NTNU
Norwegian University of
Science and Technology

Software Tool for Analysis and Design of Antenna Arrays for Small Satellites

Even Birkeland

Master of Science in Electronics
Submission date: June 2017
Supervisor: Egil Eide, IES

Norwegian University of Science and Technology
Department of Electronic Systems

Even Birkeland

Software Tool for Analysis and Design of Antenna Arrays for Small Satellites

Master's thesis in Electronics

Trondheim, June 2017

Supervisor: Egil Eide, IE
Co-supervisor: Irene Jensen, SINTEF

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering (IE)
Department of Electronic Systems

NTNU – Trondheim
Norwegian University of Science and Technology

Abstract

The Norwegian Space Centre wants a commitment to national nano- and micro-satellite projects to cover strategical national interests. AISSat-1 was launched in 2010, and AISSat-2 in 2014. AIS operates on 160 MHz, and on a satellite platform that is significantly smaller than the wavelength, which poses a challenge for the design of the AIS antenna.

To aid the design of the AIS antenna, and other antenna systems for small satellites, it is useful to have a tool that allows for simulation of the antenna system's radiation characteristics with decent accuracy and short computational time, and without the need to create a 3D CAD model of the antennas and satellite platform.

The software tool *SmallsatArray* was developed in this thesis work to fill this need. The software targets antenna arrays where the satellite platform is electrically small compared to the wavelength, thus having a minimal impact on the radiation from the antennas. Through rotation of imported radiation patterns from CST or HFSS and 3D Array calculations, the software lets the user construct arbitrary three-dimensional antenna arrays and calculate the total field of the antenna systems. The results can be displayed using a wide range of plotting options, and can easily exported.

The SmallsatArray software offers a very fast alternative to full-wave simulation software and was found to provide satisfactory accuracy even as the wavelength is reduced close to the satellite dimensions.

Sammendrag

Norsk romsenter ønsker en satsing på nasjonale småsatellittprosjekter for a dekke strategiske nasjonale behov. AISSat-1 ble skutt opp i 2010, og AISSat-2 i 2014. AIS opererer på 160 MHz, og på satellittplattformer som er betydelig mindre enn bølgelengden, noe som er en utfordring for design av AIS antennen.

For å forenkle designprosessen av AIS-antennen, samt andre antennesystemer for små satellitter, er det nyttig å ha et designverktøy som muliggjør simulering av antennesystemets strålingskarakteristikk med god nøyaktighet og med rask beregningstid, og uten å behøve en 3D CAD-modell av antennene og satellitten.

Programvaren *SmallsatArray* ble utviklet i arbeidet med denne oppgaven for å svare dette behovet. Programvaren fokuserer på antennearray hvor satellittplattformen har liten elektrisk størrelse i forhold til bølgelengde, og som følge gir liten innvirkning på strålekarakteristikken fra antennene. Ved å rotere stålingsdiagram importert fra CST eller HFSS og kombinere disse med 3D-antenneteori, lar programmet brukeren designe vilkårlige tredimensjonale antennearray og beregne det totale, samlede strålingsdiagrammet for antennesystemet. Resultatene kan fremvises med et bredt utvalg av plottalternativer, og kan enkelt eksporteres.

SmallsatArray programvaren gir et hurtig alternativ til "full-wave" simuleringsverktøy med god nøyaktighet, selv når bølgelengden reduseres til å nærme seg satellittens størrelse.

Preface

This report is a master's thesis in Electronics written during the spring semester at NTNU, Trondheim 2017. The challenge of designing the antenna simulation software was provided by my co-supervisor, who had seen the SatAF software developed by Gabriele Roseti (EPFL) and wanted a similar tool.

Working on this thesis and the SmallsatArray software has been a fantastic learning experience for me in understanding electromagnetism and antennas. I feel that I have transitioned from seeing electromagnetic waves as a confusing formula in a book, to having an intuitive understanding of the composition of the electromagnetic fields and antennas' radiation mechanism. Being my first major programming project, the work in this thesis has taught many coding techniques and skills that I expect to be very useful later in life.

I have chosen to make SmallsatArray available publicly under an MIT License, giving anyone the right to use and modify the software, in the hope that others will find my work useful.

I want to than my supervisors Egil Eide and Irene Jensen for trusting me with this exciting task and for their feedback and help throughout the project. I also want to thank the people in the mechanical workshop at the institute for building the satellite model for my physical measurements very swiftly.

Contents

List of Tables

List of Figures

Abbreviations

AIS	Automatic Identification System
CAD	Computer Aided Design
CST	Computer Simulation Technology
EM	Electromagnetism/electromagnetic
EPFL	École polytechnique fédérale de Lausanne
GNB	Generic Nanosatellite Bus
GUI	Graphical User Interface
HFSS	High Frequency Electromagnetic Field Simulation
HPBW	Half-Power Beam-Width
ISO	International Organization for Standardization
RMS	Root Mean Square

Symbols

α	First rotation angle
β	Second rotation angle
γ	Third rotation angle
η	Wave impedance
θ	Elevation angle
λ	Wavelength $\lambda = c/f$
π	Standard constant; ratio of circumference to the diameter of a circle
ϕ	Azimuth angle
Φ	Phase angle
Ω	Solid angle
c	Exact speed of light in a vacuum (299,792,458 m/s)
D	Directivity
$\mathbf{E}$	Electric field vector
e	Euler's number
f	Frequency
G	Gain
$\mathbf{H}$	Magnetic field vector
I_0	Current amplitude
j	Imaginary unit
k	Wave number, $k = 2\pi/\lambda$
n	Selected/current antenna element being evaluated
N	Total number of antenna elements in array
r	Radial distance
U	Radiation intensity
u, v	Arbitrary vectors
x, y, z or X, Y, Z	Cartesian coordinates

Chapter 1

Introduction

1.1 Context

The space industry has seen a great influx in nano- and microsatellites in later years, and the number of small satellites launched is predicted to continue to grow year over year [6]. The CubeSat standard has become a very popular way for universities to engage in space-technology activities, while there is a considerable number of other standards popular in the private and national sectors.

Norway's first national satellites, AISSat-1 (2010) and AISSat-2 (2014), are nanosatellites based on the modular *GNB* (Generic Nanosatellite Bus) [4]. These satellites receive AIS (Automatic Identification System) signals transmitted by ships. AIS was initially intended only to be received by other vessels and land based stations, but these satellites demonstrated the capability of receiving the AIS signal from space. This lets the supervision of vessels expand beyond the horizon, which is extremely important for Norway because of the vast ocean areas it controls and the increased maritime activity in the Arctic region caused by the melting of the Arctic ice.

Because of the success of these satellites, the Norwegian Space Centre wants an increased focus on small national satellites. Norsat-1 and Norsat-2, both scientific research satellites [5], are planned to be launched in 2017, while AISSat-3 is planned to be built [7]. The AIS receiver used in the AIS satellites have four available antenna ports, giving the potentioal to improve the detectability of the AIS signals using an array. Norsat-1 will, in addition to investigating solar radiation and space weather, experiment with improving the detectability of the AIS signals using two antennas.

1.2 Previous Work

In the semester before the work began on this thesis, a preparation project was was completed for 7.5 credits, the same as a regular subject at NTNU. In this project, work on the SmallsatArray software, developed during this thesis, began. The project focused on calculating array factor of arbitrary thee-dimensional antenna arrays. Through this work, the formula to calculate arbitrary three-dimensional antenna arrays was found and validated. The MATLAB GUI program developed in the project had the following capabilities:

- Import far-field radiation diagrams from CST or HFSS, limited to specific exported formats
- Generate radiation patterns for an isotropic radiation source or a variable length dipole antenna element
- Rotate the absolute value of radiation pattern by two rotation angles (ZY-rotation)
- Calculate *array factor* on the total field of arbitrary three-dimensional antenna arrays
- Plot the normalised field pattern or normalised directivity in either polar plot or 3D

This project was not only limited by the time frame dedicated to the project, but also by the way the software was structured. Important strategical decisions of how the data is stored and handled needed to be decided at an early stage in the project, which proved to limit the potential of the software at a later stage when more functionality was added. For example; all the antennas analysed in the project had a radiation pattern that was symmetrical around the Z-axis, and it was therefore concluded that two rotational axes would be sufficient. Another example is that the radiation patterns did not consider the polarisation of field, because the plots they were validated against were only given as absolute values.

It was discovered in the later stages of the preparation project that to add the desired functionalities, the whole program would have to be restructured and rebuilt from the ground up.

1.3 Aim of the Thesis

The thesis' aim is to continue the work on and improve upon the software developed in the preparation project. A list of the desired functionalities of the software is given below:

- Import far-field radiation diagrams from CST or HFSS in all formats
- Separating the $\hat{\theta}$- and $\hat{\phi}$-components of the field to evaluate the polarisation
- Have the ability to rotate by three rotation angles, making it possible to achieve any orientation of the antenna element
- Rotation of the field components
- Expand plot options to include rectangular and 2D plot options, plotting true directivity, and the choice of plotting individual field components and axial ratio
- Improving the user experience by, for example, allowing for saving and opening arrays configured in the software, shortening computational time, and automatically updating input fields with known values

The thesis also aims to give a good description of the software such that it would be possible for others to continue to improve the software and add functionalities.

1.4 Outline

First, chapter 2 gives an insight into the methods available for analysing antenna radiation and reasoning behind the strategy for designing the SmallsatArray software. Chapter 3 gives the theory behind analysis and calculation of the radiation patterns that is used in the thesis and in the program.

In chapter 4, the SmallsatArray software is presented, giving an overview of the core variables and data structures in the program, the core calculations of the program and how the theory from chapter 3 is implemented. The chapter also explores some of the features of the program and its usage, as well as giving an insight into the structure of how the GUI connects to the associated script.

Chapter 5 explains the experiments that were set up in order to test and validate the functionality of the software. The result from the tests are the given in chapter 6. Finally in chapter 7 we reach the conclusion, and suggestions for future work related to the software are presented.

Chapter 2

Background

2.1 EM Simulation Software

The most accurate software tools available to analyse antenna radiation are full-wave EM-simulation software such as ANSYS HFSS and CST Microwave Studio. These require a 3D model of the antennas and the satellite platform, and then solves Maxwell's equations with respect to the boundary conditions introduced by the 3D model.

The results are usually very close to physical experiments, but have the disadvantage of being expensive software and being time consuming in both the creation of the 3D model, and the computational time. For example; simulating the the model described in section 5.2 in CST on a powerful desktop computer took about three minutes, and this simulation must be repeated for any change in the 3D model, such as changing the antenna position, no matter how small the change is.

2.2 SatAF

Gabriele Roseti at EPFL developed a program named ”Satellite Array Factor”, or ”SatAF” for short, as part of his PhD thesis [8]. The SatAF software covers the needs described in section 1.1, however, nor the SatAF source code or the software itself has been made publicly available. Some source code was provided in the appendix the PhD thesis, however it was found to be more difficult to analyse and implement this code, or parts of it, than to find solutions for the SmallsatArray software independently.

The PhD thesis does provide useful analyses on the effects that the satellite platform has on radiation characteristics of the antenna system. Roseti found that for low directive radiation sources in the presence of a metallic object which size is comparable to the wavelength, the radiation pattern would be greatly affected by the scattering and diffraction caused by the object. However, for more directive radiation sources the currents induced in the satellite platform would be smaller and therefore affect the radiation pattern to a lesser extent. Roseti also states that the satellite platform will have a negligible effect on the radiation pattern for satellites where the wavelength is significantly larger than the satellite platform.

2.3 Strategy

Roseti states that the radiation from sources around a satellite body is the result of three components:

- Direct radiation from the sources
- Diffraction and scattering of the field by the satellite body
- Mutual coupling between the elements

Because this thesis is mainly focusing on AIS on nanosatellites, which means the wavelength is about ten times the size of the platform, it is assumed that the radiation of the elements should not be greatly affected by the satellite platform. In addition the latter two points are very difficult to analyse. The SmallsatArray software will therefore only take the direct radiation from the radiation sources into consideration.

The SmallsatArray software tool is meant to analyse the far-field radiation pattern of the antenna system and the polarisation of the field. It is therefore only interesting to look at the relative values and parameters between the elements and at the general shape of the radiation pattern. Because of this, the radiation patterns of each element should be normalised to their average radiated power so that their contribution to the total field is weighed fairly. The relative excitation of the fields can then be used to cause some elements to contribute more than others.

Considering the formerly mentioned criteria and assumptions, the software needs to preform the following operations:

1. Calculate or import the radiation pattern
2. Convert radiation pattern to RMS-normalised E-field
3. Rotate the radiation pattern to represent the desired orientation of the antenna element
4. Calculate the array factor and the total field from all the elements in the array with respect to their position relative to each other
5. Plot the radiation diagram

In addition to preforming these operation, the software should be easy to use, have an intuitive interface, and have a fast computational time.

Chapter 3

Basic Theory

This chapter covers the basic theory for analyses of radiation patterns and the theory used in the SmallsatArray software. First, section 3.1 and 3.2 covers the coordinate systems and different field parameters. Section 3.3 present some analytical formulas for radiation patterns before section 3.4 and 3.5 explain the rotation of the fields and the calculation of the total field.

3.1 Coordinate system

The analysis of antenna arrays and their radiation patterns uses both spherical and Cartesian coordinate systems. For the position of the elements in the array, it is usually more intuitive to use Cartesian coordinates, while the radiation pattern is more intuitively seen with spherical coordinates. The spherical coordinate system used (ISO convention) in this thesis and in the SmallsatArray software is shown in figure 3.1.

Most mathematical operations and functions are much simpler to describe using Cartesian coordinates, and it is often necessary to convert points or vectors from one coordinate system to the other. The relation between the coordinate systems are

$$\begin{aligned} x(\theta,\phi) &= r\sin\theta\cos\phi \qquad & r(x,y,z) &= \sqrt{x^2+y^2+z^2} \\ y(\theta,\phi) &= r\sin\theta\sin\phi \qquad & \theta(x,y,z) &= \arccos\frac{z}{r} \\ z(\theta) &= r\cos\theta \qquad & \phi(x,y) &= \arctan\frac{y}{x} \end{aligned} \tag{3.1}$$

A special case occurs in the conversion from Cartesian to spherical when $x = 0$. As can be seen in equation 3.1, this will result in ϕ being undefined, even though for a non-zero y-value ϕ should be either $\pi/2$ or $3\pi/2$. This needs to be taken into consideration when doing this conversion.

In order to decompose arbitrary field vectors into their $\hat{\theta}$- and $\hat{\phi}$-components it is necessary to know the tangential unit vectors $\hat{\theta}$ and $\hat{\phi}$ which are also shown in figure 3.1. For an arbitrary vector $\vec{v}(\theta,\phi,r)$, the tangential unit vectors, in spherical coordinates, will be

$$\begin{aligned} \hat{\theta} &= (\theta+\pi/2,\phi,1) \\ \hat{\phi} &= (\pi/2,\phi+\pi/2,1) \end{aligned} \tag{3.2}$$

which, using the relation given in (3.1), can be written in Cartesian coordinates as

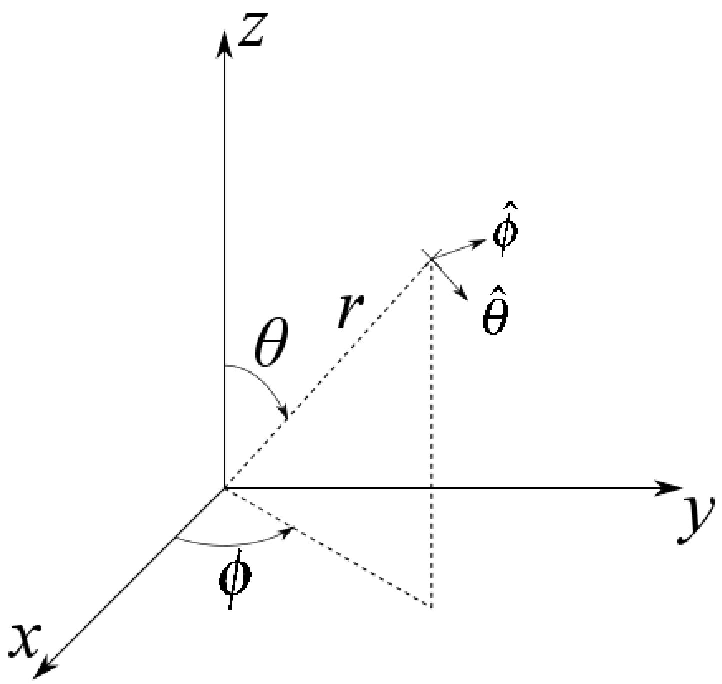


Figure 3.1: Spherical coordinates in ISO convention.

$$
\begin{aligned}
\hat{\theta} &= (\cos\theta\cos\phi, \cos\theta\sin\phi, -\sin\theta) \\
\hat{\phi} &= (-\sin\phi, \cos\phi, 0)
\end{aligned}
\tag{3.3}
$$

A mathematical proof that the vectors in equation 3.2 and 3.3 satisfies the desired properties of the tangential unit vectors is given in the appendix section A.1.

If a arbitrary field vector $\mathbf{E}(\theta, \phi)$ is given and needs to be decomposed into its $\hat{\theta}$- and $\hat{\phi}$-components, this is done by taking the scalar projection of the field vector onto the tangential unit vectors.

$$
\begin{aligned}
E_\theta(\theta, \phi) &= \hat{\theta} \cdot \mathbf{E}(\theta, \phi) \\
E_\phi(\theta, \phi) &= \hat{\phi} \cdot \mathbf{E}(\theta, \phi)
\end{aligned}
\tag{3.4}
$$

3.2 Radiation Pattern

The *radiation pattern* or *antenna pattern* is defined as: “The spatial distribution of a quantity which characterizes the electromagnetic field generated by an antenna” [3]. Although this refers to a transmitting antenna, the properties of a receiving antenna will be identical.

For the work in this thesis, the radiating antenna will be at a distance from the receiving antenna that is much greater than the wavelength, This means that far-field approximation can be used, which means that the radial component of the radiated field can be neglected, and only the tangential field components need to be evaluated. It also means that the position (translation) of the individual antenna elements will have a negligible effect on its associated radiation pattern.

The radiation pattern of an antenna or antenna system can be expressed in many different ways. Commonly used are the electric and magnetic E- and H-fields, radiation intensity, U, directivity, D, gain, G, and power, P.

The SmallsatArray software plots using directivity or E-pattern, but can import data in any of the formats mentioned above. It is therefore necessary to establish methods to covert between them.

3.2.1 Composition of Complex Field Vectors

For the calculations described in this thesis, the fields are considered harmonic and are therefore best described in phasor form. The complex vector field is decomposed into magnitude and phase for each of the $\hat{\theta}$- and $\hat{\phi}$-components in the following way:

$$\mathbf{E}(\theta,\phi) = \hat{\theta}E_\theta(\theta,\phi) + \hat{\phi}E_\phi(\theta,\phi) = \hat{\theta}E_\theta^0(\theta,\phi)e^{j\Phi_\theta(\theta,\phi)} + \hat{\phi}E_\phi^0(\theta,\phi)e^{j\Phi_\phi(\theta,\phi)} \tag{3.5}$$

where E_θ^0 and E_ϕ^0 give the magnitudes of the $\hat{\theta}$- and $\hat{\phi}$-components of the electric field respectivly, and Φ_θ and Φ_ϕ are the phase angles.

3.2.2 Electric and Magnetic Field

The E- and H-field give the electric and magnetic field strength respectively. The E-field is expressed in volts per metre and the H-field in ampere per metre. They are both complex vector-fields with magnitude, phase and direction for any point within the region they are defined. The relation between the fields are given by Maxwell's equations. In the far-field the relation simplifies to

$$\begin{aligned} E_r(\theta,\phi) &\simeq 0 \\ E_\theta(\theta,\phi) &\simeq +\eta H_\phi(\theta,\phi) \\ E_\phi(\theta,\phi) &\simeq -\eta H_\theta(\theta,\phi) \end{aligned} \tag{3.6a}$$

$$\begin{aligned} H_r(\theta,\phi) &\simeq 0 \\ H_\theta(\theta,\phi) &\simeq -\frac{E_\phi(\theta,\phi)}{\eta} \\ H_\phi(\theta,\phi) &\simeq +\frac{E_\theta(\theta,\phi)}{\eta} \end{aligned} \tag{3.6b}$$

[1, p. 137] .

From equation 3.6 it can be seen that the electric and magnetic fields are in-phase in the far-field, and the magnitude relation is given by the wave impedance η, which means that the shape of the E_θ and H_ϕ radiation patterns will be (nearly) identical, and likewise for E_ϕ and H_θ.

3.2.3 Radiation Intensity

The radiation intensity is defined by IEEE as ”In a given direction, the power radiated from an antenna per unit solid angle” [3]. It can be found from the electric field as

$$U(\theta,\phi) = \frac{r^2}{2\eta}|\mathbf{E}(r,\theta,\phi)|^2 = \frac{1}{2\eta}|\mathbf{E}(\theta,\phi)|^2 \tag{3.7}$$

As can be seen from equation 3.7, the radiation intensity is independent of the distance from the antenna, and is therefore only a far-field parameter.

3.2.4 Directivity

Directivity is defined as

$$D(\theta,\phi) = \frac{U(\theta,\phi)}{U_0} = 4\pi\frac{U(\theta,\phi)}{P_{rad}} \tag{3.8}$$

The average radiation intensity U_0 is found from dividing the total radiated power by the solid angle of a whole sphere, 4π;

$$U_0 = \frac{P_{rad}}{4\pi} \tag{3.9}$$

The total radiated power is obtained by integrating the radiation intensity over the entire solid angle of 4π.

$$P_{rad} = \oiint U(\theta,\phi)d\Omega = \int_0^{2\pi}\int_0^{\pi} U(\theta,\phi)\sin\theta\ d\theta\ d\phi \tag{3.10}$$

where $\sin\theta$ appears because the distance on the sphere for each step $d\phi$ near the poles is shorter, in the same way as the longitudinal lines on a globe are more closely spaced near the poles on a globe than at the equator.

