Machine Learning

In recent years, ML has become a well-established area where computers are trained to learn from data and make prediction based on the data. ML on images has become a significantly developed field in medical imaging, autonomous driving and facial recognition, using architectures such as CNNs.

2.4.1 Convolution Neural Network

Here is a review the structure of a CNN based on [45]. CNNs are multiple layers that break down an image and identify key features to perform tasks such as classification, object detection and image recognition. These layers consist of convolutional layers, pooling layers and fully connected layers. The layers are comprised of neurons which self-optimise through training.

An example of a CNN is shown in Figure. 2.9.

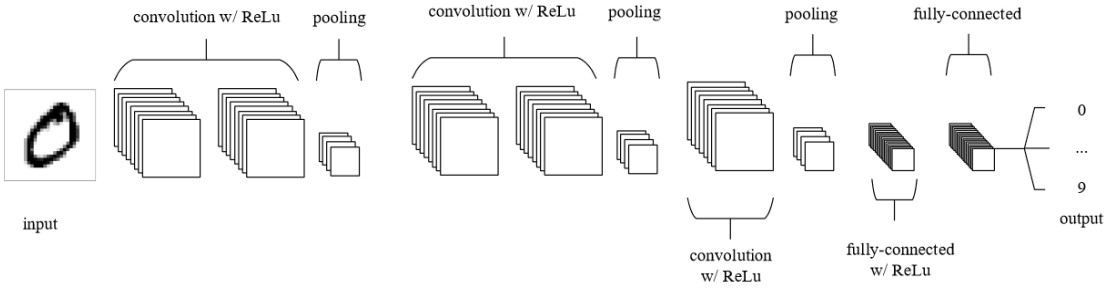

Figure 2.9: A CNN Architecture.

Convolutional Layers

Convolutional layers, Figure. 2.10, apply learnable filters (kernels) across the image and its dimensions to produce a 2D activation map for each filter. They determine

the output of the neurons that are connected to local regions in the original input by performing a scalar product of the weights (strength of connections of neurons) and input channel size. The activation output is then followed by an activation function for regularisation e.g. Rectified Linear Unit (ReLU).

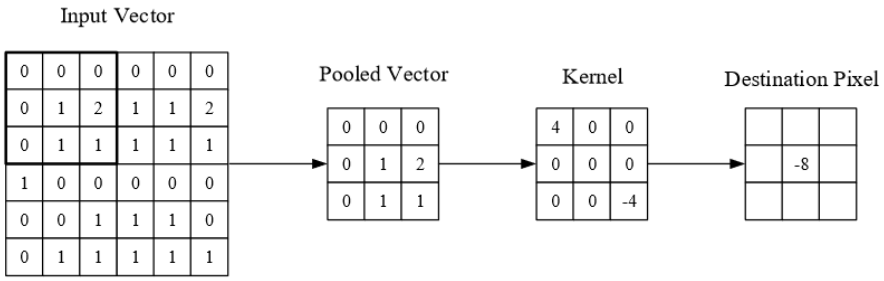

Figure 2.10: Convolutional Layer.

ReLU, is a regularisation technique that produces non-linearity in the network, represented mathematically in Eq. 2.13. It help mitigate the vanishing gradient problem, where the gradients of the loss function become very small as they back-propagate through the network.

$$f(x) = \max(0, x) \tag{2.13}$$

Parameters of the Network

Certain parameters can decrease the complexity of a convolution layer. Firstly, the depth of the output from a convolution layer can be manually set to produce a specific number of generated feature maps. By reducing this hyperparameter, the number of neurons in the layer is significantly reduced, however, it can reduce the pattern recognition capabilities.

Secondly, stride is a hyperparameter which specifies how many pixels the kernel steps by. If the stride is set to 1, the convolutional operation produced will heavily

overlapped, resulting in a larger output dimension. However, setting a larger stride will produce an output with lower spatial dimensions. Lastly, padding adds a given number of borders to the input, which gives further control to the dimensionality of the output. Usually, zero padding increases the dimensionality filled with 0 around the input data.

2.4.2 Super Resolution

SR is a ML technique to enhance the resolution of images, transforming low-resolution inputs to high-resolution outputs. SR is commonly used in Deep CNNs and Generative Adversarial Network (GAN)s to learn the mapping between low-resolution images and high-resolution images.

Example-based strategies use either a method of exploiting internal similarities of the same image or learning mapping functions from external low-resolution and high-resolution pairs.

The Sparse-coding-based method is one of the external example-based methods. The pipeline algorithm consists of encoding overlapped patches (fixed-size sections of an image with shared pixels) with a low-resolution dictionary and then using the sparse coefficients of the representation with a high-resolution dictionary respectively. This is then weight-averaged to produce a final output. This method is used in [46] and [47] where Yang *et al.* demonstrate its effectiveness in achieving high-quality image SR by leveraging the sparse representation pipeline.

2.4.3 Super-Resolution Convolutional Neural Network

In [15], Dong *et al.* propose a model called SRCNN which replaces the external example-based approaches, such as the use of dictionaries for sparse representation,

with convolutional layers. This achieves a better performance with a lightweight structure and fast execution for practical use using a moderate number of filters and layers. The only pre-processing performed is to upscale the input image to the desired size using bi-cubic interpolation. The objective of this SR is to map the low-resolution image (Y) to the high-resolution image (X),

$$X = F(Y). \tag{2.14}$$

The learning of $F(Y)$ is performed using three operations.

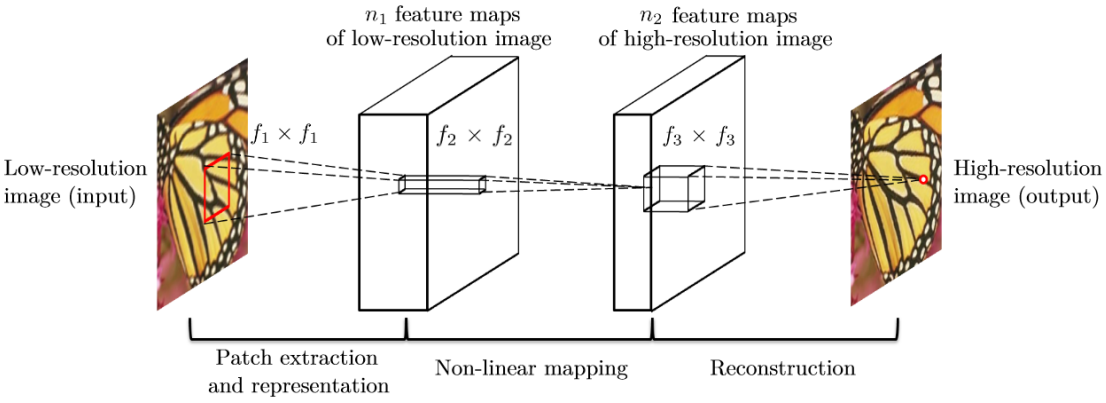

Figure 2.11: Convolutional Layer of SRCNN.

Firstly, the patch extraction extracts dense patches and represents them as a set of pre-trained bases, which is equivalent to convolving the image with filters. The max function is a ReLU activation and the weights (W_1) are set randomly from a Gaussian distribution with zero mean and standard deviation of 0.001.

$$F_1(Y) = \max(0, W_1 * Y + B_1), \tag{2.15}$$

where W_1 and B_1 represent the weights and biases of the first convolutional layer. The weight matrix W_1 corresponds to n_1 filters in the shape, each with a the shape

$C_{in} \times f_1 \times f_1$, where C_{in} is the number of input channels, and f_1 is the kernel size. The bias is an n_1-dimension vector.

The second convolution layer, 2.16, involves a mapping of the n_1-dimensional vectors into n_2-dimensional vectors. The non-linear mapping applies 1×1 filters to the output of the first layer. This output is conceptually a representation of a high-resolution patch.

$$F_2(Y) = \max(0, W_2 * F_1(Y) + B_2) \tag{2.16}$$

The last convolutional layer is the reconstruction of the high-resolution patches.

$$F(Y) = W_3 * F_2(Y) + B_3 \tag{2.17}$$

The filters used act like an averaging filter so the W_3 weights act as a set of linear filters.

The combination of the three operations acts as a form of CNN as each one acts as a convolutional layer. The network parameters must be trained and optimised through training.

This is achieved by minimising the loss between reconstructed and ground truth. The loss function Dong *et al.* use is the Mean Squared Error (MSE).

$$L(\Theta) = \frac{1}{n} \sum_{i=1}^{n} ||F(Y_i; \Theta) - X_i||^2. \tag{2.18}$$

Using MSE favours a high Peak Signal-to-Noise Ratio (PSNR) which is partially related to the perceptual quality and widely used for quantitatively evaluating image restoration quality.

The loss is minimised using stochastic gradient descent (SGD) with standard back-propagation.

$$W_{i+1}^{l} = W_{i}^{l} + \Delta_{i+1}, \tag{2.19}$$

$$\Delta_{i+1} = 0.9 \cdot \Delta_{i} - \eta \cdot \frac{\partial L}{\partial W_{i}^{l}}, \tag{2.20}$$

where $l = 1, 2, 3$ and i are the indices of layers, η is the learning rate, and $\frac{\partial L}{\partial W_{i}^{l}}$ is the derivative. The learning rate is optimised by lowering the value in the last layers to ensure the network converges.

The training data consists of high-resolution samples that went through blurring using a Gaussian filter, down-scaled and then up-scaled using bi-cubic interpolation. To train the model to contain variable size flexibility, the images are subdivided into smaller images.

The datasets used to train the SRCNN was the 5 Million sub-image dataset, ImageNet [48] and a 91-image dataset that was decomposed into $24,800$ sub-images.

The network was evaluated on serveral different metrics, including PSNR and structural similarity (SSIM) index. A higher PSNR indicates a better quality reconstructed image as a lower level of noise present in the network. The SSIM index compares the structural information between the high-quality image and the reconstructed image. Therefore, a high SSIM score indicates a better resemblance.

3

Implementation

Contents

The microwaves from the area observation reflect off the surface of the satellite dish to the focal point, where the LNB sits and performs down-conversion of the RF band to an IF band using a LO. The IF is passed straight through the satellite receiver unprocessed, and the receiver IF output port is connected to the SDR input port. The SDR generates a complex baseband signal with a band width of 3.2 MHz tuned to a carrier frequency within the satellite receiver IF band. This data is sent to a Raspberry Pi running software written in Python. This software computes sample by sample power values which are then integrated/averaged to give a final intensity value, which is simultaneously associated with a position in the sky through raster scanning and tracking movement to produce a 2D microwave intensity image. The

generated image is passed into the pre-trained convolutional network for inference (using a trained model to make predictions) on a separate PC to produce a higher-quality image of the Sun. Fully represented in Figure. 3.1

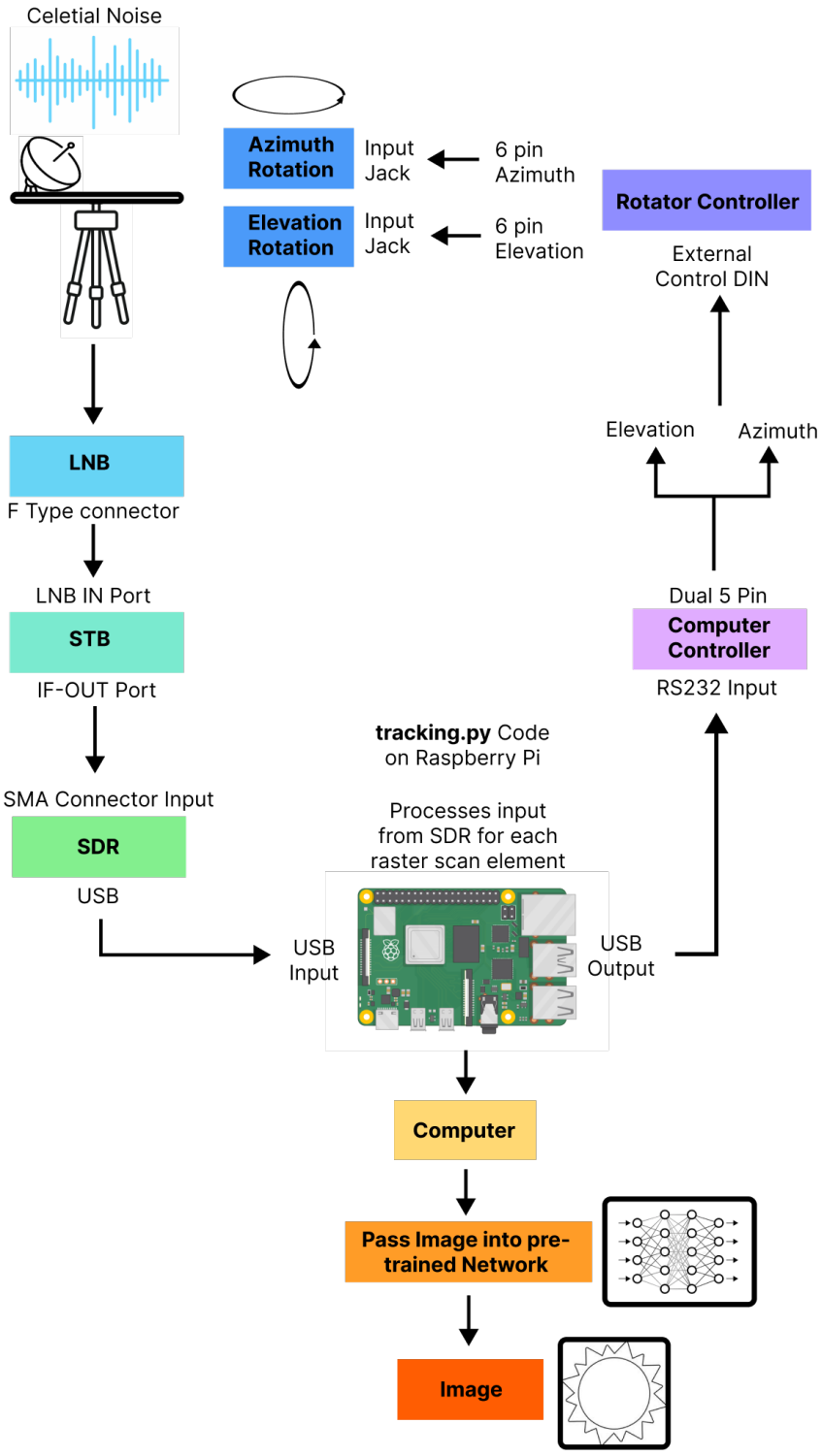

Figure 3.1: Schematic of entire system.

3.1 Satellite Dish Resolution

The diameter of the satellite dish for this project is 0.35 m, which can be used in Rawlinson's calculations, to calculate the HPBW at 12 GHz.

From Equations 2.2 and 2.3,

$$\text{Gain}_\text{A} = \frac{\pi^2 D^2}{\lambda^2} e_A = \frac{\pi^2 0.35^2}{0.025^2} 0.6 \approx 1161 \tag{3.1}$$

$$\text{HPBW} = \sqrt{\frac{\pi^2 k^2 e_A}{\text{Gain}_\text{A}}} = \sqrt{\frac{\pi^2 70^2 0.6}{1161}} = 5^\circ \tag{3.2}$$

The HPBW of the system is very large compared to the Sun's angular diameter, 0.6°, therefore, the Sun only covers 1.44% of the area covered by the the HPBW. This implied that it would be very difficult to observe any details of the Sun, let alone the Sun as a whole. Consequently, problems will occur in producing high-resolution images. Also, as movement is restricted to one degree, this implied a single scan would encapsulate the entire Sun.

Two variables influence the size of the HPBW; the wavelength and the diameter of the dish. If you increase the frequency of observation, which decreases wavelength, the HPBW of the dish decreases. Similarly, increasing the diameter of the dish, decreases the HPBW and increases sensitivity to the incoming microwave frequencies.

As the SDR has the highest input frequency of 1.75 GHz, the maximum observation RF is 12.35 GHz from an LO of 10.6 GHz.

$$1.75 + 10.6 = 12.35 \text{ GHz} \tag{3.3}$$

By increasing the centre frequency from 12 GHz to 12.3 GHz, the new HPBW would equal 4.88°, which has a minimal difference of 0.12°. Once again, the limitation of the rotator movement renders the change of the observation frequency insignificant.

3.2 Optimising LNB Integration with SDR

Provided for this project were two LNBs; the Nikkai Universal Single LNB and the Comag Single Universal LNB. Both LNBs have an input frequency range of 10.7 GHz to 12.75 GHz, which are converted to IF by the LO of values of 9.75 GHz or 10.6 GHz. These LNBs are capable of capturing microwave frequencies.

The Comag LNB was used, due to the Nikkai LNB displaying tendencies of picking up unwanted frequencies. This was investigated in 4.1.1.

To allow the connection between the LNB and the SDR, a control voltage and tone is required, as described in 2.1.2. Similar to Rawlinson [9], a bias tee was essential to provided the required 13 V or 18 V without damaging the SDR. However, the bias tee does not have the capability of supplying a tone of 22 kHz for the LNB to switch LO bands. Due to this, observing at frequencies higher than 11.7 GHz would require an extra signal generator at 22 kHz. Further investigations and research identified that the STB must have this function of the bias tee. Proved in Section 4.1.2, the change of the STB preset satellite with an allocated frequency confirmed the change in the high and low band LO.

For the STB, the check of functionality and understanding was first implemented by visualising the frequency spectrum on a monitor display. As the STB was produced in 2006, the output connection is a Syndicat des Constructeurs d'Appareils Radiorécepteurs et Téléviseurs (SCART), where access is limited in the current day,

therefore there's a shortage of any compatible monitor or TV inputs. A series of adaptors were required to create the connection between the monitor and STB. Firstly, a SCART cable runs from the TV output on the STB and is connected to the Audio Visual (AV) 3 input on a Technika SBX 84 AV Control 3, with the settings Monitor AV 3. The AV ports are connected to a Garite AV to High-Definition Multimedia Interface (HDMI) converter to enable the connection between the Audio-L, Audio-R and video on the AV Control and the monitor.

The STB has pre-programmed satellites that vary in polarity, transponder (TP) frequency and symbol rate. Polarity and symbol rate are irrelevant in the case of the Sun, however, selecting a satellite with a TP frequency greater than 11.7 GHz, triggers the STB to send the 22 kHz tone to the LNB and use the high LO band. This confirmed the STB can be used as an alternative to a bias tee.

The next challenge was to establish a connection between the STB and SDR. A required output from the STB should be the IF going into the LNB IN port. There was little indication online about this STB, so logical assumptions were constructed. From all the ports on the STB, the possibilities were Recommended Standard 232 (RS232) and IF OUT. The RS232 is the function to update the software of the STB, but the port seemed to be a uni-directional input connection. The IF OUT was the most viable option. No details about the hardware between LNB IN and IF OUT were supplied, however the manual described the port as an "LNB connection for a second satellite receiver". A reasonable assumption was that if another STB can be connected, the input to the second STB must be the same or similar to the input of the original STB, therefore, the assumption is that the output of IF OUT is the same as the IF. The assumptions were resolved in 4.1.2, by connecting the output of the IF OUT to a Spectrum analyser and observing the frequency peaks matching the theoretical IF produced by the IF OUT port.

3.3 Raspberry Pi

A Raspberry Pi 3B runs the back-end software that collects image data from the SDR, controls the dish position movement for either solo tracking or tracking and raster scanning and connects to a GPS Unit (PA1010D GPS Breakout [49]) that provides the user's current location and time. A third-party software package `ephem` uses the GPS location data to calculate the azimuth and elevation of the Sun relative to the dish. One of the USB A port connections communicates with the computer controller to send and receive angles to and from the rotators, and another connects to the SDR. Finally, a PC, connected via a Secure Shell (SSH) connection, fetches the final constructed image file from the Raspberry Pi.

