



CHALMERS
UNIVERSITY OF TECHNOLOGY



Monitoring GNSS transmit power with small radio telescopes

Master's thesis in Wireless, Photonics and Space Engineering

MARCUS MALMQUIST

MASTER'S THESIS 2018

Monitoring GNSS transmit power with small radio telescopes

MARCUS MALMQUIST



Department of Space, Earth and Environment
Onsala Space Observatory
Space Geodesy and Geodynamics
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Monitoring GNSS transmit power with small radio telescopes
Master's thesis in Wireless, Photonics and Space Engineering
MARCUS MALMQUIST

© MARCUS MALMQUIST, 2018.

Supervisor: Grzegorz Klopotek, Department of Space, Earth and Environment
Examiner: Rüdiger Haas, Department of Space, Earth and Environment

Master's Thesis 2018
Department of Space, Earth and Environment
Onsala Space Observatory
Space Geodesy and Geodynamics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Two SALSA telescopes: Brage (left) and Vale (right).

Typeset in L^AT_EX
Gothenburg, Sweden 2018

Monitoring GNSS transmit power with small radio telescopes
MARCUS MALMQUIST
Department of Space, Earth and Environment
Chalmers University of Technology

Abstract

The small radio telescopes SALSA (“Such A Lovely Small Antenna”) refers to the system consisting of two (2.3 m) radio telescopes located at the Onsala Space Observatory and built with the aim of introducing students to the world of radio astronomy. In this project, SALSA has been extended with the possibility of tracking satellites that are part of the Global Navigation Satellite Systems (GNSS). This new feature of SALSA allows to carry out observations in an automatic fashion and store all the necessary information for subsequent data analysis. The latter stage aims at deriving important satellite-based parameters, namely the transmit power of GNSS satellites as well as their antenna gain patterns. Good knowledge of transmit power of GNSS satellites is required for an enhanced GNSS orbit modelling and GNSS orbit determination. In addition, satellite antenna gain patterns are vital for the calibration of data from GNSS reflectometry (GNSS-R).

The following report describes the technical specifications, system calibration procedure and measurement performance of SALSA with emphasis on the hardware modifications and software tools required to upgrade an uncalibrated system designed for astronomical observations at 1.4 GHz to a system that can be utilized for satellite observations at frequencies between 1 and 2 GHz. This project can form a foundation of a new routine service, regularly monitoring GNSS signals and providing users with the aforementioned satellite-based products.

Keywords: SALSA, GNSS, radio signal monitoring, satellite tracking, small radio telescope

Acknowledgments

I would like to thank my supervisor, Grzegorz Klopotek, for supporting and helping me during the project and assisting with reparation of a broken telescope. I would also like to thank Peter Forkman for assisting me with repair work related to the mechanical parts of the SALSA system. The workshop at Onsala Space Observatory assisted me with assembling the telescope and the lab at Onsala Space Observatory has provided me with a workbench and taught me to solder properly. Eskil Varenius, who created the original software and website, deserves to be mentioned for addressing all my questions regarding the existing system as well as ideas related to the calibration of SALSA.

Marcus Malmquist, Gothenburg, December 2018

Contents

List of Figures	xiii
List of Tables	xv
1 Introduction	1
2 Theory	3
2.1 GNSS signal characteristics	3
2.2 System noise temperature	3
2.3 Signal propagation modeling	4
2.4 Natural radio sources	5
2.5 Traveling salesman problem	6
3 SALSA system description	7
3.1 Control program	8
3.2 USRP	8
3.3 RIO	9
3.4 Dish	10
4 Methods	13
4.1 Preparing the existing system	13
4.1.1 Identifying the core components	14
4.1.2 Creating the core components	14
4.2 Batch measurement program	15
4.2.1 Tracking model	15
4.2.2 Measurement model	15
4.2.3 Post-processing model	16
4.3 Telescope Motion	16
4.3.1 Software modifications	18
4.3.1.1 Position tracking	18
4.3.1.2 Move loop	18
4.3.1.3 Improvements	19
4.3.2 Hardware repairs	20
4.4 Optimal Telescope route	21
4.4.1 Algorithm description	22
4.4.2 Static vs. dynamic routing	23
4.5 <i>Observation procedures</i>	24

4.5.1	Measurements to Cassiopeia A	24
4.5.2	Zenith measurements	25
4.5.3	Measurements to GNSS satellites	25
5	Results	27
5.1	Preparing the existing system	27
5.1.1	Batch measurement program	27
5.1.2	Measurement model	27
5.1.3	Post-processing model	28
5.2	Telescope Motion	28
5.2.1	Hardware repairs	29
5.2.2	Software modifications	30
5.3	Optimal Telescope route	31
5.3.1	Implementation	31
5.3.2	Naive routing	32
5.3.3	TSP routing	32
5.3.4	Naive vs. TSP routing	33
5.3.5	Static vs. dynamic routing	33
5.4	Observations	34
5.4.1	Cassiopeia A	34
5.4.2	Zenith diode switch	34
5.4.3	GNSS satellites	36
6	Conclusions and Outlook	37
6.1	Calibration	37
6.2	Pathfinding	38
6.3	Software improvements	39
6.4	Hardware improvements	39
	Bibliography	41

Abbreviations

ADC	Analogue-to-Digital Converter
BeiDou	BeiDou Navigation Satellite System
CCW	Counterclockwise
CDMA	Code-Division Multiple Access
CW	Clockwise
DMC	Digital Motion Control
EIRP	Effective Isotropic Radiated Power
FDMA	Frequency-Division Multiple Access
FFT	Fast Fourier Transform
Galileo	Galileo
GLONASS	Globalnaya Navigatsionnaya Nputnikovaya Sistema
GNSS	Global Navigation Satellite System
GNSS-R	Global Navigation Satellite System Reflectometry
GPS	Global Positioning System
GUI	Graphical User Interface
LNA	Low Noise Amplifier
MA	Multiple Access
RFI	Radio Frequency Interference
RIO	Remote Input/Output
SALSA	Such A Lovely Small Antenna
SINEX	Solution Independent Exchange Format
SWIG	Simplified Wrapper and Interface Generator
TLE	Two-Line Element
TSP	Traveling Salesman Problem
USRP	Universal Software Radio Peripheral

List of Figures

3.1	The SALSA rack with USRPs and RIOs.	7
3.2	The Brage telescope with its 2.3 m dish and the antenna horn where the monopole and LNA are located. A noise diode is located in the center of the dish and is aligned with the monopole in the horn. . . .	8
3.3	The GUI of the control program that is used to make observations with SALSA.	9
3.4	USRP and wideband LNAs located in the SALSA rack.	9
3.5	The RIO that is used to control the Brage telescope.	10
3.6	SALSA radio telescopes (Vale - foreground, Brage - background) as seen from the control room.	11
4.1	The core components of the SALSA software package. Arrows represent direct interaction between the components. Dotted lines represent breakpoints between levels of abstraction.	14
4.2	The node concept, where the boxes represent an action that is performed when the node is executed and the arrows represent the relationship between the nodes. A node can execute the next node multiple times depending on its implementation.	16
4.3	Time series of power measurements with the former DMC code for a 60-hour measurement of all visible GPS satellites using Vale. Different line colors and marker shapes represent different satellites. The satellites that are included in the image have been hand-picked to illustrate the trend.	17
4.4	Best- and worst-case scenarios for the naive method (sort by azimuth).	21
4.5	The node structure of the Cassiopeia A beam switching observation.	24
4.6	The node structure of the Zenith beam-switching observation.	25
4.7	The node structure of the 2-hour observation of GPS satellite PRN 30.	25
5.1	Two node structures and their equivalent procedure.	29
5.2	A comparison of the performance of the telescope control code using the old software (upper) and the new software (lower).	30
5.3	Computation times for different pathfinding methods.	33
5.4	Observation of Cassiopeia A using beam-switching mode (20 switches) where the beam offset was 6° in azimuth. 40 observations with an integration time of 100 ms and a bandwidth of 200 kHz was made for each switch (on or off target).	35