3.2.5 Gain and Realised Gain

The gain of an antenna is closely related to its directivity, but also takes into account the efficiency of the antenna. It is defined as “The ratio of the radiation intensity, in a given direction, to the radiation intensity that would be obtained if the power accepted by the antenna were radiated isotropically.”[3], and is expressed mathematically as

$$G(\theta,\phi) = 4\pi\frac{U(\theta,\phi)}{P_{in}} \tag{3.11}$$

A possible way of converting the gain to directivity is to treat it as the radiation intensity and run it through the function (3.8)

$$D(\theta,\phi) = 4\pi\frac{G(\theta,\phi)}{\oiint G(\theta,\phi)d\Omega} = \frac{\cancel{4\pi}\frac{4\pi U(\theta,\phi)}{\cancel{P_{in}}}}{\oiint \cancel{4\pi}\frac{U(\theta,\phi)}{\cancel{P_{in}}}d\Omega} = 4\pi\frac{U(\theta,\phi)}{\oiint U(\theta,\phi)d\Omega} \tag{3.12}$$

Another property of the directivity that is exploited in the SmallsatArray software is that the directivity itself can be run through the same function and still come out as the directivity:

$$D(\theta,\phi) = 4\pi \frac{D(\theta,\phi)}{\oiint D(\theta,\phi) d\Omega} = 4\pi \frac{\cancel{4\pi} \frac{U(\theta,\phi)}{\cancel{P_{rad}}}}{\oiint \cancel{4\pi} \frac{U(\theta,\phi)}{\cancel{P_{rad}}} d\Omega} = 4\pi \frac{U(\theta,\phi)}{\oiint U(\theta,\phi) d\Omega} \tag{3.13}$$

This method works for any radiation pattern expressed in a unit of power, thus a generalised function can be made such that any radiation patten expressed in power (W) can be transformed to directivity.

3.2.6 RMS-Normalised E-Field

When calculating the total field of an array it is important to know that the contribution of each antenna in the array is accurately accounted for. When radiation patterns are imported from CST or HFSS, their values depend on the radial distance between the antenna and the observing field monitor. SmallsatArray is oriented around the relative values between each antenna element and offers the option to change the relative amplitude of the exciting signal, I_0. This is shown in more detail in section 3.5.

To ensure that the fields are represented fairly, they are all normalised to their average radiated power and then multiplied by the excitation signal. In the SmallsatArray software, the RMS-normalised E-field is used in all calculations which is found by normalising the E-field to the square root of the average radiated power:

$$\mathbf{E}_{RMS}(\theta,\phi) = \frac{\mathbf{E}(\theta,\phi)}{\sqrt{U_0}} = \sqrt{D(\theta,\phi)} \tag{3.14}$$

As the conversion from the H- to the E-field was shown in (3.6), from E-field to radiation intensity in (3.7) and from E-field to RMS-normalised in (3.14), it is possible to find $\mathbf{E}_{RMS}$ from any type of radiation pattern from a generelasid function.

3.2.7 Normalisation

A normalised radiation pattern is a radiation pattern where the maximum value is unity, or 0 dB. This is found by dividing the respective field by its maximum value:

$$\mathbf{E}_{Norm}(\theta,\phi) = \frac{\mathbf{E}(\theta,\phi)}{E_{max}} \tag{3.15a}$$

$$\mathbf{D}_{Norm}(\theta,\phi) = \frac{\mathbf{D}(\theta,\phi)}{D_{max}} \tag{3.15b}$$

3.3 Analytical Formulas for Radiation Patterns

In addition to importing radiation patterns, SmallsatArray lets the user generate isotropic radiators and dipole elements within the program.

3.3.1 Isotropic Radiator

The isotropic radiator is a useful inclusion in the software because it enables the opportunity of visualising the array factor by itself. The isotropic radiator is defined by the IEEE as: "A hypothetical, lossless antenna having equal radiation intensity in all directions"[3]. Since the isotropic radiator is defined by the radiation intensity, the polarisation of the element can be chosen. In this program the polarisation of the isotropic source is set to be purely in the $\hat{\theta}$-component.

$$\mathbf{E}(\theta,\phi) = \hat{\theta}(\theta,\phi) \tag{3.16}$$

3.3.2 Dipole Antenna

A thin dipole antenna of length l, aligned with the Z-axis and exited by a current I_0, can be described analytically by the formula

$$\mathbf{E}(\theta,\phi,r) \approx \hat{\theta}E_\theta(\theta,\phi,r) \approx \hat{\theta}\, j\eta\frac{I_0 e^{-jkr}}{2\pi r}\left[\frac{\cos\left(\frac{kl}{2}\cos\theta\right)-\cos\left(\frac{kl}{2}\right)}{\sin\theta}\right] \tag{3.17}$$

[1]

Since the fields in the SmallsatArray software are converted to the RMS-normalised E-field at a later stage, (3.17) can be simplified to

$$E_\theta(\theta,\phi) \approx j\frac{\cos\left(\frac{kl}{2}\cos\theta\right)-\cos\left(\frac{kl}{2}\right)}{\sin\theta} \tag{3.18}$$

3.4 Rotation

When an antenna element is rotated in the program, the corresponding radiation pattern needs to be rotated.

3.4.1 Rotation Matrices

A rotational transformation can be done with rotation matrices. Two of the basic rotation matrices used are given below.

$$R_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \quad \text{and} \quad R_z(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.19}$$

[9, p. 36]

These can be combined in the following way to do three consecutive rotations:

$$
\begin{aligned}
R_{ZYZ}(\alpha,\beta,\gamma) &= R_Z(\alpha)R_Y(\beta)R_Z(\gamma) \\
&= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} \cos\alpha\cos\beta\cos\gamma - \sin\alpha\sin\gamma & -\cos\alpha\cos\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta \\ \sin\alpha\cos\beta\cos\gamma + \cos\alpha\sin\gamma & -\sin\alpha\cos\beta\sin\gamma + \cos\alpha\sin\gamma & \sin\alpha\sin\beta \\ -\sin\beta\cos\gamma & \sin\beta\sin\gamma & \cos\beta \end{bmatrix}
\end{aligned}
\tag{3.20}
$$

The matrix in (3.20) is known as the *ZYZ-Euler Angle Transformation* and it can be used to rotate to any orientation [9, p. 48].

Figure 3.2 shows an illustration of the ZYZ-rotation. Reading the rotation from left to right, the object is first rotated by an angle α around the Z-axis of the fixed coordinate system, then by an angle β around the rotated Y-axis, and finally around rotated Z-axis by an angle γ. If the rotation matrix combination is read from right to left, which better explains the rotation mathematically, the object is first rotated around the fixed Z-axis by the angle γ, then around the fixed Y-axis by β and then again around the fixed Z-axis by α.

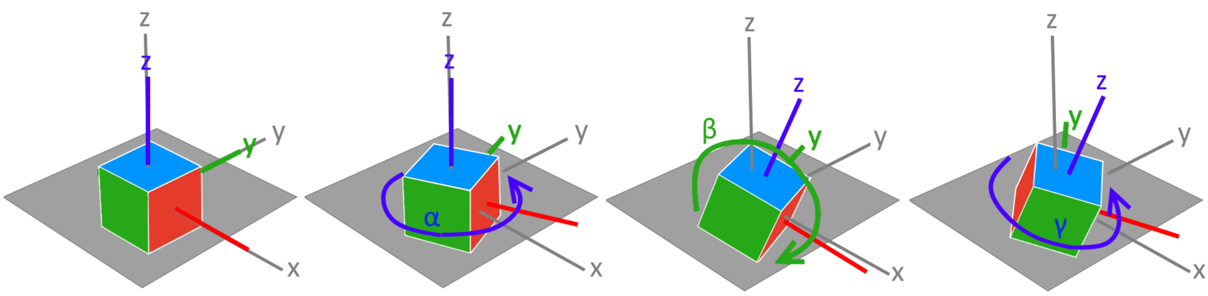


Figure 3.2: Illustration of ZYZ-rotation

It is also useful to have the inverse rotation which can be found by reversing the rotation angles and order or by transposing the matrix [9, p. 34]:

$$
\begin{aligned}
&R_{zyz}(\alpha,\beta,\gamma)^{-1} = R_z(-\gamma)R_y(-\beta)R_z(-\alpha)\\
&= \begin{bmatrix}\cos-\gamma & -\sin-\gamma & 0\\ \sin-\gamma & \cos-\gamma & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos-\beta & 0 & \sin-\beta\\ 0 & 1 & 0\\ -\sin-\beta & 0 & \cos-\beta\end{bmatrix}\begin{bmatrix}\cos-\alpha & -\sin-\alpha & 0\\ \sin-\alpha & \cos-\alpha & 0\\ 0 & 0 & 1\end{bmatrix}\\
&= \begin{bmatrix}\cos\gamma & \sin\gamma & 0\\ \sin-\gamma & \cos\gamma & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\cos\beta & 0 & \sin-\beta\\ 0 & 1 & 0\\ \sin\beta & 0 & \cos\beta\end{bmatrix}\begin{bmatrix}\cos\alpha & \sin\alpha & 0\\ \sin-\alpha & \cos\alpha & 0\\ 0 & 0 & 1\end{bmatrix}\\
&= \begin{bmatrix}\cos\alpha\cos\beta\cos\gamma+\sin-\alpha\sin\gamma & \sin\alpha\cos\beta\cos\gamma+\cos\alpha\sin\gamma & \sin-\beta\cos\gamma\\ \cos\alpha\cos\beta\sin-\gamma+\sin-\alpha\cos\gamma & \sin\alpha\cos\beta\sin-\gamma+\cos\alpha\cos\gamma & \sin-\beta\sin-\gamma\\ \cos\alpha\sin\beta & \sin\alpha\sin\beta & \cos\beta\end{bmatrix}\\
&= \begin{bmatrix}\cos\alpha\cos\beta\cos\gamma-\sin\alpha\sin\gamma & \sin\alpha\cos\beta\cos\gamma+\cos\alpha\sin\gamma & -\sin\beta\cos\gamma\\ -\cos\alpha\cos\beta\sin\gamma-\sin\alpha\cos\gamma & -\sin\alpha\cos\beta\sin\gamma+\cos\alpha\cos\gamma & \sin\beta\sin\gamma\\ \cos\alpha\sin\beta & \sin\alpha\sin\beta & \cos\beta\end{bmatrix}\\
&= R_{ZYZ}^T
\end{aligned}
\tag{3.21}
$$

3.4.2 Rotation of Radiation Patterns

The far-field radiation pattern $\mathbf{E}(\theta_b,\phi_b)$ gives the complex field components in a direction given by θ_b and ϕ_b. The rotation of the radiation pattern from a coordinate system $o_b\theta_b\phi_b r_b$ to $o_a\theta_a\phi_a r_a$ using a rotation matrix R_b^a is written as

$$
\begin{aligned}
\mathbf{E}(\theta_a,\phi_a) &= R_b^a\mathbf{E}(\theta_b,\phi_b)\\
&= R_b^a\left[\hat{\theta}_b E_\theta(\theta_b,\phi_b)+\hat{\phi}_b E_\phi(\theta_b,\phi_b)\right]
\end{aligned}
\tag{3.22}
$$

Using the Cartesian form of the tangential unit vector from (3.3), the rotation is written as

$$
\mathbf{E}(\theta_a,\phi_a) = R_b^a\left[E_\theta(\theta_b,\phi_b)\begin{bmatrix}\cos\theta_b\cos\phi_b\\ \cos\theta_b\sin\phi_b\\ -\sin\theta_b\end{bmatrix}+E_\phi(\theta_b,\phi_b)\begin{bmatrix}-\sin\phi_b\\ \cos\phi_b\\ 0\end{bmatrix}\right]
\tag{3.23}
$$

We still need to express (θ_b,ϕ_b) in terms of (θ_a,ϕ_a). For this, consider the rotation of a vector $\vec{v}(\theta_b,\phi_b,r_b=1)$ written in Cartesian form

$$
\begin{bmatrix}x_a\\ y_a\\ z_a\end{bmatrix} = R_b^a\vec{v}(\theta_b,\phi_b,r_b=1) = R_b^a\begin{bmatrix}\sin\theta_b\cos\phi_b\\ \sin\theta_b\sin\phi_b\\ \cos\theta_b\end{bmatrix}
\tag{3.24}
$$

Using the relation between spherical an Cartesian coordinates given in (3.1), θ_a and ϕ_a are given by

$$
\begin{aligned}
r_a &= \sqrt{x_a^2+y_a^2+z_a^2}\\
\theta_a &= \arccos\frac{z_a}{r_a}\\
\phi_a &= \arctan\frac{y_a}{x_a}
\end{aligned}
\tag{3.25}
$$

Finally, the rotated field vectors can be decomposed by taking the dot product between the field vector and the tangential unit vectors, which is essentially a scalar projection

$$\begin{aligned} E_\theta(\theta_a, \phi_a) &= \hat{\theta}_a \cdot \mathbf{E}(\theta_a, \phi_a) \\ E_\phi(\theta_a, \phi_a) &= \hat{\phi}_a \cdot \mathbf{E}(\theta_a, \phi_a) \end{aligned} \tag{3.26}$$

3.5 Array Factor

The calculation of the radiated field from an array of antenna elements can be broken down into the element factor and the array factor such that the total field is

$$\mathbf{E}(\text{total}) = [\mathbf{E}(\text{single element at reference point})] \times [\text{array factor}] \tag{3.27}$$

[1, p. 287].

The array factor is based on the the interference between the radiated fields of the elements in the array. The interference can be destructive or constructive depending on the phase difference between the elements at the point of observation. The phase differences at the observation point are the result of the phase differences in the excitation signals and the phase differences due to the difference in distance from the observer to each element.

The relative phase difference between the elements due to the relative spatial distance between the elements is found by relating the the phase of all elements to the origin of the coordinate system.

For an element positioned at $P(x, y, z)$, the phase angle relative to the origin from an observation angle given by θ and ϕ is

$$\begin{aligned} \Phi_x(\theta, \phi) &= kx \sin\theta \cos\phi \\ \Phi_y(\theta, \phi) &= ky \sin\theta \sin\phi \\ \Phi_z(\theta, \phi) &= kz \cos\theta \end{aligned} \tag{3.28}$$

where $k = 2\pi/\lambda$, known as the wave number, converts distance into phase angle.

For an array of elements exited by a current amplitude I_0^n and phase delay Φ_n, the array factor is

$$\text{AF} = \sum_n^N I_0^n \exp\left(j\Phi_x(\theta, \phi)\Phi_y(\theta, \phi)\Phi_z(\theta, \phi)\Phi_n\right) \tag{3.29}$$

If all the elements in the array are of the same type and orientation, the total field can be found by simply multiplying the element factor and the array factor as shown in equation 3.27, however if the elements have different radiation characteristics they must also be included in the summation

$$\text{Total field} = \sum_n^N I_0^n \mathbf{E}_n(\theta, \phi) \exp\left(j\Phi_x(\theta, \phi)\Phi_y(\theta, \phi)\Phi_z(\theta, \phi)\Phi_n\right) \tag{3.30}$$

If all the elements radiation patterns are normalised to the same power level, for example the average radiated power, the element factors can be summed separately and then multiplied by the array factor.

Chapter 4

The SmallsatArray Software

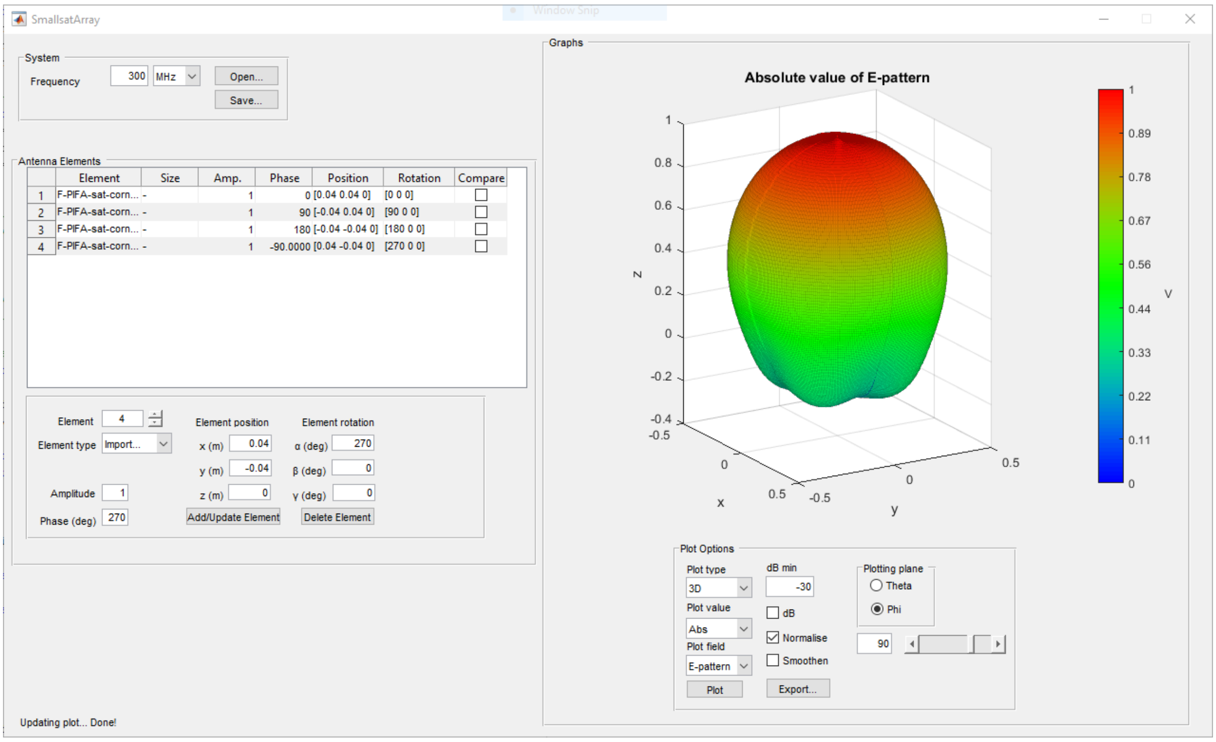


Figure 4.1: Screen-shot of the GUI

In this chapter, we first look at how the radiation patterns and element properties are represented in the software in section 4.2. Then some of the core calculation of the program are described in section 4.3. Some other features and the usage of the program is described in sections 4.4 and 4.5. Finally some insight into the structure of the program is given using flowcharts in section 4.6.

4.1 Description

The SmallsatArray software is a tool to be used for simulating antenna arrays for small satellites. The software can generate radiation patterns for isotropic radiators and variable length dipole elements. It can also import 3D radiation diagrams exported by CST Microwave Studio and HFSS. For HFSS, the data should be given in a unit of power and in decibels, for example directivity (dB), while for CST they can have any format.

The software provides a wide range of plotting options for the far-field of the antenna array and lets the user easily export the figures. It is also possible to save and open the designed antenna arrays either as an array of elements, or as a single antenna.

The software has been made publicly available under an MIT License, which can be found in appendix C, granting anyone the right use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software.

The software is available at `https://github.com/EvenBirk/SmallsatArray` both as MATLAB script and figure, and as an executable running with MATLAB Runtime allowing anyone who don't have a MATLAB license run the software.

4.2 Data Handling and Core Variables

4.2.1 Radiation pattern

Representing a radiation pattern in spherical coordinates in a data structure is similar to drawing a map of the earth on a flat map. For a rectangular map a cylindrical projection has to be used, where the longitudinal lines run parallel vertically. This causes the top and bottom parts of the map to be stretched.

If the radiation pattern has a given resolution in spherical coordinates of, for example, $1°$, so that between each point where the field is observed there is an angle of $1°$ θ or ϕ, the spatial resolution varies between the equator and the poles. For $\theta = 0°$, at the pole, any angles of ϕ refers to the same point in space. In SmallsatArray the radiation patterns are stored in a $\theta \in [0, \pi] \times \phi \in [0, 2\pi]$ matrix, so for a resolution of $1°$ the matrix dimensions will be 361×181, where the entire top and bottom row of the matrix each have 361 entries of the same point observed at different angles of ϕ.

The advantage of this representation is that it is easy to index the entries since consecutive rows and columns represent one step by the angular resolution. It is also the projection used in the files exported by CST and HFSS, simplifying the importation procedure. A flaw of this projection occurs when rotating the radiation patter. This is discussed more in section 4.3.3.

To keep track of the polarisation of the field, SmallsatArray use two of the formerly mentioned matrix containing the $\hat{\theta}$- and $\hat{\phi}$-components of the field. Using complex entries in the matrices, both the magnitude and phase of the field are represented.

While the indices represent degrees, the calculations need the angular values given in radians. There is therefore a 1×181 array vector holding the radian values for $\theta \in [0, \pi]$ and a 1×361 array vector for $\phi \in [0, 2\pi]$ to compliment the field matrices.

Property	Data Type	Description
Type	String	Isotropic, Dipole or Imported
Excitation	Complex double	Complex variable with the amplitude and phase of the excitation
Position	1×3 double	Contains the [x,y,z] position of the element
Rotation	1×3 double	Contains the $[\alpha, \beta, \gamma]$ rotation angles in radians for the desired rotation of the element
CurrentRotation	1×3 double	Contains the $[\alpha, \beta, \gamma]$ rotation angles in radians for the current rotation state of the element
Dimensions	Struct	Struct with the field 'L' holding the length of the antenna, only for dipole
Tag	String	Holds the elements name/label, which is shown in the GUI element table
Compare	Boolean	Marks the element to be used for *compare* mode
E	Struct	Struct containing the fields 'Theta' and 'Phi', each of which are 361×181 complex doubles for the $\hat{\theta}$- and $\hat{\phi}$-component of the RMS-normalised E-field
E_nonrot	Struct	Same as the E-property, but this is not changed when the field is rotated.

Table 4.1: Description of the *Element*-objects properties

The program initially had the option to change the angular resolution so that the user could choose to prioritise either computational time or detail. During the development of the program it became apparent that a resolution of 1° gave sufficient levels of detail while maintaining very fast computations. This option was therefore removed as it simplified the programming tasks, but the remains of this functionality is still present in many of the functions.

4.2.2 Element Properties

All the properties of the elements are kept in a variable named *Element*. This is an 1xN object array, where N is the total number of elements in the array, containing all the parameters of each element and its field. Table 4.1 gives an overview of the properties of the *Element* object.

4.3 Core Calculations

The source code for the functions described in this chapter is provided in appendix B. The following subsections describe the functions in detail to aid in interpreting the code and connecting it to the theory given in chapter 3.

4.3.1 Importing and Generating Radiation patterns

CST and HFSS use different file formats (.txt and .csv) for the exported far fields, but both files consist of a long table where the first two columns give the θ and ϕ angles, while the following columns contain various field parameters. The source code for importing function and its nested functions is given in appendix B.1. The import function prompts the user to choose whether they want to import from CST or HFSS before letting the user select the file to import. The first step in the importing process is to determine the angular resolution of field to be imported, then $\phi \times \theta$ matrices are created whose dimensions is based on the resolution, as explained in section 4.2.1. The the function then simply goes through the list line-by-line, converts the listed angles to indices and stores the listed field values in the matrices after converting to linear field values. Even H-fields are converted using (3.6), for CST-files. Finally the field is interpolated to the resolution used by the program.