The system is divided into multiple files containing classes representing the different aspects, Table 3.3.

Files	**Description**
config.py	Contains the configuration parameters.
main.py	Initialises the system.
tracking.py	The main class that executes the tracking loop.
gui.py	Initialises and updates the open-source Tkinter software library.
sdr.py	Performs the collection of SDR data and image creation.
raster_scanner.py	Executes the raster scanning algorithm.
position_control.py	Executes the tracking algorithm.
position_calculation.py	Calculates positioning for the rotators.
location.py	Handles GPS and updates location.

Table 3.1: File structure of Python script.

3.3.1 SDR

An SDR is required to sample the input IF from the STB, while performing similarly to a radiometer.

SDR++ Software

The open-source software, SDR++ [50], which performs DSP and produces a visual representation of the samples in the frequency spectrum, was initially used to ensure the correct functionality of the SDR as well as a progress step to understanding what to expect from the Python code. In this software, an In-phase Quadrature (IQ) correction tick box performs a DC blocker to counteract the DC Spike introduced by the SDR. With the IQ correction enabled, a peak appeared at $\approx$ 17.18 MHz when an oscilloscope signal generator produced a sine wave of frequency 17 MHz with an amplitude of 0.1 ppV. The SDR has a frequency offset (error) due to the limited calibration of the low-cost tuner chip [51], which a parts per million (PPM) correction value is required to adjust for this offset. The use of SDR++ software indicated the parameter values needed to implement into the Python code as seen in Table 3.2.

Parameter	**Setting**
Direct Sampling	Disabled
PPM Correction	-67
Gain	49.6dB

Table 3.2: SDR++ settings.

PySDR Library

The Python library `pyrtlsdr` [51] was used to configure and extract complex samples from the SDR. The configurations required for the SDR to sample can be seen in Table 3.3, where `freq_correction` is the PPM value and the `gain` to a value in dB. The `freq_correction` is set to the value found using SDR++ with the gain set to the maximum limit of the SDR to improve the SNR and aid the SDR in detecting the weaker signals. The variable, `center_freq`, is set to a predicted IF based on the relationship between the observation RF of 12 GHz and LO, from Eq. 2.4.

Parameter	Value
sample_rate	3.2e6
center_freq	1.4e9
freq_correction	-67
gain	49.6

Table 3.3: PySDR settings.

Processing Samples

As seen in Section 2.2.2, a RC integrator can be digitally programmed to produce an integrated output. Using Eq. 2.11, the function `rc_integrator`, 3.2, takes an array of samples and integrates over the already collected samples.

```
def rc_integrator(x, tau, dt):
    y = np.zeros_like(x)
    k = dt / tau
    for n in range(1, len(x)):
        y[n] = y[n-1] + k * (x[n-1] - y[n-1])
    return y
```

Figure 3.2: RC integrator function.

This function imitates an RC integrator circuit where the value of τ can equate to any value as there are no physical restrictions.

Given the raw samples are available in memory, there is little benefit in running an RC like integrator compared to just computing a block average. However, an RC integrator was implemented as a learning exercise. The integrator and running mean methods will produce different results because the integrator computes an approximate average. The comparison can been seen in Section 4.2.1.

The other method to process the samples is to calculate the block average of all the samples by using the `mean` function from the open-source `NumPy` library.

```
avg_pwr = np.mean(abs(samples)**2)
```

The absolute of the samples squared calculates $I^2 + Q^2$, of each element in the

array and divides it by the total number of samples to find the average values of all the samples.

As sampling in the SDR involves collecting data and storing all samples, using the RC integrator results in a lower output value than taking the average power of the array if the time window is longer than τ. To achieve a value closer to the equilibrium output of the RC integrator, Table. 3.4, the number of samples taken is chosen to equate to Eq. 3.4. Therefore, the integrator would never achieve its equilibrium value compared to taking the average power of all samples.

$$N = 2\tau B \tag{3.4}$$

Due to the accumulation of the input samples, the output of the integrator accumulates to 63.2% of the total voltage in one τ, shown in Table. 3.4.

Time Constant	**Capacitor Charging**	**Capacitor Discharging**
0.5	39.4%	60.6%
0.7	50%	50%
1	63.2%	36.7%
2	86.4%	13.5%

Table 3.4: Percentage output of RC integrator for each time constant [52].

The bandwidth of the SDR is set equal to the sample rate used through Quadrature sampling, which results in the runtime of the SDR to be twice τ. From Table. 3.4, the capacitor charging percentage is 86.4% at the time constant, 2τ.

Test	NumPy average	RC Integrator	Ratio: Integrator by Average
1	3.02e-05	2.61e-05	0.864
2	3.02e-05	2.61e-05	0.864
3	3.02e-05	2.61e-05	0.864

Table 3.5: Ratio of NumPy and RC integrator outputs.

For each test, the ratio between the RC integrator output and the array average output equates to 0.864, the expected value of the RC integrator after 2τ, which indicates the average power output equates to the integrator's equilibrium value. This concludes that taking the average power of all the samples is a better representation for the SDR.

The average power method has influences of τ on the run time of the SDR through Eq. (3.4). The greater the τ value implies the greater the number of samples, as shown in Table. 3.6, and the more refined average power the DSP will produce. For accuracy in data collection of a point of observation, too large of a τ value is not applicable.

τ (s)	Total Number of Samples
0.00001	64
0.0001	640
0.001	6,400
0.01	64,000
0.1	640,000
1	6,400,000

Table 3.6: τ values with 3.2 MHz Bandwidth.

The Sun's movement across the sky also impacted the choice of τ value. As the Sun moves approximately 0.25° every minute and the rotators are restricted to whole degrees of movement, to collect an accurate representation of the Sun, the threshold of encapsulation time is 2 minutes. Under these conditions, the SDR could run between 1 to 2 minutes of sampling, however, the rotators' runtime opposes this.

DC Blocker

A DC blocker was implemented similar to the IQ correction in SDR++ with the following the transfer function [53],

$$H(z) = \frac{1 - z^{-1}}{1 - Rz^{-1}} \tag{3.5}$$

with the difference equation of this recursive filter,

$$y[n] = x[n] - x[n-1] + Ry[n-1] \tag{3.6}$$

where R is a parameter between 0.9 and 1, which acts as a DC attenuation controller.

As the DC spike is generated within the SDR and impacts every sample taken, all samples are proportionally affected equally. The average power calculation accounts for the DC spike, so no adjustments are necessary.

SDR Python Class

The code structure of the `SDR` class has three functions: The `sampling()` function extracts and processes the samples, and `close_sdr()` executes the closing of the SDR connection. `create_image()` is executed after the scanning sequence has entirely executed, combining the average power with the collection position. An image of the Sun is formed through the Python package `matplotlib.pyplot` from the Python plotting library `Matplotlib`.

3.3.2 GPS

A script communicated with the GPS location of the Raspberry Pi, which, when first tested, the unit responded with no readings. With further research, the particular GPS unit does not function indoors. This meant indoor testing used manually entered location. Upon taking the unit outdoors, the GPS unit started to read the location. It provides the longitude, latitude, altitude, and timestamp that are all used in the `ephem` to calculate the relative positioning of the Sun.

3.3.3 Movement Code

A Moon-tracking script following a hybrid-loop strategy was supplied by Cook [8] which uses the Python package `ephem` from the library `PyEphem` for performing high-precision astronomy computations in order to locate the position of astronomical bodies in the sky. The package returns real-time high-precision orbital routines of the Sun, the Moon, planets, and major planet moons. Given the user's longitude, latitude and altitude, `ephem` will return the azimuth and elevation of the astronomical body every 30 seconds which are sent to the rotators. The position is then finely corrected by up to 3° using feedback from a camera which is passed through a CNN to identify the discrepancy between the centre of the body and that of the image.

The fine-correction setup however is not relevant to the implementation of this project as it is concerned with raster scanning, and so positioning to the exact centre of the body is unnecessary.

To only undertake 5 minutes of intermittent movement, a trigger activates a pause to the system, which resumes after the 15 minute cool-down [20]. Within those 15 minutes, the user must turn off the dual controller unit. When the system is ready to restart, the program waits for recognition that the dual controller unit

is back on.

Feedback from rotators

The original code of Cook includes a sleep command in between adjustments that allows time (5 seconds) to ensure that the rotators have completed their movement. In this implementation, the rotators possess a "C2" command[1] which returns their current position. To confirm completion of movement, the sleep function is replaced by a new function `wait_position_match()` that repeatedly sends the "C2" command and checks whether the result complies with the desired angles within a given tolerance[2].

Algorithm	Tolerance
Tracking	2
Raster Scanning	1

Table 3.7: Rotator angle tolerances for tracking and raster scanning.

Sleep commands still existed in the code for sending write and read commands to the computer controller, giving the device time to respond before receiving another command.

3.3.4 Scanning

The combined effect of the Earth's orbit and axial rotation cause the Sun to move across the sky. When implementing the raster scan, as the raster elevation rotation occurred, both elevation and azimuth would continuously need updating to ensure

[1] See operating manual [18] [19]

[2] It was noted that when the rotators were sent to 0° azimuth and 0° elevation, the azimuth angle feedback returned values between 2° and 4°. To account for this, tolerances are included which are different for tracking and raster scanning. The need for higher accuracy in raster scanning necessitates a smaller tolerance. See Table 3.7

that each scan line is positioned correctly with respect to the centre of the Sun. To achieve this, the most recent position of the Sun was calculated by the `ephem` package with every command sent to the computer controller.

A scanning range of twice the HPBW was chosen to ensure that the image produced would have a differentiable border around the Sun, with steps of 1°. Given that the HPBW is 5° and the angular diameter of the Sun is 0.6°, the Sun will be within the antenna's HPBW for five measurements. For each iteration, the offset of the azimuth angle is passed into the `move_elevation` function which calculates the current elevation offset. These offsets are totalled with the Sun's position to provide the next position of the raster scan.

Final code structure

Cook's [8] code was restructured into an Object Oriented Programming (OOP) framework consisting of five Classes:

Class	Description
`GUI`	Visual display of all variables and inputs
`PositionCalculation`	Calculation and setting of the rotation angles.
`PositionControl`	Serial Communication with rotators.
`Location`	GPS calibration and manual location.
`Tracking`	The main class for tracking executing tracking.

Table 3.8: Cook's [8] in separate classes.

A new class, `RasterScanner`, was created to calculate the new values of azimuth and elevation (`raster_azimuth` and `raster_elevation`) with a boolean to trigger either the raster scanning algorithm or the tracking algorithm. The `RasterScanner` class consisted of four functions (see Table. 3.3.4), where an embedded elevation

offset adjustment loop ran within an azimuth offset adjustment loop generating vertical raster scan lines.

Function name	Description
`updating_rotator_position`	Adjusts angles based on offsets.
`raster_elevation`	Adjusts the elevation during scanning.
`raster_azimuth`	Adjusts the azimuth during scanning.
`send_raster_command`	Serial communicates the raster scanning commands.

Table 3.9: Functions within Raster Scanner Class.

3.4 Super Resolution

The SRCNN was programmed on Google Colab to use the fast-performing Graphics Processing Unit (GPU)s available, to maximum capacity. This allowed for fast performing training and gave availability to various configuration changes.

Dataset Implementation

One of the most crucial tasks for the network training was the dataset implementation. In [15], the authors used the ImageNet dataset, which provides over 5 Million images. However, due to storage limitations, this project could not use ImageNet. Instead, the DIV2K dataset [54] [55] was chosen, which offers 800 high-quality images. Each image in DIV2K [54] [55] is of high resolution and includes corresponding low-quality versions created through bicubic interpolation, specifically designed for SR networks. This dataset is suitable because it aligns with the requirements for high-quality images needed to train the SRCNN effectively.

One of the most crucial tasks for the network training was the dataset implementation. In [15], Dong *et al.* used the ImageNet dataset [48], which can provide over 5 Million sub-images. However, again due to storage limitations, this project could not the ImageNet dataset. The package `torch`, from the Python library `PyTorch`, had several datasets which can be imported, however, none matched the correct specifications,i.e images in a Red, Green, Blue (RGB) colour system and the ability to apply transforms. The most important factor was due the lack of quality in the datasets. For SR the network requires high-quality images to map low-quality images too.

The dataset used for training the SRCNN is DIV2K [54] [55], which provides 800 high resolution images with the corresponding low resolution images created

through bicubic interpolation, specifically used for SR networks.

Low resolution transformation

To recreate a model similar to [15], the low-quality images provided by DIV2K were not used and instead the high resolution images were transformed through a Gaussian blur, then a sub-sample bicubic interpolation by an up-scaling factor of 3, followed by another bicubic interpolation by the same up-scaling factor, coded as follows, 3.3.

```
def low_res_transform():
    return transforms.Compose([
        transforms.GaussianBlur(kernel_size=(5, 5),
        sigma=(1.5, 1.5)),
        transforms.Resize(224 // 3,
        interpolation=transforms.InterpolationMode.BICUBIC),
        transforms.Resize(224,
        interpolation=transforms.InterpolationMode.BICUBIC),
    ])
```

Figure 3.3: Low resolution transform.

The SRCNN structure used is based on the most efficient network found by [15]. It consists of a three convolutional layers with the kernels sizes of 9, 5, 5 and the channel numbers as 64 then 32, 3.4. The up-scaling factor of 3 was chosen, resulting in a total of $69,251$ parameters.

```
SRCNN(
    (patch_extraction): Sequential(
        (0): Conv2d(3, 64, kernel_size=(9, 9), stride=(1, 1),
        padding=(4, 4))
        (1): ReLU()
    )
    (non_linear_mapping): Sequential(
        (0): Conv2d(64, 32, kernel_size=(5, 5), stride=(1, 1),
        padding=(2, 2))
        (1): ReLU()
    )
    (reconstruction): Sequential(
        (0): Conv2d(32, 3, kernel_size=(5, 5), stride=(1, 1),
        padding=(2, 2))
    )
)
```

Figure 3.4: Network structure of the SRCNN.

4

Testing

Contents

4.1 Spectrum Analyser Test

The satellite observation equipment, Figure. 4.1, has multiple functionality checkpoints throughout the project, using a spectrum analyser to prove the LNB and STB operated as expected and testing the connection between LNB and STB.

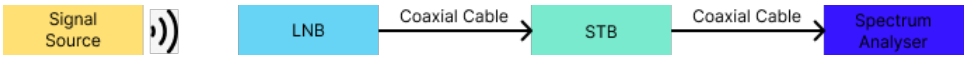

Figure 4.1: Setup for Spectrum Analyser Tests.

4.1.1 LNB

Nikkai LNB

The expected output on the Spectrum Analyser should have displayed a peak close to the IF calculated from the input RF, (2.4). The received signal of the LNB should have mixed with the selected LO. Peaks unrelated to the IF showed when using the Nikkai LNB, Figure. 4.2.

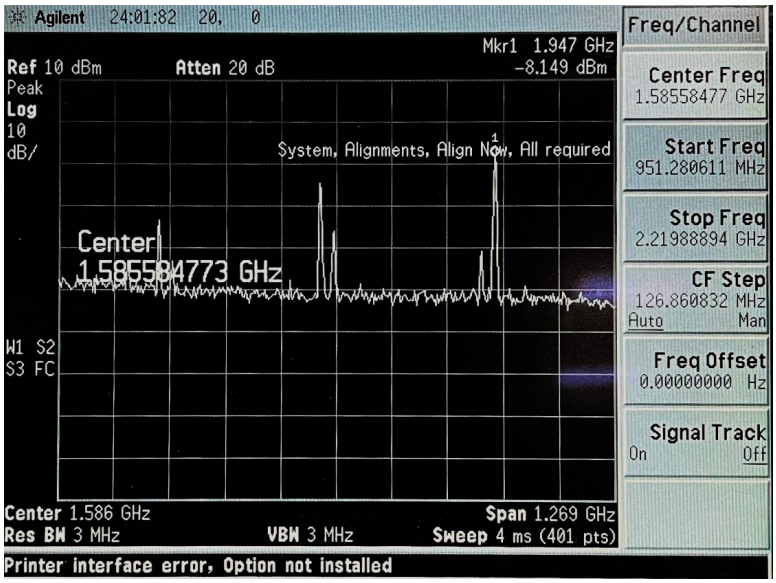

Figure 4.2: NIKKAI Universal LNB at 11.7 GHz.

The peak at 1.947 GHz was assumed to be the expected peak (where the theoretical is 11.7 GHz −9.75 GHz = 1.96 GHz). The unwanted peaks are at the approximate frequencies in Table. 4.1 as seen from right to left, Figure. 4.2.

Peak	**Frequency (GHz)**
1	1.851
2	1.582
3	1.538
4	1.205
5	1.173

Table 4.1: Approximate frequencies of unwanted peaks.

None of the unwanted peaks were a harmonic of the input signal. There was no definitive conclusion for the unwanted peaks, but potential causes were explored.

As the LNB and STB have internal noise, the signal-generating source was switched off to discover whether the unwanted peaks are generated by the devices. No peaks were observed on the spectrum analyser, indicating they were probably not generated by these devices.

As tin foil can shield microwaves, the sheets of tin foil were progressively increased in front of the LNB to see the change in response. The difference observed was minimal with one sheet of foil and no foil. With two sheets of foil, the spectrum analyser only displayed a peak of frequency 1.226 GHz, spanning 240 MHz across the spectrum. With the IF set to 9.75 GHz, an unknown source of RF at 10.976 GHz would be causing this peak. A reason for this phenomenon could have been a faulty LNB, causing the unwanted peaks. Another possible reason was the sensitivity of the LNB. The NIKKAI LNB has a NF of 0.3 dB, which means less noise is added to the incoming signal, making the LNB more sensitive to weak signals.

Comag LNB

Another LNB (the Comag LNB) was integrated with the system. The Comag LNB showed no unexpected peaks other than harmonics of the IF. As each LNB had different NF, this may have added enough noise to the signal for the output on the Spectrum Analyser to disappear below the noise level.

4.1.2 Different LO Bands

With the Comag LNB in the system, an investigation was conducted to understand how the LNB switched between the different LO bands. After assuming the STB has a signal generator for the 22 kHz tone, there must be a trigger to activate this. Checking the visual display of the STB indicated a change in frequency for each satellite channel, which led to switching between two satellites while connected to the spectrum analyser. From experimentation, it was demonstrated that choosing channels above/below 11.7 GHz resulted in the switch between high and low LO.

The satellite channel "HELLAS39EAST", is set to a TP frequency of $12,606$ MHz which produces a set of results in Table 4.1.2.

RF (GHz)	**IF (GHz)**	**Power (dBm)**
11.60	1.00	-9.59
11.70	1.10	-7.77
11.79	1.19	-18.12
11.89	1.29	-12.73
11.99	1.39	-18.85
12.091	1.49	-18.57

Table 4.2: IF and power for an LO of 10.6 GHz. (HELLASAT satellite).

The RF input of 11.60 GHz and 11.70 GHz have second harmonics, as shown in Table. 4.3

IF (GHz)	**Harmonic**	**Power (dBm)**
1.00	1.99	34.00
1.10	2.19	39.04

Table 4.3: Second harmonics Presents.

The satellite channel "EXPRESS53EAST", was set to a TP frequency of $11,096$ MHz which produces a set of results in Table 4.1.2.

RF(GHz)	**IF (GHz)**	**Power (dBm)**
10.68	0.93	-8.33
10.78	1.03	-7.25
10.89	1.14	-6.81
11.99	1.24	-6.35
11.09	1.34	-6.42
11.19	1.44	-10.47
11.29	1.54	-11.35
11.39	1.64	-9.07
11.49	1.74	-12.41
11.59	1.84	-4.24
11.69	1.94	-6.67
11.79	2.04	-10.52
11.89	2.14	-12.73
11.99	2.24	-49.43

Table 4.4: IF and power for an LO of 9.75 GHz.(EXPRESS53EAST satellite).