5.5	A 12-hour observation at zenith (telescope points at 90° in elevation) using frequencies 1000 MHz through 2000 MHz in steps of 4 MHz and an integration time of 3 seconds with a bandwidth of 5 MHz.	35
5.6	Time series of power measurements shown for a 2-hour measurement of GPS PRN 30 satellite at L1 for an integration time of 3 s using signal mode (top) and an example of the recorded spectrum for one of the measurements (bottom).	36

List of Tables

2.1	Frequency bands and the corresponding frequencies for GPS [12], Galileo [13] and GLONASS [14].	4
5.1	Data format used to store the measurement results.	34

Chapter 1

Introduction

Global position determination of objects finds its applications in science and industry. One way for obtaining such information is utilization of signals transmitted by GNSS satellites. GNSS is a general name for systems that use artificial satellites to provide navigation services. Examples of GNSS are the American GPS, the Russian GLONASS and the European Galileo. GNSS satellites broadcast navigation signals towards the Earth and these signals can be recorded at different frequency bands by dedicated receivers. The position of the device that receives these signals can be estimated using navigation signals from at least four satellites. The positioning accuracy of the receivers is directly related to the accuracy of the position of the GNSS satellites (i.e. the satellite orbits). This is especially important for high-precision global positioning.

One of the factors that needs to be considered for an improved knowledge on satellite orbits is so-called antenna thrust that causes a change in the orbital radius of the satellite due to the signal transmission. It is therefore important to know how strong the transmitted signal is. Unfortunately, there is no official information on transmit power or satellite antenna gain pattern available. This has been addressed by [1] using a 30 m dish, but the signals from GNSS satellites are very strong and a smaller dish can be used instead. Larger telescopes are often expensive to operate and have limited tracking capabilities due to their size. A smaller telescope such as SALSA is much cheaper to operate and maintain. If properly calibrated, SALSA can be utilized for monitoring of transmit power of GNSS satellites on a regular basis, which would be an important contribution for orbit modeling and determination as well as GNSS reflectometry.

Another factor that needs to be considered for an improved knowledge on satellite orbits is solar radiation pressure. State-of-the-art models for solar radiation pressure effect on satellite orbits have accuracies on the order of about 5 cm [2]. Thus the antenna thrust effects with an impact on the radial component of the satellite orbits of up to 2.7 cm [1] come close to this uncertainty and therefore deserve further investigation.

GNSS signals need to be monitored on a regular basis, which should preferably be done in an automated process. The system needs also to be flexible enough to handle

many different types of measurement setups.

In order to solve this problem, the existing SALSA system will be used as a basis for the new system and it should reuse as much of the current features as possible.

For the reasons that will be presented in this report, the scope of this report has been limited to making sure that the SALSA system as a whole is stable, accurate and flexible enough to create a variety of observing modes that can be used for satellite tracking, system calibration or by users in order to create complex and automated observations.

In order to address this problem, the system needs to be verified on all levels and prepared for these new requirements that this extension may bring.

First a brief introduction to the topic is given in Chapter 1 and the corresponding theory in Chapter 2. The technical specification of SALSA is given in Chapter 3 followed by the description of used methods which are given in Chapter 4. The obtained results are summarized in Chapter 5. The last part of this thesis consists of the conclusions and future work related to this project, including system capabilities that can be further developed based on the results from this report, and will be discussed in Chapter 6.

Chapter 2

Theory

This chapter provides a brief theoretical background related to the topic of this project.

Section 2.1 describes the structure of GNSS signals and Section 2.3 concerns propagation of the radio signals through the atmosphere and modeling of the signal strength received at the observation site. Section 2.2 introduces terminology with regard to system noise temperature and Section 2.4 describes how natural radio sources can be used to calibrate the utilized measurement system. Section 2.5 describes a problem of finding the optimal path between a number of connected nodes while visiting all of them.

2.1 GNSS signal characteristics

Each GNSS transmits signals on at least two different frequencies to handle the delay that is caused by the ionosphere [11]. The signals consists of ranging codes and downlink system data. The ranging codes are used to estimate the position of a receiving device while the downlink system data are used to convey information about the satellite's status (e.g. its orbit model and on-board oscillator/clock offsets).

In order to distinguish signals transmitted by different satellites, multiple-access (MA) techniques are used by GNSS. The MA techniques that are used today in GNSS are FDMA and CDMA [11], which allow a receiver to distinguish signals coming from different satellites. Satellites from GPS and Galileo use CDMA while satellites from GLONASS use FDMA and CDMA [11].

The frequency bands that are used by GPS, Galileo and GLONASS are listed in Table 2.1.

2.2 System noise temperature

The measured satellite signals have components other than the original signal that is transmitted by the satellite. As the signal propagate through the system, system noise is added so that the signals sampled at the receiver are not just the signal sent

Table 2.1: Frequency bands and the corresponding frequencies for GPS [12], Galileo [13] and GLONASS [14].

Band	Center Frequency
L1 (GPS)	1575.42 MHz
L2 (GPS)	1227.60 MHz
L5 (GPS)	1176.45 MHz
E1 (Galileo)	1575.42 MHz
E6 (Galileo)	1278.75 MHz
E5 (Galileo)	1191.795 MHz
L1 (GLONASS)	$1602 \text{ MHz} + k \cdot 562.5 \text{ kHz}, x \in \mathbb{Z} : -7 \leq x \leq 6.$
L2 (GLONASS)	$1246 \text{ MHz} + k \cdot 437.5 \text{ kHz}, x \in \mathbb{Z} : -7 \leq x \leq 6.$
L3 (GLONASS)	$1201 \text{ MHz} + k \cdot 437.5 \text{ kHz}, x \in \mathbb{Z} : -7 \leq x \leq 6.$

from the observed satellite. If the noise that is added by the system is known, it can be subtracted from the sampled signal to yield the observed signal.

Noise power generated by the measurement system is often expressed as a system noise temperature. This comes from the fact that components in a system introduce thermal noise [7]. The noise that a component introduces to the system can be expressed in terms of an equivalent noise temperature. The conversion between noise power and noise temperature is given by (2.1), where P is noise power (in watts) generated by the component, $k_B \approx 1.38 \cdot 10^{-23} \text{ JK}^{-1}$ is the Boltzmann constant, T is the equivalent noise temperature of the component, and B is the observed bandwidth (in Hertz).

$$P = k_B T B \quad (2.1)$$

From (2.1) it is clear that the noise temperature is invariant with respect to bandwidth.

The noise that is added by the system can be represented by its equivalent noise temperature T_{sys} . For radio telescopes like SALSA, T_{sys} contains the noise temperature that is introduced by all components of the receiving chain, including antenna and noise that it picks up (T_{ant}) and the receiver (T_{rx}) itself. While T_{sys} is invariant with respect to bandwidth it may be different for different observation frequencies because the components of the system have noise figures that are frequency dependent. The antenna introduces noise that varies with an elevation angle as a parabolic antenna can pick up additional noise through its sidelobes (e.g. when tracking satellites at low elevations).