For HFSS, the given radiation pattern is expected to be expressed in decibels and in a unit of power. For CST-files, the function reads the headers of imported file to determine what values are provided. Another difference between HFSS and CST is that the function only reads phase values for CST-files. For HFSS-files the phase is set to be 0 over the whole field.

4.3.2 Calculating Field Values

The normalisation of the fields to the average radiated power is done by a function called *RMS-Field*, which source code is provided in appendix B.2. The function first converts the input fields to radiation intensity (U) using (3.7) and then finds the total radiated power P_{rad} using a numeric version of (3.10). Since the integration is done numerically, the total solid angle is summed up during the integration, instead of using 4π to find U_0 as in (3.9). The input fields are finally divided by the average power (U_0) as in (3.8), or the square root of it, as in (3.14) depending on the options chosen in the input.

4.3.3 Rotating Radiation Patterns

The program does the rotations for one cell in the radiation pattern matrices at a time. The first step in the rotation process is to find the relation of the θ and ϕ angles in the initial and rotated coordinate systems. This is using the formula given in (3.24) and (3.25. In the program, however, this relation is found in the reverse order using the rotation matrix (3.21). The rotation is done reversely to ensure that all cells in the output matrices for the fields are filled.

When the radiation pattern is rotated around the Y-axis by, for example, 90°, many cells from the top and bottom rows will map into the same cells for the matrices of the rotated pattern, while cells that are mapped from $\theta = 90°$ to the poles will only fill one of the 361 cells in that row. By rotating reversely, each cell in the matrices of the rotated radiation pattern is mapped to a cell in the matrices representing the radiation pattern in the initial coordinate system.

Many of the cells for the rotated radiation pattern will map to the same cells in the initial radiation pattern, while some cells in the initial radiation pattern will not be mapped to by any cells in the rotated pattern. Thus some information is lost in the rotation.

The source code for the function rotating the radiation patterns can be found in appendix section B.3.

Variable	Options
Plot style	Polar, rectangular, 2D, 3D
Field value	Absolute value, $\hat{\theta}$-component, $\hat{\phi}$-component, AR
Field type	RMS E-pattern, directivity
dB	linear, dB
Normalisation	off, on
Smoothing	off, on
dB min	Any value

Table 4.2: Overview over available plot options

4.3.4 Calculating the Total Field

The summation of the elements radiation pattern is preformed using (3.30) and the implementation is shown in the source code in the appendix, section B.4. Before each elements radiation pattern is summed, matrices for the phase variations due to the position of the element are set up, similarly to (3.28). After the total field is calculated, it calls the *RMSField* function to normalise the field to the average power.

This function is a good example of how the computational time of the program has been kept low. In the function created in the preparation project, only one cell of the matrices were summed at a time, causing the computation of the total field to be as much as about 10 seconds for arrays of 10 elements. With the new implementation of the function, the computation time is reduced to a fraction of a second.

4.3.5 Plotting

The software offers many options for the plotting of the radiation patterns. Table 4.2 gives an overview on the options that can be chosen for the plot.

In addition there is also the option of choosing the intersecting plane for polar and rectangular plots. Finally there is also an option to select to select any number of the antenna elements to compare by plotting their radiation patterns on top of each other.

Figure 4.2 showcases the four different plot styles available. Observe also the markers generated for the main lobe and the half-power beam-width and the display of the main-lobe's value.

It is also possible to export the plots generated by the program and save them as a .png image. To make this work the program has to open a figure temporarily in a new window in which the a copy of the plot is generated. After saving the file, this window is closed automatically, however there is an issue in the case where there is a discontinuity in the plot where one of the HPBW markers should be placed. When this happens, the plotting function will generate an error which causes the program to abort all calculations. When this happens, the temporary plot window is left open and the plot is not saved. The plot can be saved manually in the temporary window, and the user can continue to use the program after closing the window.

The full source code for the functions related to plots are provided in the appendix B.5.

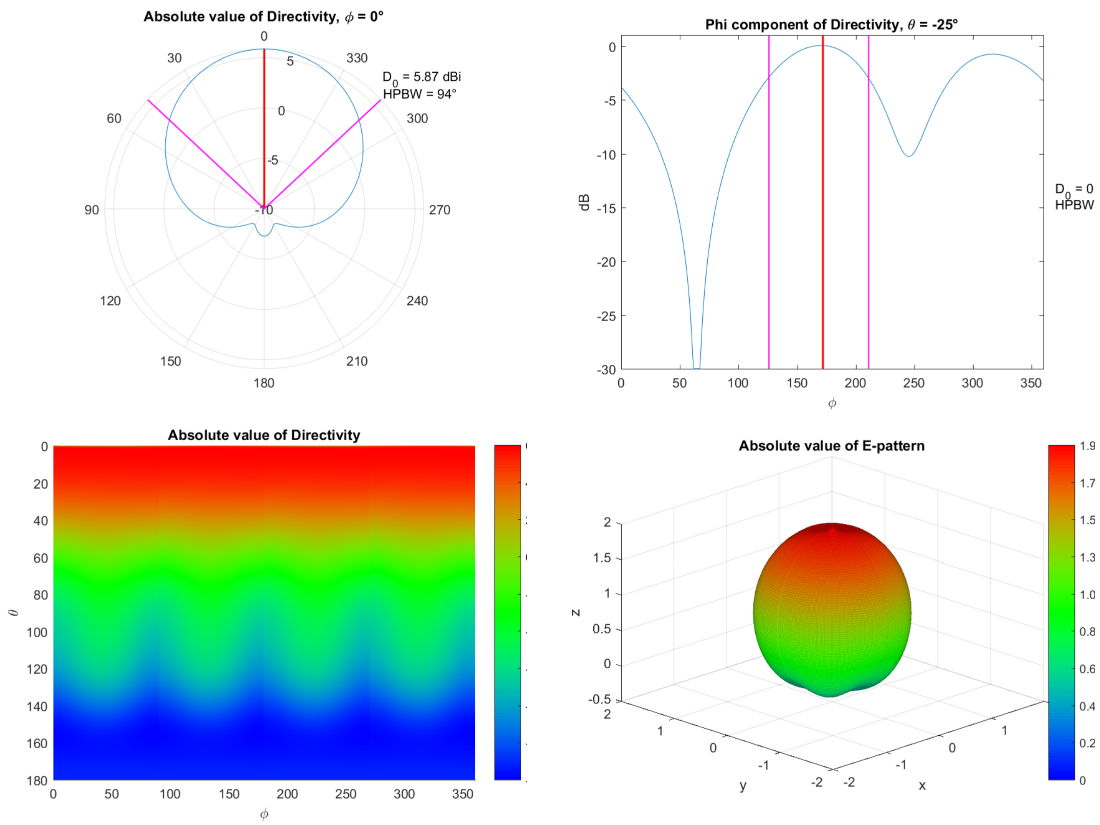


Figure 4.2: The four different plot styles available in SmallsatArray

4.4 Other Features

The Software tool also offers the options of saving and opening arrays created in the program. The flowcharts in figure 4.10 and 4.11 gave an overview of how these functions work. The file-format for these files is *.rpt*. When saving an array of of antennas, the user has the option of saving the session as an array, in which case opening the file will bring back the exact configuration open in the program when saving, or to combine the array into a single element. The latter option gives the lets the user, for example, configure two different arrays which they save as a single element, and later open these two arrays as single elements and compare them with the compare functionality in the program.

4.5 Using the Software

The software is, of course, meant to be played and experimented with, and to be used in any way the user wants. A lot of time has gone into making sure that errors do not occur, and that the calculations are correct, no matter what button the user presses at any time. It is though difficult to predict all actions other users could think of, and therefore bugs and errors can occur. If the program is dysfunctional, it is best to simply close and reopen it. If this does not work, MATLAB must also be relaunched. This is because MATLAB stores global variables even when the SmallsatArray software is restarted. All global variables are cleared or set to their default value during the initialisation of the program, so this is unlikely to happen.

A possible work-flow to design an antenna array is provided below.

1. Set the system frequency
2. Select the element type
3. Enter the antenna's parameters (excitation, position, rotation)
4. Press the (Add/Update Element button)
5. Repeat from point 2 until all the elements are configured
6. Use the plot options to analyse the field
7. Make adjustments to the elements' properties or system frequency
8. Repeat from step 6 to find the desired radiation pattern

If the user wants to edit a parameter of an element, they can simply click on the cell in the GUI table for that parameter and the input fields will automatically be with the properties of the selected element, and the parameter that was selected will be highlighted, making it very easy to make changes to an array.

When importing a radiation pattern, it is important to remember that the position of the antenna(s) in the CST or HFSS simulation is not noted in the far-field radiation diagrams that are imported. Therefore an imported radiation pattern will show up in the program, by default, as being positioned at the origin, while the phase values of the radiation patterns are from a simulation where the position could be different.

Consider, for example the test case in section 5.2, where the satellite platform is centred on the Z-axis, while the element is placed at the corner of the satellite. If this radiation pattern is rotated by $\alpha = 180°$ in SmallsatArray, it will also translate the element to the opposite corner.

4.6 Structure

For a GUI program created in MATLAB, the associated script does not run continuously. When opening the GUI, an *opening function* is run from the script. In the SmallsatArray software, the opening function is used to set global functions to default parameters. The global variables that don't have default values are cleared so that there is no possibility that values from previous sessions will interfere with the program.

After the opening function, the GUI remains passive until an object in the GUI that is associated with a function in the script is manipulated by the user. The flow chart in figure 4.3 shows how manipulation of the different objects in the GUI shown in figure 4.1 connects to the functions in the code. The functions that are called by GUI objects are known as *callback functions*. The following flow charts show the operations from each of the callback functions called by the GUI in figure 4.3.

The callback functions have the input variables *handles*, *eventdata* and *hObject*. Only the variables that are used by the function are listed in the flow charts. The *handles* variable contains all the values and properties of the objects in the GUI. It is from this variable that data such as the element properties are read and stored in global variables in the MATLAB script. After running the functions that have been called, the *handles* variable is returned to the GUI, so by editing its values, it is possible to change the GUI. The *eventdata* variable contains information about what actions the user has done, for example what cell in the GUI table of elements the user clicked and what happened to it. The *hObject* variable simply refers to the object that initialised the function.

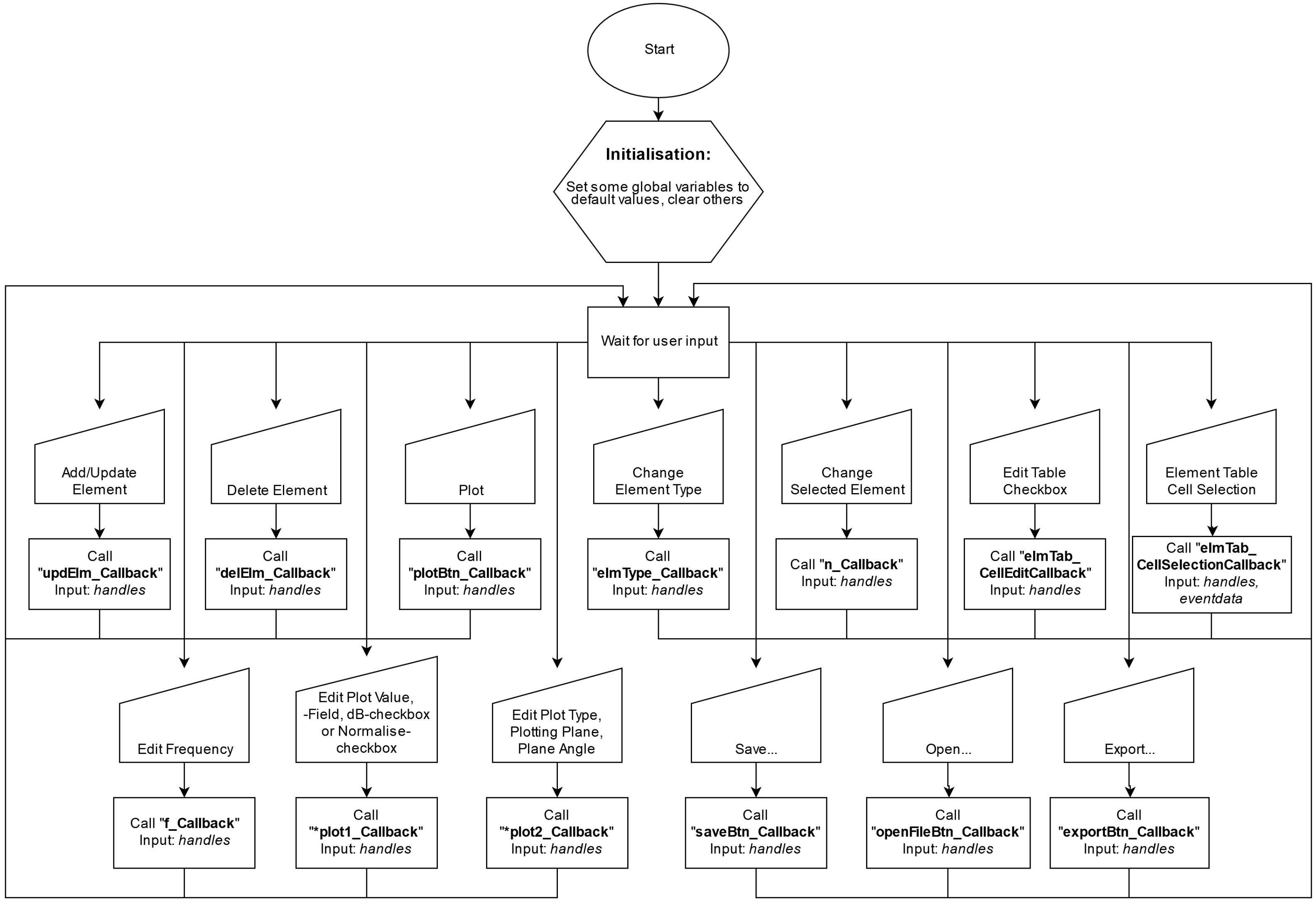


Figure 4.3: Flow chart showing the connection between the GUI and the associated script.

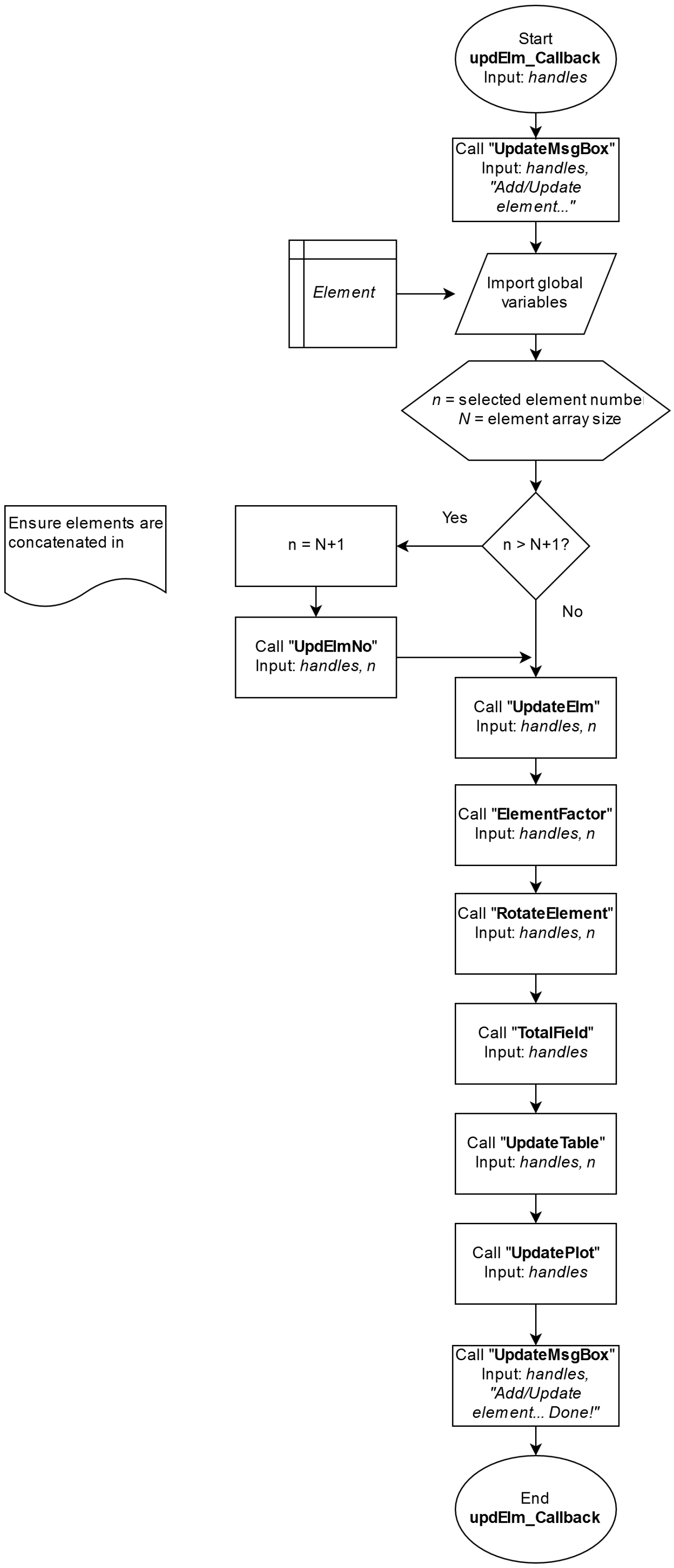


Figure 4.4: Flow chart for the function called when pressing the *Add/Update Element* button.

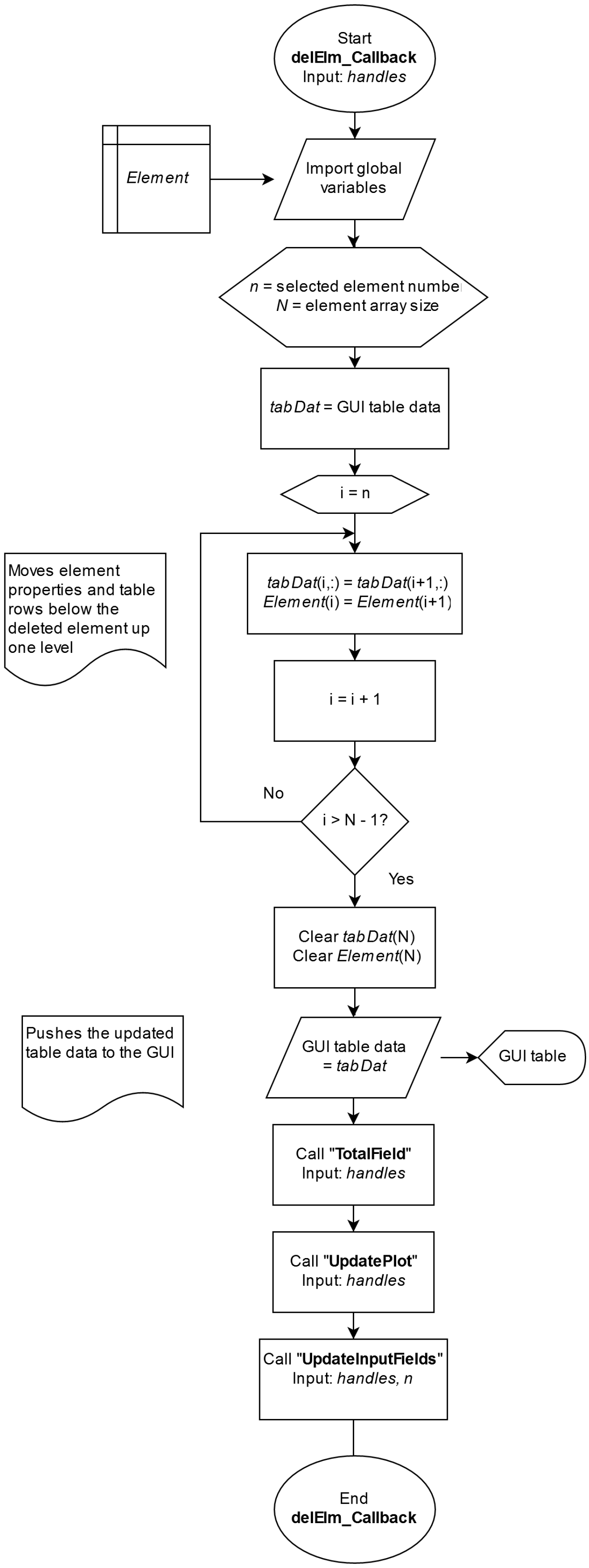


Figure 4.5: Flow chart for the function called when pressing the *Delete Element* button.