For the input RF of 10.68 GHz and 10.78 GHz have second harmonic, as shown in Table. 4.5.

IF (GHz)	Harmonic	Power (dBm)
0.93	1.87	-20.12
1.03	2.08	-28.03

Table 4.5: Second harmonics present.

These results concluded that the IF OUT suffices to output without affecting the IF from the LNB, proving that the system would output the correct IF to the SDR when connected.

4.2 SDR Experiments

By configuring the signal generator to generate a sine wave at 17 MHz and amplitude of 100 mVpp, the software SDR++ running on the laptop observed the signal at the value 17.14 MHz, Figure. 4.3.

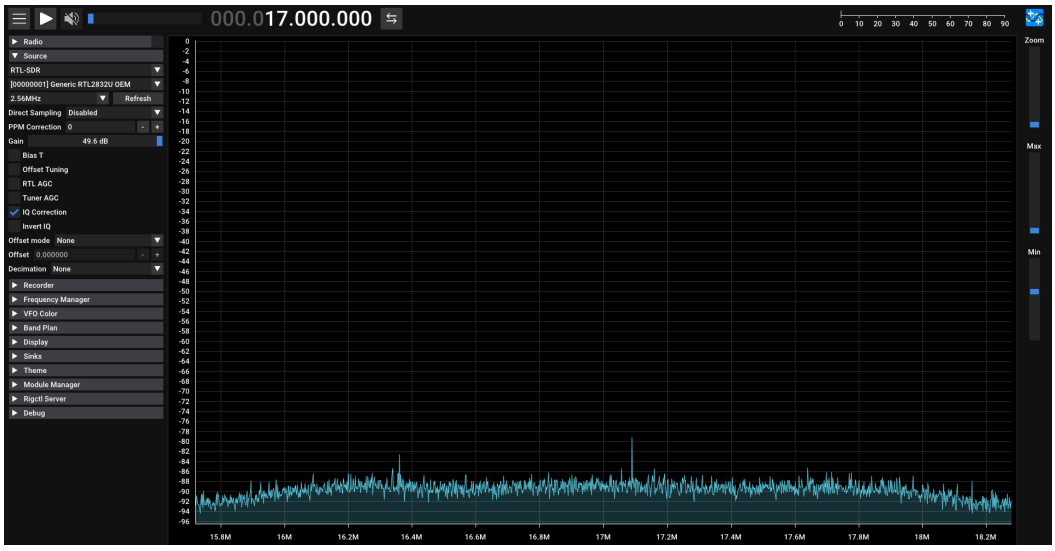

Figure 4.3: Sine wave generated at frequency 17 MHz misaligned on SDR++ software.

The difference between measured and expected frequencies indicated that a PPM correction was required to align the signal to the expected frequency. The PPM value was measured at -67, which centred the input signal at 17 MHz, as depicted in Figure 4.4.

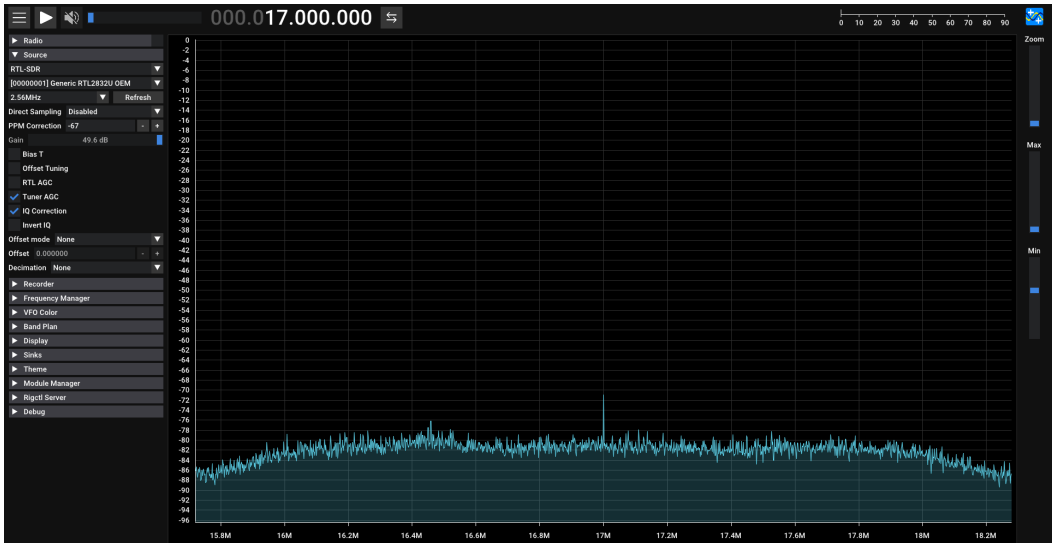

Figure 4.4: Output of SDR++ with adjusted PPM.

This correction ensures the frequencies are accurately represented.

4.2.1 Python SDR Code

Integrator Implementation

An RC integrator must obey the rule that after one time constant τ, the charging of the capacitor is at 63.2%. No capacitor exists digitally, but the output of the digital integrator also obeys this rule.

Testing the integrator involved keeping all parameters constant apart from τ, where two values were used, 0.1 seconds and 0.01 seconds. A step function was used as the input to the integrator.

The results for the two values of tau are shown in Figure. 4.5 and Figure. 4.6.

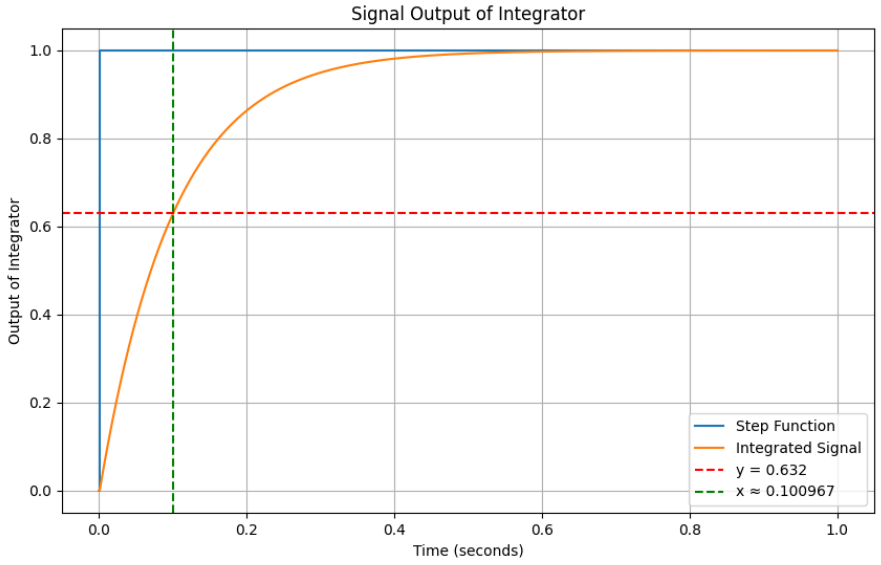

Figure 4.5: Output of integrator, time constant = $\frac{1}{10}$.

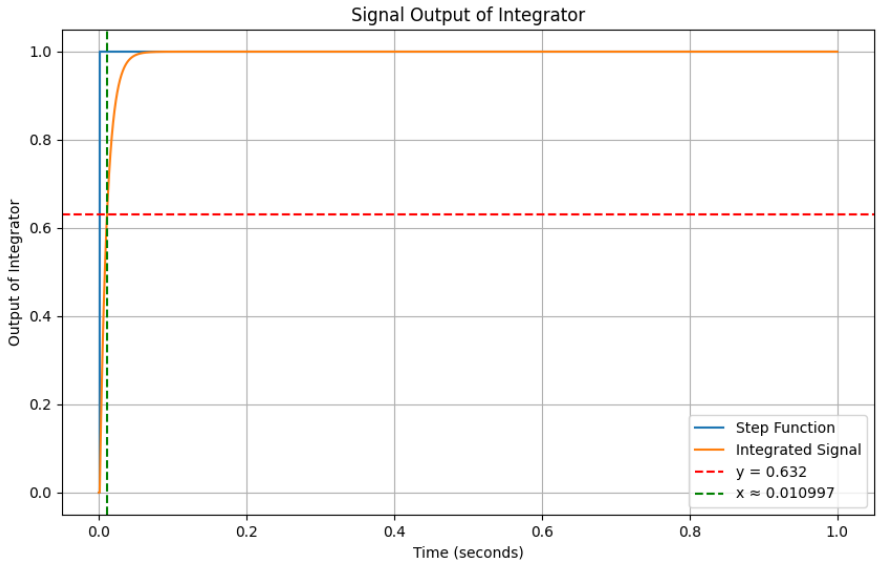

Figure 4.6: Output of integrator, time constant = $\frac{1}{100}$.

As seen on the graphs, the time on the x-axis at which the output value of the integrator was 0.632 (63.2%) [52], was approximately equal to the time constant,

which matched the typical integrator conditions, proving the integrator function was behaving as expected.

4.2.2 Python Image Formation Input

The Python library `matlabplotlib` was used to create the image from the collected data. The function, `create_image`, executes after the completion of the entire raster scanner and normalises the data collected to differentiate the differences between the values collected.

A pre-coded input array produced a 5-by-3 image (Figure. 4.7) to investigate the different colour maps and possibilities in using the `matlabplotlib` package. The colour map chosen was the `'jet'` to replicate a similar style of image to Rawlinson [9], where the intensities of each red, green, and blue colours are within the range of 0 and 1.

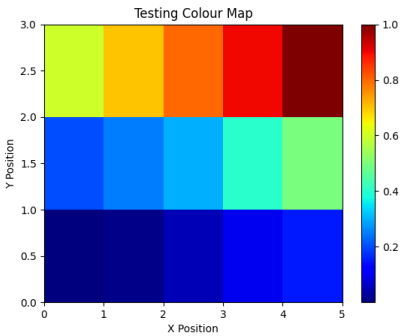

Figure 4.7: Colour Map.

With no signal input to the SDR, a map of pure blue is produced, Figure. 4.8, as all the values are very small (between 3.168×10^{-5} and 3.180×10^{-5}).

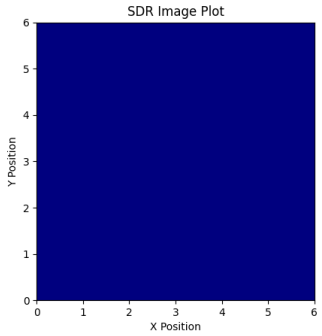

Figure 4.8: Image produced with no input to SDR.

Through the ”auto” argument in the `plt.imshow`, the aspect ratio is automatically matched to the data so the differences between the intensity values produced an image (Figure. 4.9) with different colours for the same values used in Figure. 4.8.

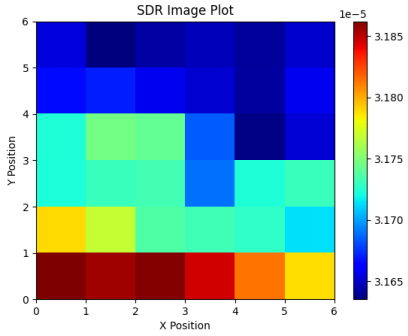

Figure 4.9: Image with ”auto” in plotting of intensities.

The equipment needed testing with a source similar to the Sun, where the noise level should rise between a cold and hot source, so a heat emitting bulb was set to point towards the LNB. The LNB to the SDR setup was assembled and a script was executed that collects a 10-by-10 image. Halfway through this program (about $X = 5$ and $Y = 0$ on the pixels), the lamp was positioned towards the LNB. A 0.7 dB increase, as seen in Figure. 4.10, between the two heat differences and a jump in colour on the colour map, as seen in Figure. 4.11.

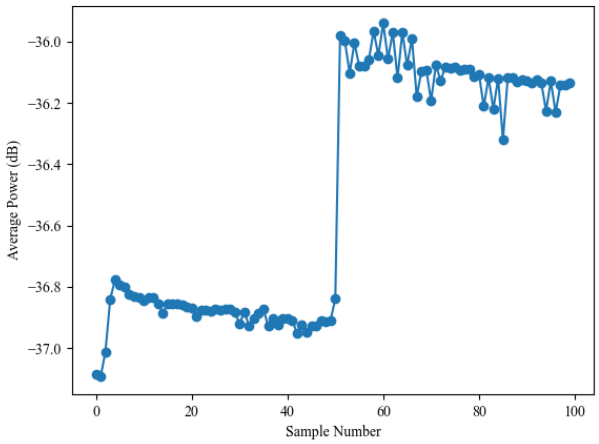

Figure 4.10: Average power output of the SDR over the total averaged samples.

The system from LNB to SDR detects an intensity change between hot and cold sources. The sensitivity of the system cannot be measured from these values as they are measured in arbitrary units. The SDR has no reference power to read any absolute power values, Figure. 4.11.

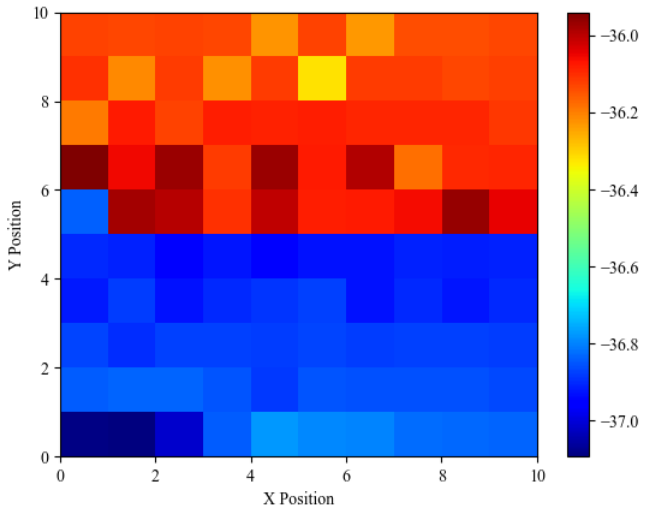

Figure 4.11: Colour map of averaged power for each pixel.

4.3 Movement Experiments

4.3.1 Overview

Continuous checks were implemented to ensure the code continued to work as expected. The `ephem` produced the correct angles for the Sun based on the GPS location by verifying through an online GPS Sun angle calculator. The feedback of the angles from the rotators showed that the rotators were moving correctly for the tracking of the Sun.

4.3.2 Raster Scanning Tests

The raster scanning algorithm was simulated to ensure the correctness of the values, that are intended to be sent to the rotators, Figure. 4.12. s

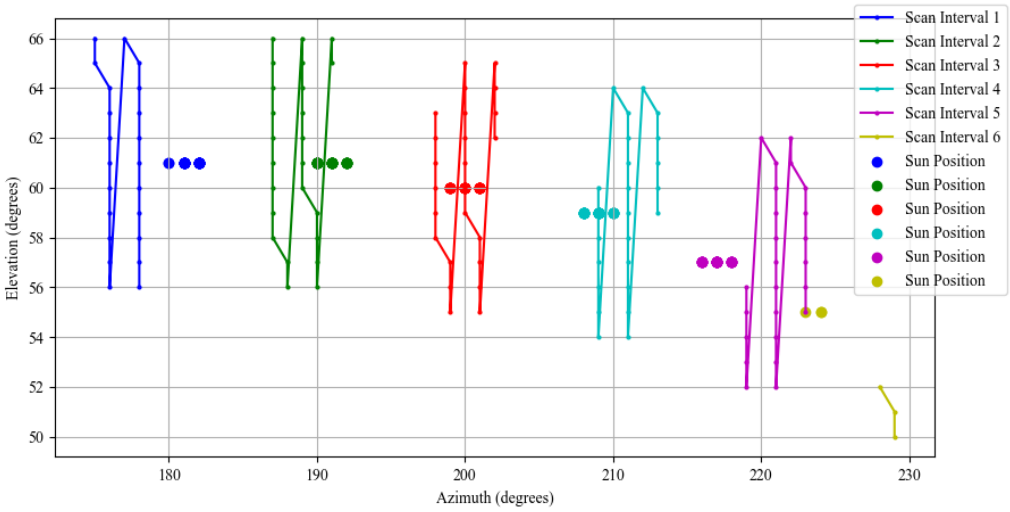

Figure 4.12: Simulation of raster scan algorithm with 5 minutes run time and 15 minutes pause.

Each calculated Sun position had a number of scans within the 5 minutes interval,

which are represented by different colours. Once the cool-down had finished, the new Sun position was calculated and the raster scanning algorithm would adjust to these new positions, which then was represented in a another colour.

Once this was finalised, the integration of raster scanning code and the tracking code was confirmed through the feedback from the computer controller.

4.4 Collection System Testing

The entire collection system was run multiple times in the lab to ensure the integration of the entire setup executed correctly. This was monitored by displaying the correct input angles, the correct feedback angle, and the data mapping on the terminal.

A test run with a heat emitter placed in view of the satellite dish was ran in the lab. The system was positioned to face the heat emitter and an number of scans to produce a 7-by-7 image.

As the heat emitter was placed approximately 1 m from the satellite dish, capturing the entire heat lamp would require a larger number of scans. Therefore with only a 7-by-7 scope was run, Fig. 4.13.

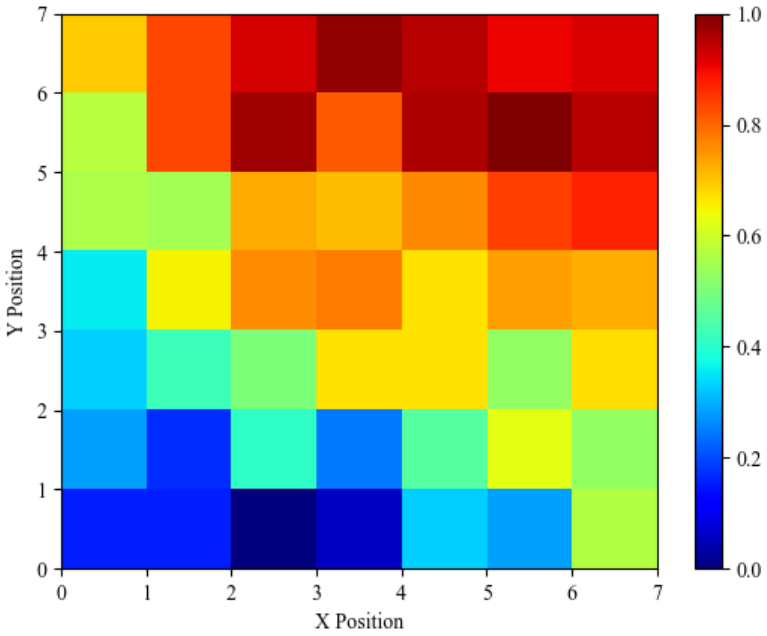

Figure 4.13: Heat Emitter image.

4.5 SRCNN model variation testing

Multiple networks were implemented and trained on batches of 16 images for 200 epochs, totalling 160, 200 backpropagations. The training length for each network accumulated to 6 hours and approximately 70 computing units on Google Colab.

4.5.1 SGD

Initially, training the network that used the same optimiser as [15], SGD, showed little learning over 200 epochs. The network lacked learning, which could be due to the fewer number of backpropagations performed on the model compared to the 8×10^8 backpropagations completed by Dong *et al.* [15]. To execute 8×10^8 backpropagations, decomposition of the dataset into multiple sub-images and

training for longer would be required. Upon implementation, each epoch was to cost 31 hours of runtime and, therefore, not be explored.