2.3 Signal propagation modeling

The signal strength of GNSS signals that is received at an observing site can be represented (in a linear scale) by (2.2) [1]

$$P_r = P_t \frac{G_t G_r}{L_{\text{pol}} L_{\text{fs}} L_{\text{atm}}}, \quad (2.2)$$

where P_r is the received power (in watt), P_t is the transmitted power (in watt), G_t is the transmit antenna gain, G_r is the receive antenna gain, L_{pol} is the polarization mismatch loss, L_{fs} is the free-space loss and L_{atm} is the atmospheric attenuation. P_t and G_t are sometimes combined (2.3) to represent an effective isotropic radiated power (EIRP), which is the effective radiated power in the direction of the main lobe of the antenna.

$$\text{EIRP} = P_t G_t \quad (2.3)$$

G_r for a parabolic antenna can be expressed (in a linear scale) as (2.4) [1]

$$G_r = k \left(\frac{\pi D}{\lambda} \right)^2, \quad (2.4)$$

where k is an aperture efficiency factor, D is the diameter of the dish and λ is the wavelength of an observed signal. For SALSA the gain for L-band observations ($\sim 1 - 2$ GHz) is about 30 dB

L_{fs} is given by (2.5) and is about 180 dB in zenith and increases to 184.5 dB for an elevation angle of 5° on L1 [1]

$$L_{\text{fs}} = \left(\frac{\lambda}{4\pi d} \right)^2, \quad (2.5)$$

where λ is the wavelength of the signal and d is the distance between the satellite and the receiver antenna.

L_{atm} can be calculated in accordance to ITU recommendation P.676 (attenuation by atmospheric cases) [16]. The value of L_{atm} for L-band observations is approximately 0.03 dB in zenith and increases to 0.4 dB for an elevation angle of 5° [1].

The unknown parameters in (2.2) is the transmit power (P_t) and gain pattern of the transmit antenna (G_t). In the case of GNSS satellites, the satellite antenna gain is not constant and varies with the nadir angle.

2.4 Natural radio sources

Natural radio sources are objects in outer space that emit thermal radiation producing a continuous signal (Gaussian noise) spectra [8]. An example can be Cassiopeia A, which is a supernova remnant in the Cassiopeia constellation, and is one of the brightest radio sources for frequencies above 1 GHz [9]. The flux density of Cassiopeia A has been theoretically calculated and also measured many times. One such model was proposed by [10]. For Cassiopeia A the flux density can be computed from (2.6).

$$\begin{aligned} d(\nu) [\% \text{ per year}] &= (0.97(\pm 0.04) - 0.30(\pm 0.04) \log \nu [\text{GHz}]), \\ S(\nu, \Delta t) [\text{Jy}] &= 10^{5.745 - 0.77 \log \nu [\text{MHz}] \frac{100 - d(\nu) \Delta t}{100}} \end{aligned} \quad (2.6)$$

Here, $d(\nu)$ is the frequency dependency (ν in Hz) of secular decrease in the flux density (d in % per year). $S(\nu, \Delta t)$ is the flux density in Jansky's ($1 \text{ Jy} = 10^{-26} \text{ Wm}^{-2} \text{ Hz}^{-1}$), ν is the frequency (in Hz) and Δt is the time (in years) since the year 1980.

The spectral power density that is received by an antenna can be then computed from (2.7)

$$k_B T_a = \frac{1}{2} \eta A S \quad (2.7)$$

Here, T_a is the antenna temperature, η is the aperture efficiency of the dish, A is the geometrical area of the dish and S is the source flux from (2.6). The factor $\frac{1}{2}$ comes from the polarization loss (SALSA antenna receives only one polarization).

2.5 Traveling salesman problem

The traveling salesman problem (TSP) defines a problem where the cheapest path is sought, given a set of connected nodes and weights between them, and applies to a wide variety of problems. The complexity of solving the TSP is up to $\mathcal{O}(n!)$ (where n is the number of nodes), making it very impractical for even a few nodes (if they are all connected). This complexity is associated with a brute force search, where all combinations are tested to find the cheapest solution.

An approach that has the potential to perform better is to use a branch-and-bound algorithm, where a path must be cheaper than some upper bound in order to be considered. This means that some branches can be discarded if that branch is guaranteed to yield a solution that is worse than the upper bound.

A third approach is to use a heuristic and approximation algorithm that has a high probability of being close to the optimal solution.

Chapter 3

SALSA system description

SALSA refers to the two small radio telescopes located at the Onsala Space Observatory and is used to bring interactive astronomy to students [3]. Two radio telescopes, Vale and Brage, are available for anyone to use and can be booked and accessed through a website¹. The user interacts with a control program that can be used to set up observations, view the results and also upload the results to a database. The results can be then retrieved from the database through the website for further processing.

The main components of the SALSA system are shown in Figure 3.1 and Figure 3.2.



Figure 3.1: The SALSA rack with USRPs and RIOs.

¹<https://vale.oso.chalmers.se/salsa/observe>



Figure 3.2: The Brage telescope with its 2.3 m dish and the antenna horn where the monopole and LNA are located. A noise diode is located in the center of the dish and is aligned with the monopole in the horn.

3.1 Control program

The user interacts with the system and make observations through a control program. The control program is accessed using the SALSA GUI that is shown in Figure 3.3. The control program (and the GUI) is written in Python 2.7 and combines different components of SALSA to create an easy-to-use interface allowing to work with the telescope and the receiver. The position of GNSS satellites are predicted using TLE data from CelesTrack². In the case of natural radio sources and objects like Moon, PyEphem³ is used.

3.2 USRP

The USRP is an Ettus Research⁴ USRP N210 [4] that can operate in the frequency range [0 6] GHz and has a 100 MS/s (megasamples per second) ADC. It is equipped with an Ettus Research DBSRX2 daughterboard that is used as a receiver and can operate at frequencies 800-2300 MHz and a bandwidth of 8-80 MHz. The control program communicates with the USRP using the free software toolkit GNU Radio⁵. Before each USRP the received signal is amplified by 3 wideband (1-2.4 GHz) LNAs (one near the horn, two directly before the USRP), each with noise figure of 0.6 dB and gain of 30 dB. USRP and LNAs for both Vale and Brage are shown in Figure 3.4.

²<http://celestrak.com/>

³<https://rhodesmill.org/pyephem/>

⁴<https://www.ettus.com/>

⁵<https://www.gnuradio.org/>

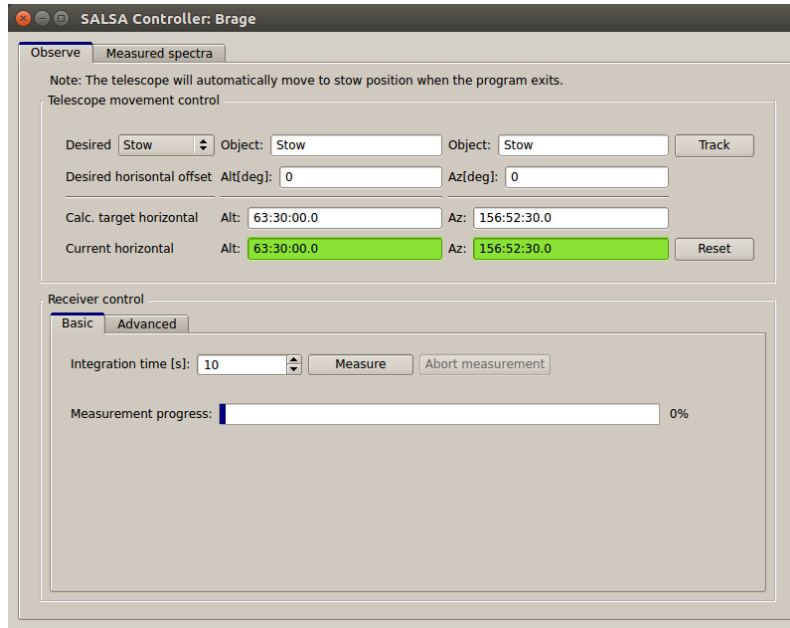


Figure 3.3: The GUI of the control program that is used to make observations with SALSA.