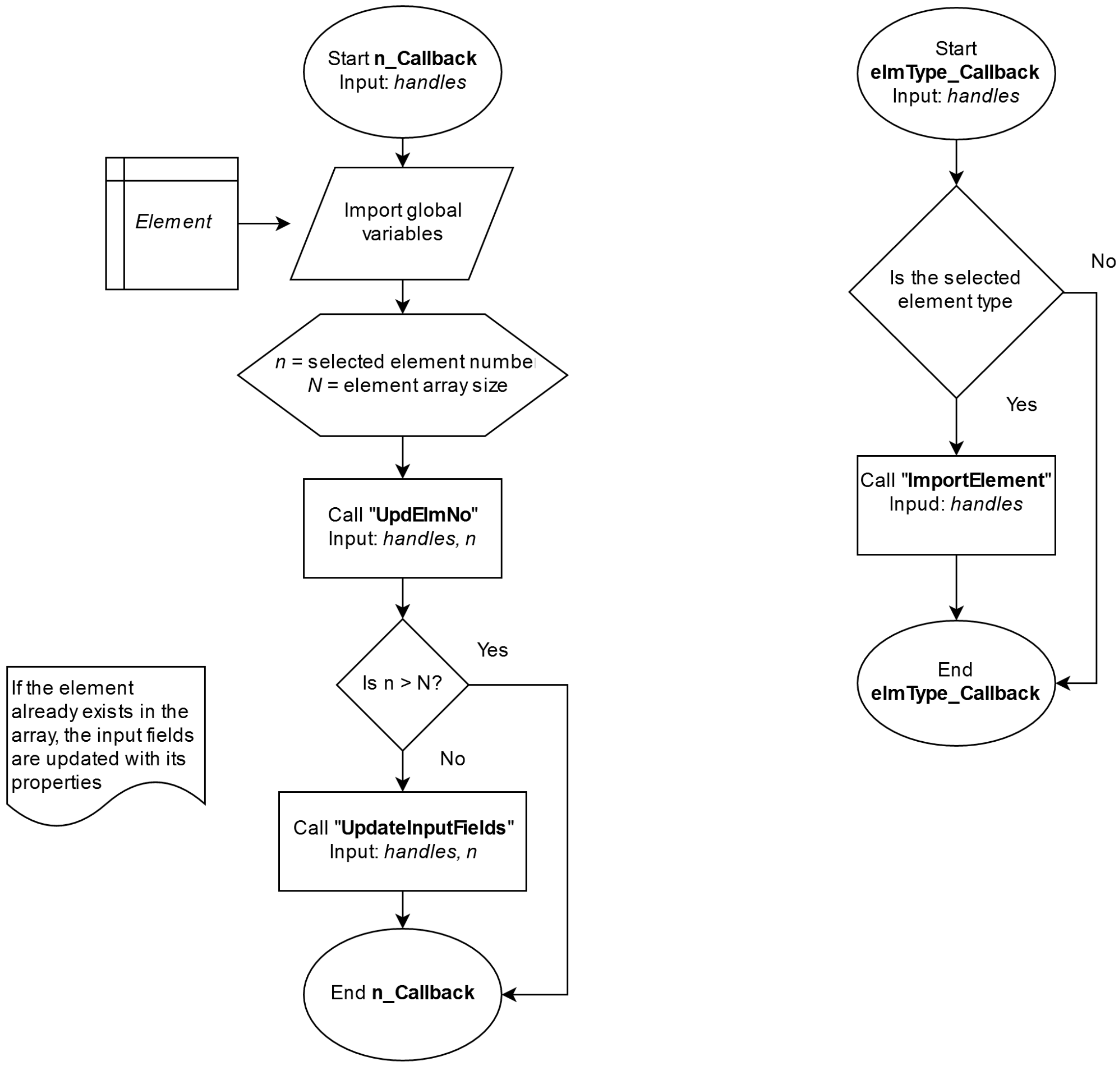


Figure 4.6: Flow charts for the functions called when changing the element number in the GUI (left) and when selecting an element type from the drop-down menu (right).

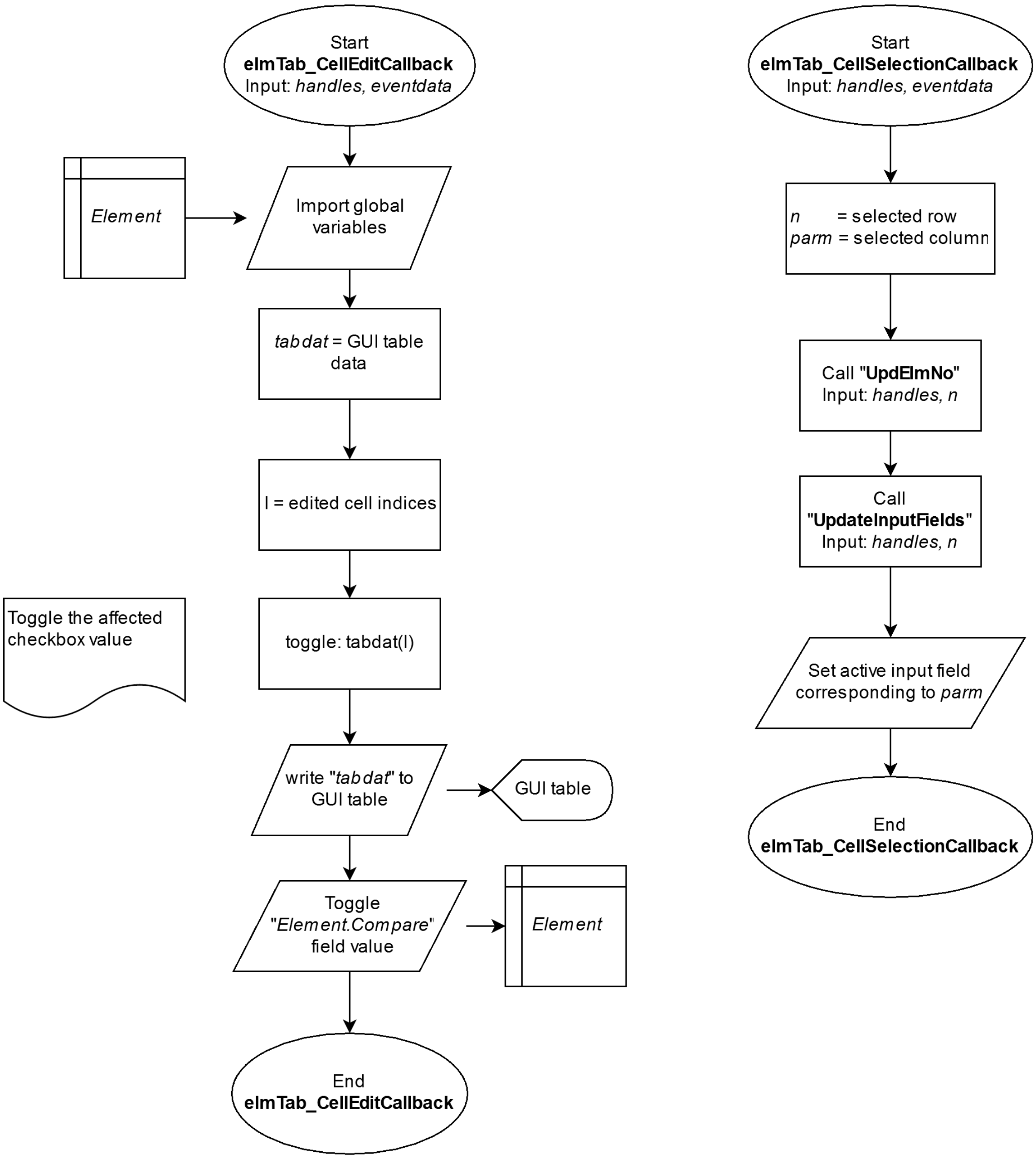


Figure 4.7: Flow charts for the functions called when changing a checkbox in the table of elements in the GUI (left), and when selecting a cell in the in the table (right).

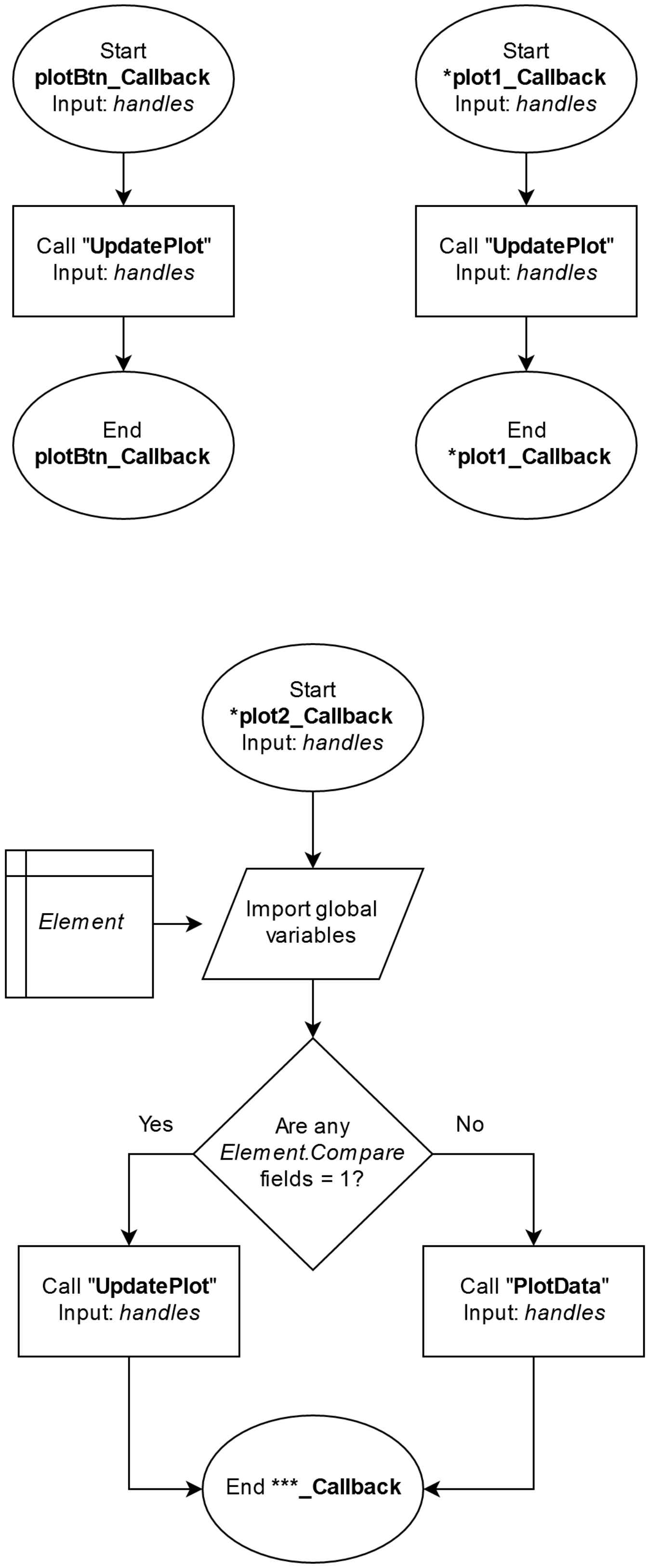


Figure 4.8: Flow charts for the functions called when pressing the *Plot* button (top left), and two different plot callbacks that are called by many actions in the GUI. *plot1_Callback* (top right) is called when the plotting values needs to be recalculated. *plot2_Callback* (bottom) is called when there is just a change in the plotting plane or the plot style.

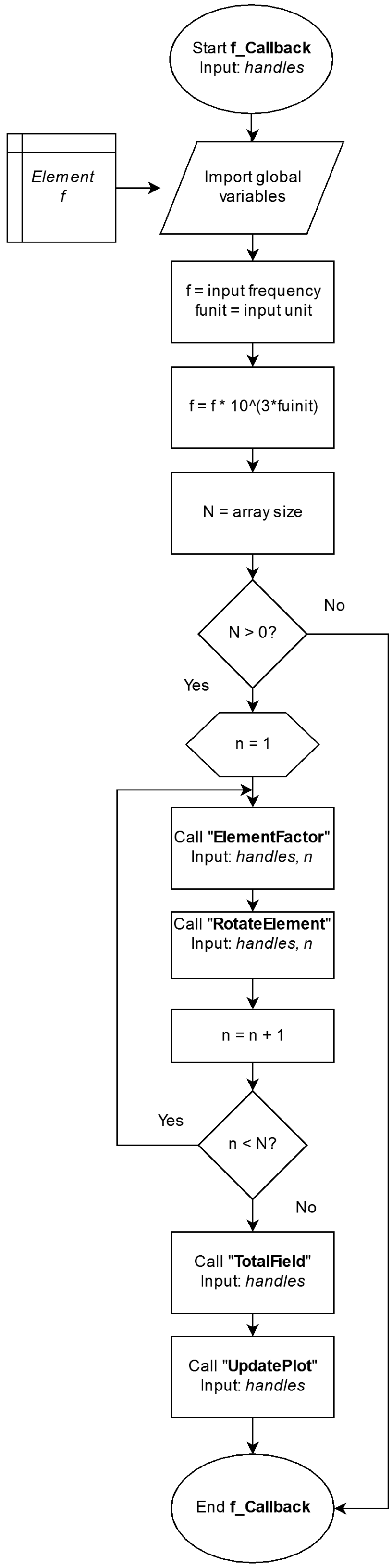


Figure 4.9: Flow chart for the function called when the system frequency is changed.

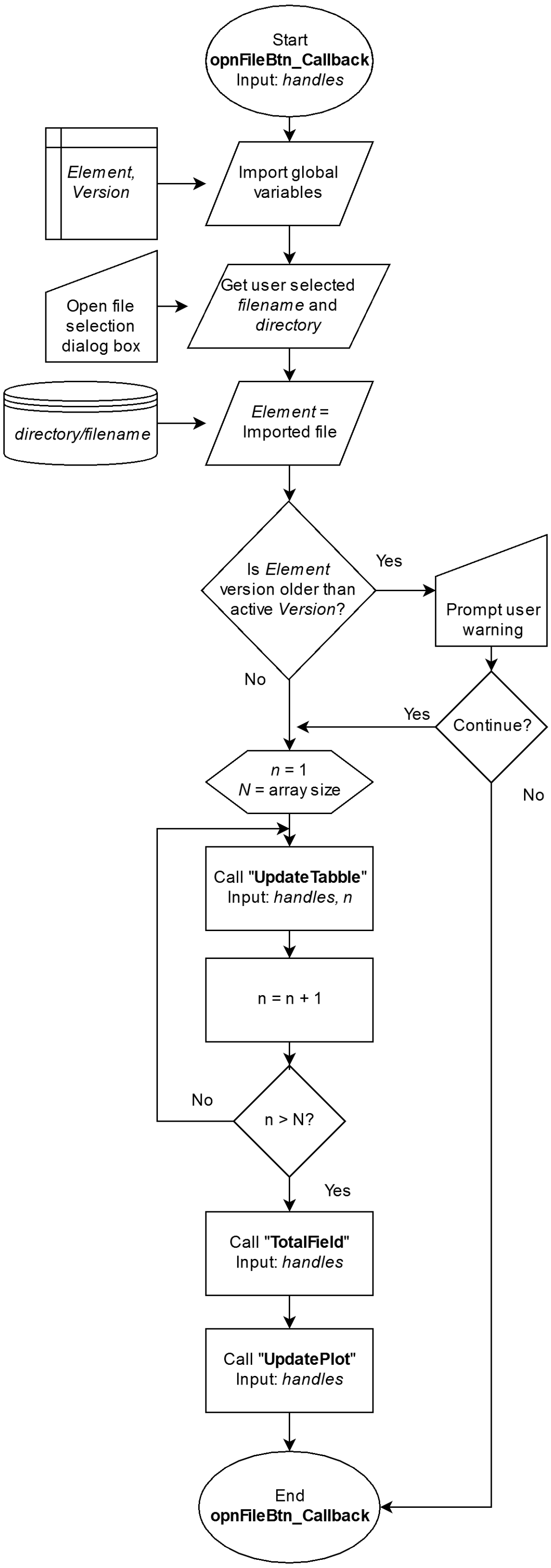


Figure 4.10: Flow chart for the function called when pressing the *Open...* button

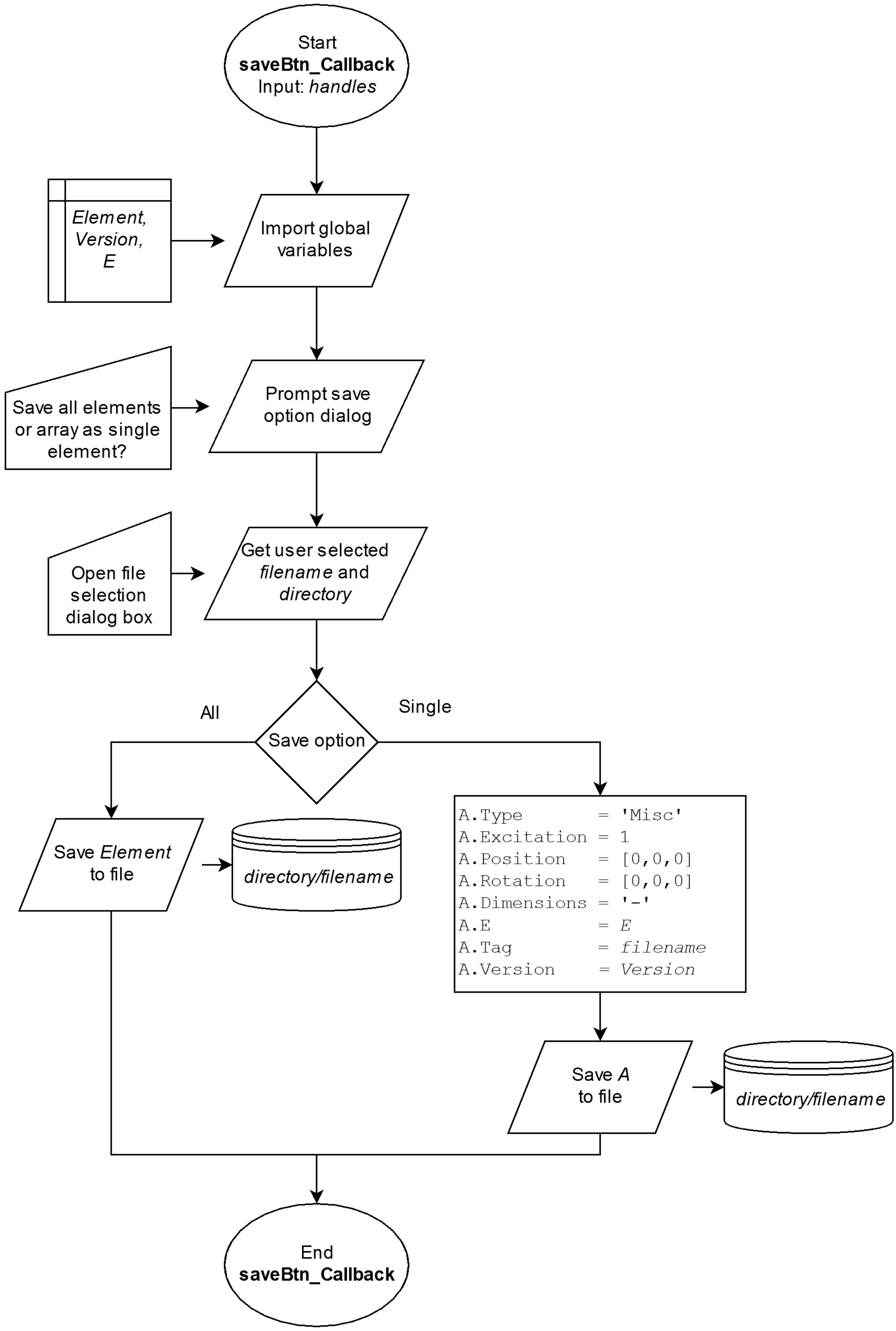


Figure 4.11: Flow chart for the function called when pressing the *Save...* button

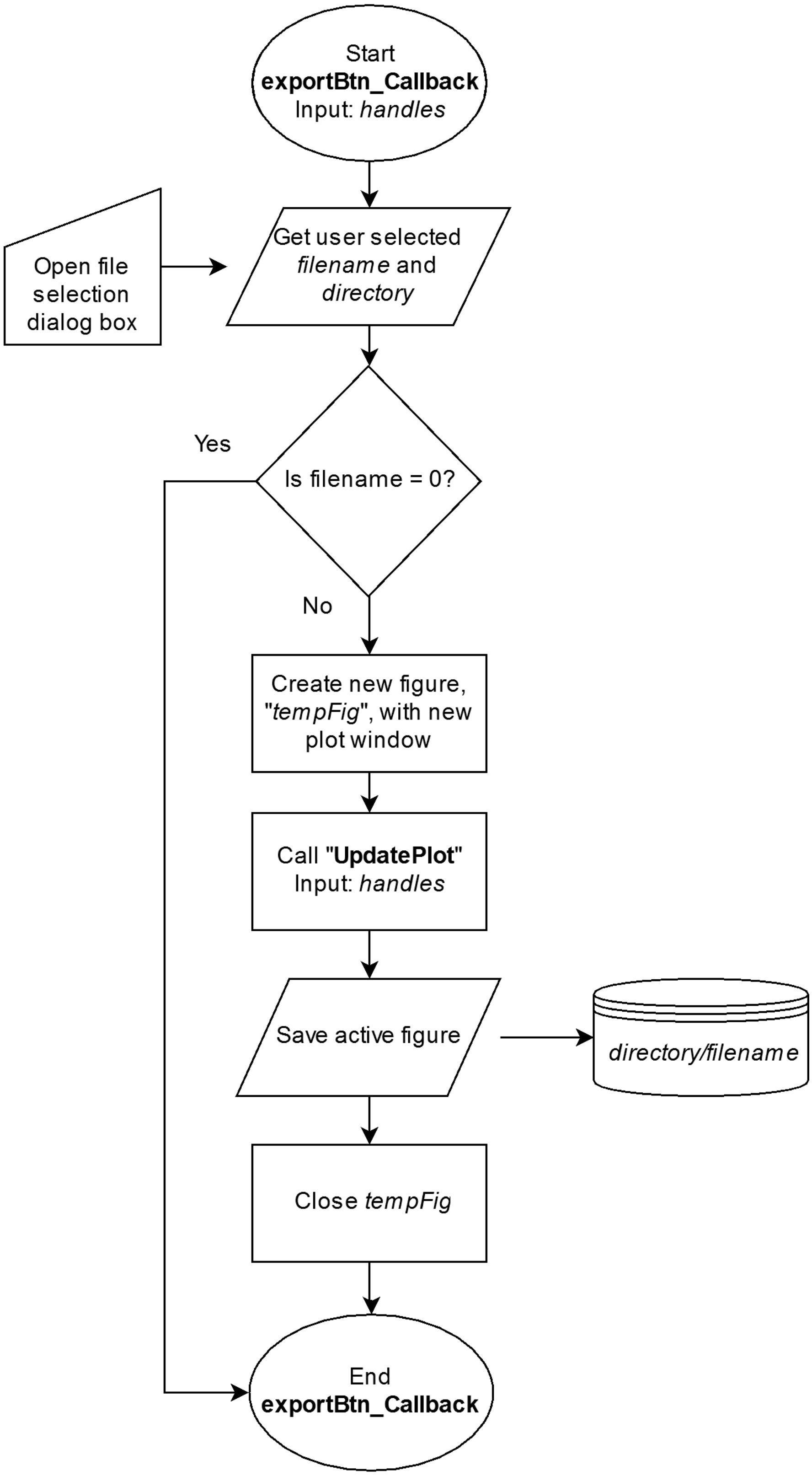


Figure 4.12: Flow chart for the function called when pressing the *Export...* button

Chapter 5

Experiments

In order to validate the results calculated by the SmallsatArray software, three test cases have been set up. Firstly we want to ensure that the radiation patterns that are imported are interpreted and displayed correctly. The second test case considers an antenna array of four folded PIFA elements on a satellite platform which size is less than a third of the wavelength. The third test case uses an array of two monopole elements on a 2U CubeSat platform where the wavelength is only slightly longer than the size of the platform.