4.5.2 Adam Optimiser

The SGD optimiser was replaced with an Adam (Adaptive Moment Estimation) optimiser to stabilise the training. The Adam leverages both the first-order moment (mean) and the second-order moment (uncentred variance) of the gradients to adapt the learning rate for each parameter. This adaptation helps in addressing the issues of slow convergence and oscillations during training. This network did converge converge after 200 epochs.

4.5.3 Adam Optimiser with Learning Rate Scheduler

The `Adam` SRCNN was adapted to include a learning rate scheduler (LRS), 4.14, which changes the learning rate periodically by a certain number of epochs. The LRS refined the learning of the images by changing the learning rate by a factor of 0.1 every 50 epochs.

```
def initialise_optimizer(self):
    params_to_optimize = [
        {"params": self.model.patch_extraction.parameters(),
        "lr": 1e-4, "weight_decay": 1e-6},
        {"params": self.model.non_linear_mapping.parameters(),
        "lr": 1e-4, "weight_decay": 1e-6},
        {"params": self.model.reconstruction.parameters(),
        "lr": 1e-5, "weight_decay": 1e-6}
    ]
    optimiser = torch.optim.Adam(params_to_optimize)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
    step_size=50, gamma=0.1)
    return optimiser, scheduler
```

Figure 4.14: LRS with Adaptive Moment Estimation (Adam) optimiser.

4.5.4 Adam optimiser with data augmentation.

When training another network, data augmentation was applied to all images in the dataset to prevent overfitting. This process generated a randomly transformed set of images. The data augmentation consisted of random horizontal flips, random rotations up to 10° and a colour jitter that changes the colour aspects of the image, 4.15.

```
transforms.RandomHorizontalFlip()
transforms.RandomRotation(10)
transforms.ColorJitter(brightness=0.2
contrast=0.2, saturation=0.2, hue=0.1)
```

Figure 4.15: Additional transforms to entire high-resolution dataset.

The low-resolution transform was also adjusted to involve more Gaussian blurring, at a probability of 50% and additional Gaussian noise at a standard deviation of 0.05, with a probability of 50%, 4.16.

```
transforms.RandomApply(
[transforms.GaussianBlur(kernel_size=(5, 5),
sigma=(2.0, 2.0))], p=0.5),
transforms.RandomApply([transforms.Lambda(lambda x:
x + torch.randn_like(x) * 0.05)], p=0.5),
```

Figure 4.16: Additional transforms to produce low-resolution images.

4.5.5 Comparison of Loss and PSNR

The SRCNN trained with an SGD optimiser did not converge within 200 training epochs. The validation loss, Figure. 4.17, decreased linearly, showing that the model is performing well on the unseen validation dataset, indicating the improvement of generalisation. The PSNR of the validation showed a linear increase, which indicated a steady improvement in the quality of the reconstructed images. However, the training PSNR and loss purely fluctuated with no indication of convergence 4.18.

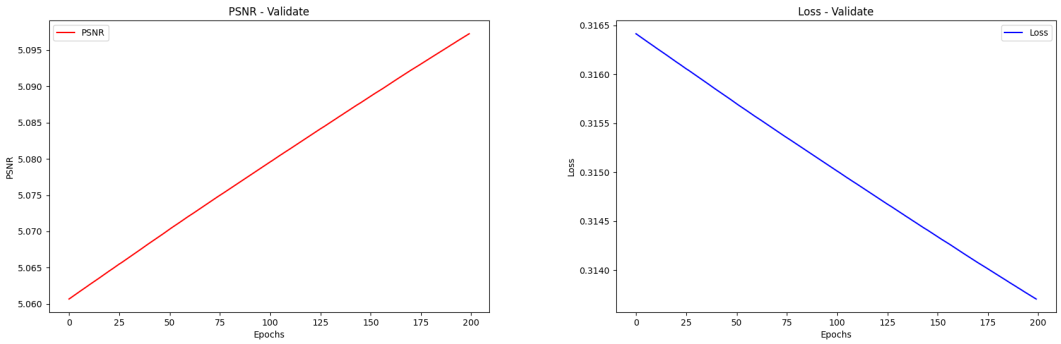

Figure 4.17: Validation PSNR and Loss of SGD SRCNN.

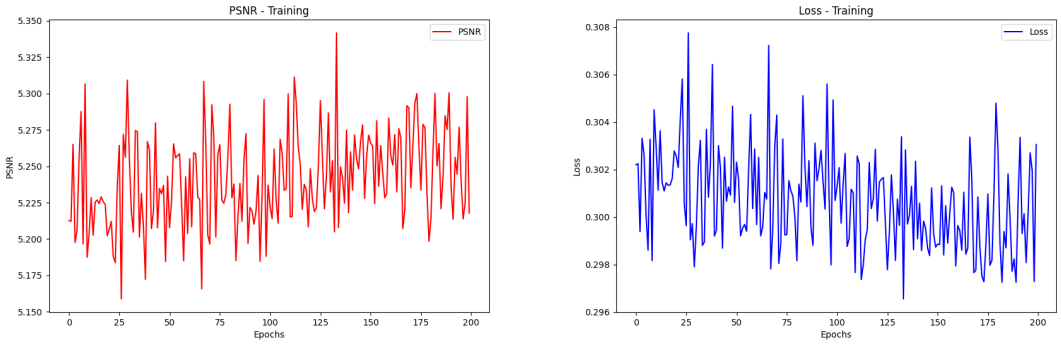

Figure 4.18: Training PSNR and Loss of SGD SRCNN.

The SRCNN trained with the Adam optimiser showed significant improvement over 200 epochs compared to the SGD SRCNN. The plateau of the training loss, Fig. 4.19, indicates that the network has started to converge. The slow increase of

the training PSNR shows that the reconstructed image quality is slowly improving. A continuous rise in PSNR reached a peak validation PSNR of 18.68, which showed that the network has high performance on the unseen validation set. The validation loss confirmed that the SRCNN did perform and generalise well on the validation set.

The LRS SRCNN followed similar trends to the Adam SRCNN. The validation loss plateaued in two regions, one in early training and the other around 50 epochs. At the 50 epoch, the learning rate changed by a factor of 0.1, therefore there was less rapid training.

The final model of the SRCNN with learning rate scheduler and data augmentation showed a general trend similar to the Adam SRCNN but varied along the epochs slightly. This training PSNR has plateaued fully by 200 epochs, though the validation PSNR did not smooth out like the Adam SRCNN.

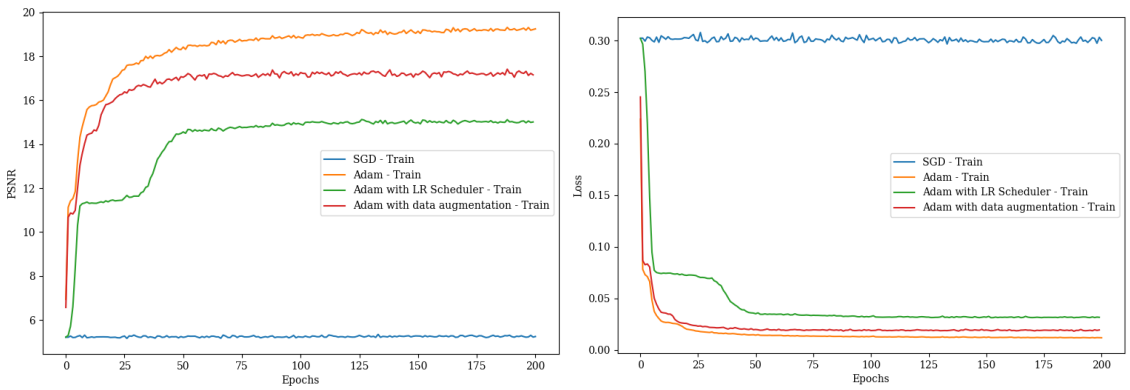

Figure 4.19: Training PSNR and Loss.

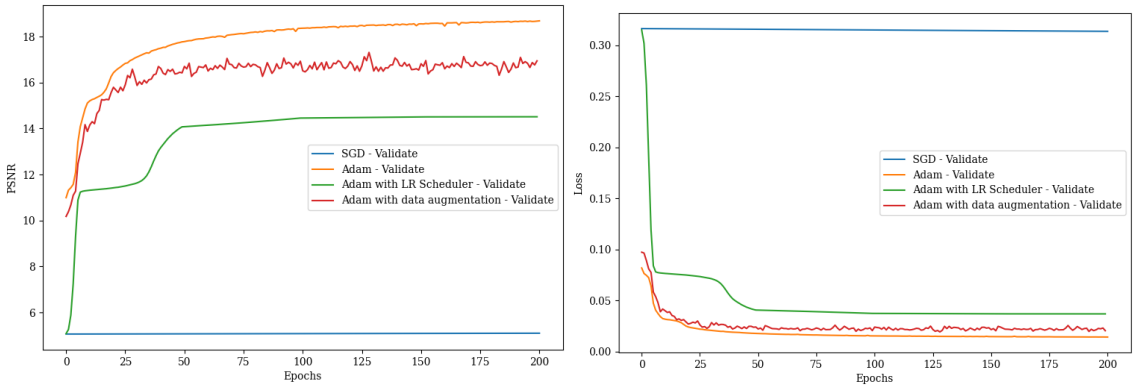

Figure 4.20: Validation PSNR and Loss.

The SGD, compared to the other models, does not perform well as there is little change in loss or PSNR for training and validation. Among the models, the Adam SRCNN performs best in loss and PSNR with the highest PSNR and lowest loss. The data augmented SRCNN performed better than the one with just the learning rate. This may have been due to the LRS SRCNN potential underfitting to the data due to small batch size or insufficient training length, so the augmented data model performs better as there's more variation in the images.

4.5.6 Comparison of images produced

For the SGD SRCNN, the reconstructed images have no appearance similar to the high or low-resolution image 4.21, which is potentially due to the image batch size being too small, introducing high variance into the gradients. Another reason was the training length was too low to impact the reconstruction of images however, due to time and cost restrictions, this was not resolved.

Figure 4.21: SGD SRCNN low-resolution (top) and reconstructed (bottom).

For the Adam SRCNN, the confirmation that the network performed is observed through the difference between the low-resolution images and the reconstructed images, Figure. 4.22

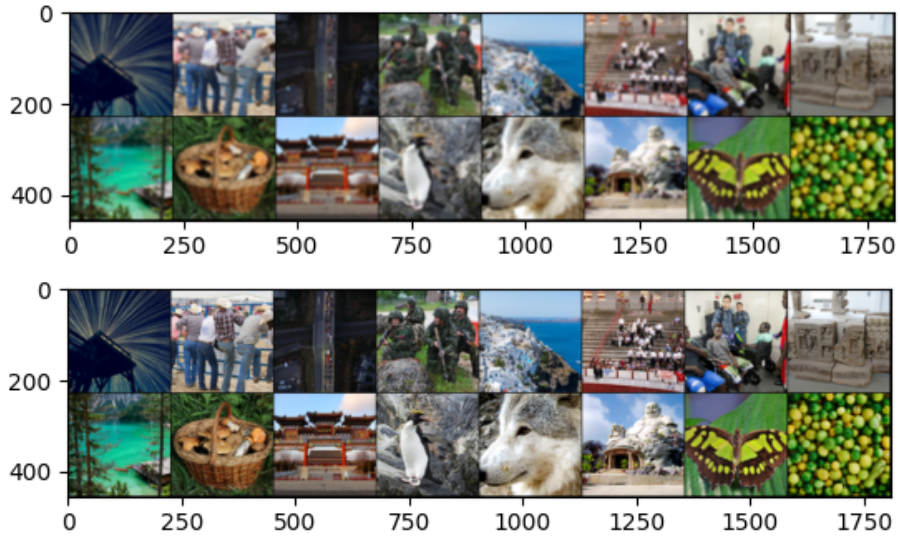

Figure 4.22: Adam SRCNN low-resolution (top) and reconstructed (bottom).

The reconstructed images from the LRS SRCNN indicated that the network did not fully train. Some of the colours within the images were mismatched between the high-resolution images and the reconstructed images.

Figure 4.23: LRS SRCNN low-resolution (top) and reconstructed (bottom).

In the data augmented SRCNN, the images were different in particular colours, but the reconstructed images compared to the low-resolution images have signifi-

cantly increased in resolution.

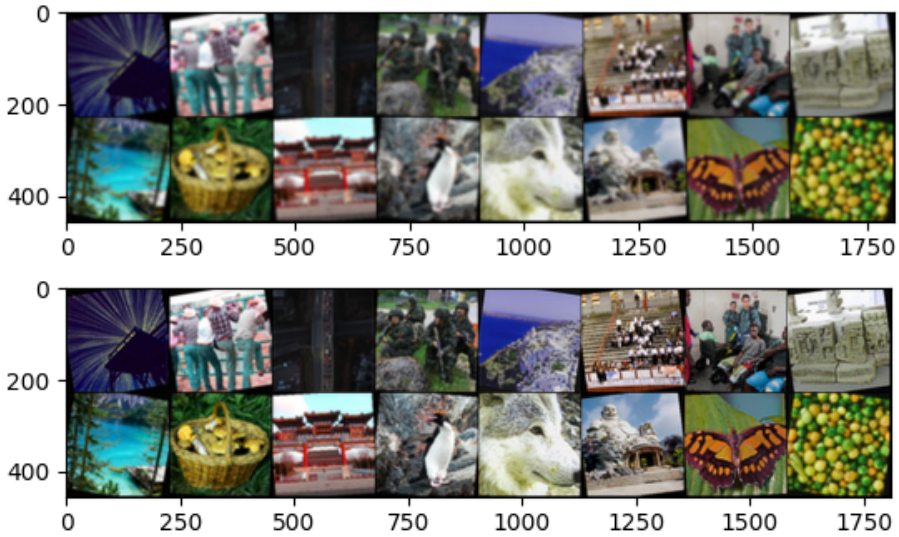

Figure 4.24: Data augmented SRCNN low-resolution (top) and reconstructed (bottom).

Ultimately, the Adam SRCNN and the data augmented SRCNN were the two models that produced the best performance.

5

Results

Contents

5.1 Image of the Sun

The total time taken for an 11×11 image was 2.5 hours. During the experiment, it was seen that the system was misaligned from the Sun. This caused the output image to have similar values, Figure. 5.1.

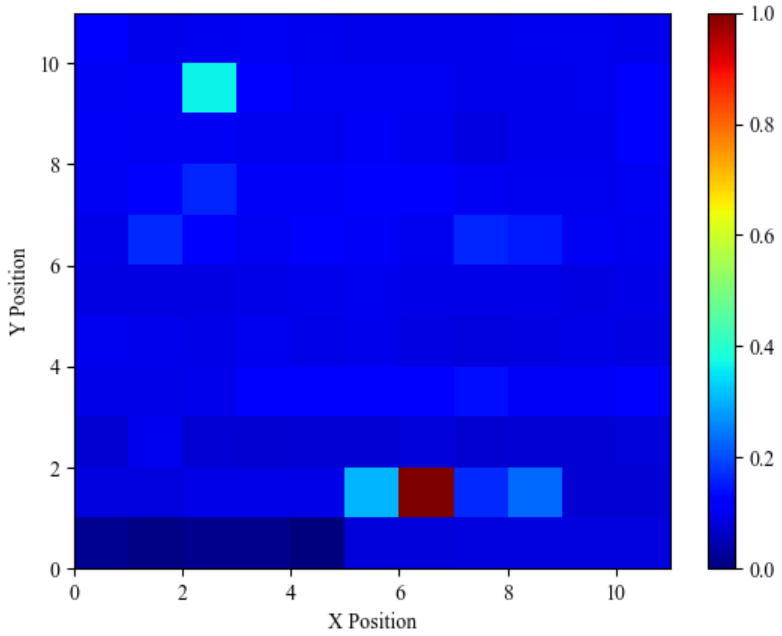

Figure 5.1: First Full Test of scanning with misaligned rotator movement.

Possible causes to the misalignment included: the raster scanning algorithm, the initial calibration of the rotators, the GPS unit providing false information or miscalibration of the rotators from the cool-down period.

As shown in Section 4.3.2, the code generated for the raster scan modifies the changes in the Sun's position after the 15 minute cool-down and therefore was not the cause. A full re-calibration of the rotators caused no change in the misalignment.

The GPS unit was re-tested and returned the time as one hour behind, which would impact the Sun's position provided by the Python package `ephem`. However,

as the alignment towards the Sun was checked before the scanning, this was unlikely to be the cause.

Figure 5.2: Test run of the tracking and capture system showing satellite dish alignment.

Therefore, the conclusion was drawn that the rotators themselves were the cause due to the turning off and on of the dual controller, which caused misaligned in the

calibration. Code was added to save the positions of the rotators before the dual controller turn off for the cool-down period to ensure the rotators were correctly aligned when restarted.

As the system had a generally low resolution, with no clear sky, the microwaves from the Sun were unlikely to be observed, so no further imaging was taken.

5.2 ML

The models were tested on three datasets: 102 Category Flower Dataset [56], Set5 [57] [58], and Set14 [59] [58]. Each was evaluated the metrics by PSNR and SSIM values, see Table. 5.1.

Database	**Method**	**PSNR**	**SSIM**
Flower	SGD	10.02	0.0071
	Adam optimiser	30.12	0.8729
	Adam with LRS	21.83	0.6296
	Adam with augmentation	26.55	0.7767
Set5	SGD	10.08	0.0084
	Adam optimiser	29.83	0.8848
	Adam with LRS	21.61	0.6351
	Adam with augmentation	27.82	0.8292
Set14	SGD	11.35	0.0119
	Adam optimiser	26.12	0.8091
	Adam with LRS	20.30	0.5323
	Adam with augmentation	24.91	0.7589

Table 5.1: PSNR and SSIM values for different methods across various datasets.

The model proposed in [15] achieved a PSNR of 32.75 and a SSIM of 0.9090 on

the Set5 dataset. In comparison, the Adam SRCNN achieved 97.3% of the SSIM score and 91.1% of the PSNR score reported by [15]. While the Adam-optimised network did not outperform the model by Dong *et al.*, it reached these percentages with only $160,200$ backpropagations, a significant reduction compared to the 8×10^8 backpropagations used by Dong *et al.*.

For the Set14 dataset, the Adam SRCNN achieved an SSIM score of 98.5% and a PSNR score of 89.1% relative to the scores reported by Dong *et al.* Although the SSIM score is closer to that of Dong *et al.*, the PSNR score is comparatively lower.

Additionally, the networks were evaluated on the 102 Category Flower Dataset, yielding PSNR and SSIM scores similar to those obtained on the Set5 dataset. This consistency indicates that the Adam SRCNN is a robust model with strong performance across different datasets.

5.3 Sun Image through SRCNN

With no images produced from the system, Rawlinson's [9] image was passed through the two best neural networks, Adam SRCNN and data augmented SRCNN.