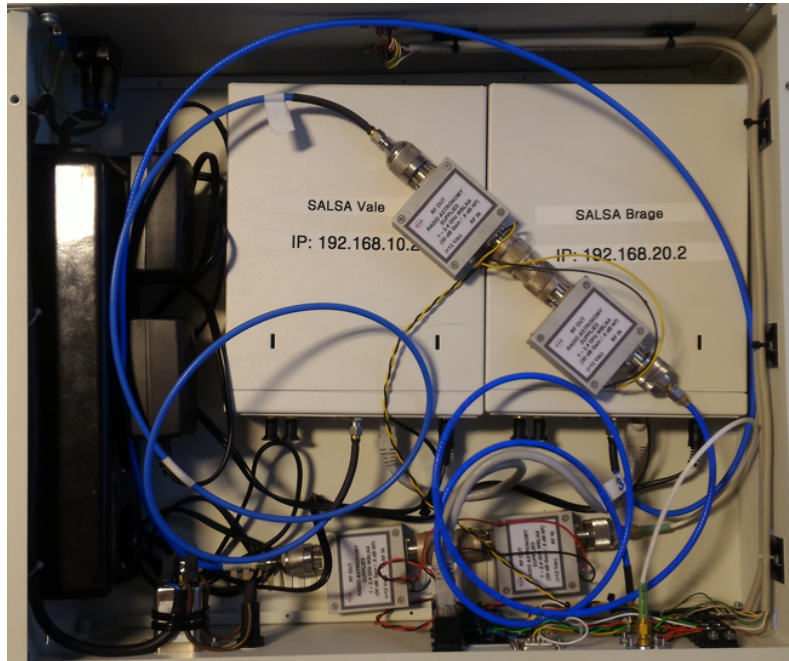


Figure 3.4: USRP and wideband LNAs located in the SALSA rack.

3.3 RIO

The RIO is a Galil⁶ RIO-47200 [6] and is used for controlling the telescope motors. It has an ethernet and RS-232 port to enable communication with other devices (i.e. the SALSA software host can communicate with the RIO through the Ethernet

⁶<http://www.galil.com/>

port). The RIO-47200 can hold a custom script containing up to 200 lines and less than 40 columns in memory. The script is written in Galil's proprietary language DMC [17] using Galil's proprietary tool GalilTools [18].

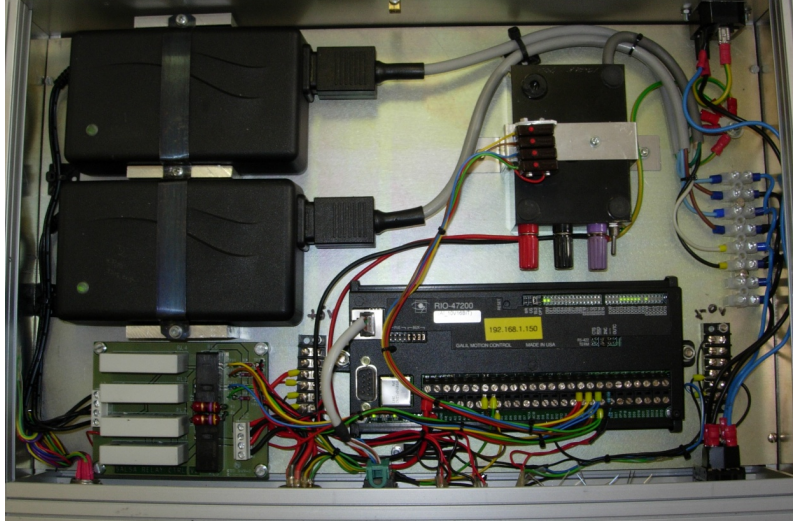


Figure 3.5: The RIO that is used to control the Brage telescope.

3.4 Dish

A SALSA telescope consists of a 2.3 m parabolic dish and a receiver horn with an inner radius of 16 cm. Inside the horn there is a 4 cm monopole that is connected to an LNA. The dish along with its counterweights is mounted on the motor unit which is mounted on a 2 m pole. The SALSA telescopes are shown in Figure 3.6.

The noise diode is mounted in the center of the telescope dish and is aligned to the monopole in the antenna horn. When turned on, the diode will emit a wideband signal that will be picked up by the receiver. If the noise diode is stable and its noise temperature is known, it can be used to calculate the system noise temperature for different frequencies. Currently, only Brage is equipped with a noise diode.



Figure 3.6: SALSA radio telescopes (Vale - foreground, Brage - background) as seen from the control room.

Chapter 4

Methods

This chapter is divided into sections that describe how the existing system was set up so that it could be extended with the possibility of satellite tracking and also how the system was then extended in order to achieve this goal. A substantial amount of work has been carried out in order to repair the parts of the system that had not been working correctly. This chapter will also cover this topic.

The existing system was mainly prepared by revising smaller modules of the large main file and by extracting different parts of the code into separate modules. The main program (measurement program) has been created by combining the existing modules and extending their functionalities.

Repairing the broken and malfunctioning parts of the system required a fair amount of trial-and-error work as the documentation for the hardware was both difficult to find and often misleading.

4.1 Preparing the existing system

The existing program was designed to observe hydrogen in the spiral arms of our galaxy at 1.4 GHz. In this observing procedure, first the telescope moves to a point on the sky. The user can choose a proper observation setup (such as integration time, bandwidth or frequency) and then start the observation. When the observation is finished the resulting spectra are displayed in a figure that can be uploaded to a dedicated database and accessed via the SALSA website.

The existing program had the core of the program in the same module as the GUI, and it utilized this design to reduce the number of parameters that needed to be passed around. This meant that the core of the program was heavily coupled with the GUI and also required that the parts of the GUI were disabled or hidden during tracking and observation in order to get consistent results.

Before any attempts of extending the program, one needed to make the program more modular. The main program, or batch measurement program, would be very different from the existing program, but still use the same core features, such as moving the telescope and making observations. It was therefore necessary to identify the core components of the program and try to isolate them from the GUI.

4.1.1 Identifying the core components

The main purpose of SALSA is to observe signals from natural radio sources. A reasonable abstraction is therefore to have two separate modules; one for tracking objects and one for observing signals.

The tracking module must be able to compute or estimate the motion of the objects relative to the telescope and point the telescope to these locations. The observation module must be able to control the receiver and process the recorded data.

The identified core components are shown in Figure 4.1. This structure did not directly exist in the original program, but the following is the good approximation of what the control program is doing.