5.1 Importing Radiation Patterns From CST

To test that the importation and display of the fields are correct, a loop antenna was simulated in CST and the far-field was exported in all available formats. These formats include, in both linear and decibel values:

- Directivity
- Gain
- Realised gain
- E-field
- E-pattern
- H-field
- Power pattern

All the 14 exported radiation pattern were then imported into the SmallsatArray software and plotted on top of each other using the compare-functionality.

Importing radiation patterns in decibel values of the directivity from HFSS has already been validated in the preparation project [2] to this thesis. Because the program normalises all imported radiation patterns to the average radiated power, all files from HFSS containing values of power expressed in decibels are expected to work with the program.

Results from the next test case were used to test the axial ratio calculations of the program.

5.2 4 PIFA elements on a Cubic Satellite

This test is set up to test the abilities of the program for combining radiation patterns when the satellite platform is small compared to the wavelength. When this is the case, the placement of the element on the satellite platform is expected to have a smaller impact on the radiation pattern.

In this test case (see figure 5.1a), an array of four folded PIFA elements, operating at 956 MHz ($\lambda = 314\,\text{mm}$), where an element is placed in each of the corners of the upwards-facing side of the $10 \times 10 \times 10\ \text{cm}^3$ satellite platform, and consecutive elements are given a rotation and phase delay of 90° to the prior, is considered. This is a configuration which gives excellent circular polarisation and moderate directivity on a small platform.

In addition to the control (figure 5.1a), two configurations of a single element were simulated in CST. One where a single element was placed in the corner of the platform (figure 5.1b) and one where a single element was placed at the centre of the satellite (figure 5.1c). All the simulations were done with the satellite platform centred on the Z-axis.

The exported far-fields from the two latter simulations were then imported into the SmallsatArray software and placed in an array. For the simulation of the element that was placed at the corner of the platform, only a rotation around the Z-axis and a phase delay was configured for the elements. This is because the simulation from CST already contains the phase delays caused by the placement of the object. Reading from the exported file, the listed phase at $\theta = 90°, \phi = 0°$ is 257°, while at $\theta = 90°, \phi = 180°$ is 175°. For the centred element simulation, the imported elements were translated by $\pm 40\,\text{mm}$ on the X- and Y-axis in addition to the rotation and phase delay. This test set-up was shown in figure 4.1.

5.3 2 Angled Monopole Elements on a 2U CubeSat

This test set-up consists of a 2U CubeSat model with two monopole elements, angled at 45° away from the platform, are placed near opposite edges at the top of the satellite, see figures 5.2 and 5.3. The wavelength here is also $\lambda = 314\,\text{mm}$ while the height of the satellite model is $h = 213\,\text{mm}$, thus the placement of the antenna on the satellite is expected to have a greater impact on the radiation pattern.

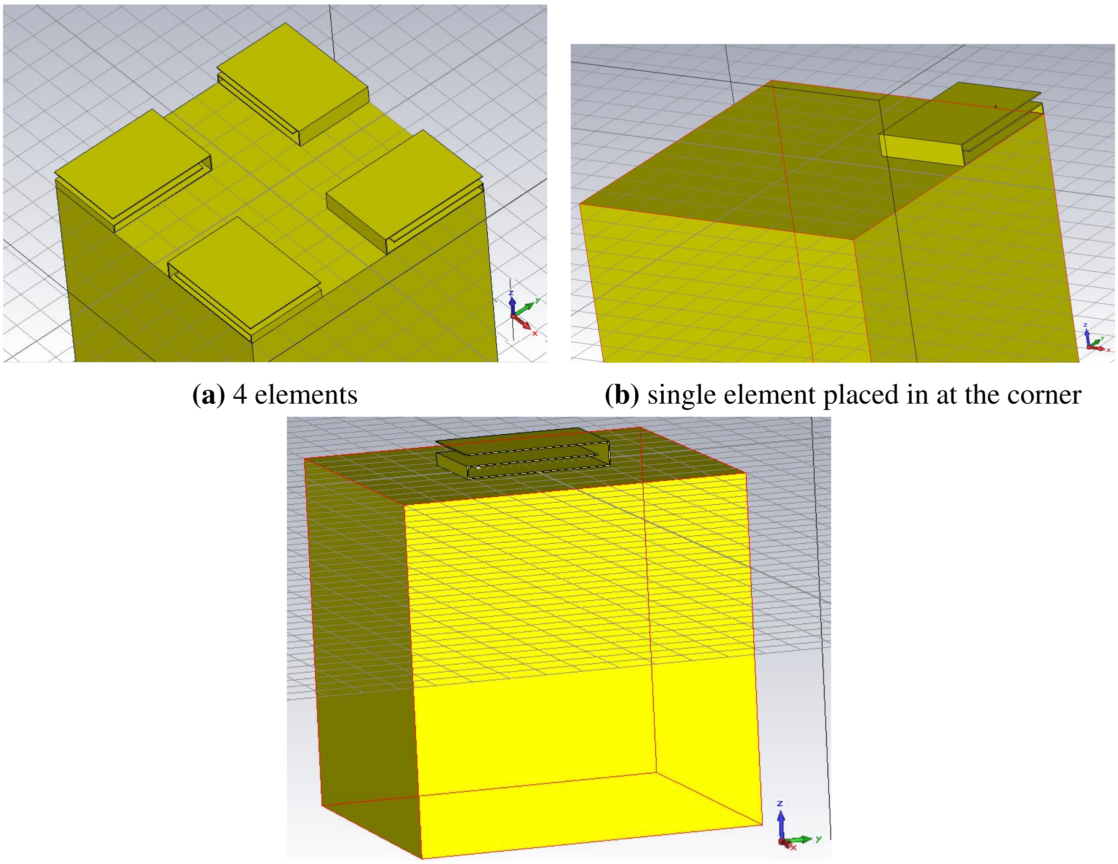
This test will fully utilise the functionality of the software, including; importing radiation patterns, rotation about three axes and translation and combination of the fields.

5.3.1 CST

Five simulations were done in CST:

1. One antenna was excited and the antenna was aligned with and centred on the Z-axis (figure 5.2a)

2. The satellite upright and centred on the Z-axis, and one element was exited (figure 5.2b)



(a) 4 elements

(b) single element placed in at the corner

(c) Single centred element

Figure 5.1: Test set-up for 4 PIFA elements on a cubic platform

3. The satellite upright and centred on the Z-axis, and both elements were exited without phase delay
4. The satellite upright and centred on the Z-axis, and both elements were exited with a 90° phase delay
5. With a single element centred on the satellite and the Z-axis (figure 5.2c)

5.3.2 SmallsatArray

In the SatelliteArray software, three of the radiation patterns from the CST simulations were imported and combined into the array; configurations (1) (2) and (5) from the list in the previous subsection. All these were simulated at 0° and 90° phase delay between the elements.

For configuration (2), only a rotation around the Z-axis was used to place the second element in the array because, as discussed previously, the phase resulting from the position of the element is already present in the exported radiation pattern.

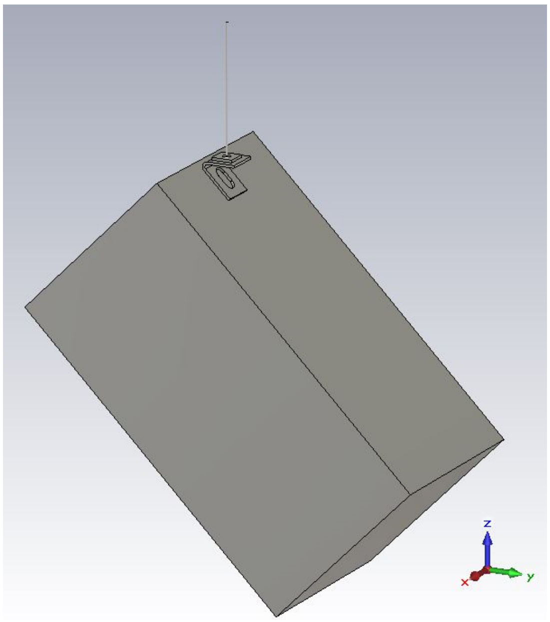
For configurations (1) and (6), the elements were rotated by $\alpha = \pm 90°, \beta = -45°, \gamma = 90°$ and translated to $\pm 50\,\text{mm}$. The base of the antenna was placed at the origin in the simulation, and this remains true after the rotation in SmallsatArray. Therefore the translation is with reference to the base of the antenna and not the centre.

5.3.3 Physical Experiment

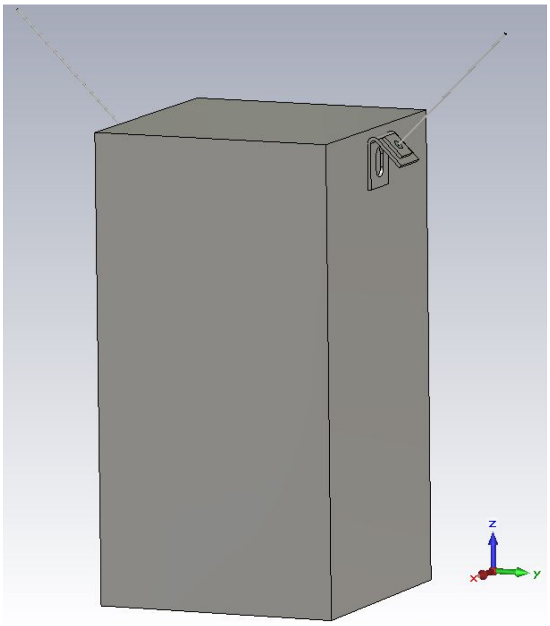
The physical experiment was conducted in the antenna laboratory at NTNU Trondheim. The anechoic chamber measures $10\,\text{m} \times 6\,\text{m} \times 4\,\text{m}$, and is reflection-free for frequencies above $\sim$1 GHz. The satellite model was operating at a frequency of 956 MHz. Three planes of the field were evaluated; $\theta = 90°, \phi = 0°$ and$\phi = 90°$ (XY, XZ and YZ). The satellite position on the rotating platform can be seen in figure 5.3. The transmitting antenna, figure 5.4, was rotated to receive either the $\hat{\theta}$- or $\hat{\phi}$-component of the field.

To get the desired phase delay, different combinations of cables were used. It was not possible to achieve the exact phase delay that were wanted. The phase delays between the cables were measured to be:

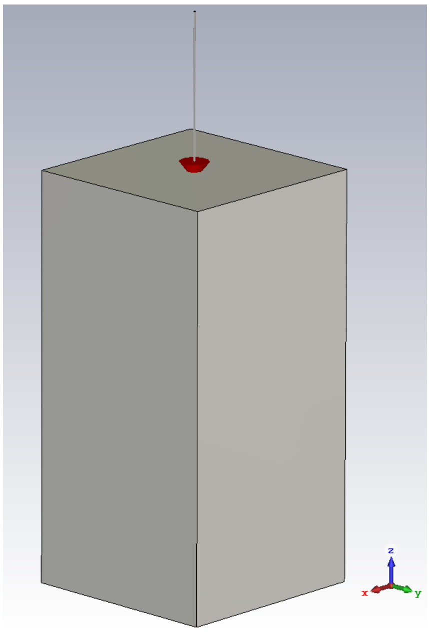
Goal		**Realised Phase**
$\beta =$	0°	14°
$\beta =$	90°	89°



(a) Antenna along Z-axis

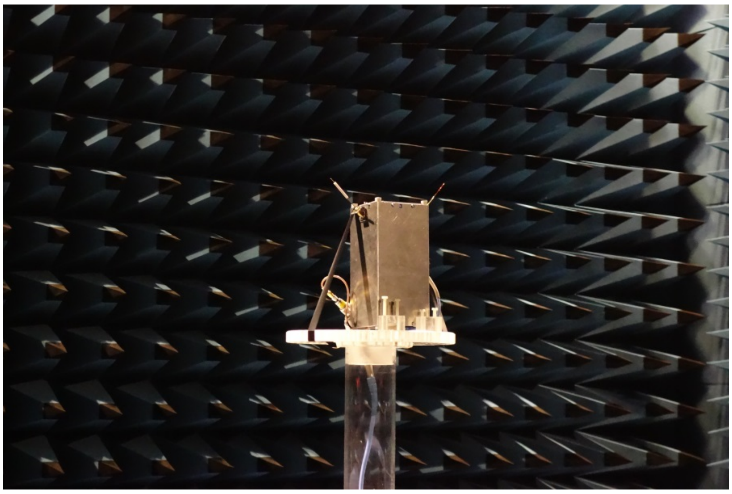


(b) Satellite placed flat

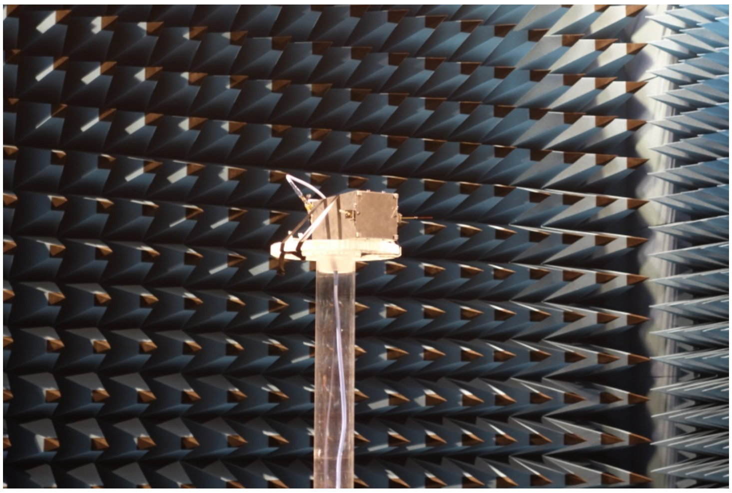


(c) Centred antenna element

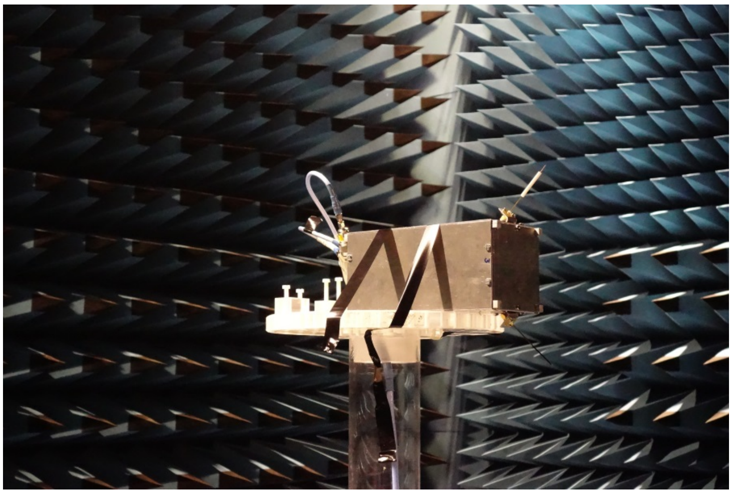
Figure 5.2: CST model of a 2U CubeSat with monopole antennas



(a) Position 1, XY-plane, $\theta = 90°$



(b) Position 2, YZ-plane, $\phi = 90°$



(c) Position 3, XZ-plane, $\phi = 0°$

Figure 5.3: The three satellite orientations evaluated in the physical experiment for the CubeSat model

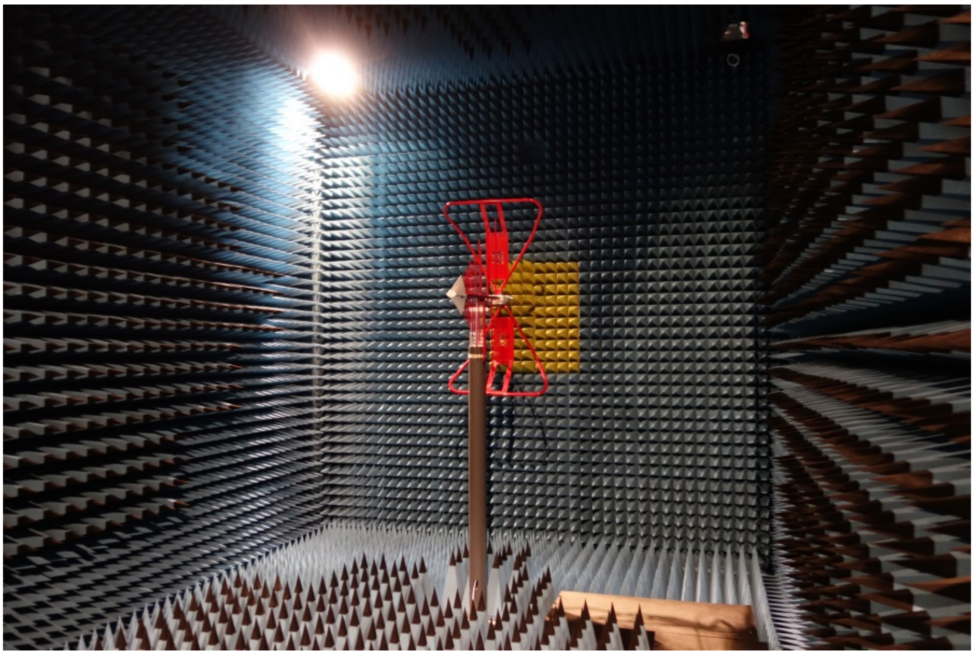


Figure 5.4: The receiving antenna in the anechoic chamber oriented to receive the vertical field component

Chapter 6

Results

This chapter gives the results from the three experiments presented in chapter 5.

6.1 Importing Radiation Patterns From CST

Figure 6.1 shows a comparison between a screen-shot of CST and the imported fields in the SmallsatArray software. While the plot from SmallsatArray appears to only have one plot, it does show 14 plots on top of each other that are identical. They are also identical to the plots from CST.

Figure 6.2 compares the 3D-plot of the axial ratio from CST and SmallsatArray. In this case the difference between CST and SmallsatArray are more substantial. Although the results from SmallsatArray cannot be trusted entirely, it can be used to give an indication that there is circular polarisation present.

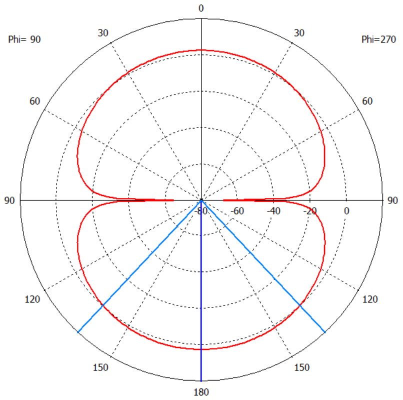
6.2 4 PIFA elements on a Cubic Satellite

In this section, we will compare the results from

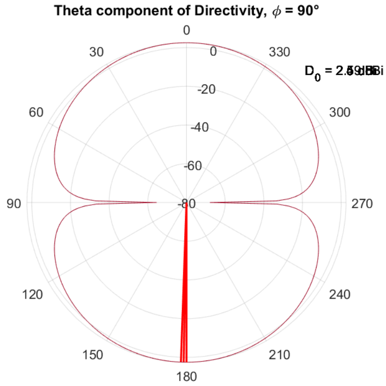
1. The control test of the four-element array simulated in CST (figure 5.1a)
2. The four-element array calculated in SmallsatArray by combining the imported simulation result of the single element placed at the corner of the platform (figure 5.1b)
3. The four-element array calculated in SmallsatArray by combining the imported simulation result of the single element placed in the centre of the satellite platform (figure 5.1c)

A 3D-plot of the radiation pattern from the array can be seen in figure 6.3. It is evident from the plot that the radiation pattern is quite symmetrical about the XZ- and YZ-plane. We will therefore only analyse the radiation pattern in the XZ-plane, where $\phi = 0$.

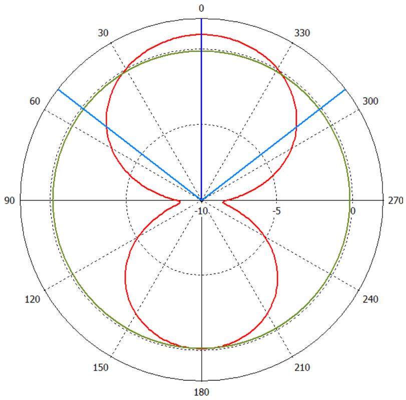
Figure 6.4 shows the results for the $\hat{\theta}$-component of the field. The results from the corner-placed element (3.) appears to be nearly identical to the CST-control, while the results found



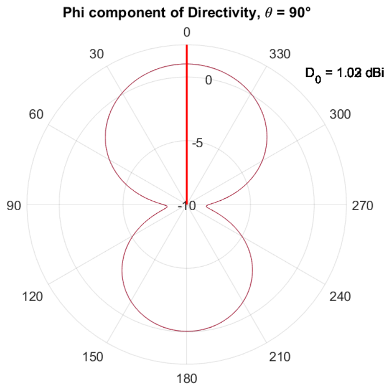
(a) CST, θ-component, $\phi = 90°$



(b) AISSim, θ-component, $\phi = 90°$

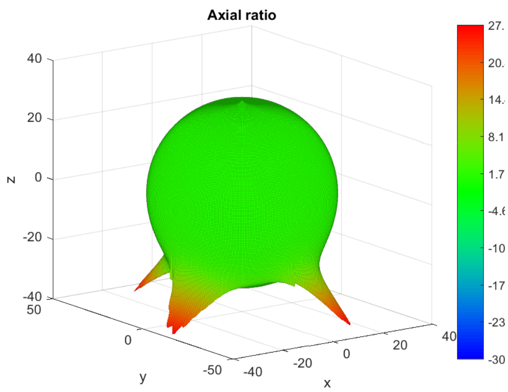


(c) CST, ϕ-component, $\theta = 90°$

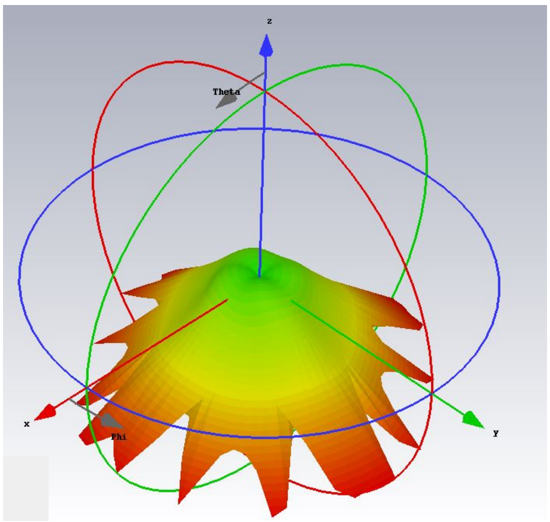


(d) AISSim, ϕ-component, $\theta = 90°$

Figure 6.1: Results of importing radiation patterns in various formats



(a) CST (SmallsatArray)



(b) CST

Figure 6.2: Comparing 3D-plots of axial ratio from CST and SmallsatArray

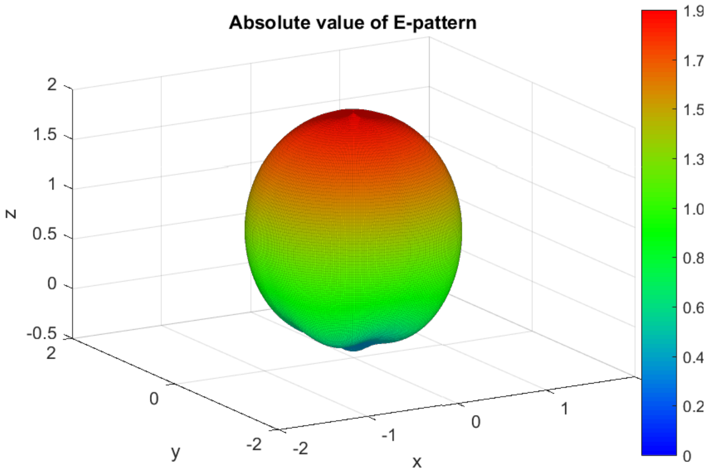


Figure 6.3: 3D-plot of the RMS-normalised E-pattern from the 4 PIFA element array

using the centred element has a larger back-lobe than the CST-control.