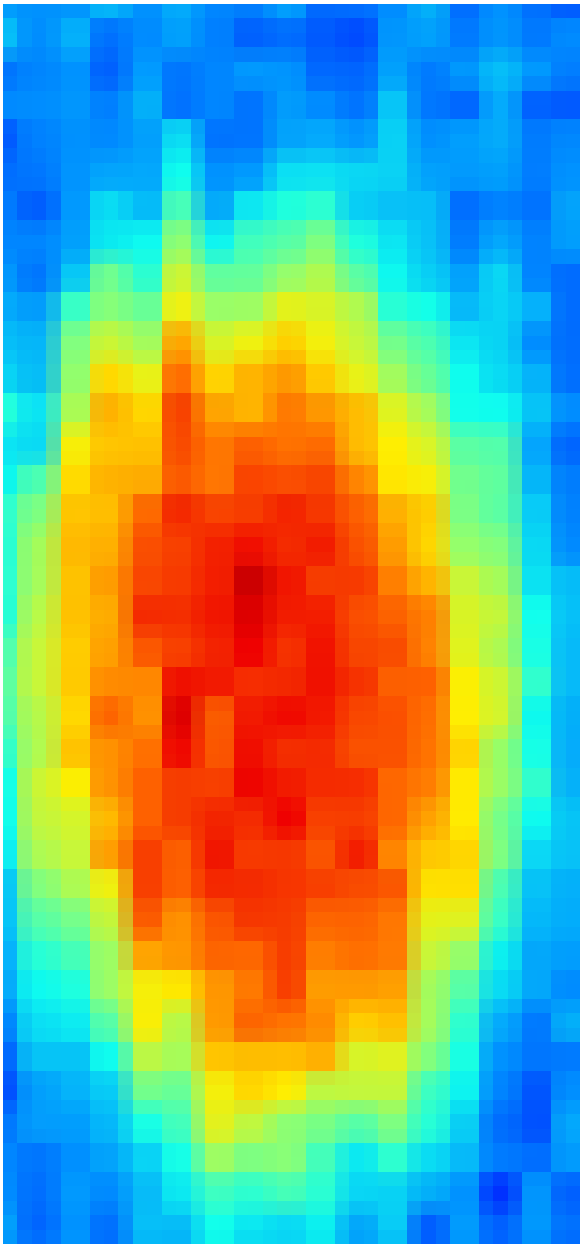

Figure 5.3: Rawlinson's Sun image.

Before the image, 5.3, was passed through the neural networks, it was processed

through a script to resize the image by scaling the pixels to match the pixel size of the network's training data. When processed through the SRCNN with Adam optimiser, the image is then scaled to 224 pixels which distorts the transformed image of Rawlinson's Sun. As the image has very little data and the Adam SRCNN model is trained on more pixelated data, the model responds worse to Rawlinson's image than the other datasets tested, Figure. 5.4.

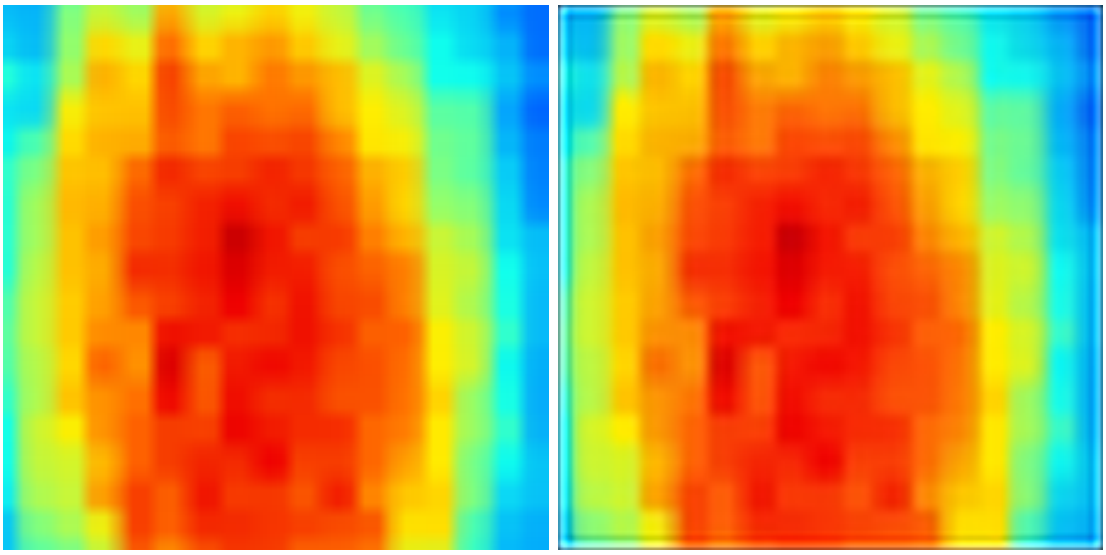

Figure 5.4: Transformed low-resolution image (left), reconstructed image (right).

6

Evaluation

Contents

The system combines a single satellite dish system with a SDR to sample the microwave intensities of the signals. These intensities are mapped to a location calculated and moved by the raster scanning algorithm and rotators to produce an image of the Sun. This image is then passed into a trained neural network to increase the resolution of the image. The equipment entails the satellite dish, LNB, STB and SDR, which in combination collected and sampled the microwave intensities. The project successfully made use of off the shelf equipment to configure a system which tracks the sun, collects data, generates images, and enhances those images. In the scope of celestial imaging, several improvement remain to be explored to finalise and optimise the image creation, these are explored in the following sections.

6.1 Key Contributions

From existing literature, three factors are identified as affecting the spatial resolution of images; the size of the dish, the smallest angle of rotation available by the motors, and the spacing between dishes in interferometry setups. Interferometry effectively functions as a dish with a larger radius. For this project, it was chosen to investigate the capability of a single dish to capture data. Rawlinson [9] utilised a 1.2 m and Morgan [26] used a 3 m. In this project, the capabilities of a single dish 0.35 m was chosen to be investigated to determine the minimum size required. A single dish not only reduces the quantity of equipment needed, but also eliminates the additional data manipulation necessary when combining the data from multiple antennas. Overall, this simplifies the installation process and allows this to be set up anywhere.

It was validated that a STB can be used as an alternative to a satellite finder. This approach offers flexibility by utilising equipment commonly purchased with a satellite dish. Compared to Rawlinson, who uses a line power inserter, this setup requires a STB to power the LNB. This implementation proved effective because the STB also functioned as a power source for the LNB. This method successfully expands the range of equipment available to future engineers, reducing complexity and cost.

The connection between LNB and SDR functioned effectively. By testing the system with a heat emitter, microwave data was successfully captured, allowing for the observation of differences between the emitter and the surrounding area. This confirms that the implementation is effective and can be used in the same manner as demonstrated by Rawlinson [9] and Ogechukwu [60].

Cook [8] developed a tracking algorithm for the Moon and Sun, while Rawlinson

[9] achieved raster scanning with a single axis of rotation. A key contribution of this project is the implementation of code that can track the Sun while performing a raster scanning algorithm. The dual-axis system in this project extends the available capture time to potentially the entire day, rather than relying on the Sun passing over the dish. The code was designed to operate at any time and location. Additionally, the feedback response from the rotators ensured accurate positioning, further enhancing the precision and reliability of the code. This advancement is significant as it demonstrates that even with low-fidelity equipment, the entire process can be automated, providing greater flexibility for data collection, such as continuous operation throughout the day or activation at any time.

As discussed in the introduction, the use of ML in celestial imaging has shown significant potential to enhance image resolution. Developing and training a SRCNN for this project is crucial for leveraging lower fidelity systems while obtaining meaningful data. Although the network produced slightly lower output metric scores compared to the original paper, it performed well on the 102 Category Flower Dataset [56]. By using the Adam optimiser, the project achieved 88.48% similarity on the Set5 dataset [58], which is very close to Dong *et al.* [15] similarity of 90.90 %, validating the effectiveness of this CNN. When implementing this system, engineers can utilise the pre-trained CNN, which adds minimal complexity to the system while providing substantial value.

6.2 Limitations

The implementation of a simple system using off-the-shelf equipment resulted in several limitations, the most significant being spatial resolution. The observed angular resolution was very low due to the diameter of the dish and the rotators' inability to achieve increments smaller than 1 degree. This limitation affected the system's

ability to detect precise intensity variations between points in the sky.

In the physical setup, the motion code experienced misalignment in test trials. This issue was likely caused by the stop-and-start operation of the dual controller and the intervals required for the rotators to cool down. By plotting the output of the code against the Sun's position, it was confirmed that the code functioned correctly. Therefore, the misalignment must have originated from the physical components.

The extended cooling periods required for the rotators significantly increased the total scan time to 2.5 hours. This limitation restricts the observable phenomena to longer-lasting events such as sunspots, plages, and prominences, which typically persist for days or weeks. Consequently, shorter events like solar radio bursts and solar flares may not be captured.

The SDR used was limited to a sampling rate of 3.2 MSPS, by using a SDR with a high sampling rate could allow for the capture of more detailed information from the signal, improving the resolution and precision of the data. Higher sampling rates allow for finer temporal resolution, which translates into better capture of the signal's variations and more precise data.

Although the Adam SRCNN was successfully trained on high-definition images, it was not suitable for enhancing Rawlinson's 19-by-60 pixel image [9]. By breaking down the coarse resolution into smaller pixels, we adjusted it to meet the network's 224-by-224 pixel requirement and ran the enhancement. Despite the output image appearing sharper in certain areas, it did not exhibit significant changes to provide more information than the initial input. The network has considerable potential for further exploration and customisation for the specific output of the raster scan. For instance, training the network on a dataset with the same pixel count as the input image — similar in quality to the microwave intensity image of the Sun — could yield better results. Furthermore, the training process of the network was limited by the resources and costs. The final network was trained at 200 epochs for 6 hours

through Google Colab. The NVIDIA [61] A100 GPU used in Google Colab is a professional-grade solution highly suited for creating and training complex models like the SRCNN is this project. However longer run times could still improve the final performance of the network.

6.3 Future Developments

By evaluating whether the rotators can achieve precision of half a degree or less, we could potentially enhance the spatial resolution and capture more detailed intensity measurements. This improvement may also optimise continuous rotation, leading to faster execution times and increased data collection. The rotators used in this project had a large step size, resulting in reduced information collection. If the rotators had the ability to move continuously, the calibration errors caused by switching the dual controller on and off would be eliminated, saving time during scans.

Therefore, it is essential to investigate alternative rotators that strike the best balance between availability, cost-effectiveness, and smaller step sizes — ideally around 0.1 degrees, which corresponds to approximately one sixth of the Sun's diameter as observed from Earth. Additionally, it is crucial to select motors that do not require breaks for cooling, ensuring uninterrupted operation and further improving efficiency.

To achieve higher spatial resolution, a meaningful next step is to extend the system to perform interferometry. As explored in [6] this approach effectively simulates a larger dish and captures higher precision data. Using two smaller dishes is likely to be more cost-effective and easier to implement than a single very large dish. The addition in this system would require the connection of a second dish and a second rotator. Implementing a correlation process would then identify the time delay between signals arriving at each dish. This time delay information would then

be used to create an interferometric image with enhanced resolution. The rest of the system and the SRCNN would remain unchanged.

To further increase the robustness of the system, a secondary verification could be set up to regularly check that the system is still aligned. This could be implemented by adding a camera with a tube around the lens as a field restrictor, which would continuously capture images to ensure correct synchronisation of the raster scan. When the rotators are out of sync, a simple alarm system could be set up to warn the engineer or to pause the system.

To improve upon the Adam SRCNN, a more adapted data set could be used to re-train the network. In this project, high quality images were used and a transform applied to create the low-resolution dataset. This means the CNN was trained with a different band of quality, much larger than the one needed to enhance the low-resolution output from the raster scan. By using an input dataset of lower-resolution, the enhancement could be more tailored to the output of the raster scan.

Secondly, the amount of image blurring in the training could to be further explored. The goal is to match the amount of blurring applied during training to the level of detail and quality improvement (enhancement) desired in the final images. In machine learning, it is common and reliable to achieve a 4x enhancement in resolution. By experimenting with different degrees of blurring in the SRCNN, the enhancement can be optimised to ensure that the resulting images are meaningful and not distorted or misleading.

The advantage of this project lies in its specific focus on imaging the Sun, allowing the SRCNN to be trained exclusively on solar images and data. By understanding the typical shapes of solar phenomena, such as sunspots or flares, and the likelihood of concurrent events, the network can learn to generate accurate and meaningful enhancements from lower resolution images. Advanced models like GAN-based super-resolution models and AI can achieve higher levels of enhance-

ment. However, these methods must be used cautiously, as they generate new data rather than simply improving existing content. In contrast, the model used in this project is less likely to introduce bias, as it is trained on a diverse set of varied images.

To thoroughly explore this potential and train a high-quality network, the system could be paired with high-fidelity imaging equipment. By comparing the machine learning output to these high-quality control images, the model's performance can be assessed to prevent the generation of misleading data. This comparison would serve as a control for calibrating the extent of image enhancement, ensuring that the generated content is accurate and useful for solar imaging purposes.

Conclusions

The results of this study demonstrate the feasibility of using low-cost equipment combined with machine learning techniques to enhance solar imaging. The implemented system successfully captured and processed microwave signals from the Sun, producing images that were further improved using a SRCNN.

Key contributions include validating the use of a set-top box as an alternative to more specialised equipment, effectively integrating a satellite dish with an SDR, and developing a dual-axis tracking system that extends capture times throughout the day. The project also highlighted the potential of machine learning to enhance low-resolution images, achieving significant improvements in image quality metrics such as PSNR and SSIM across multiple datasets.

Despite these successes, the study faced limitations, particularly in spatial resolution due to the dish size and motor precision. The system's ability to detect short-lived solar phenomena was also constrained by the long scan times and cooling times required. Additionally, while the Adam SRCNN model showed promise, further training with more tailored datasets could yield better results.

Future developments could focus on improving motor precision, reducing scan times, and exploring interferometry to enhance spatial resolution. By addressing these areas, the system can become a more robust tool for solar imaging, providing valuable data for both educational and preliminary research purposes.

Overall, this project demonstrates that low-cost, accessible technology, when combined with advanced data processing techniques, can significantly contribute to the field of radio astronomy and solar observation.

A

Github link

All files can be found and executed via this Github link. My Master Project on GitHub

B

Set-top box rear side

LNB IN	LNB connection for the antenna cable
IF-OUT	LNB connection for a second satellite receiver
VCR	SCART connection for the video set
TV	SCART connection for the TV set
RS 232	Serial port
DC IN 12 Volt	Connection 12V/power pack
SPDIF	Digital coaxial audio output
Audio-L	Analogue audio connection L(eft)
Audio-R	Analogue audio connection R(ight)

Table B.1: Rear side of the receiver

C

Interferometry

Interferometry is a technique where two or more satellite dishes use the principle of superposition to combine the observed waves and increase resolution. This mitigates the low aperture of a single dish, fluctuations of atmospheric emissions, receiver gain and radio-frequency interference [36].

The background to interferometry is reviewed here, based upon the discussion in [36].

The new collecting area is dependent on the number of satellites in use N and the diameter of the dishes D.

$$D_{tot} \approx \sqrt{N} D. \tag{C.1}$$

The simplest radio interferometer is with two radio telescopes (elements), where the voltage outputs are correlated (multiplied and averaged). The two elements are separated by a baseline vector $\vec{b}$ of length b and both elements point in the same direction $\hat{s}$. The angle between $\vec{b}$ and $\hat{s}$ is θ.

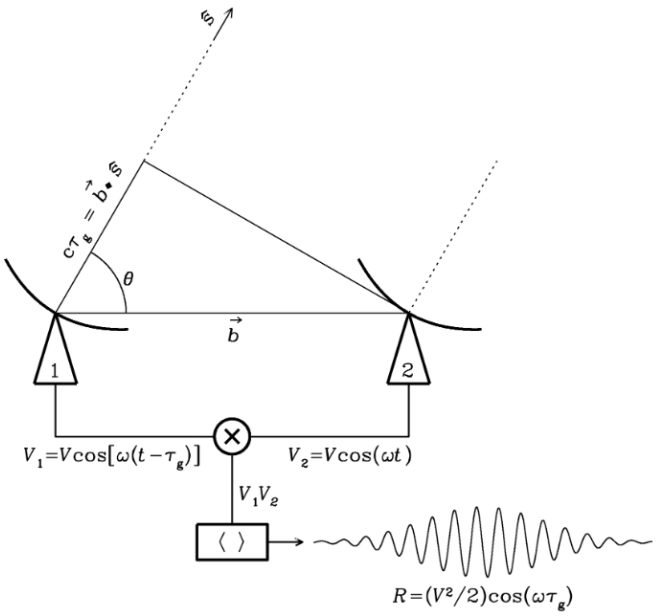

Figure C.1: Interferometer Setup. [36]

Waves travel a longer distance to one of the elements and the extra distance is measured a time delay τ_g,

$$\tau_g = \frac{\vec{b}\cdot\hat{s}}{c}. \tag{C.2}$$

For a very narrow bandwidth Eq.(C.3) centred on a frequency Eq.(C.4), the desired output voltages of the two elements are Eq.(C.5) and Eq.(C.6).

$$\Delta\nu << \frac{2\pi}{\tau_g}, \tag{C.3}$$

$$\nu = \frac{\omega}{2\pi}, \tag{C.4}$$

$$V_1 = V\cos[\omega(t-\tau_g)], \tag{C.5}$$

$$V_2 = V\cos(\omega t). \tag{C.6}$$

The correlator will multiply the voltages V_1 and V_2 and yield a product,

$$V_1V_2 = (\frac{V^2}{2})[\cos(2\omega t - \omega\tau_g) + \cos(\omega\tau_g)]. \tag{C.7}$$

To remove the high-frequency terms, the correlator also time averages and produces an output response R.

$$R = \langle V_1V_2\rangle = (\frac{V^2}{2})\cos(\omega\tau_g), \tag{C.8}$$

with a time average that is long enough to obey,

$$\Delta t >> \frac{1}{2\omega}. \tag{C.9}$$

The correlator output amplitude $\frac{V^2}{2}$ is proportional to the flux density S at the source point multiplied by the effective collecting areas of the elements $(A_1A_2)^{\frac{1}{2}}$. This is because V_1 and V_2 are proportional to the electric field produced by the source multiplied by the voltage gains of the two dishes.

Due to the Earth's rotation, the baseline vector changes, ergo the correlation output voltage varies in a sinusoid. These sinusoids are called fringes and have a phase of,

$$\phi = \frac{\omega}{c}b\cos\theta. \tag{C.10}$$

These fringe phases are based on times, therefore small tracking errors of the individual elements do not affect the measured source position. A interferometer with a horizontal baseline has equal delay of signal as the plane-parallel component of the atmospheric refraction has no effect.

The complex visibility function is a mathematical representation of the spatial distribution of brightness through the amplitude and phase information of the EM radiation represented by,

$$V(u,v) = \int\int I(x,y)e^{-2\pi i(ux+vy)}dxdy. \tag{C.11}$$

$I(x,y)$ represents the brightness distribution at the sky coordinate (x,y) and (u,v) is the coordinate in the Fourier plane. Scientists sample different parts of the Fourier plane with various baselines and frequencies and then apply an inverse Fourier transform to construct an image.

In [22], Latief *et al.* design a two-element adding radio interferometer using satellite equipment. With a maximum baseline b, the resolution of an interferometer is inversely proportional to the maximum baseline separation. The combined correlated signal is fed into a square law detector output,

$$V_o = V^2[\sin(2\pi\nu_0 t) + sin(2\pi\nu_0(t-\tau_g)]^2, \tag{C.12}$$

where ν_0 is the centre frequency.

After a LPF, the output voltage (fringes) of the final receiver are,

$$V_{rec} = P[1+\cos 2\pi\nu_0\tau_g)], \tag{C.13}$$

$$V_rec = P[1 + cos(\frac{2\pi\nu_0 b \sin(\theta_s)}{c})]. \tag{C.14}$$

The fringe frequency is proportional to the baseline as per Eq.(C.14).

To construct a radio image, both cosine and sine components are required. To achieve a sine component an additional correlator unit delays one element's signal by a phase of 90.

$$V_{cosine} = P[1 + cos(\frac{2\pi v_0 b \sin(\theta_s)}{c}], \tag{C.15}$$

$$V_{sine} = P[1 + sin(\frac{2\pi v_0 b \sin(\theta_s)}{c}]. \tag{C.16}$$

As there is no down-conversion taking place, the fringes are dependent on the geometric time delay. However, if the LNBs are down-converted by the same LO, all equations are still applicable.