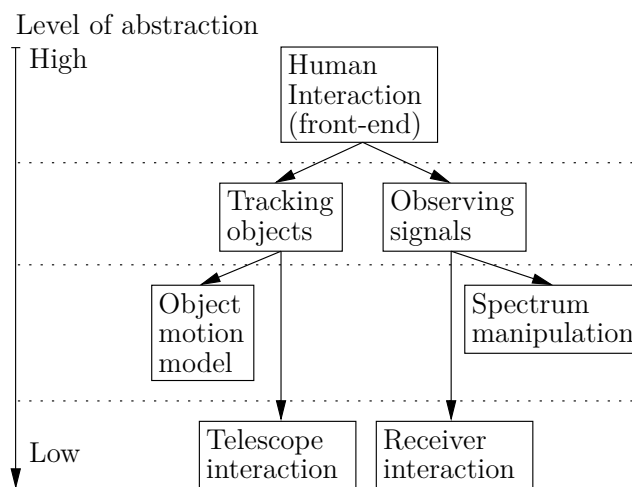


Figure 4.1: The core components of the SALSA software package. Arrows represent direct interaction between the components. Dotted lines represent breakpoints between levels of abstraction.

4.1.2 Creating the core components

One of the biggest challenge when isolating parts of a larger module is the data flow. Since the original program contained most of the core components which required access to the GUI, no data flow control had been established. If a component needed information from the GUI it could just get it directly from the GUI. This dependency also required that when the program was tracking objects or recording signals, the parts of the GUI related to those actions had to be disabled to avoid unexpected results. A necessary change was therefore to store information related to tracking once the tracking process started, and the same for data related to observation once the observation process started. This removed the requirement of disabling the GUI, allowing the user to set up the next observation (if any) while the current observation was in progress. The next step was to create the core components and move the relevant parts of the code there, while creating the interface to that module.

4.2 Batch measurement program

Selecting targets to track and observe their signals had to be done in an automatic fashion. A simple solution was to replace the GUI with a configuration file and completely remove the need for a GUI. A further modification of the model of the original program was to move the core components into a separate library, and identify a common interface to the library required by the GUI and the batch measurement program.

The purpose of the parts of the program a user would interact with would be to supply a tracking procedure, an observation procedure and a post-processing procedure. This meant that the library would have to provide an interface to set up a tracking procedure and a combination of observation and post-processing procedures.

4.2.1 Tracking model

The telescope itself provides a simple interface: it can be sent to a position, which it will hold until a new position is given. The tracking module would have to continuously compute the position of the object that is to be tracked and send the coordinates to the telescope.

A wide variety of objects would have to be tracked, as defined in the original program. The objects ranged from stars to artificial satellites orbiting Earth to fixed coordinates relative to the local sky of the telescope. A common interface had to be created which the tracking module could use to direct the telescope to a defined target. The most basic interface is the ability to get an up-to-date position of the object. Any object that is tracked had to provide such a possibility.

4.2.2 Measurement model

This model required a flexible observation procedure and needed to be different from what the user required in the existing program. This made it impractical to use a static observation model, e.g.

```

1: for all satellites do
2:   for  $i$  from 1 to  $k$  do
3:     for all frequencies do
4:       OBSERVE
5:     end for
6:   end for
7: end for

```

because all of the steps might not be required. Introducing flags to toggle the unwanted parts could be a solution, but it would reduce the readability of the code and not address the issue in case one might want to change the order of the steps.

A more flexible solution is to create observation nodes, which could be connected to create an observation procedure. The purpose of each node might be to modify the observation settings, such as frequency, before invoking the next node. This concept is illustrated in Figure 4.2.

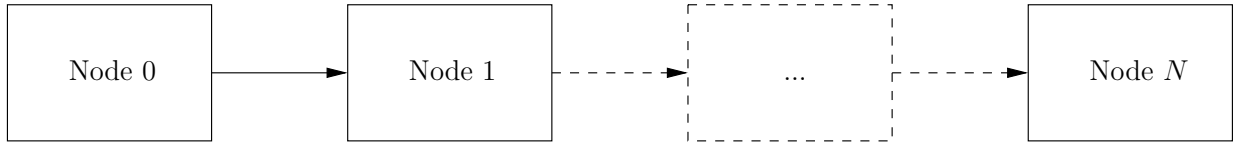


Figure 4.2: The node concept, where the boxes represent an action that is performed when the node is executed and the arrows represent the relationship between the nodes. A node can execute the next node multiple times depending on its implementation.

4.2.3 Post-processing model

The post-processing, much like the measurement model, required a flexible procedure that could vary depending on the application.

A node structure similar to the observation model was used. The post-processing nodes could be hooked up to the final observation node which meant that the selected post-processing procedure would be repeated for every observation.

The observation node would set up the receiver to stream the signal it has sampled (a vector of complex-valued voltages) to a file. The signal data was then loaded from the file and converted into a voltage spectra (by applying an FFT on it) and converted into a power spectra. The power spectra could then be loaded from the file and processed using a node structure.

The purpose of each node might be to modify the observed data (such as removing RFI) or writing the data into a file, before invoking the next node.

4.3 Telescope Motion

The telescope motion is controlled by turning on or off the motors that the telescope is mounted on. A cogwheel is mounted on the same axis that drives the motor. The cogwheel is also mounted between a light-sensitive detector and a light source, which means that whenever the motor is running the cogwheel will alternate between blocking the light (with a cog) or not blocking the light (no cog). A cog-hole pair represents a movement of $\frac{1}{8}$ degrees and takes about 70 ms to pass at full motor speed. The signal from the detector can be used to keep track of the telescopes current position. A RIO is used to create an simple-to-use interface to move the telescope. The RIO has outputs connected to the motors and inputs connected to the detector. A program running on the RIO keeps track of the position and moves the telescope to any position that is requested by the user. The goal of using the RIO is that the telescope should be as autonomous as possible.

The telescope seemed to perform just fine when used with the original software, but once it was used with the batch measurement program which was much more demanding, small errors started to appear. In particular, when looking at the data from a 60-hour tracking, the power vs. time data did not look as expected. There should not be any trend in the power with respect to time, but the power vs. time plot shown in Figure 4.3 clearly indicates a noticeable trend in the data that repeats

about every 12 hours. The script would automatically reset the telescope pointing relative to a reference position every 12 hours, and from Figure 4.3 it can be seen that the received power gradually decreases by about 10 dB for about 12 hours, after which it increases by the same amount from one observation to the next. This was an indication that there was something wrong with how the telescope keeps track of the position. This will be discussed in Section 4.3.1.