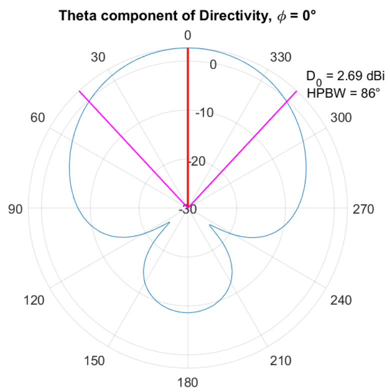
Figure 6.5 shows the $\hat{\phi}$-component in the same plane. Again, the results found using the corner-placed element are very similar to the CST-control, while the results found using the centred element show a greater radiation in the back.

6.3 2 Angled Monopole Elements on a 2U CubeSat

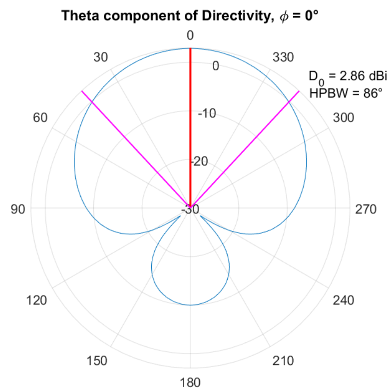
In this section we compare

1. Simulation of the array done in CST
2. Results from the physical measurements
3. The field calculated in SmallsatArray from the CST-simulation where the antenna was aligned with the Z-axis (figure 5.2a)
4. The field calculated in SmallsatArray from the CST-simulation where the satellite was placed upright (figure 5.2b)
5. The field calculated in SmallsatArray from the CST-simulation where antenna was centred on the platform (figure 5.2c)

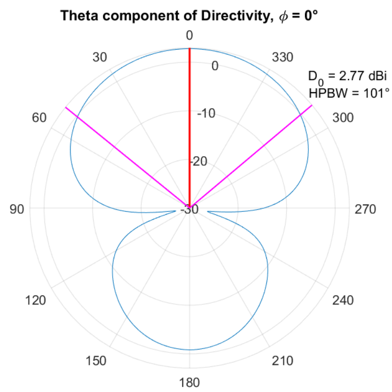
Figures 6.6, 6.7 and 6.8 shows the results from having the elements in-phase for the three evaluated planes; XY ($\theta = 90°$), XZ ($\phi = 0°$) and YZ ($\phi = 90°$), respectively. The results from CST and SamallsatArray (4) appear to be identical. The results from the physical measurements also lie close to these, however the radiation pattern has been shifted slightly because of the 14° phase delay between the elements. We can also see that the calculations done in SmallsatArray using the Z-aligned antenna simulation (3), are still very close to the formerly mentioned plots. Finally the configuration where the antenna was centred on the satellite platform (5) has



(a) CST (1.)

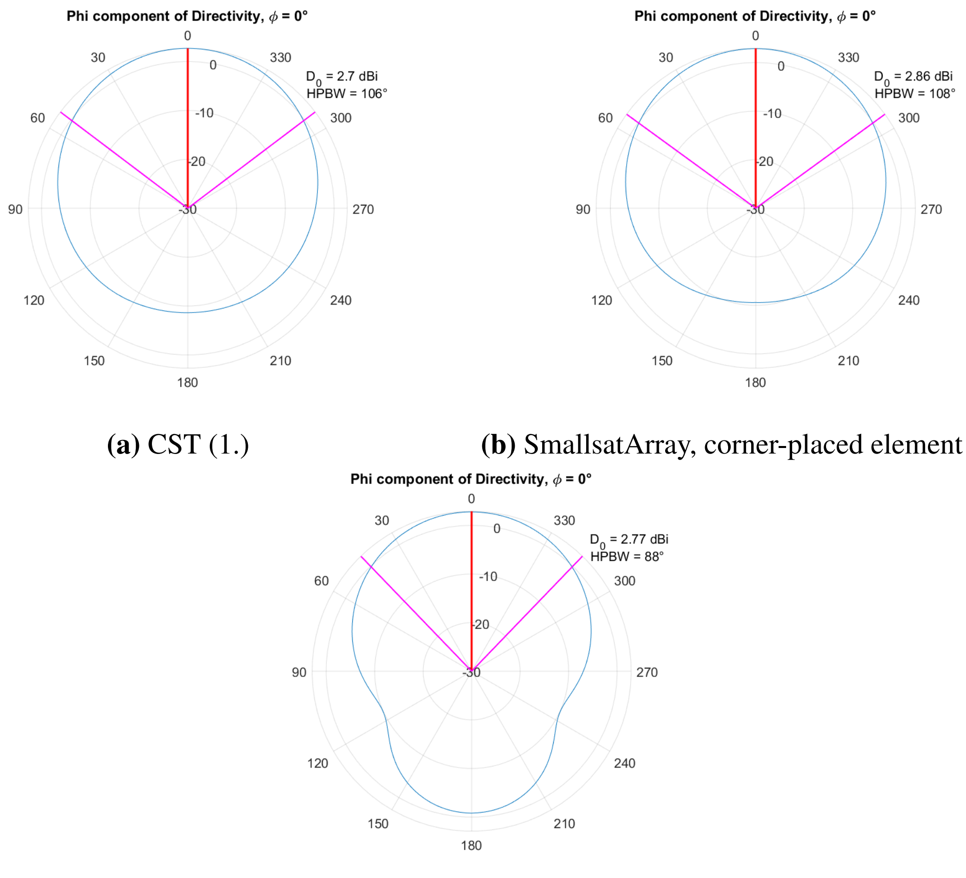


(b) SmallsatArray, corner-placed element (2.)



(c) SmllsatArray, centred element (3.)

Figure 6.4: Directivity (dB) of $\hat{\theta}$-component from four PIFA elements on a cubic satellite, $\phi = 0°$



(a) CST (1.)

(b) SmallsatArray, corner-placed element (2.)

(c) SmllsatArray, centred element (3.)

Figure 6.5: Directivity (dB) of $\hat{\phi}$-component from four PIFA elements on a cubic satellite, $\phi = 0°$

been affected by the different placement on the satellite body, however it does provide a good estimation of the field.

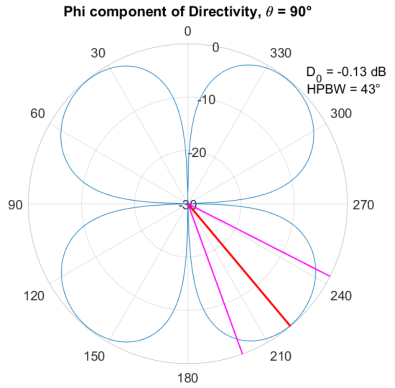
The test with a phase delay of 90° are shown in figures 6.9, 6.10 and 6.11, for the tree different evaluated planes. Again, the two simulations done in SmallsatArray using plots from simulations where the element was placed on the edge of the platform ((3) and (4)) are very close to the simulation done in CST. The results from the physical measurements are also very close to these results except for a few abnormalities, most notably the plot in figure 6.9a. The calculations done using the element centred on the platform differs the most from the other plots, but the essential features of the radiation patten are visible.

6.3.1 Computational Time

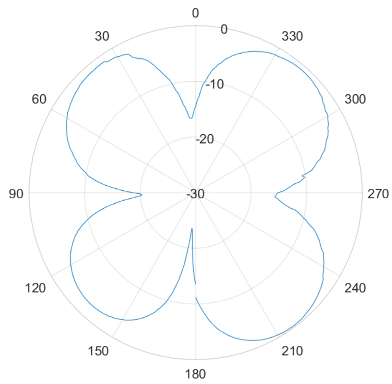
It is, of course, important to take into account the time it took to preform the analysis using the different methods. Table 6.1 gives on overview of the approximate time is took to prepare and conduct the calculations or experiment. The preparation time for SmallsatArray does not include the set-up and simulation of an antenna in CST, which is needed for calculations of antennas other than dipole elements. However, it is rare to get the array right on first attempt, and it is here that the fast computations of the SmallsatArray is a big advantage.

Method	**Experiment Preparation**		**Computation time**	
SmallsatArray	3	minutes	<0.5	seconds
CST	30	minutes	3	minutes
Physical experiment	1	week	1	workday

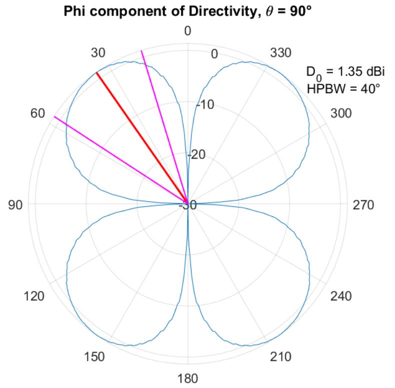
Table 6.1: Comparison of preparation time and computational time of the methods of analysis for the test described in section 5.3 and analysed in section 6.3



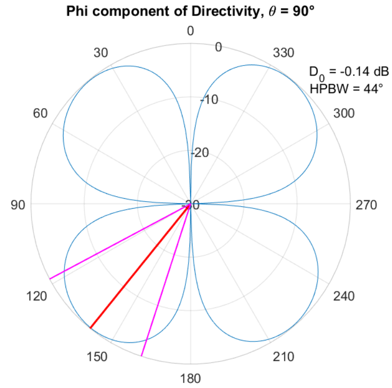
(a) CST simulation (1)



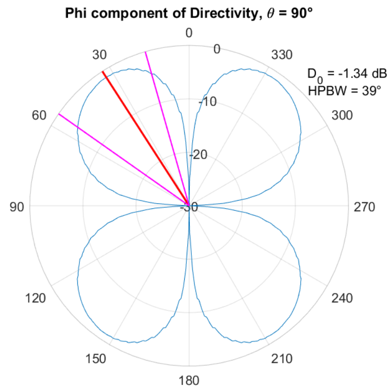
(b) Physical experiment (2)



(c) SmallsatArray using simulation of Z-aligned antenna (3)

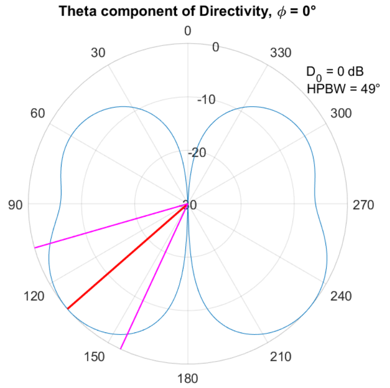


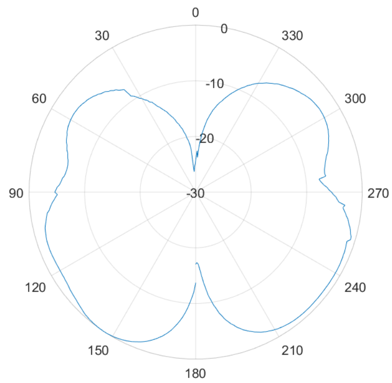
(d) SmallsatArray using simulation of upright satellite (4)



(e) SmallsatArray using simulation of centred element (5)

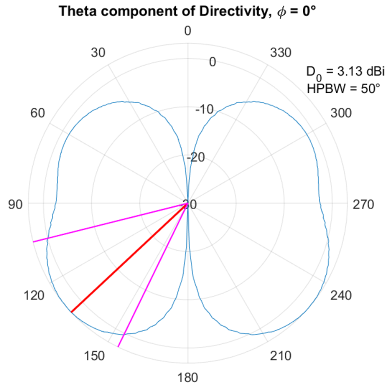
Figure 6.6: CubeSat measurements. Normalised directivity for $\hat{\phi}$-component at $\theta = 90°, \beta = 0°$

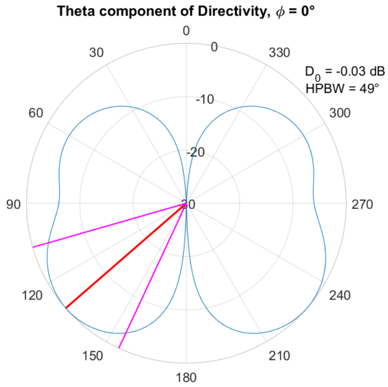




(a) CST simulation (1)

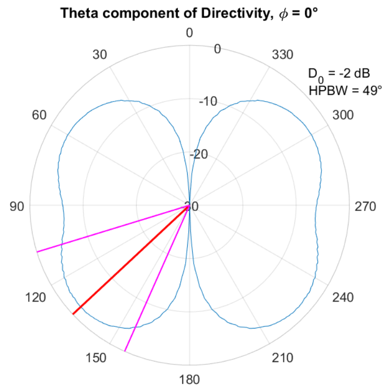
(b) Physical experiment (2)





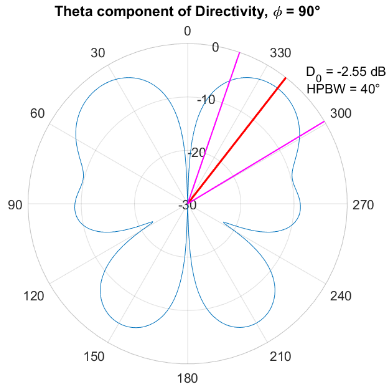
(c) SmallsatArray using simulation of Z-aligned antenna (3)

(d) SmallsatArray using simulation of upright satellite (4)

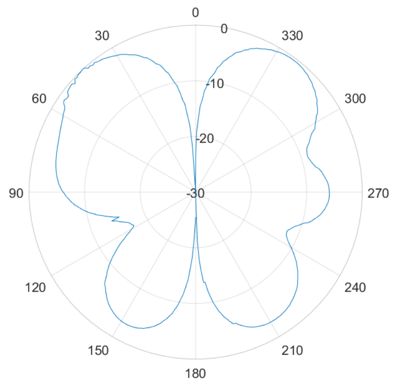


(e) SmallsatArray using simulation of centred element (5)

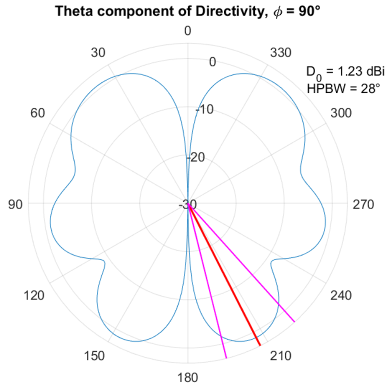
Figure 6.7: CubeSat measurements. Normalised directivity for $\hat{\theta}$-component at $\phi = 0°, \beta = 0°$



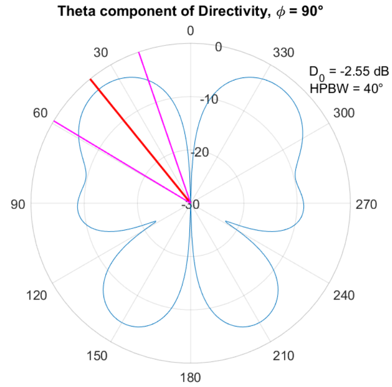
(a) CST simulation (1)



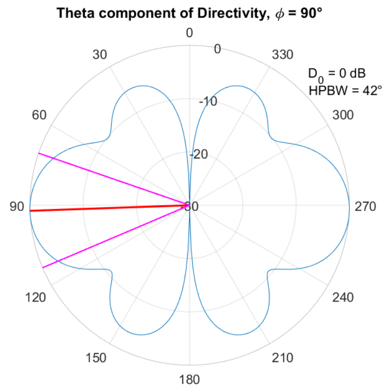
(b) Physical experiment (2)



(c) SmallsatArray using simulation of Z-aligned antenna (3)

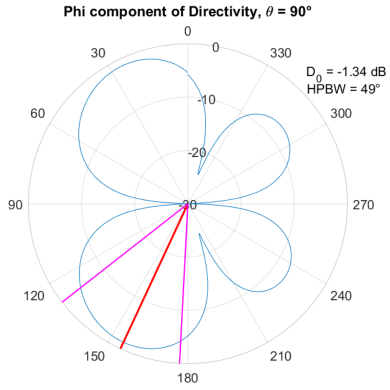


(d) SmallsatArray using simulation of upright satellite (4)

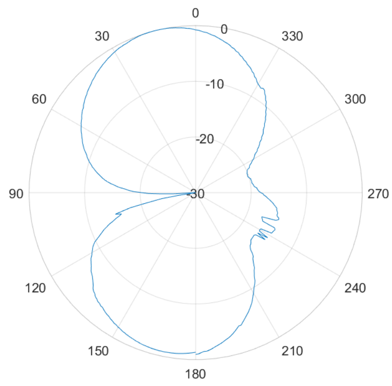


(e) SmallsatArray using simulation of centred element (5)

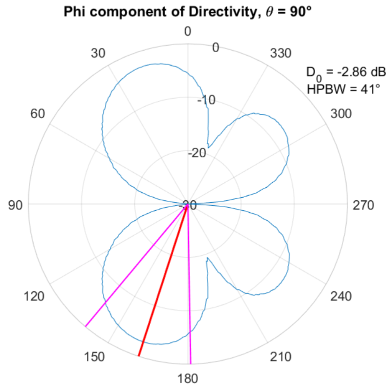
Figure 6.8: CubeSat measurements. Normalised directivity for $\hat{\theta}$-component at $\phi = 90°, \beta = 0°$



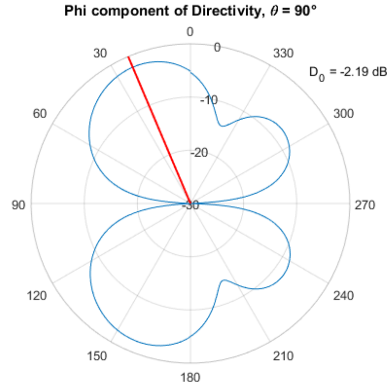
(a) CST simulation (1)



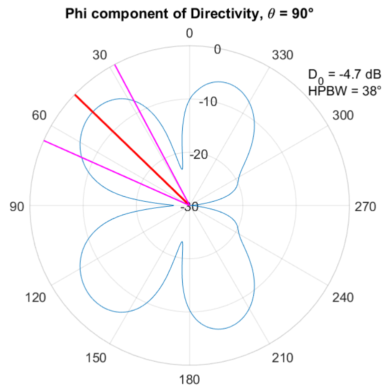
(b) Physical experiment (2)



(c) SmallsatArray using simulation of Z-aligned antenna (3)

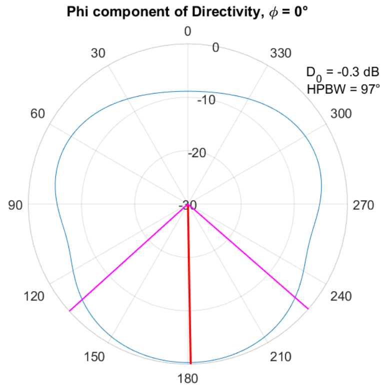


(d) SmallsatArray using simulation of upright satellite (4)

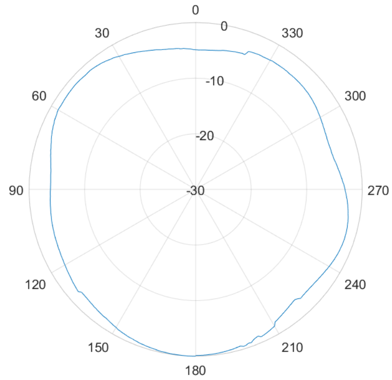


(e) SmallsatArray using simulation of centred element (5)

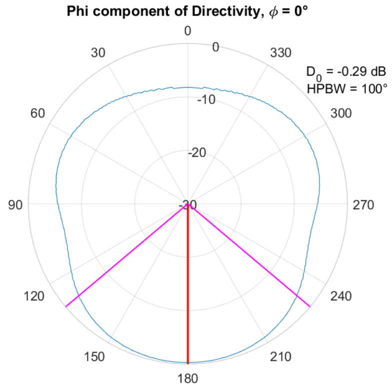
Figure 6.9: CubeSat measurements. Normalised directivity for $\hat{\phi}$-component at $\theta = 90°, \beta = 90°$



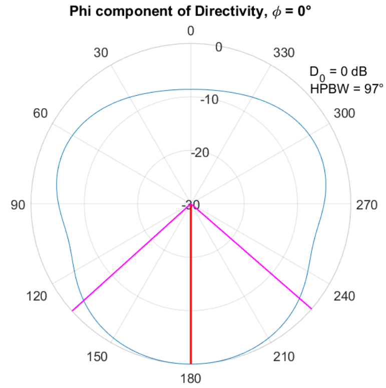
(a) CST simulation (1)



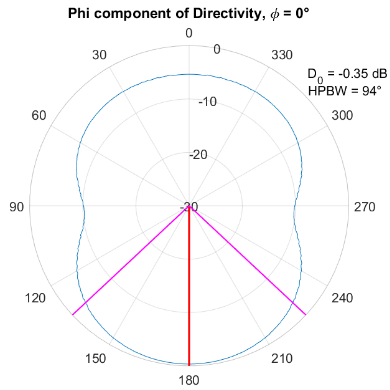
(b) Physical experiment (2)