In [6], Gireesh *et al.* implement an interferometer to observe the Sun using commercial dish TV antennas, operating in the K_u band of frequencies. The brightness temperature of the Sun $T_b \sim 10^4$K is a basic property used widely as a diagnostic tool to understand the different types of temporal changes in solar emission. The correlation interferometer is separated by $\sim$ 2.5m and is oriented in the East-West direction. The theoretical angular resolution $\sim 37'$ is specified by the separation between the interference fringes. As the angular size of the Sun is smaller than that of the dish, the "point" source is the Sun for the observations. Each of the resulting IFs are converted from analogue to digital using a 2-level (+1 or 0) high speed comparator and sampled in a D-type flip-flop. They are then processed an Ex-NOR gate for the correlation step, that is counted with a 24-bit counter that acts like an integrator and read by a computer via a 9-bit micro-controller with an integration time $\sim$ 1. This means the signal is split into a in-phase and a quadrature phase

components using a quadrature power splitter and connected to a 1-bit correlator.

D

config.py

```
# Antenna and signal parameters.
HPBW = 5
data_size = 2 * HPBW +1
sample_rate = 3.2e6
centre_freq = 1.4e9
gain = 49.6
freq_correction = -67

# Serial communication parameters.
serial_port = '/dev/ttyUSB_DEVICE1'
baud_rate = 9600
timeout = 1

# Files.
image_path = "/home/pi/Desktop/Sun_Image_Project/Moon.jpg"

# Default GPS.
```

```
default_latitude = 51.49915

default_longitude = -0.17529166666666668

default_altitude = 74.9
```

E

gui.py

```
import datetime
import ephem
import tkinter as tk
from tkinter import messagebox
from PIL import Image, ImageTk

# GUI class - TKInter setup.
class GUI:
    def __init__(self, tracking):
        self.tracking = tracking
        self.initialise_tkinter()
        self.initialise_objects()
        self.create_labels()
        self.setup_gui_grid()
        self.initialise_gui_buttons()

    # Initialize tkinter module.
```

```
def initialise_tkinter(self):
    self.root = tk.Tk()
    self.manual_calibration = tk.IntVar()

# Tracking Bodies Setup.
def initialise_objects(self):
    self.Track = tk.StringVar()
    self.Method = tk.StringVar()
    self.Bodies = ["Moon", "Sun", "Manual"]
    self.Methods = ["Lookup Table", "Neural Network
    (Moon Only)"]

# Initialize labels.
def create_labels(self):
    self.TrackMenu = tk.OptionMenu(self.root, self.Track,
    *self.Bodies)
    self.LatLabel = tk.Label(self.root, text="Latitude:")
    self.LongLabel = tk.Label(self.root, text="Longitude:")
    self.AltLabel = tk.Label(self.root, text="Altitude:")
    self.AzLabel = tk.Label(self.root, text="Target Azimuth")
    self.ElLabel = tk.Label(self.root, text="Target Elevation:")
    self.TrackLabel = tk.Label(self.root, text="Tracking:")
    self.MethodLabel = tk.Label(self.root, text="Detection
    Method:")
    self.ManAzLabel = tk.Label(self.root, text='Manually
    Set Azimuth:')
    self.ManElLabel = tk.Label(self.root, text='Manually
    Set Elevation:')
```

```
        self.c1 = tk.Checkbutton(self.root, text='Automatically

        Calibrate Azimuth? Warning - Magnetometer is Inaccurate',
        variable=self.manual_calibration, onvalue=1, offvalue=0)
        self.ChangeTime = tk.Label(self.root, text='Edit Time
        (YYYY-MM-DD HH:MM):')

        self.LatDisplay = tk.Label(self.root, text='0')
        self.LongDisplay = tk.Label(self.root, text='0')
        self.AltDisplay = tk.Label(self.root, text="0")
        self.AzDisplay = tk.Label(self.root, text="0")
        self.ElDisplay = tk.Label(self.root, text="0")
        self.az = tk.OptionMenu(self.root, self.Track,
        *self.Bodies)
        self.MethodMenu = tk.OptionMenu(self.root, self.Method,
        *self.Methods)
        self.ManAzEntry = tk.Entry(self.root)
        self.ManElEntry = tk.Entry(self.root)
        self.TimeEntry = tk.Entry(self.root)

        self.picture = tk.Label(self.root, image=ImageTk.PhotoImage(
        Image.open("/home/pi/Desktop/Sun_Image_Project/Moon.jpg")))

    # Setting position of TK Labels on GUI.
    def setup_gui_grid(self):
        self.LatLabel.grid(row=0, column=0)
```

```
        self.LongLabel.grid(row=1, column=0)
        self.AltLabel.grid(row=2, column=0)
        self.AzLabel.grid(row=3, column=0)
        self.ElLabel.grid(row=4, column=0)
        self.TrackLabel.grid(row=5, column=0)
        self.MethodLabel.grid(row=6, column=0)
        self.ManAzLabel.grid(row=7, column=0)
        self.ManElLabel.grid(row=8, column=0)
        self.ChangeTime.grid(row=9, column=0)
        self.c1.grid(row=10, column=0, columnspan=3)

        self.LatDisplay.grid(row=0, column=1)
        self.LongDisplay.grid(row=1, column=1)
        self.AltDisplay.grid(row=2, column=1)
        self.AzDisplay.grid(row=3, column=1)
        self.ElDisplay.grid(row=4, column=1)
        self.TrackMenu.grid(row=5, column=1)
        self.MethodMenu.grid(row=6, column=1)
        self.ManAzEntry.grid(row=7, column=1)
        self.ManElEntry.grid(row=8, column=1)
        self.TimeEntry.grid(row=9, column=1)
        self.picture.grid(row=0, column=2, rowspan=7)

    # Types of tracking and objects.
    def initialise_tracking_method(self):
        self.Track.set(self.Bodies[1])
        self.Method.set(self.Methods[0])
```

```
# Updates the time if user wishes to change it from
the initial input.
def update_time_gui(self):
    NewTime = self.TimeEntry.get()
    print(type(NewTime))
    StartTime = datetime.datetime.now()
    Time0 = NewTime + ':' + '00'
    print(Time0)
    self.tracking.obs.date = ephem.Date(Time0)

# Allows user to manually set Azimuth and Elevation.
def set_manual_azimuth_elevation(self):
    self.tracking.azimuth = float(self.ManAzEntry.get())
    self.tracking.elevetaion = float(self.ManElEntry.get())
    print(self.tracking.azimuth)

# Initialises the Button on the GUI.
def initialise_gui_buttons(self):
    self.EnterButton = tk.Button(self.root, text =
    'Enter Manual Az/El', command = self.set_manual_azimuth_elevation)
    self.UpdateTime = tk.Button(self.root, text =
    'Set Time', command = self.update_time_gui)
    self.EnterButton.grid(row = 7, column = 2)
    self.UpdateTime.grid(row = 9, column = 2)

# Updates the GUI to new values.
def gui_update(self):
    print("Updating GUI")
```

```
self.LatDisplay.configure(text = self.tracking.latitude)
self.LongDisplay.configure(text = self.tracking.longitude)
self.AltDisplay.configure(text = self.tracking.altitude)
self.AzDisplay.configure(text = self.tracking.azimuth)
self.ElDisplay.configure(text = self.tracking.elevation)
self.AltDisplay.configure(text = self.tracking.altitude)
image = Image.open("/home/pi/Desktop/Sun_Image_Project/Moon.jpg")
photo = ImageTk.PhotoImage(image)
self.picture.configure(image = photo)
self.picture.image = photo
```

F

location.py

```
import datetime
import ephem
import math
import smbus
from pa1010d import PA1010D
import time

# Location class - deals with GPS and manual locations.
class Location:
    def __init__(self, tracking, add):
        self.tracking = tracking
        self.gps = PA1010D()
        self.bus = smbus.SMBus(1)
        self.add = add
        self.StartTime = datetime.datetime.now()

    # Transform data into twos complement.
```

```
def twos_complement(self, data):
    if data >= 32768:
        return data - 65536
    else:
        return data


# Update the GUI settings.
def gps_update(self):
    self.gps.update()
    if (self.gps.data['altitude'] is not None):
        latitude = self.gps.data['latitude']
        longitude = self.gps.data['longitude']
        altitude = self.gps.data['altitude']
        self.tracking.obs.lat = latitude
        self.tracking.obs.long = longitude
        self.tracking.obs.elevation = altitude
    current_time = datetime.datetime.now()
    self.tracking.obs.date = ephem.Date(current_time)


# Sets up new values for position.
def recalibrate_system(self):
    print("Setting Offsets")
    self.gps_update()
    if (self.tracking.gui.manual_calibration.get() == 1):
        self.bus.write_byte_data(self.add, 0x60, 0x80)
        self.bus.write_byte_data(self.add, 0x62, 0x01)
        val0 = self.bus.read_byte_data(self.add, 0x68)
        val1 = self.bus.read_byte_data(self.add, 0x69)
```

```
            val2 = self.bus.read_byte_data(self.add, 0x6A)

            val3 = self.bus.read_byte_data(self.add, 0x6B)
            x_raw = self.twos_complement(float(val1 << 8 | val0))
            y_raw = self.twos_complement(float(val3 << 8 | val2))
            yaw = math.atan2(y_raw, x_raw) * 180 / 3.14159
            if yaw < 0:
                yaw = yaw + 360
        else:
            yaw = 0

    # Get updated location from GPS Unit.
    def auto_location(self):
        result = self.gps.update()
        while (self.gps.data['altitude'] is None):
            print('Connecting to GPS')
            result = self.gps.update()
            time.sleep(10.0)
        self.tracking.latitude = self.gps.data['latitude']
        self.tracking.longitude = self.gps.data['longitude']
        self.tracking.altitude = self.gps.data['altitude']

    # Get manual inputted location from user.
    def manual_location(self):
        self.tracking.latitude = input('Latitude: ')
        self.tracking.longitude = input('Longitude: ')
```

```
self.tracking.altitude = float(input('Altitude: '))
year = input('Year(YYYY):')
month = input('Month(MM) :')
day = input('Day(DD):')
hour = input('Hour(HH) :')
minute = input('Minute(MM) :')
self.StartTime = datetime.datetime.now()
current_time = year + '-' + month + '-' + day + ' '
+ hour + ':' + minute + ':' + '00'
current_time = datetime.datetime.now()
self.tracking.obs.long = self.tracking.longitude
self.tracking.obs.lat = self.tracking.latitude
self.tracking.obs.elevation = self.tracking.elevation
self.tracking.obs.date = ephem.Date(current_time)
```

position_calculation.py

```
import ephem
import time

# PositionCalculaton class - Calculates the celestial objects positions.
class PositionCalculation:
    def __init__(self, tracking):
        self.tracking = tracking

    # Moon azimuth and elevation.
    def get_moon_position(self):
        self.tracking.location.gps_update()
        moon = ephem.Moon(self.tracking.obs)
        print("Moon Azimuth and Elevation: ", moon.az, moon.alt)
        self.tracking.azimuth = moon.az * 180 / 3.14159
        self.tracking.elevation = moon.alt * 180 / 3.14159

    # Sun azimuth and elevation.
```

```
def get_sun_position(self):
    self.tracking.location.gps_update()
    Sun = ephem.Sun(self.tracking.obs)
    print("Sun Azimuth and Elevation: ", Sun.az, Sun.alt)
    self.tracking.azimuth = Sun.az * 180 / 3.14159
    self.tracking.elevation = Sun.alt * 180 / 3.14159

# Gets position of selected body.
def get_target_position(self):
    if (self.tracking.gui.Track.get() == "Moon"):
        print("Getting Moon")
        self.get_moon_position()
    if (self.tracking.gui.Track.get() == "Sun"):
        print("Getting Sun")
        self.get_sun_position()
    if (self.tracking.gui.Track.get() == "Manual"):
        print("Getting manual")
        self.tracking.azimuth = ManAz
        self.tracking.elevation = ManEl

# Sets rotators to azimuth = 0 and elevation = 0.
def reset_rotators(self):
    self.tracking.Ser.reset_input_buffer()
    self.tracking.Ser.reset_output_buffer()
    self.tracking.Ser.write(b'\r')
    input("Switch on Rotator Computer Controller and then press enter")
    time.sleep(1)
    self.tracking.azimuth = 0
```

```
self.tracking.elevation = 0


self.tracking.position_control.send_rotator_command()
print(self.tracking.Ser.read())
self.yaw = 0
```

H

position_control.py

```
import time

# PositionControl class sends positions to the rotators.
class PositionControl:
    def __init__(self, tracking):
        self.tracking = tracking
        self.clear_buffers()

    # Ensure the buffers are clear for next command.
    def clear_buffers(self):
        self.tracking.Ser.reset_input_buffer()
        self.tracking.Ser.reset_output_buffer()

    # Sends the current azimuth and elevation to the computer controller.
    def send_rotator_command(self):
        try:
            self.tracking.azimuth = round(self.tracking.azimuth -
```

```
            self.tracking.yaw)
            self.tracking.elevation = round(self.tracking.elevation)
            self.tracking.azimuth = (self.tracking.azimuth % 360 +
            360) % 360
            azimuth_serial = str(self.tracking.azimuth).zfill(3)
            if self.tracking.elevation > 180:
                elevation_serial = "180"
            elif self.tracking.elevation < 0:
                elevation_serial = "000"
            else:
                elevation_serial = str(self.tracking.elevation).zfill(3)
            print("Sending: w" + azimuth_serial + " "
            + elevation_serial + "\r")
            Cmd = bytes("w" + azimuth_serial + " " + elevation_serial
            + "\r", 'utf-8')
            self.tracking.Ser.write(Cmd)
            time.sleep(1.5)
            self.wait_for_position_match(self.tracking.azimuth,
            self.tracking.elevation)
        except Exception as e:
            print(f"Error in send_rotator_command: {e}")

    # Reads from computer controller the current angles of the rotators.
    def read_current_angles(self):
        try:
            self.tracking.Ser.reset_input_buffer()
            self.tracking.Ser.reset_output_buffer()
            command = b'C2\r'
```

```
                self.tracking.Ser.write(command)


                time.sleep(1.5)
                max_retries = 3
                for _ in range(max_retries):
                    self.tracking.Ser.write(b'\r')
                    time.sleep(1.5)
                    response = self.tracking.Ser.readline().decode().strip()
                    print(f"C2 Response: {response}")
                    if response.startswith("AZ=") and " EL=" in response:
                        parts = response.split()
                        az = int(parts[0].split('=')[1])
                        el = int(parts[1].split('=')[1])
                        return az, el
                    else:
                        print("Invalid response received from C2 command.
                        Retrying...")
                        time.sleep(1)
                print("Failed to get valid response after multiple retries.")
                return None, None
            except Exception as e:
                print(f"Error while sending C2 command: {e}")
                return None, None

        # Waits until the rotators are in correct positions.
        def wait_for_position_match(self, target_azimuth, target_elevation,
```

```
tolerance=2):
    while True:
        az, el = self.read_current_angles()
        time.sleep(2)
        if az is not None and el is not None:
            if self.tracking.execute_raster_scan:
                if abs(az - target_azimuth) <= 1
                and abs(el - target_elevation) <= 1 :
                    break
                else:
                    time.sleep(5)
                    self.tracking.raster_scanner.send_raster_command()
            else:
                if abs(az - target_azimuth) <= tolerance and
                abs(el - target_elevation) <= tolerance:
                    break
            else:
                print("Failed to get current position using C2 command.")
```

I

raster_scanner.py

```
import time
import numpy as np

# RasterScanner class performs the raster scanning algorithm.
class RasterScanner:
    defs __init__(self, tracking, HPBW):
        self.tracking = tracking
        self.raster_azimuth = 0
        self.raster_elevation = 0
        self.hpbw = HPBW

    # Adjust Sun position angles by current raster position.
    def update_rotator_position(self, delta_azimuth, delta_elevation):
        self.raster_azimuth = self.tracking.azimuth + delta_azimuth
        self.raster_elevation = self.tracking.elevation + delta_elevation
        self.send_raster_command()
```

```
# Loops through all variations in raster elevation angles.
def move_elevation(self, delta_azimuth):
    for delta_elevation in np.arange(round(self.hpbw),
    -round(self.hpbw)-1, -1):
        self.tracking.check_rotator_duration()
        self.tracking.position_calculation.get_target_position()
        self.update_rotator_position(delta_azimuth, delta_elevation)
        self.tracking.sdr.sampling(delta_azimuth, delta_elevation)
        # self.tracking.capture_image()

# Loops through all variations in raster azimuth angles.
def move_azimuth(self):
    for delta_azimuth in np.arange(-round(self.hpbw),
    round(self.hpbw)+1, 1):
        self.tracking.check_rotator_duration()
        self.move_elevation(delta_azimuth)
        print("Finished scanning")
    self.tracking.sdr.close_sdr()
    self.tracking.sdr.create_image()

# Sends the current raster azimuth and elevation to the computer
controllers.
def send_raster_command(self):
    try:
        self.raster_azimuth = round(self.raster_azimuth -
        self.tracking.yaw)
        self.raster_elevation = round(self.raster_elevation)
        self.raster_azimuth = (self.raster_azimuth % 360 + 360) % 360
```

```
            azimuth_serial = str(self.raster_azimuth).zfill(3)


            if self.raster_elevation > 180:
                elevation_serial = "180"
            elif self.raster_elevation < 0:
                elevation_serial = "000"
            else:
                elevation_serial = str(self.raster_elevation).zfill(3)
            print("Sending: w" + azimuth_serial + " "
            + elevation_serial + "\r")
            Cmd = bytes("w" + azimuth_serial + " "
            + elevation_serial + "\r", 'utf-8')
            self.tracking.Ser.write(Cmd)
            time.sleep(1.5)
            self.tracking.position_control.wait_for_position_match(
            self.raster_azimuth, self.raster_elevation)
        except Exception as e:
            print(f"Error in send_raster_command: {e}")
```

J

sdr.py

```
from rtlsdr import *
import numpy as np
import matplotlib.pyplot as plt

# SDR class - processes SDR sampling and image creation.
class SDR:
    def __init__(self, tracking, data_size = 11, sample_rate = 3.2e6,
    centre_freq = 1.4e9, gain = 49.6, freq_correction = -67, tau = 1/5):
        self.tracking = tracking
        self.sdr = RtlSdr()
        self.data = np.zeros((data_size,data_size))
        self.sdr.sample_rate = sample_rate
        self.sdr.center_freq = centre_freq
        self.sdr.gain = gain
        self.sdr.freq_correction = freq_correction
        self.sdr.set_direct_sampling(0)
        self.num_samples = 2 * tau * sample_rate
```

```
# Single scan for particular position of antenna.
def sampling (self, az_element, el_element):
    samples = self.sdr.read_samples(self.num_samples)
    samples = samples[2500:]
    az_index = int(az_element + self.data.shape[0] // 2)
    el_index = int(-el_element + self.data.shape[1] // 2)
    avg_pwr = np.mean(abs(samples)**2)
    self.data[az_index, el_index] = avg_pwr
    print(az_index, el_index, avg_pwr)

# Creates image of average power from the samples.
def create_image(self):
    self.data = self.data.astype(np.float64)
    np.savetxt('radio_image_data.txt', self.data)
    fig, ax = plt.subplots()
    cax = ax.imshow(self.data, interpolation='nearest',
    cmap='jet', aspect='auto', origin='lower',
    extent=[0, self.data.shape[1], 0, self.data.shape[0]])
    cbar = fig.colorbar(cax)
    ax.axis('off')
    plt.tight_layout()
    plt.savefig('radio_image_sun.png')
    plt.show()

# Shut down sampling of SDR.
def close_sdr(self):
    return
```

```
        self.sdr.close()
```