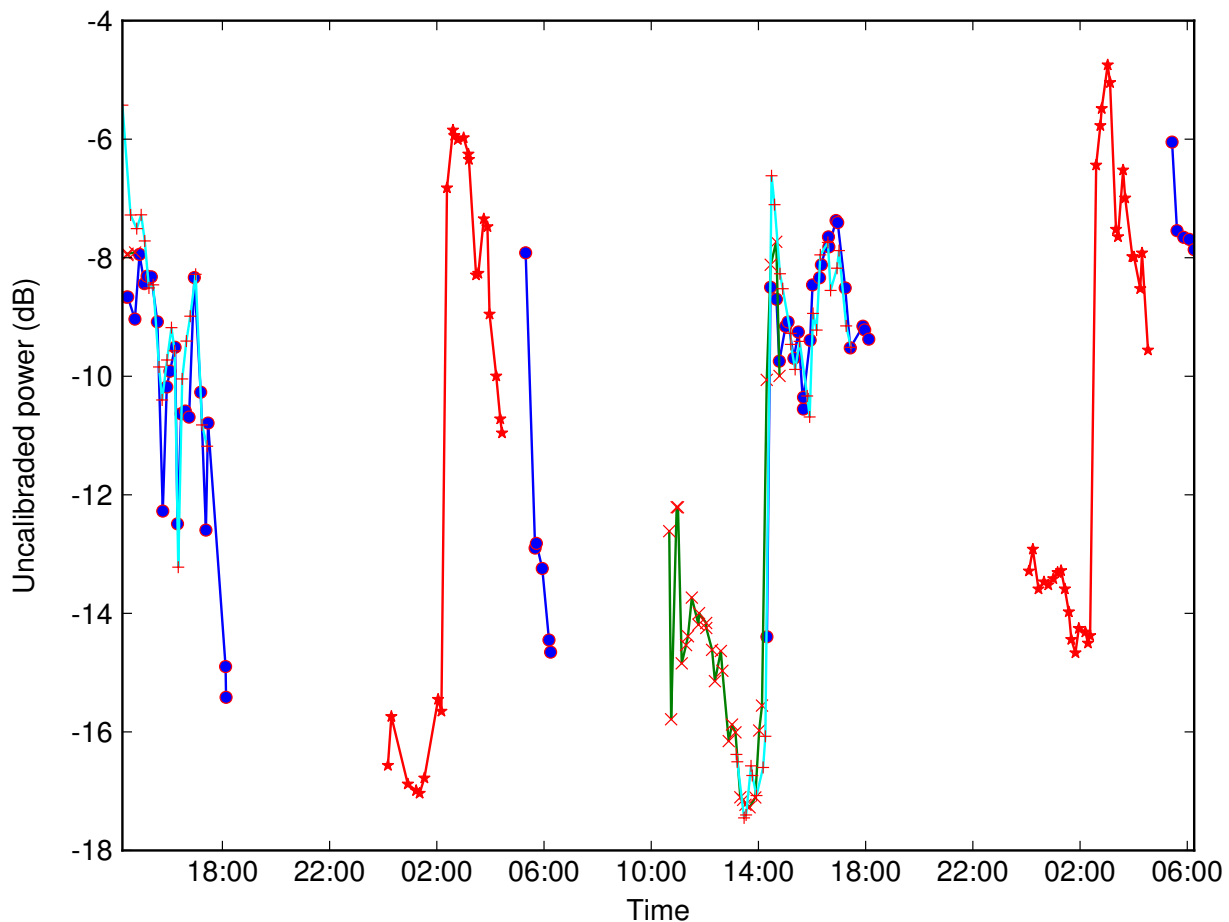


Figure 4.3: Time series of power measurements with the former DMC code for a 60-hour measurement of all visible GPS satellites using Vale. Different line colors and marker shapes represent different satellites. The satellites that are included in the image have been hand-picked to illustrate the trend.

SALSA has normally two operational telescopes, Vale and Brage, but one of them, Brage, had been disassembled for maintenance. Users had noticed that the telescope quite often stopped responding to commands and required a reset to a power-on state as well as reset the pointing to start working again. Since the operational telescope, Vale, was going to be out of service for maintenance, it was worth taking a look at Brage as well. This is discussed in Section 4.3.2

4.3.1 Software modifications

The DMC program is restricted to 200 lines and 40 columns to fit into memory and must be written in a language called DMC which is created by the manufacturer. A DMC program consists of commands that are two characters long and supports variables and subroutines. Each operation takes about 40 μ s to execute, which means that it is sometimes necessary to take the execution time into account when writing the program.

The former DMC code consists of four main parts: initialization, azimuth position tracking, elevation position tracking and a move loop. The initialization step moves the telescope to a known location and starts the azimuth and elevation position tracking. The move loop can then be started later by the user.

The greatest challenge of writing the telescope control program is to get all of the necessary features to meet the size constraints imposed by the operating environment. The updated software has a lot more features but is more dense, making it harder to read, and there is also little room for comments to explain the code (those are located in a separate file).

4.3.1.1 Position tracking

The position tracking procedure consists of a loop that checks the state of the detector several times (for redundancy). The state is 0 when a cog is blocking the detector and 1 when the detector is not blocked. The state is then established based on the median value of the samples. When the state is deduced it is compared to the previous state and if they do not match the telescope must have moved. The direction can be deduced from the value of the CCW output (1 means CCW direction, 0 means CW direction).

The main issue with this part of the code is that it uses 50 samples at a rate of 1 Hz. At full motor speed it takes about 70 ms to pass a cog which is quite close to the time it takes to sample and deduce the state. Reducing the sampling rate (to about 10 ms) solved this issue.

The second issue is that in some rare situations the telescope can stop at the border between a cog and a hole. At those locations the mean value can fluctuate between 0 and 1 even though the telescope is not actually moving. With the reduced sampling time it is possible to require that all samples are either 0 or 1 in order to deduce a state. It is much more unlikely that the samples fluctuate between all samples to be 0 and all samples to be 1.

4.3.1.2 Move loop

The move loop is a procedure that runs the motors until the target position is reached. The time it takes to run each loop depends on whether or not the telescope is close to its target position in azimuth or elevation or both. Since the hardware only supports turning on or off the motors, the only way to vary the speed is to vary the time the motors are activated during a given interval. If the telescope

is close to the target position the time the motors are active is reduced.

The main issue with this part of the code is that when the telescope changes direction (in azimuth, elevation or both) the CCW output changes. This means that the position tracking procedure starts counting any position changes in the new direction, but since the telescope has a momentum, the direction is not changed immediately when the polarity of the voltage changes. So every time the movement direction changes, there is a significant risk that the position tracking starts to drift by a few cogs. This drift might explain why the telescope performs fine when it is sent to track one or two objects before being reset. Turning off the motors and waiting for the telescope to be stopped completely before changing the directions solved this issue.

The biggest challenge is implementing this in the DMC program since the entire code is limited to 200 lines that are less than 40 columns each. While the RIO has an internal clock with millisecond resolution, the time is stored in a 16-bit integer and resets every 65 536 ms. So instead of measuring elapsed time, the number of move loops can be counted. This means that the move loop should preferably have a fixed execution time. Having a fixed execution time makes it beneficial to have the azimuth and elevation move frames overlap. Doing so makes the code shorter and actually reduces the duration of the move loop which makes the telescope more responsive. Related tests have shown that it takes about 3 seconds from turning off the motors until the telescope has stopped.

Since it takes such a long time to change the direction of the telescope, it is important that the telescope does not need to stop if the direction is changed when the telescope is not moving and also that it does not need to stop when the direction is not changed, since this would make tracking impossible. One solution is to look at the difference between the current position and compare it to the current target position and the target position from the previous loop, as well as comparing the difference between the current position and the target position from the current loop and the previous loop. The comparison is made by multiplying the differences; if the result is negative the direction has changed.

4.3.1.3 Improvements

While the telescope should perform correctly after the changes made so far, it is not as autonomous as it should be. For example the move loop is not started after initialization, and the telescope itself does not signal when the initialization is complete. A more serious problem is that there is no bounds check on the target position, which means that one has to rely on the kill switches (a physical switch that cuts the power to the motors when hit) to cut the power if the telescope moves too far (moving to invalid positions may severely damage the telescope). To sort these and some other minor issues the code has to be rewritten.