(c) SmallsatArray using simulation of Z-aligned antenna (3)

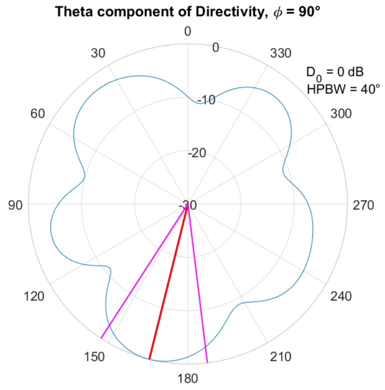


(d) SmallsatArray using simulation of upright satellite (4)

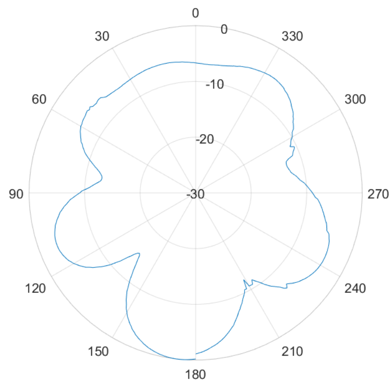


(e) SmallsatArray using simulation of centred element (5)

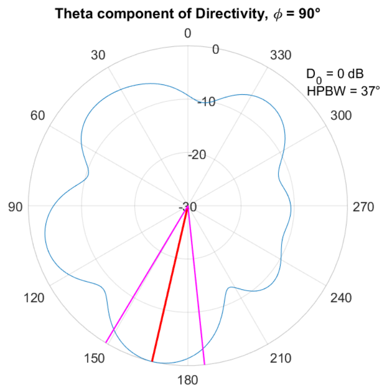
Figure 6.10: CubeSat measurements. Normalised directivity for $\hat{\phi}$-component at $\phi = 0°, \beta = 90°$



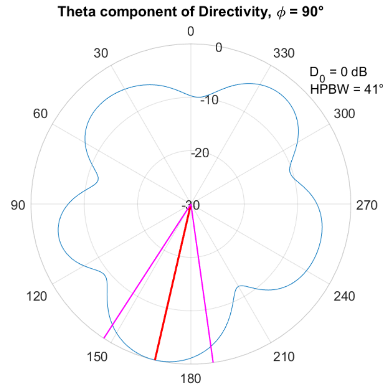
(a) CST simulation (1)



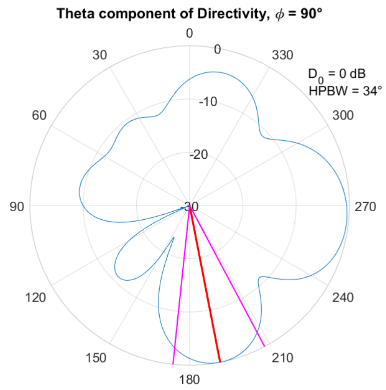
(b) Physical experiment (2)



(c) SatelliteArray (1)



(d) SmallsatArray using simulation of upright satellite (4)



(e) SmallsatArray using simulation of centred element (5)

Figure 6.11: CubeSat measurements. Normalised directivity for θ-component at $\phi = 90°, \beta = 90°$

Chapter 7

Conclusion

The SmallsatArray software was developed on the assumption that the total field of an antenna array, on a satellite platform with electrically small dimensions, could be calculated accurately using only the contribution from the direct radiation from the antenna elements, neglecting the contributions from coupling between the elements, and the scattering and diffraction of the field by the satellite platform.

The simulations in SmallsatArray where the imported radiation patterns came from simulations where the antenna elements' placement on the satellite platform had the same symmetry to the platform as in their placement in the array, the results were highly accurate. From this we can conclude that the coupling between the elements did not make a significant contribution to the radiation pattern and can rightfully be neglected. These results also verify the methods for rotation of the radiation pattern and the calculation of the array factor with respect to the antenna elements position in the array.

In the simulations where the imported radiation pattern came from a simulation where the antenna element had a different symmetry in its placement on the satellite platform than in the simulated array, the results show that the radiation patterns were affected by the scattering and diffraction caused by the satellite platforms. Though the simulations from SmallsatArray still provided decent accuracy. The test-cases evaluated in this thesis had satellite platform that was only slightly smaller than the wavelength, so the calculations are expected to be even more accurate when the electrical size of the satellite is decreased.

In summary, the SmallsatArray software provides very fast simulation of arrays with respectable accuracy. It offers a wide range of plot options and has many useful features to make the program user friendly and responsive. Personally, I am very satisfied with the user interface of the program and its ease-of-use, but I have spent hundreds of hours with the program, so I can only hope others will find it intuitive.

7.1 Future Work

A problem with the current state of the SmallsatArray software is that it is difficult for the user to visualize the elements in the antenna array. It would be very helpful to have a visual representation of the array and its elements. A possible solution is to have a separate figure where the antenna elements are drawn in, however having two figures that needs to be accessed by the scrip would, from my experience, add a lot of complexity to the code.

Another place where the program is lacking is in visualising the polarisation. A simple plot of the axial ratio exits in the program, but as seen in the results in figure 6.2, this is not accurate enough. The implementation of the axial ratio is probably incorrect as it does not consider the phase between the field components, only their magnitude. There are also other ways to plot the polarisation in CST such as different options for Ludwig 2AE and Ludwig 3 that could be implemented in this program.

It would also be very useful to relate the radiation patterns to the satellite in orbit. This could be accomplished by plotting the contour of the radiation pattern for different power levels on a globe or map of the earth, given the height and position in the orbit. The ship AIS antenna is vertically polarized, so it would be a very useful feature to see polarisation loss factor also included in the view.

To further test current state of the program, I would like to see a test of the program versus an array where the coupling between the antenna elements is strong, possibly strong enough to invalidate the results from the SmallsatArray software. I am also interested in seeing simulations in the program with satellites where the wavelength is even greater compared to the satellite than the cases tested in this thesis. I predict that when the relative wavelength is larger, the results will be even more accurate when moving the antennas around on the platform.

Having a larger selection of elements built-in to the program would also be a useful addition. Common element types for satellites such as monopoles and patch antennas could possibly be calculated using analytical formulas or some default radiation patterns from CST simulations can be stored within the program somehow.

The function for calculating the total field was, for a long time during the thesis work, the slowest calculation in the program. This was an issue because it had to be run every time an element was added, and took longer and longer the more elements were added. By changing the function from calculating single cells in the matrices at a time to simply adding or multiplying entire matrices together after setting up matrices handling the transformation of each cell, the computational time was reduced to almost nothing. The slowest calculation currently in the program is the function for rotating the field. It is not a big issue because it takes less than a second and the computational time remains constant for any number of elements in the array, but if this function can be made with matrix multiplication and -addition instead of evaluating each cell individually, the program can be made even more responsive.

EM-simulation software such as CST offers optimization tools find the best antenna parameters for the desired radiation characteristics. Because of the extremely fast calculations in the SmallsatArray software compared to CST, there is a potential for preforming very fast optimization in this software. This perhaps not so useful for fine-tuning the array, as this should be done in EM-simulation tools for serious design considerations, but could be used to discover array configurations that are unlikely to be found through an iterative or intuitive design approach.

Bibliography

[1] C. A. Balanis, *Antenna Theory, Analysis and Design*, Fourth. Noboken, NJ, US: John Wiley & Sons, 2016.

[2] E. Birkeland, "Forprosjekt: Antenner til små satellitter", NTNU, Tech. Rep., 2016.

[3] T. I. of Electrical and I. Electronics Engineers, *IEEE Standard Definitions of Terms for Antennas*, 345 East 47th Street, New York, NY 10017, 1983.

[4] ESA. (). Aissat-1 and 2. Accessed: 10. June 2017, [Online]. Available: `https://directory.eoportal.org/web/eoportal/satellite-missions/a/aissat-1-2`.

[5] ——, (). Norsat-1. Accessed: 10. June 2017, [Online]. Available: `https://directory.eoportal.org/web/eoportal/satellite-missions/n/norsat-1`.

[6] *Nano/Microsatellite Market Forecast*, SpaceWorks Enterprises, Atlanta, USA, 2017.

[7] NSC. (). Norske satellitter. Accessed: 10. June 2017, [Online]. Available: `www.romsenter.no/Bruk-av-rommet/Norske-satellitter`.

[8] G. Roseti, "Numerical analisys and design of antenna systems for micro/nano satellites", PhD thesis, EPFL, 2013.

[9] M. W. Spong, S. Hutchinson, and M. Vidyasagar, *Robot Modeling and Control*, First. New York, NY, US: John Wiley & Sons, 2006.

Appendix A

Mathematical Proofs

A.1 Tangential Unit Vectors for Spherical Coordinates

The tangential unit vectors described in section 3.1 needs to satisfy the following conditions:

1. They must both have a length of unity
2. They must both be normal to the vector they represent
3. $\hat{\phi}$ must be only in the xy-plane
4. They must be normal to each other

(1) is evident from the spherical form of the vectors where $r = 1$. (2) is prooven in (A.1a) and (A.1b). (3) is seen in (A.1b) where the z-component is zero. (4) is prooven in (A.1c)

$$\begin{aligned}\vec{v}\cdot\hat{\theta} &= [\theta_v,\phi_v,r]^{spherical}\cdot[\pi/2,\phi_v,1]^{spherical}\\ &= [r\sin\theta_v\cos\phi_v, r\sin\theta_v\sin\phi_v, r\cos\theta_v]\\ &\cdot[\sin(\theta_v+\pi/2)\cos\phi_v,\sin(\theta_v+\pi/2)\sin\phi_v,\cos(\theta_v+\pi/2)]\\ &= [r\sin\theta_v\cos\phi_v, r\sin\theta_v\sin\phi_v, r\cos\theta_v]\cdot[\cos\theta_v\cos\phi_v,\cos\theta_v\sin\phi_v,-sin\theta_v]\\ &= r(\sin\theta\cos\theta\cos^2\phi+\sin\theta\cos\theta\sin^2\phi-\sin\theta\cos\theta)\\ &= r\sin\theta\cos\theta(\sin^2\phi+\cos^2\phi-1)=0\end{aligned} \tag{A.1a}$$

$$\begin{aligned}\vec{v}\cdot\hat{\phi} &= [r\sin\theta\cos\phi, r\sin\theta\sin\phi, r\cos\theta]\cdot[\sin\pi/2\cos(\phi+\pi/2),\sin\pi/2\sin(\phi+\pi/2),\cos\pi/2]\\ &= r[\sin\theta\cos\phi,\sin\theta\sin\phi,\cos\theta]\cdot[-\sin\phi,\cos\phi,0]\\ &= r(-\sin\theta\sin\phi\cos\phi+\sin\theta\sin\phi\cos\phi+0)=0\end{aligned} \tag{A.1b}$$

$$\begin{aligned}\hat{\theta}\cdot\hat{\phi} = & \quad [\sin(\theta+\pi/2)\cos\phi,\sin(\theta+\pi/2)\sin\phi,\cos(\theta+\pi/2)]\\ & \cdot[\sin\pi/2\cos(\phi+\pi/2),\sin\pi/2\sin(\phi+\pi/2),\cos\pi/2]\\ = & \quad [\cos\theta\cos\phi,\cos\theta\sin\phi,-sin\theta]\cdot[-\sin\phi,\cos\phi,0]\\ = & \quad -\cos\theta\sin\phi\cos\phi+\cos\theta\sin\phi\cos\phi+0=0\end{aligned} \tag{A.1c}$$

Appendix B

Source Code

B.1 Importing Radiation Patterns

Listing B.1: *ImportElement* function and nested functions

```
%-------------------Import Element-----------------------------------------
function ImportElement(handles)
UpdateMsgBox(handles,'Importing element data...');

% Fetching element number
n = round(str2num(handles.n.String));

global Element thetar phir

% Prompting format selection
format          = questdlg('Select input format',...
    'Select input format','CST','HFSS','CST');

switch format
    case 'CST'
        [cstfile,directory]     = uigetfile('*.txt','Select a file to import');

        % Checking valid input
        if cstfile == 0
            UpdateMsgBox(handles,'Importing element data failed.');
            return;
        end

        % Clearing variable
        if not(size(Element,2)>n)
            Element(n).E_nonrot=[];
        end

        Element(n).Tag = cstfile;


        %Assuming input format:
        %Theta|Phi|Abs(tot)|Abs(theta)|Phase(theta)|Abs(phi)|Phase(phi)|Ax.Ratio

        A = cstread(strcat(directory,cstfile));
        H = cstheadread(strcat(directory,cstfile));

        %Checking angle units units
        if strcmp(H{2},'[deg.]')
            angleunit = 'deg';
        else
            error('elmTyp_Callback: Unknown angle unit');
```

```
        end

        % Calculating resolution
        ImportedEF.res = A(2,1)-A(1,1);

        % Reading data and storing in desired data structures and
        % units
        for i=1:size(A,1)
            t_idx = A(i,1)/ImportedEF.res+1;
            p_idx = A(i,2)/ImportedEF.res+1;
            ImportedEF.theta(p_idx,t_idx)   = A(i,4);
            ImportedEF.phi(p_idx,t_idx)     = A(i,6);
            %ImportedEF.tot(p_idx,t_idx)=A(i,3);
            ImportedEF.t_phase(p_idx,t_idx) = A(i,5);
            ImportedEF.p_phase(p_idx,t_idx) = A(i,7);
        end

        % Checking scale
        if any(strfind(H{6},'dB'))
            ImportedEF.theta    = dB2lin(ImportedEF.theta);
            ImportedEF.phi      = dB2lin(ImportedEF.phi);
        end

        % Checking unit for power
        if any(strcmp(H{5},{'Abs(Grlz)','Abs(Dir.)','Abs(Gain)','Abs(P)'})) || any(strcmp(H{5},{'Abs(E)','Abs(V)','Abs(H)'})) && any(strfind(H{6},'dB'))
            ImportedEF.theta    = sqrt(ImportedEF.theta);
            ImportedEF.phi      = sqrt(ImportedEF.phi);
        end

        if any(strcmp(H{5},{'Abs(H)'}))
            thetamag   = ImportedEF.theta;
            thetaphase = ImportedEF.t_phase;

            ImportedEF.theta    = ImportedEF.phi;
            ImportedEF.t_phase = ImportedEF.p_phase;

            ImportedEF.phi      = thetamag;
            ImportedEF.p_phase = thetaphase;
        end


    case 'HFSS'
        [FileName,PathName,FilterIndex] = uigetfile('*.csv','Select a file to import');
        A = csvread(strcat(PathName,FileName),1,0);

        Element(n).Tag = FileName;

        % Calculating resolution
        ImportedEF.res = abs(A(1,1)-A(2,1));

        % Reading data and storing in desired data structures and
        % units
        for i=1:size(A,1)
            [i_p, i_t]=r2t(d2r(A(i,2)), d2r(A(i,1)), ImportedEF.res);

            ImportedEF.theta(i_p,i_t)=dB2lin(A(i,3));
            ImportedEF.phi(i_p,i_t)=dB2lin(A(i,4));
            ImportedEF.tot(i_p,i_t)=...
                sqrt(ImportedEF.theta(i_p,i_t)^2+...
                ImportedEF.phi(i_p,i_t)^2);
        end
        ImportedEF.t_phase=zeros(size(ImportedEF.theta));
        ImportedEF.p_phase=zeros(size(ImportedEF.phi));

    otherwise
```

```
        return;
end


Element(n).E_nonrot.Theta =...
    Interpol2(ImportedEF.theta,ImportedEF.t_phase,thetar,phir);
Element(n).E_nonrot.Phi   =...
    Interpol2(ImportedEF.phi,ImportedEF.p_phase,thetar,phir);

Element(n).Type = 'Imported';

Element(n).CurrentRotation = [0,0,0];


UpdateMsgBox(handles,'Importing element data... Done!');


%------------------Reading text files exported by CST--------------------
function [A] = cstread(filename)
% Import data from text file.
% Script for importing data from the following text file:
%
%    C:\Users\Ola\git\master-thesis\NorSat-3_FF.txt
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.

% Auto-generated by MATLAB on 2017/03/28 15:35:09

% Initialize variables.
%filename = 'C:\Users\Ola\git\master-thesis\NorSat-3_FF.txt';
%filename = uigetfile('*.txt','Select a file to import');
startRow = 3;

% Format for each line of text:
%   column1: double (%f)
%   column2: double (%f)
%   column3: double (%f)
%   column4: double (%f)
%   column5: double (%f)
%   column6: double (%f)
%   column7: double (%f)
%   column8: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%8f%16f%21f%20f%20f%20f%20f%20f%[^\n\r]';

% Open the text file.
fileID = fopen(filename,'r');

% Read columns of data according to the format.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', '', 'WhiteSpace', '', '
    EmptyValue' ,NaN,'HeaderLines' ,startRow-1, 'ReturnOnError', false, '
    EndOfLine', '\r\n');

% Close the text file.
fclose(fileID);

% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

% Create output variable
A = [dataArray{1:end-1}];
```

```
% Clear temporary variables
clearvars filename startRow formatSpec fileID dataArray ans;


%-------------------Read the header from the CST-file----------------------
function [H] = cstheadread(filename)

fileID = fopen(filename,'r');
s     = fscanf(fileID,'%c');
fclose(fileID);

h{1} = '';
i     = 1;
j     = 1;
while s(i) ~= '-'
    if strcmp(s(i),' ') && (not(strcmp(s(i-1),' ')) || not(strcmp(s(i-1),'')))
        j = j + 1;
        h{j} = '';
    elseif s(i) == ']' || s(i) == ')'
        h{j} = strcat(h{j},s(i));
        j = j + 1;
        h{j} = '';
    elseif strcmp(s(i),'[')
        j = j + 1;
        h{j} = s(i);
    else
        h{j} = strcat(h{j},s(i));
    end

    i = i + 1;
end

j = 1;
%H = cell(1);
for i=1:length(h)
    if strcmp(h{i},']') || strcmp(h{i},')')
        H{j-1} = strcat(H{j-1},h{i});
    elseif strcmp(h{i},'')
        continue;
    else
        H{j} = h{i};
        j = j + 1;
    end
end


%-------------------Interpolate complex field-----------------------------
function Field = Interpol2(V,W,Xq,Yq)
% Interpolates field and phase values and constructs complex matrix
% V is a matrix containing field values
% W is a matrix containing phase values
% Xq is an array of theta meassurement points for the output field
% Yq is an array of phi meassurement points for the output field
x        = 0:pi/(size(V,2)-1):pi;
y        = 0:2*pi/(size(V,1)-1):2*pi;
[X,Y]    = meshgrid(x,y);
[Xq,Yq]  = meshgrid(Xq,Yq);

U = V.*exp(1i.*d2r(W));

Field    = interp2(X,Y,U,Xq,Yq);
```

B.2 Calculating RMS-Normalised E-field

Listing B.2: Source code for normalising fields to average power

```matlab
%-------------------Normalise field to average power-----------------------
function [varargout] = RMSField(varargin)
% RMSFIELD Normalises the field to the average power
%   RMSFIELD(A) normalises E to its average power
%   RMSFIELD(A,B) normalises Et and Ep to the average power of
%   A^2+B^2
%   RMSFIELD(_,'Name',Value)
%
% Calculates the RMS-value of input fild(s)
% If the input has one field, it is normalised to its RMS value
% If the input has two fields, they are normalised to the RMS value of
% their combined value
% Options: dB, boolean          power, boolean

global thetar

narginchk(1,6);

args   = varargin;

%Default values
dB    = false;
power = false;
Z0    = 377;

%Checking dB option
for i=1:length(args)
    if strncmpi('dB',args{i},3)
        if varargin{i+1} == true
            dB = true;
        end
        args{i}      = [];
        args{i+1}    = [];
        args         = args(~cellfun('isempty',args));
        break
    end
end

%Checking power option
for i=1:length(args)
    if strncmpi('Power',args{i},3) || strncmpi('power',args{i},3)
        if varargin{i+1} == true
            power = true;
        end
        args{i}      = [];
        args{i+1}    = [];
        args         = args(~cellfun('isempty',args));
        break
    end
end

nfields = length(args);

%Calculating rms
switch nfields
    case 1
        if dB == true
            U = 10.^(abs(args{1})./10);
        else
            U = args{1};
        end
        if power == false
            U = U.^2;
```

```
        end

    case 2
        if dB == true
            U = 10.^(abs(args{1})./10)+10.^(abs(args{2})./10);
        else
            U{1} = abs(args{1});
            U{2} = abs(args{2});
        end
        if power == false
            U    = U{1}.^2+U{2}.^2;
        else
            U    = U{1}+U{2};
        end

    otherwise
        error('RMSField: number of inputfields must be 1 or 2');
end

% U is now radiation intensity. (Power, linear)
Prad   = 0;
Totang = 0;
%thetar = linspace(0,pi,size(U,2));
for ii=1:size(U,1)
    for i=1:size(U,2)
        Prad = Prad + U(ii,i)*sin(thetar(i))*d2r(1)^2;
        Totang = Totang + sin(thetar(i))*d2r(1)^2;
    end
end

U0 = Prad./(Totang);

if power == false
    U0 = sqrt(U0);
end

%Outputing fields
switch nfields
    case 1
        if dB == true
            varargout{1} = args{1}-10.*log10(U0);
        else
            varargout{1} = args{1}./U0;
        end
    case 2
        if dB == true
            varargout{1} = args{1}-10.*log10(U0);
            varargout{2} = args{2}-10.*log10(U0);
        else
            varargout{1} = args{1}./U0;
            varargout{2} = args{2}./U0;
        end
end
```