K

tracking.py

```
from gui import GUI
from sdr import SDR
from raster_scanner import RasterScanner
from position_control import PositionControl
from position_calculation import PositionCalculation
from location import Location
import serial
import ephem
import cv2 as cv
import picamera
import time

# Tracking class - Top class, Processes when the cool down times happen.
class Tracking:
    def __init__(self, HPBW, data_size, sample_rate, centre_freq,
    gain, freq_correction, azimuth=0, elevation=90, latitude=0, longitude=0,
    altitude=0, yaw=0, add=0x1E, body="Sun"):
```

```
        self.azimuth = azimuth
        self.elevation = elevation
        self.latitude = latitude
        self.longitude = longitude
        self.altitude = altitude
        self.yaw = yaw
        self.execute_raster_scan = False
        self.obs = ephem.Observer()
        # self.camera = picamera.PiCamera()
        self.Ser = serial.Serial('/dev/ttyUSB_DEVICE1', 9600,
        bytesize=8, timeout=1, stopbits=serial.STOPBITS_ONE,
        parity=serial.PARITY_NONE)
        self.gui = GUI(self)
        self.raster_scanner = RasterScanner(self, HPBW)
        self.sdr = SDR(self, data_size, sample_rate,
        centre_freq, gain, freq_correction)
        self.position_control = PositionControl(self)
        self.position_calculation = PositionCalculation(self)
        self.location = Location(self, add)
        self.rotator_operation_start_time = time.time()
        self.is_paused = False
        self.cool_down_start_time = None
        self.cool_down_duration = 15 * 60
        self.saved_azimuth = None
        self.saved_elevation = None

    # Take image from camera and save to jpg file.
    def capture_image(self):
```

```
        print("Taking Picture")


        self.camera.capture('/home/pi/Desktop/Moon.jpg')
        img = cv.imread("/home/pi/Desktop/Sun_Image_Project/Moon.jpg")
        if img is not None:
            img = cv.resize(img, (256, 256))
            img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
            cv.imwrite("/home/pi/Desktop/Sun_Image_Project/Moon.jpg", img)
            img = cv.resize(img, (64, 64))
            cv.imwrite("/home/pi/Desktop/Sun_Image_Project/Moon_CNN.jpg",
            img)
        else:
            print("Error: Image not found.")

    # Check if computer controller is on after cool down.
    def is_controller_on(self):
        try:
            self.Ser.write(b'C2\r')
            time.sleep(1)
            response = self.Ser.readline().decode().strip()
            if response in ["AZ=-001  EL=-002", "AZ=-001  EL=-001"]:
                print("Waiting for the controller to be switched on.")
                return False
            return bool(response)
        except Exception as e:
            print(f"Error checking controller status: {e}")
```

```
            return False

    # Restart scanning process after cool down.
    def start_rotator_operation(self):
        self.rotator_operation_start_time = time.time()
        self.is_paused = False
        time.sleep(2)

    # Chooses between cool down and continue scanning.
    def check_rotator_duration(self):
        estimated_time_for_scan = 12

        if self.is_paused:
            while time.time() - self.cool_down_start_time <
            self.cool_down_duration or not self.is_controller_on():
                cool_down_time_left = self.cool_down_duration -
                (time.time() - self.cool_down_start_time)
                if cool_down_time_left > 0:
                    print(f"Rotator is paused for cool down.
                    Time left: {int(cool_down_time_left)} seconds.")
                else:
                    print("Waiting for the controller to be switched on.")
                time.sleep(1)
            if self.is_controller_on():
                self.is_paused = False
                self.start_rotator_operation()
                self.restore_rotators()
        else:
```

```
            time_left = (5*60) - (time.time() -
            self.rotator_operation_start_time)
            print(f"Time left until cool down: {int(time_left)}")
            if time_left < estimated_time_for_scan:
                self.is_paused = True
                self.cool_down_start_time = time.time()
                self.save_rotator_angles()
                self.shutdown_rotators()
                self.check_rotator_duration()

    # Save the rotator angles before cool down for recalibration.
    def save_rotator_angles(self):
        try:
            self.Ser.write(b'AZ?\r')
            time.sleep(1)
            self.saved_azimuth = self.Ser.readline().decode().strip()
            self.Ser.write(b'EL?\r')
            time.sleep(1)
            self.saved_elevation = self.Ser.readline().decode().strip()
            print(f"Saved azimuth: {self.saved_azimuth},
            elevation: {self.saved_elevation}")
        except Exception as e:
            print(f"Error saving rotator angles: {e}")

    # Restore the rotator angles after cool down for recalibration.
    def restore_rotators(self):
        if self.saved_azimuth and self.saved_elevation:
            try:
```

```
                self.Ser.write(f'AZ{self.saved_azimuth}\r'.encode())
                time.sleep(1)
                self.Ser.write(f'EL{self.saved_elevation}\r'.encode())
                print(f"Restored azimuth to {self.saved_azimuth} and elevatio
            except Exception as e:
                print(f"Error restoring rotator angles: {e}")

    # Show users the cool down has started.
    def shutdown_rotators(self):
        print("Shutting down rotators for cooldown.")

    # Close the GUI interface and Serial communication.
    def close_application(self):
        if messagebox.askokcancel("Quit", "Do you want to quit?"):
            self.Ser.close()
            self.gui.root.destroy()

    # Executes the main loop of tracking and raster scanning.
    def main_loop(self):
        self.location.recalibrate_system()
        self.check_rotator_duration()
        # self.capture_image()
        self.position_calculation.get_target_position()
        self.position_control.send_rotator_command()
        try:
            user_input = input("Execute raster scan? True or False ")
            if user_input.lower() in ['true', 'false']:
                self.execute_raster_scan = user_input.lower() == 'true'
```

```
            else:
                raise ValueError("Invalid input")
        except ValueError:
            self.execute_raster_scan = False
        print(self.execute_raster_scan)
        self.gui.gui_update()
        if self.execute_raster_scan:
            self.raster_scanner.move_azimuth()
        self.gui.root.after(1000, self.main_loop)
```

L

Raspberry Pi - main.py

```
from tracking import Tracking
import config

# main calls the system.
def main():
    # Configuration set up.
    HPBW = config.HPBW
    data_size = config.data_size
    sample_rate = config.sample_rate
    centre_freq = config.centre_freq
    gain = config.gain
    freq_correction = config.freq_correction

    # Initialise classes.
    module = Tracking(HPBW, data_size, sample_rate,
    centre_freq, gain, freq_correction)
    module.gui.initialise_tracking_method()
```

```
        module.position_calculation.reset_rotators()
        manual_input = input('Automatically Detect Location? Y/N ')
        if (manual_input == 'Y' or manual_input == 'y'):
            module.location.auto_location()
        else:
            module.location.manual_location()

        module.gui.root.after(1000, module.main_loop)
        module.gui.root.protocol("WM_DELETE_WINDOW", module.close_application)
        module.gui.root.mainloop()

    if __name__ == '__main__':
        main()
```

M

Train Adam SRCNN

```
import os
import numpy as np
import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms, datasets
from torchvision.utils import save_image, make_grid
import matplotlib.pyplot as plt
import tqdm
from PIL import Image
from google.colab import drive
from pathlib import Path

%load_ext google.colab.data_table

# configuration setup
class Config():
```

```
SRCNN_path = 'SRCNN/'
content_path = f'/content/drive/MyDrive/{SRCNN_path}'
data_path = './data/'

content_path = Path(content_path)

GPU = True
device = torch.device("cuda" if torch.cuda.is_available()
and GPU else "cpu")

batch_size = 16
num_epochs = 100
# Change this file path to change checkpoint model
# Set to "None" to start training from scratch
checkpoint_path = content_path / 'Models/SRCNN_checkpoint.pth

# tranforms high-resolution images
transform = transforms.Compose([
    transforms.Resize(224),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
])

# converts high-resolution to low-resolution
@staticmethod
def low_res_transform():
    return transforms.Compose([
```

```
            transforms.GaussianBlur(kernel_size=(5, 5), sigma=(1.5, 1.5)),

            transforms.Resize(224 // 3, interpolation=
            transforms.InterpolationMode.BICUBIC),
            transforms.Resize(224, interpolation=
            transforms.InterpolationMode.BICUBIC),
        ])

    hr_train_dir = content_path / 'DIV2K_train_HR'
    hr_val_dir = content_path / 'DIV2K_valid_HR'

# Setup drive paths
class Setup():
    def __init__(self, config):
        self.config = config
        self.mount()
        self.make_dir()
        self.seed()

    def mount(self):
        drive.mount('/content/drive/')

    def make_dir(self):
        if not os.path.exists(self.config.content_path / 'Models/'):
            os.makedirs(self.config.content_path / 'Models/')
        if not os.path.exists(self.config.data_path):
```

```
            os.makedirs(self.config.data_path)

    def seed(self):
        if torch.cuda.is_available():
            torch.backends.cudnn.deterministic = True
            torch.manual_seed(0)

# Setup dataset
class SRDataset(Dataset):
    def __init__(self, hr_image_dir, transform=None, low_res_transform=None):
        self.hr_image_dir = hr_image_dir
        self.hr_image_filenames = os.listdir(hr_image_dir)
        self.transform = transform
        self.low_res_transform = low_res_transform

    def __len__(self):
        return len(self.hr_image_filenames)

    def __getitem__(self, idx):
        hr_image_path = os.path.join(self.hr_image_dir,
        self.hr_image_filenames[idx])
        hr_image = Image.open(hr_image_path).convert('RGB')

        if self.transform:
            hr_image = self.transform(hr_image)

        lr_image = self.low_res_transform(hr_image)
        if self.low_res_transform else hr_image
```

```
        return lr_image, hr_image


# SRCNN model
class SRCNN(nn.Module):
    def __init__(self):
        super(SRCNN, self).__init__()
        self.patch_extraction = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=9, padding=4),
            nn.ReLU()
        )
        self.non_linear_mapping = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=32, kernel_size=5, padding=2),
            nn.ReLU()
        )
        self.reconstruction = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=3, kernel_size=5, padding=2),
        )
        self.initialise_weights()

    # initialises weights
    def initialise_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.normal_(m.weight, mean=0.0, std=0.001)
                nn.init.constant_(m.bias, 0)

    # performs a forward pass on the network
```

```
    def forward(self, x):
        x = self.patch_extraction(x)
        x = self.non_linear_mapping(x)
        x = self.reconstruction(x)
        return x


# SRCNN Training process
class SRCNNTrainer():
    def __init__(self, config):
        self.config = config
        self.device = config.device
        self.batch_size = config.batch_size
        self.num_epochs = config.num_epochs
        self.transform = config.transform
        self.low_res_transform = config.low_res_transform()
        self.psnr = {'Train': [], 'Validate': []}
        self.loss = {'Train': [], 'Validate': []}
        self.model = SRCNN().to(self.device)
        self.optimizer = self.initialise_optimizer()
        self.train_dataloader, self.val_dataloader = self.initialise_dataset()
        self.start_epoch = self.load_checkpoint()


    # optimiser
    def initialise_optimizer(self):
        params_to_optimize = [
            {"params": self.model.patch_extraction.parameters(), "lr": 1e-4},
            {"params": self.model.non_linear_mapping.parameters(), "lr": 1e-4},
            {"params": self.model.reconstruction.parameters(), "lr": 1e-5}
```

```
        ]
        optimizer = torch.optim.Adam(params_to_optimize)
        return optimizer

    # initialise the training dataset
    def initialise_dataset(self):
        train_dataset = SRDataset(self.config.hr_train_dir,
        transform=self.transform, low_res_transform=self.low_res_transform)
        train_dataloader = DataLoader(train_dataset, batch_size=self.batch_size,
        shuffle=True)
        val_dataset = SRDataset(self.config.hr_val_dir, transform=self.transform,
        low_res_transform=self.low_res_transform)
        val_dataloader = DataLoader(val_dataset, batch_size=self.batch_size,
        shuffle=False)
        return train_dataloader, val_dataloader

    # Number of parameter in model
    def model_output(self):
        params = sum(p.numel() for p in self.model.parameters() if p.requires_grad)
        print("Total number of parameters is: {}".format(params))
        print(self.model)

    # MSE loss
    def loss_MSE(self, input, target):
        loss = nn.MSELoss()
        mse_loss = loss(input, target)
        max_pixel = 1.0
        psnr = 20 * torch.log10(max_pixel / torch.sqrt(mse_loss))
```

```
        return mse_loss, psnr


    # save checkpoint model in training
    def save_checkpoint(self, epoch):
        checkpoint_path = self.config.content_path /
        'Models/SRCNN_checkpoint_epoch_{}.pth'.format(epoch)
        torch.save({
            'epoch': epoch,
            'model_state_dict': self.model.state_dict(),
            'optimizer_state_dict': self.optimizer.state_dict(),
            'loss': self.loss,
            'psnr': self.psnr
        }, checkpoint_path)


    # loads checkpoint model for training
    def load_checkpoint(self):
        checkpoint_path = self.config.checkpoint_path
        if checkpoint_path.exists():
            checkpoint = torch.load(checkpoint_path)
            self.model.load_state_dict(checkpoint['model_state_dict'])
            self.optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
            self.loss = checkpoint['loss']
            self.psnr = checkpoint['psnr']
            start_epoch = checkpoint['epoch'] + 1
            return start_epoch
        else:
            return 0
```

```
# saves model metrics to .txt file
def save_metrics(self):
    metrics_path = self.config.content_path / 'Models/metrics.txt'
    with open(metrics_path, 'w') as f:
        for epoch in range(self.num_epochs):
            if epoch % 20 == 0 or epoch == self.num_epochs - 1:
                f.write(f"Epoch {epoch}:\n")
                f.write(f"Train Loss: {self.loss['Train'][epoch]}\n")
                f.write(f"Train PSNR: {self.psnr['Train'][epoch]}\n")
                f.write(f"Validate Loss: {self.loss['Validate'][epoch]}\n")
                f.write(f"Validate PSNR: {self.psnr['Validate'][epoch]}\n")

# Train loop
def training(self):
    start_epoch = self.load_checkpoint()
    self.model.train()
    try:
        for epoch in range(start_epoch, self.num_epochs):
            training_loss = 0
            psnr_total = 0
            with tqdm.tqdm(self.train_dataloader, unit="batch") as tepoch:
                for batch_idx, (low, high) in enumerate(tepoch):
                    high_res = high.to(self.device)
                    low_res = low.to(self.device)
                    self.optimizer.zero_grad()
                    reconstructed_images = self.model(low_res)
                    loss, psnr = self.loss_MSE(reconstructed_images, high_res)
                    loss.backward()
```

```
                    psnr_total += psnr.item()
                    training_loss += loss.item()
                    self.optimizer.step()
                    if batch_idx % 20 == 0:
                        tepoch.set_description(f"Epoch {epoch}")
                        tepoch.set_postfix(loss=loss.item()/len(high_res),
                        psnr=psnr.item())
            self.psnr['Train'].append(psnr_total/len(tepoch))
            self.loss['Train'].append(training_loss/len(tepoch))
            self.validate()
            if epoch % 20 == 0 or epoch == self.num_epochs - 1:
                self.save_checkpoint(epoch)
        self.save_metrics()
    except Exception as e:
        print(f"Training interrupted at epoch {epoch}: {e}")
        self.save_checkpoint(epoch)

# Validation loop
def validate(self):
    self.model.eval()
    with torch.no_grad():
        valid_loss = 0
        psnr_total = 0
        with tqdm.tqdm(self.val_dataloader, unit="batch") as tepoch:
            for batch_idx, (low, high) in enumerate(tepoch):
                high_res = high.to(self.device)
                low_res = low.to(self.device)
                reconstructed_images = self.model(low_res)
```

```
                    loss, psnr = self.loss_MSE(reconstructed_images, high_res)
                    valid_loss += loss.item()
                    psnr_total += psnr.item()
                    if batch_idx % 20 == 0:
                        tepoch.set_description(f"Test")
                        tepoch.set_postfix(loss=loss.item()/len(high_res))
            self.psnr['Validate'].append(psnr_total/len(tepoch))
            self.loss['Validate'].append(valid_loss/len(tepoch))


def main():
    config = Config()
    setup = Setup(config)
    trainer = SRCNNTrainer(config)
    trainer.training()


if __name__ == "__main__":
    main()
```