The reference position is determined by moving the telescope in CCW mode for azimuth and elevation until the kill switches are hit. To ensure consistency with the reference position the motors are turned on for a few hundred milliseconds in CW

mode before moving to the kill switches (see Section 4.3.2) in CCW mode a second time. This step is necessary because the reference position can change by up to ± 10 cogs (about 1°) depending on the telescope's momentum when it hits the kill switch. When the initialization is complete the RIO sets a flag which can be checked by the host computer and the move loop is started automatically.

An additional routine has been added with the purpose of checking if the telescope is stuck and terminating the program if so. It is necessary to implement such a subroutine if for example the telescope is blocked by an object or unable to move because of strong wind to avoid damaging the telescope. The condition for being stuck is that the telescope is not at its target position and not moving.

The telescope is moved towards different locations by changing the target position, which means that the target position can change in the middle of a move loop. Before implementing the bound checking, it is necessary to change this, so the target position requested by the user is copied to an internal variable in the beginning of the loop, and any changes to the target position during the loop only have an effect on the next loop.

4.3.2 Hardware repairs

One of the telescopes, Vale, suddenly stopped responding in the azimuth direction. At the time this was the only operational telescope so the only option was to attempt to repair it. The measurements from Vale appeared to be rather noisy (a measurement precision of about 2 dB). If Brage was to be repaired the measurement results from the two systems could be compared to see if this was true, so repairing Brage would be the next logical step.

After some debugging it was found that the motors were fully operational, but one of the azimuth kill switches was broken and there was a problem with the conductor that delivered the power to the azimuth motor.

The kill switch was rather simple to replace. It was part of a circuit that, when the switch was pressed, cut the power in the direction the telescope was moving, but not in the other, thus stopping the telescope from moving further in its current direction but allowing it to move back.

On the contrary, the conductor connects the telescope to the SALSA rack in the nearby building and there was no way of knowing where it was broken. Fortunately the cable had twelve conductors but only eight of them were in use, so the solution was to replace the broken conductor with one of the spare ones.

With information and experience gained from repairing Vale, it was decided to attempt to repair the other telescope, Brage, which had been under maintenance for about six months. The problem with Brage was that it would occasionally stop responding and it was believed to be caused by a faulty motor. A new motor was being prepared to replace the old one but it was far from ready.

The old motor had been used when trying to understand how to repair the kill switch circuit for Vale, and it seemed to be fully functional. Since the problem with

Vale had been a broken conductor, it was possible that the problem with Brage was similar.

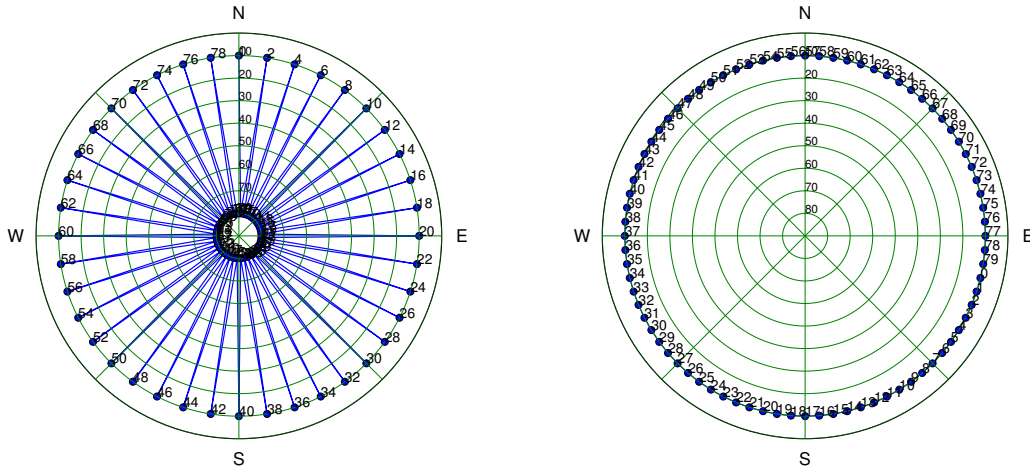
After running some tests in the lab the same problem appeared as before, only that the motor started moving again when the power cable was moved slightly. The problem seemed to be a loose contact or bad soldering in the connector to the motor. The connector near the telescope in the power cable was replaced and the telescope was ready to be assembled.

One of the telescopes, Brage, had a noise diode mounted on the dish, but turning it on made no difference in the recorded signal. After some debugging it was found that connectors were incorrectly soldered on both Vale and Brage, so the diode was never turned on. The connector was re-soldered and the noise diode's dipole was aligned with the telescope's monopole which fixed the issue.

4.4 Optimal Telescope route

For a slow-moving telescope such as the SALSA telescopes, the majority of the time in an observation lap is spent on moving between targets, so a faster route will yield many more observations.

The initial approach was to use a simple routing algorithm, such as the nearest neighbor or sorting positions by azimuth. Figure 4.4 demonstrates an important problem with such solutions: sometimes you may get the best solutions, but sometimes you may also get the worst solution. Using the worst solutions for a slow-moving telescope such as SALSA, a single observation lap can take hours. While it is very unlikely that one can get even close to the worst-case solution in a real environment, it would be preferable to use a routing algorithm that produces better and more even results.



(a) Solution for the worst-case scenario (b) Solution for the best-case scenario with naive routing.

Figure 4.4: Best- and worst-case scenarios for the naive method (sort by azimuth).

4.4.1 Algorithm description

Finding the most optimal route while visiting all nodes is known as the Traveling Salesman Problem, or TSP (see Section 2.5). For this system, the nodes are the satellites and the cost are the distance between them. It is possible to reach every satellite from each satellite, so this problem has a worst-case complexity of $\mathcal{O}(n!)$ (where n is the number of nodes), making it unfeasible to use for only a few satellites. Algorithm 1 describes the procedure for TSP using brute force. Such a procedure simply tries all permutations to find the cheapest one and have a complexity of $\mathcal{O}(n!)$. Even if a better algorithm is found, it is important to know how long the brute-force method will take for a given number of nodes, since this will be the upper bound for the computation time of that algorithm.

Algorithm 1 TSP (brute force)

```

1: function ROUTING(nodes, cheapest)
2:   for all permutations in nodes do
3:     if permutation  $\geq$  cheapest then
4:       cheapest  $\leftarrow$  permutation
5:     end if
6:   end for
7:   return cheapest
8: end function

```

A better algorithm than Algorithm 1 is Algorithm 2, which describes the procedure for TSP using branch-and-bound method. The difference here is that the algorithm keeps track of the most optimal route it has encountered so far. This means that some branches can be exited early because they are guaranteed to yield a route that is less optimal. Algorithm 2 is set up so the permutations are generated in a way that is favorable to both memory and caching, which are important properties to keep in mind when a small piece of a program is ran over and over again. It is favorable to memory because when the algorithm is complete, the order of the input nodes is maintained, so as long as the nodes are not shared between multiple threads, the nodes do not have to be copied before they are rotated. It is favorable to caching because the cost of the route up to the point where the program is executing can be precomputed and passed along with the subset of nodes when the function calls itself.

Even if Algorithm 2 is many orders of magnitude faster, it will only make an unfeasible algorithm slightly less unfeasible. The greatest performance gain will come from partitioning the nodes into smaller groups and apply the algorithm to the subgroups. If the partitioning was done correctly the optimal route is very likely to be the same or slightly more expensive than the optimal route. Using this technique the algorithm have to only be fast enough to be able to handle subgroups of nodes that will yield meaningful results.