B.3 Rotating Radiation Pattern

Listing B.3: Source code for the *RoteteElement* function

```
%--------------------Rotates element n--------------------------------------
function RotateElement(handles, n)
global thetar phir res Element

Rotation = Element(n).Rotation-Element(n).CurrentRotation;

if Element(n).Rotation == [0,0,0]
    Element(n).E = Element(n).E_nonrot;
    return;
elseif Rotation == [0,0,0]
    return;
else
    Rotation = Element(n).Rotation;
end

UpdateMsgBox(handles, strcat('Rotating element ',num2str(n),'...'));


% Reverse order rotation without interpolation

% Setting up rotation matrix for reverse rotation
Rr       = rotZ(-Rotation(3))*rotY(-Rotation(2))*...
           rotZ(-Rotation(1));

% Setting up rotation matrix for regular rotation
Rf       = rotZ(Rotation(1))*rotY(Rotation(2))*...
           rotZ(Rotation(3));


for i=1:length(thetar)
    for ii=1:length(phir)
        %Setting up index vector for rotation
        [x,y,z]      = s2c(thetar(i),phir(ii),1);

        %Rotating index vector reversely
        v            = Rr*[x;y;z];

        %Converting index vector to spherical coordinates
        [u(1),u(2),u(3)] = c2s(v(1),v(2),v(3));

        %Converting index vector to matrix indecies
        [i_p,i_t]    = r2t(u(1),u(2),res);

        %Setting up field vector for rotation
        %Theta component spherical vector coverted to Cartesian
        [Et(1),Et(2),Et(3)] = s2c(thetar(i_t)+pi/2,phir(i_p),Element(n).E_nonrot
            .Theta(i_p,i_t));

        %Phi component spherical vector coverted to Cartesian
        [Ep(1),Ep(2),Ep(3)] = s2c(pi/2,phir(i_p)+pi/2,Element(n).E_nonrot.Phi(
            i_p,i_t));

        %Combining field components
        E   = Et+Ep;

        %Rotating field vector
        rE = Rf*[E(1),E(2),E(3)]';

        %Creating reference tangential vectors at target rotation
        [v_ref_t(1),v_ref_t(2),v_ref_t(3)]=s2c(thetar(i)+pi/2,phir(ii),1);
        [v_ref_p(1),v_ref_p(2),v_ref_p(3)]=s2c(pi/2, phir(ii)+pi/2,1);

        %Decomposing rotated field vector using tangential reference vectors
```

```
        Et = dot(rE,v_ref_t);
        Ep = dot(rE,v_ref_p);

        %Removing NaN values
        if isnan(Et)
            Et = 0;
        end
        if isnan(Ep)
            Ep = 0;
        end

        %Writing to element variable
        Element(n).E.Theta(ii,i) = Et;
        Element(n).E.Phi(ii,i)   = Ep;
    end
end

Element(n).CurrentRotation = Element(n).Rotation;

UpdateMsgBox(handles, strcat('Rotating element ',' ',num2str(n),'... Done!'));
```

B.4 Calculating Total Field

Listing B.4: *TotalField* function

```
%-------------------Calculate total field----------------------------------
function TotalField(handles)

UpdateMsgBox(handles, strcat('Calculating total field...'));

global E thetar phir f Element c

%Reading array size
N    = size(Element,2);

%Setting up constants for calculation
k    = 2*pi*f/c;

%Clearing E variable
E.Theta      = zeros(length(phir),length(thetar));
E.Phi        = zeros(length(phir),length(thetar));

for n=1:N
    PHIx     = exp(1i*k*Element(n).Position(1)*cos(phir')*sin(thetar));
    PHIy     = exp(1i*k*Element(n).Position(2)*sin(phir')*sin(thetar));
    PHIz     = exp(1i*k*Element(n).Position(3)*ones(length(phir),1)*cos(thetar));
    PHIsp    = PHIx.*PHIy.*PHIz;

    E.Theta  = E.Theta + Element(n).E.Theta.*PHIsp.*Element(n).Excitation;
    E.Phi    = E.Phi +   Element(n).E.Phi   .*PHIsp.*Element(n).Excitation;
end

UpdateMsgBox(handles,'Normalising field...');
[E.Theta,E.Phi] = RMSField(E.Theta,E.Phi,'dB',false);
UpdateMsgBox(handles,'Normalising field... Done!');
E.Abs = sqrt(abs(E.Theta).^2+abs(E.Phi.^2));


UpdateMsgBox(handles, strcat('Calculating total field... Done!'));
```

B.5 Plotting Field

Listing B.5: Source code for functions related to plotting

```
%-------------------Update plot-----------------------------------------
function UpdatePlot(handles)
UpdateMsgBox(handles,'Updating plot...');

global Element

compare = {Element(:).Compare};

props = whos('compare');

switch props.class
    case 'cell'
        compare = cell2mat(compare);
end

if ~any(compare) || nnz(compare) == 1

    CalcPlotData(handles)

    ScalePlotData(handles);

    PlotData(handles);

else

    s = handles.plotType.String(handles.plotType.Value);
    if strcmp(s,'2D') || strcmp(s,'3D')
        return;
        error('Cannot compare in 2D or 3D plot');
    end

    k = 1;
    for n=1:length(compare)
        if compare(n) && k == 1
            CalcCompPlotData(handles,n)

            ScalePlotData(handles);

            PlotData(handles);

            k = k + 1;

        elseif compare(n)
            CalcCompPlotData(handles,n)

            ScalePlotData(handles);

            hold on
            PlotData(handles);
            hold off

            k = k + 1;
        end
    end
end
UpdateMsgBox(handles,'Updating plot... Done!');


%------------------Calculate plot data------------------------------------
function CalcPlotData(handles)
UpdateMsgBox(handles,'Calculating plot data...');
global E PlotData

```

```
PlotData     = [];
s            = handles.plotValue.String;
k            = handles.plotValue.Value;

switch s{k}
    case 'Abs'
        PlotData = sqrt(abs(E.Theta).^2 + abs(E.Phi).^2);
    case 'Theta'
        PlotData = abs(E.Theta);
    case 'Theta Phase'
        PlotData = angle(E.Theta);
    case 'Phi'
        PlotData = abs(E.Phi);
    case 'Phi Phase'
        PlotData = angle(E.Phi);
    case 'Axial Ratio'
        Em = E.Theta - 1i.*E.Phi;
        Ep = E.Theta + 1i.*E.Phi;

        t  = angle(Em./Ep)./2;

        Ex = E.Theta.*sin(t)-E.Phi.*cos(t);
        Ey = E.Theta.*cos(t)-E.Phi.*sin(t);

        PlotData = abs(Ey./Ex);
    case 'Theta/Phi'
        PlotData = abs(E.Theta)./abs(E.Phi);
    case 'Phi/theta'
        PlotData = abs(E.Phi)./abs(E.Theta);
end

% Field pattern or power pattern
s2  = handles.plotField.String;
k2  = handles.plotField.Value;

if strcmp(s2{k2},'Directivity') && not(strcmp(s{k}, 'Axial Ratio'))
    PlotData = PlotData.^2;
% elseif strcmp(s2{k2},'E-pattern') && not(strcmp(s{k}, 'Axial Ratio'))
%     PlotData = PlotData.*4.0862;
%     if handles.dbCheck.Value == true
%         PlotData = PlotData.^2;
%     end
end
UpdateMsgBox(handles,'Calculating plot data... Done!');

%-------------------Calculates plot data for compare mode------------------
function CalcCompPlotData(handles, n)
UpdateMsgBox(handles,strcat({'Calculating plot data for element '},num2str(n),'
    ...'));
global Element PlotData

PlotData     = [];
s            = handles.plotValue.String;
k            = handles.plotValue.Value;

switch s{k}
    case 'Abs'
        PlotData = sqrt(abs(Element(n).E.Theta).^2 + abs(Element(n).E.Phi).^2);
    case 'Theta'
        PlotData = abs(Element(n).E.Theta);
    case 'Theta Phase'
        PlotData = angle(Element(n).E.Theta);
    case 'Phi'
        PlotData = abs(Element(n).E.Phi);
    case 'Phi Phase'
        PlotData = angle(Element(n).E.Phi);
    case 'Axial Ratio'
        Em = Element(n).E.Theta - 1i.*Element(n).E.Phi;
```

```
        Ep = Element(n).E.Theta + 1i.*Element(n).E.Phi;

        t  = angle(Em./Ep)./2;

        Ex = Element(n).E.Theta.*sin(t)-Element(n).E.Phi.*cos(t);
        Ey = Element(n).E.Theta.*cos(t)-Element(n).E.Phi.*sin(t);

        PlotData = abs(Ey./Ex);
    case 'Theta/Phi'
        PlotData = abs(Element(n).E.Theta)./abs(Element(n).E.Phi);
    case 'Phi/theta'
        PlotData = abs(Element(n).E.Phi)./abs(Element(n).E.Theta);
end

% Field pattern or power pattern
s2  = handles.plotField.String;
k2  = handles.plotField.Value;

if strcmp(s2{k2},'Directivity') && not(strcmp(s{k}, 'Axial Ratio'))
    PlotData = PlotData.^2;
elseif strcmp(s2{k2},'E-pattern') && not(strcmp(s{k}, 'Axial Ratio'))
    PlotData = PlotData.*4.0862;
    if handles.dbCheck.Value == true
        PlotData = PlotData.^2;
    end
end
UpdateMsgBox(handles,strcat({'Calculating plot data for element '},num2str(n),'
    ... Done!'));


%-------------------Scale plot data for plot options-----------------------
function ScalePlotData(handles)
UpdateMsgBox(handles,'Scaling plot data...');
%Manual settings
smoothen.bool = handles.smoothen.Value;
%smoothen.bool = true;
smoothen.n = 10;

global PlotData E plotlims thetar phir

norm     = handles.normCheck.Value;
dB       = handles.dbCheck.Value;

s            = handles.plotValue.String;
k            = handles.plotValue.Value;

switch s{k}
    case {'Abs', 'Theta', 'Phi'}

        switch norm
            case true
                m = max(max(PlotData));

                PlotData = PlotData./m;
            case false

        end

        switch dB
            case true
                PlotData = 10.*log10(PlotData);
            case false

        end

    case 'Axial Ratio'

end
```

```
% Removing NaN, Inf and too small values
for i=1:size(PlotData,2)
    for ii=1:size(PlotData,1)
        switch dB
            case true
                if PlotData(ii,i) > plotlims.dB(2)
                    PlotData(ii,i) = plotlims.dB(2);
                elseif PlotData(ii,i) < plotlims.dB(1) || isnan(PlotData(ii,i))
                    PlotData(ii,i) = plotlims.dB(1);
                end
            case false
                if PlotData(ii,i) > plotlims.lin(2)
                    PlotData(ii,i) = plotlims.lin(2);
                elseif PlotData(ii,i) < plotlims.lin(1) || isnan(PlotData(ii,i))
                    PlotData(ii,i) = plotlims.lin(1);
                end
        end
    end
end

if smoothen.bool == true
    if 1 %Stable
    for i=1:size(PlotData,2)
        PlotData(:,i) = smooth(PlotData(:,i),smoothen.n);
    end

    for i=1:size(PlotData,1)
        PlotData(i,:) = smooth(PlotData(i,:),smoothen.n);
    end

    else %Experimental

    [theta3,phi3] = meshgrid(thetar,phir);

    if dB == true
        rho3 = PlotData - plotlims.dB(1);
    else
        rho3 = PlotData;
    end

    [X,Y,Z] = s2c(theta3,phi3,rho3);

    V(:,:,1) = X;
    V(:,:,2) = Y;
    V(:,:,3) = Z;

    W = smooth3(V);

    %[theta3,phi3,rho3] = c2s(W(:,:,1),W(:,:,2),W(:,:,3));

    PlotData=sqrt(W(:,:,1).^2+W(:,:,2).^2+W(:,:,3).^2);
    end
end
UpdateMsgBox(handles,'Scaling plot data... Done!');


%------------------Plot Data----------------------------------------------
function PlotData(handles)
UpdateMsgBox(handles,'Plotting...');
global PlotData res thetar phir plotlims Element

if isempty(PlotData)
    return;
end

PlotMat = [];

```

```
s    = handles.plotType.String;
k    = handles.plotType.Value;

dB       = handles.dbCheck.Value;
norm     = handles.normCheck.Value;

D0 = max(max(PlotData));
limits(2) = D0;

if dB == true
    limits(1) = plotlims.dB(1);
    if limits(2) < 0
        limits(2) = 0;
    end
else
    limits(1) = plotlims.lin(1);
    if limits(2) < 1
        limits(2) = 1;
    end
end

limits(2) = ceil(limits(2)*100)/100;

if limits(2) > 100
    limits(2) = 100;
end

if dB == true
    i = 2;
    ticks(1) = plotlims.dB(1)+10;
    while ticks(i-1)<limits(2)-10
        ticks(i) = plotlims.dB(1)+10*i;
        i = i + 1;
    end
    ticks(i) = limits(2);
else
    i = 2;
    ticks(1) = plotlims.lin(1)+0.25;
    while ticks(i-1)<limits(2)-0.25
        ticks(i) = plotlims.lin(1)+0.25*i;
        i = i + 1;
    end
    ticks(i) = limits(2);
end

switch s{k}
    case {'Polar', 'Rectangular'}
        Plane         = handles.plotplane.SelectedObject.String;
        PlaneAngle    = round(handles.plotAngSlider.Value);

        PlotMat(1,:) = 0:2*pi/360*res:2*pi;

        switch Plane
            case 'Phi'
                for i = 1:size(PlotMat,2)
                    [i_p, i_t]    = r2t(PlotMat(1,i),d2r(PlaneAngle),res);

                    PlotMat(2,i) = PlotData(i_p,i_t);

                    label = '\theta';
                end
            case 'Theta'
                PlotMat(2,:) = PlotData(:,mod(PlaneAngle,180)+1);

                label = '\phi';

                % For tilted theta plane:
%                 R = rotY(-(pi/2-d2r(PlaneAngle)));
```

```
%                   for i = 1:size(PlotData,1)
%                       [x,y,z]      = s2c(pi/2,phir(i),1);
%                       q            = R*[x,y,z]';
%                       [u,v,r]      = c2s(q(1),q(2),q(3));
%                       [i_p,i_t]    = r2t(u,v,res);
%                       PlotMat(2,i)= PlotData(i_p,i_t);
%                   end
        end

        switch s{k}
            case 'Polar'
                polarplot(PlotMat(1,:),PlotMat(2,:));
                rlim(limits);

                ax = gca;
                d  = ax.ThetaDir;
                ax.ThetaDir           = 'counterclockwise';
                ax.ThetaZeroLocation = 'top';

            case 'Rectangular'
                PlotMat(1,:) = r2d(PlotMat(1,:));
                plot(PlotMat(1,:),PlotMat(2,:));
                xlim([0,360]);
                ylim(limits);

                xlabel(label);

                if dB == true
                    ylabel('dB');

                end
        end

        case '2D'
            [Y,X] = meshgrid(thetar,phir);

            X = r2d(X);
            Y = r2d(Y);

            R = PlotData-min(min(PlotData));
            Rmax = max(max(R));

            % Red
            C(:,:,1) = subplus(-cos(pi.*R./Rmax));
            % Green
            C(:,:,2) = sin(pi.*R./Rmax);
            % Blue
            C(:,:,3) = subplus(cos(pi.*R./Rmax));


            surf(X,Y,PlotData,C,'FaceColor','interp','MeshStyle','none');
            view(0,90);
            axis([0,360,0,180]);
            rotate3d off
            xlabel('\phi');
            ylabel('\theta');
            set(gca,'Ydir','reverse');

        case '3D'
        [az,el] = view;

        [theta3,phi3] = meshgrid(thetar,phir);

        if dB == true
            rho3 = PlotData - plotlims.dB(1);
        else
            rho3 = PlotData;
        end
```

```
        [X,Y,Z] = s2c(theta3,phi3,rho3);

        R       = sqrt(X.^2+Y.^2+Z.^2);
        Rmax    = max(max(R));

        % Colour matrix
        % Red
        C(:,:,1) = subplus(-cos(pi.*R./Rmax));
        % Green
        C(:,:,2) = sin(pi.*R./Rmax);
        % Blue
        C(:,:,3) = subplus(cos(pi.*R./Rmax));

        surf(X,Y,Z,C,'FaceColor','interp','MeshStyle','both','LineWidth',0.001,'
            EdgeAlpha',0.1,'LineStyle','-','EdgeLighting','none');
        set(gca,'DataAspectRatio',[1 1 1])
        h=rotate3d;
        set(h,'Enable','on');
        xlabel('x');
        ylabel('y');
        zlabel('z');

        view(az,el);
    otherwise
        error('Unsupported plot type');


end

%Filling inn supplimentary information
FieldTypeStr    = handles.plotField.String(handles.plotField.Value);
FieldValue      = handles.plotValue.String(handles.plotValue.Value);

switch FieldValue{1}
    case 'Abs'
        FieldValueStr = 'Absolute value';
    case 'Theta'
        FieldValueStr = 'Theta component';
    case 'Phi'
        FieldValueStr = 'Phi component';
    case 'Axial Ratio'
        FieldValueStr = 'Axial ratio';
end


if strcmp(FieldTypeStr,'Directivity')
    maxstr = 'D_0';
else
    maxstr = 'E_0';
end


% Unit label
if dB == true && strcmp(FieldTypeStr,'Directivity') && norm == false
    unitlabel = 'dBi';
elseif dB == true && strcmp(FieldTypeStr,'Directivity') && norm == true
    unitlabel = 'dB';
elseif dB == false && strcmp(FieldTypeStr,'Directivity')
    unitlabel = '';
elseif dB == true && strcmp(FieldTypeStr,'E-pattern')
    unitlabel = 'dBV';
elseif dB == false && strcmp(FieldTypeStr,'E-pattern')
    unitlabel = 'V';
end

compare = {Element(:).Compare};

```

```
props = whos('compare');

switch props.class
    case 'cell'
        compare = cell2mat(compare);
end

switch s{k}
    case {'2D', '3D'}
        % Colormap
        r          = linspace(0,Rmax);

        map(:,1)  = subplus(-cos(pi.*r./Rmax));
        map(:,2)  = sin(pi.*r./Rmax);
        map(:,3)  = subplus(cos(pi.*r./Rmax));

        map       = abs(map)./max(max(map));

        % Colorbar
        if dB == true
            tickLbl = linspace(plotlims.dB(1),max(max(PlotData)),10);
        else
            tickLbl = linspace(0,Rmax,10);
        end

        tickLbl = round(tickLbl,2);

        c = colorbar('TickLabels',tickLbl,'Ticks',linspace(0,1,10));
        colormap(map);
        c.Position = c.Position + [0.12,0,0,0];

        % Colorbar label
        c.Label.String = unitlabel;
        c.Label.Rotation = 0;
        c.Label.Position = c.Label.Position + [0.5,0,0];

        % Title
        if strcmp(FieldValue,'Axial Ratio')
            title(FieldValueStr);
            return;
        end
        title(strcat(FieldValueStr,{' of '},FieldTypeStr));
        return;
    case {'Polar','Rectangular'}

        % Title
        if strcmp(Plane,'Theta')
            planestr = '\theta';
        else
            planestr = '\phi';
        end



        if strcmp(FieldValueStr,'Axial ratio')
            title(strcat(FieldValueStr,{', '},planestr, {' = '},num2str(PlaneAngle), {char(176)}));
            return;
        else
            title(strcat(FieldValueStr,{' of '},FieldTypeStr,{', '},planestr, {' = '},num2str(PlaneAngle), {char(176)}));
        end

        if strcmp(s{k},'Polar')
            D0pos    = [d2r(315),D0+(D0-limits(1))*0.05];
            HPBWpos = [d2r(312),D0];
        else
            D0pos    = [370,(D0-limits(1))/2+limits(1)];
```

```
            HPBWpos = [370,(D0-limits(1))/2+limits(1)-1];
        end

        % Mainlobe marker
        [M,I] = max(PlotMat,[],2);

        text(D0pos(1),D0pos(2),strcat(maxstr,{' = '},...
            num2str(round(PlotMat(2,I(2)),2)),{' '},unitlabel),...
            'VerticalAlignment','bottom','HorizontalAlignment','left');
        hold on
        line([PlotMat(1,I(2)),PlotMat(1,I(2))],[limits(1),D0],'LineWidth',1.5,'Color','red');
        hold off

        if any(compare)
            return;
        end

        % HPBW markers
        if dB == true && strcmp(FieldTypeStr,'Directivity')
            HP = max(PlotMat(2,:))-10*log10(2);
        elseif dB == false && strcmp(FieldTypeStr,'Directivity')
            HP = max(PlotMat(2,:))*0.5;
        elseif dB == false && strcmp(FieldTypeStr,'E-pattern')
            HP = max(PlotMat(2,:))*sqrt(0.5);
        elseif dB == true && strcmp(FieldTypeStr,'E-pattern')
            HP = max(PlotMat(2,:))-10*log10(sqrt(2));
        end

        HPi = [];
        i   = 1;

        while isempty(HPi) && i<length(PlotMat)
            if PlotMat(2,mod(I(2)-i-1,length(PlotMat))+1) < HP
                HPi(1) = mod(I(2)-i-1,length(PlotMat))+1;
                break;
            end
            i = i+1;
        end

        if ~isempty(HPi) && abs(PlotMat(2,HPi(1))-HP) > abs(PlotMat(2,HPi(1)+1)-HP)
            HPi(1) = HPi(1)+1;
        end

        HPi(2) = 1i;
        i      = 1;

        while ~isreal(HPi(2)) && i<length(PlotMat)
            if PlotMat(2,mod(I(2)+i-1,length(PlotMat))+1) < HP
                HPi(2) = mod(I(2)+i-1,length(PlotMat))+1;
            end
            i = i+1;
        end

        if real(HPi(1)) == real(HPi(2)) || ~isreal(HPi(2))
            HPi = [];
        end

        if ~isempty(HPi) && abs(PlotMat(2,HPi(2))-HP) > abs(PlotMat(2,HPi(2)-1)-HP)
            HPi(2) = HPi(2)-1;
        end



        hold on
        for i=1:length(HPi)
```

```
            line([PlotMat(1,HPi(i)),PlotMat(1,HPi(i))],[limits(1),D0],'LineWidth',1,'Color','m');
        end
        hold off

        % HPBW label
        if ~isempty(HPi)
            HPBW = mod((PlotMat(1,HPi(2))-PlotMat(1,HPi(1))),2*pi);


            if strcmp(s{k},'Polar')
                HPBW = round(r2d(HPBW),1);
            else
                HPBW = round(HPBW,1);
            end

            text(HPBWpos(1),HPBWpos(2),strcat({'HPBW = '},...
                num2str(HPBW),char(176)),...
                'VerticalAlignment','bottom','HorizontalAlignment','left');
        end




end
UpdateMsgBox(handles,'Plotting... Done!');
```

Appendix C

Copyright Licence

The software is available at `https://github.com/EvenBirk/SmallsatArray` both as MATLAB script and figure, and as an executable running with MATLAB Runtime allowing anyone who don't have a MATLAB license run the software.

Listing C.1: MIT License for SmallsatArray

```
MIT License

Copyright (c) 2017 Even Birkeland

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```