N

SRCNN anaylsis and execution

This script plots the PSNR and loss of training and validation and processes inputted low quality images.

```
import os
import torch
from torchvision.utils import make_grid, save_image
import matplotlib.pyplot as plt
from PIL import Image
from torchvision import transforms
from pathlib import Path
import numpy as np
import torch.nn as nn
import tqdm
from google.colab import drive
from torch.utils.data import Dataset, DataLoader
from torchvision import datasets

drive.mount('/content/drive/')
```

```
# Define config
class Config():
    SRCNN_path = 'SRCNN/'
    content_path = f'/content/drive/MyDrive/{SRCNN_path}'
    data_path = './data/'

    content_path = Path(content_path)

    GPU = True
    device = torch.device("cuda" if torch.cuda.is_available()
    and GPU else "cpu")

    batch_size = 16
    num_epochs = 100

    # tranforms high-resolution images
    transform = transforms.Compose([
        transforms.Resize(224),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5])
    ])

    # converts high-resolution to low-resolution
    @staticmethod
    def low_res_transform():
        return transforms.Compose([
```

```
                transforms.GaussianBlur(kernel_size=(5, 5), sigma=(1.5, 1.5)),

                transforms.Resize(224 // 3, interpolation=
                transforms.InterpolationMode.BICUBIC),
                transforms.Resize(224, interpolation=
                transforms.InterpolationMode.BICUBIC),
            ])


    hr_train_dir = content_path / 'DIV2K_train_HR'
    hr_val_dir = content_path / 'DIV2K_valid_HR'

# Define Dataset
class SRDataset(Dataset):
    def __init__(self, hr_image_dir,
    transform=None, low_res_transform=None):
        self.hr_image_dir = hr_image_dir
        self.hr_image_filenames = os.listdir(hr_image_dir)
        self.transform = transform
        self.low_res_transform = low_res_transform

    def __len__(self):
        return len(self.hr_image_filenames)

    def __getitem__(self, idx):
        hr_image_path = os.path.join(self.hr_image_dir,
```

```
            self.hr_image_filenames[idx])
            hr_image = Image.open(hr_image_path).convert('RGB')

            if self.transform:
                hr_image = self.transform(hr_image)

            lr_image = self.low_res_transform(hr_image) if
            self.low_res_transform else hr_image

            return lr_image, hr_image

    # Plots images from dataset: high, low and reconstructed
    # Plots PNSR and loss of training and validation
    class Results():
        def __init__(self, model, device, content_path):
            self.model = model
            self.device = device
            self.content_path = content_path

        def denorm(self, x):
            device = x.device
            mean = torch.tensor([0.5, 0.5, 0.5], device=device).view(3, 1, 1)
            std = torch.tensor([0.5, 0.5, 0.5], device=device).view(3, 1, 1)
            x = x * std + mean
            return x

        def display_dataset(self, val_dataloader, low_res_transform):
            sample_inputs, _ = next(iter(val_dataloader))
```

```
        fixed_input = sample_inputs[0:32, :, :, :]
        img = make_grid(self.denorm(fixed_input), nrow=8, padding=2, normaliz
                        value_range=None, scale_each=False, pad_value=0)
        plt.figure()
        self.show(img)
        plt.savefig(self.content_path / "high_res.png")

        sample_inputs, _ = next(iter(val_dataloader))
        fixed_input = sample_inputs[0:32, :, :, :]
        img = make_grid(self.denorm(low_res_transform(fixed_input)), nrow=8,
        normalize=False, value_range=None, scale_each=False, pad_value=0)
        plt.figure()
        self.show(img)
        plt.savefig(self.content_path / "low_res.png")

        print('-' * 50)
        with torch.no_grad():
            recon_batch = self.model(fixed_input.to(self.device))
            recon_batch = recon_batch.cpu()
            recon_batch = make_grid(self.denorm(recon_batch), nrow=8, padding
            normalize=False,value_range=None, scale_each=False, pad_value=0)
            plt.figure()
            self.show(recon_batch)
            plt.savefig(self.content_path / "recon_images.png")

    def plot_loss_psnr(self, psnr, loss):
        plt.figure(figsize=(9, 6))
        plt.plot(psnr['Train'], color='red', label='PSNR')
```

```
plt.title("PSNR - Training")
plt.xlabel('Epochs')
plt.ylabel('PSNR')
plt.legend()
plt.savefig(self.content_path / "training_psnr.png")

plt.figure(figsize=(9, 6))
plt.plot(loss['Train'], color='blue', label='Loss')
plt.title("Loss - Training")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig(self.content_path / "training_loss.png")

plt.figure(figsize=(9, 6))
plt.plot(psnr['Validate'], color='red', label='PSNR')
plt.title("PSNR - Validate")
plt.xlabel('Epochs')
plt.ylabel('PSNR')
plt.legend()
plt.savefig(self.content_path / "validate_psnr.png")

plt.figure(figsize=(9, 6))
plt.plot(loss['Validate'], color='blue', label='Loss')
plt.title("Loss - Validate")
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
```

```
            plt.savefig(self.content_path / "validate_loss.png")


        def save_metrics(self, loss, psnr):
            metrics_path = self.content_path / 'Models/metrics.txt'
            with open(metrics_path, 'w') as f:
                for epoch in range(len(loss['Train'])):
                    if epoch % 20 == 0 or epoch == len(loss['Train']) - 1:
                        f.write(f"Epoch {epoch}:\n")
                        f.write(f"Train Loss: {loss['Train'][epoch]}\n")
                        f.write(f"Train PSNR: {psnr['Train'][epoch]}\n")
                        f.write(f"Validate Loss: {loss['Validate'][epoch]}\n")
                        f.write(f"Validate PSNR: {psnr['Validate'][epoch]}\n")


        def show(self, img):
            npimg = img.cpu().numpy()
            plt.imshow(np.transpose(npimg, (1, 2, 0)))


    # Setup up drive
    class Setup():
        def __init__(self, config):
            self.config = config
            self.mount()
            self.make_dir()
            self.seed()


        def mount(self):
            drive.mount('/content/drive/')
```

```
    def make_dir(self):
        if not os.path.exists(self.config.content_path / 'Models/'):
            os.makedirs(self.config.content_path / 'Models/')
        if not os.path.exists(self.config.data_path):
            os.makedirs(self.config.data_path)


    def seed(self):
        if torch.cuda.is_available():
            torch.backends.cudnn.deterministic = True
            torch.manual_seed(0)


# SRCNN model deffined
class SRCNN(nn.Module):
    def __init__(self):
        super(SRCNN, self).__init__()
        self.patch_extraction = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=64, kernel_size=9,
            padding=4),
            nn.ReLU()
        )
        self.non_linear_mapping = nn.Sequential(
            nn.Conv2d(in_channels=64, out_channels=32, kernel_size=5,
            padding=2),
            nn.ReLU()
        )
        self.reconstruction = nn.Sequential(
            nn.Conv2d(in_channels=32, out_channels=3, kernel_size=5,
            padding=2),
```

```
            )
            self.initialise_weights()

        def initialise_weights(self):
            for m in self.modules():
                if isinstance(m, nn.Conv2d):
                    nn.init.normal_(m.weight, mean=0.0, std=0.001)
                    nn.init.constant_(m.bias, 0)

        def forward(self, x):
            x = self.patch_extraction(x)
            x = self.non_linear_mapping(x)
            x = self.reconstruction(x)
            return x

    # Process low quality images and form higher quality images
    class ImageProcessor():
        def __init__(self, model, device, content_path):
            self.model = model
            self.device = device
            self.content_path = content_path

        def load_image(self, image_path):
            img = Image.open(image_path).convert('RGB')
            transform = transforms.Compose([
                transforms.Resize(224),
                transforms.CenterCrop(224),
                transforms.ToTensor(),
```

```
        ])
        img = transform(img).unsqueeze(0)
        return img

    def process_image(self, image_path):
        img = self.load_image(image_path)
        img = img.to(self.device)
        self.model.eval()
        with torch.no_grad():
            output = self.model(img)
        denorm_output = self.denorm(output)
        return img, output

    def denorm(self, x):
        device = x.device
        mean = torch.tensor([0.5, 0.5, 0.5], device=device).view(3, 1, 1)
        std = torch.tensor([0.5, 0.5, 0.5], device=device).view(3, 1, 1)
        x = x * std + mean
        x = torch.clamp(x, 0, 1)
        return x

    def display_and_save_image(self, input_image, output_image):
        input_image = make_grid(input_image, nrow=8, padding=2,
        normalize=False)
        output_image = make_grid(output_image, nrow=8, padding=2,
        normalize=False)

        plt.figure(figsize=(10, 5))
```

```
            plt.subplot(1, 2, 1)
            self.show(input_image)
            plt.subplot(1, 2, 2)
            self.show(output_image)
            plt.show()

            save_image(output_image, os.path.join(self.content_path,
            'reconstructed_image.png'))

    def show(self, img):
            npimg = img.cpu().numpy()
            plt.imshow(np.transpose(npimg, (1, 2, 0)))
            plt.axis('off')

# Load checkpoint model
def load_model(checkpoint_path, device):
    model = SRCNN().to(device)
    checkpoint = torch.load(checkpoint_path, map_location=device)
    model.load_state_dict(checkpoint['model_state_dict'])
    loss = checkpoint['loss']
    psnr = checkpoint['psnr']
    model.eval()
    return model, loss, psnr

# Setup
config = Config()
setup = Setup(config)
# Change path to model
```

```
checkpoint_path = config.content_path /
'Adam Optimiser/SRCNN_checkpoint_epoch_199.pth'
model, loss, psnr = load_model(checkpoint_path, config.device)

# Dataset
val_dataset = SRDataset(config.hr_val_dir, transform=config.transform,
low_res_transform=config.low_res_transform())
val_dataloader = DataLoader(val_dataset, batch_size=config.batch_size,
shuffle=False)

# Results of the network training and validation
results = Results(model, config.device, config.content_path)
results.display_dataset(val_dataloader, config.low_res_transform())
results.plot_loss_psnr(psnr, loss)
results.save_metrics(loss, psnr)


# Pass image through
image_path = config.content_path / 'Rawlinsons_Sun.png'
image_processor = ImageProcessor(model, config.device, config.content_path)
input_image, output_image = image_processor.process_image(image_path)
image_processor.display_and_save_image(input_image, output_image)
```

Bibliography

[1] ALMA Observatory, *Alma observatory*, https://www.almaobservatory.org/en/home/, Accessed: 2024-06-12.

[2] National Radio Astronomy Observatory, *Virgo's ngc 4567 and ngc 4568*, https://public.nrao.edu/gallery/ngc4567_4568_compositehst_v2/, Accessed: 2024-06-12.

[3] B. Malański and S. Malański, "Build your own radio telescope," *Science in School*, vol. 23, pp. 38–42, 2012. [Online]. Available: https://www.scienceinschool.org/wp-content/uploads/2014/11/issue23_telescope.pdf.

[4] A. Spanakis-Misirlis, *Observing the radio sky with pictor*, Proofread by Dr. Cameron Van Eck, Sep. 2019. [Online]. Available: https://www.pictortelescope.com.

[5] J. G. Mangum, D. T. Emerson, and E. W. Greisen, "The on the fly imaging technique," *Astronomy & Astrophysics*, vol. 474, no. 2, pp. 679–687, Nov. 2007. DOI: 10.1051/0004-6361:20077811.

[6] G. V. S. Gireesh, C. Kathiravan, I. V. Barve, and R. Ramesh, "Radio interferometric observations of the sun using commercial dish tv antennas," *Solar Physics*, vol. 296, no. 8, p. 121, Aug. 2021. DOI: 10.1007/s11207-021-01871-9. [Online]. Available: https://doi.org/10.1007/s11207-021-01871-9.

[7] A. Fadul, A. Abker, A. Ibrahim, M. Khiar, and H. Abdalla, "Converting a tv satellite dish into a small radio telescope," *Journal of The Faculty of Science and Technology (JFST)*, vol. 1, no. 9, pp. 128–134, Oct. 2022. DOI: 10.52981/jfst.v1i9.2748. [Online]. Available: https://www.researchgate.net/

publication/364671943_Converting_a_TV_Satellite_Dish_into_a_Small_Radio_Telescope.

[8] T. Cook, *Moon tracking rotator user guide*, Accessed: 2024-06-12, 2022.

[9] J. Rawlinson, "Amateur radio astronomy imaging in the ku band," *2013*,

[10] "Ragazine volume 1 issue 1," British Astronomical Association. (2013), [Online]. Available: http://www.britastro.org/radio/ragazine/RAGazine_v1_iss1_Sep_2013_hires.pdf (visited on 06/09/2024).

[11] "Armms conference," ARMMS RF and Microwave Society. (2024), [Online]. Available: https://www.armms.org/conferences/?conference=58 (visited on 06/09/2024).

[12] P. Kurczynski *et al.*, "New technology is a 'science multiplier' for astronomy," *Phys.org*, Sep. 2020, Retrieved June 1, 2024. [Online]. Available: https://phys.org/news/2020-09-technology-science-astronomy.html.

[13] K. Schmidt, F. Geyer, S. Fröse, *et al.*, "Deep learning-based imaging in radio interferometry," *Astronomy & Astrophysics*, vol. 664, A134, Aug. 2022. DOI: 10.1051/0004-6361/202142113. [Online]. Available: http://dx.doi.org/10.1051/0004-6361/202142113.

[14] C. Ledig, L. Theis, F. Huszar, *et al.*, *Photo-realistic single image super-resolution using a generative adversarial network*, 2017. arXiv: 1609.04802 [cs.CV].

[15] C. Dong, C. C. Loy, K. He, and X. Tang, *Image super-resolution using deep convolutional networks*, Available: https://arxiv.org/abs/1501.00092v3, arXiv:1501.00092 [cs], 2014.

[16] M. Honma, K. Akiyama, M. Uemura, and S. Ikeda, "Super-resolution imaging with radio interferometer using sparse modeling," *Publications of the Astronomical Society of Japan*, vol. 66, Jul. 2014. DOI: 10.1093/pasj/psu070.

[17] "Raspberry pi 3 model b," Raspberry Pi Foundation. (2024), [Online]. Available: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/ (visited on 06/09/2024).

[18] "Yaesu computer controller (gs-232b) manual," Yaesu. (2024), [Online]. Available: https://www.yaesu.com/jp/manuals/yaesu_m/GS-232B_EAA14X002_1803E-AS-1.pdf (visited on 06/09/2024).

[19] "Yaesu computer controller (gs-232b)," Manualslib. (2024), [Online]. Available: https://www.manualslib.com/manual/1000473/Yaesu-Gs-232b.html?page=5#manual (visited on 06/09/2024).

[20] "Dual controller," Yaesu. (2024), [Online]. Available: https://www.yaesu.com/indexVS.cfm?cmd=DisplayProducts&ProdCatID=104&encProdID=79A89CEC477AA3B819EE02831F3FD5B8#:~:text=The%20Azimuth%20Rotator%20features%20a,rotation%20range%20of%20180%C2%B0.&text=If%20you're%20just%20starting,for%20most%20all%20Amateur%20applications. (visited on 06/09/2024).

[21] "Google colaboratory," Google. (2024), [Online]. Available: https://colab.research.google.com (visited on 06/09/2024).

[22] T. Latief, S. Winberg, and A. Mishra, "The design of a two-element radio interferometer using satellite tv equipment," in *2021 International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCE)*, Jun. 2021, pp. 1–6. DOI: 10.1109/ICECCE52056.2021.9514076.

[23] K. Abood, "Study of some parabolic radio telescope antenna parameter," Jan. 2016.

[24] R. S. Bhatia, J. Marti-Canales, C. D. Matos, *et al.*, "A simple radio telescope operating at ku band for educational purposes," *IEEE Antennas and Propagation Magazine*, vol. 48, no. 5, pp. 144–152, Oct. 2006. DOI: 10.1109/MAP.2006.277116.

[25] J. E. Barnes. "Ast 110 homework answers - assignment 3." (1999), [Online]. Available: https://home.ifa.hawaii.edu/users/barnes/ast110_99/homework/ans3.html (visited on 06/09/2024).

[26] D. Morgan, *[pdf] experiments with a software defined radio telescope - free download pdf*, Available: http://www.britastro.org/radio/projects/An_SDR_Radio_Telescope.pdf, Accessed: 2024-01-09.

[27] D. Patel and B. Patel, "Low cost and robust solar tracking system based on data of daily and seasonal variation in sun position regard to specific location on earth," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 3, pp. 15 888–15 893, Sep. 2014. DOI: 10.15680/IJIRSET.2014.0309014.

[28] J. Agajo, O. Nosiri, O. Chukwuejekwu, and N. Okoro, "Modelling, simulation and analysis of a low-noise block converter (lnbc) used for communication satellite reception using matlab," *International Journal of Engineering and Technical Research (IJETR)*, vol. 3, pp. 67–75, Sep. 2015.

[29] "Satellite frequency bands," European Space Agency (ESA). (2024), [Online]. Available: https://www.esa.int/Applications/Connectivity_and_Secure_Communications/Satellite_frequency_bands (visited on 06/09/2024).

[30] FUNcube Dongle, *Funcube dongle*, Available: https://www.funcubedongle.com/, Accessed: 2024-06-12.

[31] RFSPACE, *Spectravue*, Available: http://www.rfspace.com/RFSPACE/SpectraVue.html, Accessed: 2024-06-12.

[32] Nooelec, *Nooelec nesdr smart v5 sdr*, Available: https://www.nooelec.com/store/sdr/sdr-receivers/nesdr-smart-sdr.html, Accessed: 2024-06-12.

[33] Realtek Semiconductor Corp., *Realtek*, Available: https://www.realtek.com/, Accessed: 2024-06-12.

[34] A.H. Systems, *Frequency band designations*, https://www.ahsystems.com/notes/frequency%20band%20designations.php, Accessed: 2024-06-12.

[35] everything RF, *What is a bias tee?* Available: https://www.everythingrf.com/community/what-is-a-bias-tee, Accessed: 2024-06-12.

[36] J. J. Condon and S. M. Ransom, *Essential Radio Astronomy (Princeton Series in Modern Observational Astronomy): 2*, Illustrated edition. Princeton University Press, 2016.

[37] D. Stuart, *Lab2: Ac signals, capacitors, inductors, and frequency dependence of filters*, Available: https://dstuart.physics.ucsb.edu/Lgbk/pub/E41013.dir/E41013.html, Accessed: 2024-06-12.

[38] K. Sankar, *Digital implementation of rc low pass filter*, Available: https://dsplog.com/2007/12/02/digital-implementation-of-rc-low-pass-filter/, Accessed: 2024-06-12, 2007.

[39] Gnuplot, *Gnuplot*, Available: http://www.gnuplot.info/, Accessed: 2024-06-12.

[40] W. Nsengiyumva, S. G. Chen, L. Hu, and X. Chen, "Recent advancements and challenges in solar tracking systems (sts): A review," *Renewable and Sustainable Energy Reviews*, vol. 81, pp. 250–279, Jan. 2018. DOI: 10.1016/j.rser.2017.06.085.

[41] A. R. Bhowmik, D. Dey, and S. Mukherjee, *Estimation and analysis of solar parameters in north-east india*, Available: https://papers.ssrn.com/abstract=3492866, Nov. 2019. DOI: 10.2139/ssrn.3492866.

[42] K.-K. Chong and C.-W. Wong, "Open-loop azimuth-elevation sun-tracking system using on-axis general sun-tracking formula for achieving tracking accuracy of below 1 mrad," in *2010 35th IEEE Photovoltaic Specialists Conference*, Jun. 2010, pp. 003 019–003 024. DOI: 10.1109/PVSC.2010.5614088.

[43] S. Bularka and A. Gontean, "Hybrid-loop controlled solar tracker for hybrid solar energy harvester," in *2017 25th Telecommunication Forum (TELFOR)*, Nov. 2017, pp. 1–4. DOI: 10.1109/TELFOR.2017.8249413.

[44] R. Woods, H. Apfelbaum, and E. Peli, "Dlp-based dichoptic vision test system," *Journal of Biomedical Optics*, vol. 15, p. 016 011, 2010. DOI: 10.1117/1.3292015.

[45] K. O'Shea and R. Nash, *An introduction to convolutional neural networks*, Available: http://arxiv.org/abs/1511.08458, arXiv:1511.08458 [cs], 2015.

[46] J. Yang, J. Wright, T. S. Huang, and Y. Ma, "Image super-resolution via sparse representation," *IEEE Transactions on Image Processing*, vol. 19, no. 11, pp. 2861–2873, Nov. 2010. DOI: 10.1109/TIP.2010.2050625.

[47] J. Yang, J. Wright, T. Huang, and Y. Ma, "Image super-resolution as sparse representation of raw image patches," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[48] O. Russakovsky, J. Deng, H. Su, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015. DOI: 10.1007/s11263-015-0816-y.

[49] Pimoroni, *Pa1010d gps breakout*, Available: https://shop.pimoroni.com/products/pa1010d-gps-breakout?variant=32257258881107, Accessed: 2024-06-12.

[50] SDR++, *Sdr++: The bloat-free sdr receiver*, Available: https://www.sdrpp.org/, Accessed: 2024-06-12.

[51] M. Lichtman, *Pysdr: A guide to sdr and dsp using python*, Available: https://pysdr.org/index.html, Accessed: 2024-06-12.

[52] Electronics Tutorials, *Rc integrator theory of a series rc circuit*, Available: https://www.electronics-tutorials.ws/rc/rc-integrator.html, Accessed: 2024-06-12.

[53] J. O. S. III, *Dc blocker*, Available: `https://www.dsprelated.com/freebooks/filters/DC_Blocker.html`, Accessed: 2024-06-12.

[54] E. Agustsson and R. Timofte, "Ntire 2017 challenge on single image super-resolution: Dataset and study," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jul. 2017.

[55] R. Timofte, E. Agustsson, L. V. Gool, M. H. Yang, and L. Zhang, "Ntire 2017 challenge on single image super-resolution: Methods and results," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, Jul. 2017.

[56] M. Nilsback and A. Zisserman, *102 category flower dataset*, Available: `https://www.robots.ox.ac.uk/~vgg/data/flowers/102/`, Accessed: 2024-06-12, 2008.

[57] M. Bevilacqua, A. Roumy, C. Guillemot, and M. L. A. Morel, "Low-complexity single-image super-resolution based on nonnegative neighbor embedding," in *British Machine Vision Conference*, 2012.

[58] ll01dm, *Set 5 & 14 super resolution dataset*, Available: `https://www.kaggle.com/datasets/ll01dm/set-5-14-super-resolution-dataset`, Accessed: 2024-06-12, 2021.

[59] R. Zeyde, M. Elad, and M. Protter, "On single image scale-up using sparse-representations," in *Curves and Surfaces*, 2012, pp. 711–730.

[60] O. Nnadih, A. C. Ugwuoke, and B. Okere, "Calibration of radio telescope using a satellite dish," *The International Journal of Engineering and Science*, vol. 4, pp. 38–44, Jul. 2015.

[61] NVIDIA, *World leader in artificial intelligence computing*, Available: `https://www.nvidia.com/en-gb/`, Accessed: 2024-06-12.