There are many ways to come up with an optimal partitioning scheme, but one needs to keep in mind that SALSA is unlikely to track more than about 40 satellites (e.g. all visible satellites from BeiDou, GLONASS, GPS and Galileo). A simple partitioning should be enough. The telescope routing is a small part of the project

Algorithm 2 TSP (depth-first branch-and-bound)

```

1: function ROUTING(nodes, cheapest)
2:   if nodes  $\geq$  cheapest then
3:     return cheapest
4:   end if
5:   if remaining nodes = 2 then
6:     return nodes
7:   end if
8:   if remaining nodes  $\geq$  2 then
9:     for all rotations in NEXT_SUBSET(nodes) do
10:      cheapest  $\leftarrow$  ROUTING(rotation, cheapest)
11:    end for
12:   end if
13:   return cheapest
14: end function

```

so the method of partitioning was chosen to first sort the group of nodes by their azimuth position and then split it in half until all subgroups were smaller than some maximum value. Apart from some cases where a more optimal path would be achieved by moving the telescope past 90° in elevation, the chosen partitioning scheme should be sufficient to yield a near-optimal route.

4.4.2 Static vs. dynamic routing

The simplest solution is referred to as *static* routing, where the route is computed at the beginning of a lap and the satellites are observed one by one until the lap is complete. For any single observation lap, that involves multiple satellites and spans several minutes, there are some potential problems. Any satellite that is not geostationary will move relative to the telescope all the time. For example, a GNSS satellite with an orbit of about 12 hours will move an angular distance of 30° per hour. This means that the route that was computed at the beginning of the lap is only (near-)optimal at the time it was computed, and may become less and less optimal with time. Another potential problem is that satellites may disappear under the local horizon during a lap (which should be handled anyway). Depending on the current route, one could save some time by detecting such events and remove the satellites from the path.

A slight performance gain can be achieved by removing satellites from the path as you observe them and then recompute the optimal route. This requires that the routing algorithm is efficient enough so that any potential performance gained from a slightly shorter path is canceled by the extra computation time. This is referred to as *dynamic* (as-we-go) routing and will yield a more accurate route.

4.5 Observation procedures

In the observation procedures that are described in this chapter there are some modes that are worth mentioning. *Signal mode* means that the received power is a contribution of the signal of the target, antenna temperature (cosmic background, atmosphere) and system noise temperature. *Frequency-switching mode* means that the recorded power spectra using a frequency offset are subtracted from the recorded power spectra using the center frequency that are of interest. *Frequency-switching mode* is useful because it removes contributions from the system and unwanted noise picked up through the antenna's main beam (if it is the same for both frequencies). *Beam switching mode* means that an observation is made on the target(s) that is/are of interest and then using a pointing offset in azimuth or elevation. *Beam switching mode* is useful because it removes contributions from everything but the target (if the contribution from the sky is the same at both positions).

4.5.1 Measurements to Cassiopeia A

Cassiopeia A is a well known radio source that can be used for calibration at frequencies of interest for this project. A problem with Cassiopeia A is that it is very hard to detect for small antennas such as SALSA as the sensitivity of the telescope is proportional to the size of its dish. Inspired by [19], a beam-switching observation procedure was chosen. The series consists of 20 cycles, where each position (on the source or off the source) consists of 40 observations. Each observation uses an integration time of 100 ms and a bandwidth of 200 kHz (this is the lowest bandwidth that is available for the USRP, but a bandwidth of 40 kHz is used in [19]).

The node structure of the procedure is shown in Figure 4.5.

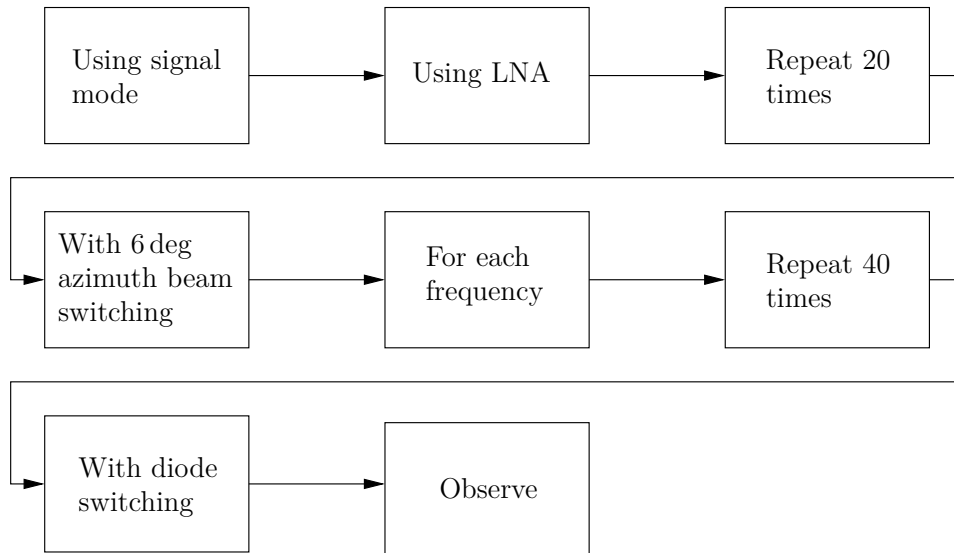


Figure 4.5: The node structure of the Cassiopeia A beam switching observation.

The measurement results from observations Cassiopeia A could be used to obtain a calibration factor by comparing the obtained values to the expected T_{ant} com-

puted using flux information of Casiopeia A calculated from the model described in Section 2.4.

4.5.2 Zenith measurements

SALSA is primarily used for observations at 1.4 GHz, but since there are wideband LNAs installed, the noise diode is used to check the frequency range that can be covered by SALSA. These measurements were carried out with an integration time of 3 s and the frequencies that were used spanned from 1000 MHz to 2000 MHz in steps of 4 MHz. The node structure of the procedure is shown in Figure 4.6 and was carried out for 12 hours.

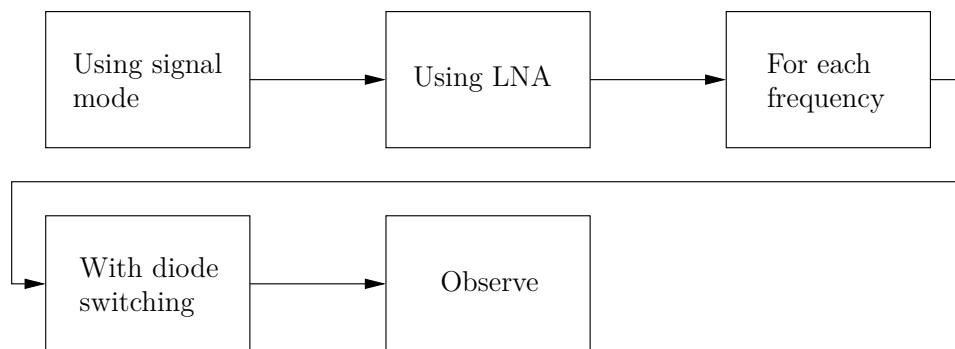


Figure 4.6: The node structure of the Zenith beam-switching observation.

4.5.3 Measurements to GNSS satellites

The original goal of this project was to monitor the transmit power of GNSS satellites, so some measurements of GNSS satellites should be included for demonstrational purposes.

The GPS satellite PRN 30 was observed with Vale on the L1 band for 2 hours. The measurements were carried out with an integration time of 3 s and the recorded bandwidth was 5 MHz. The node structure of this procedure is shown in Figure 4.7 and was carried out for two hours.

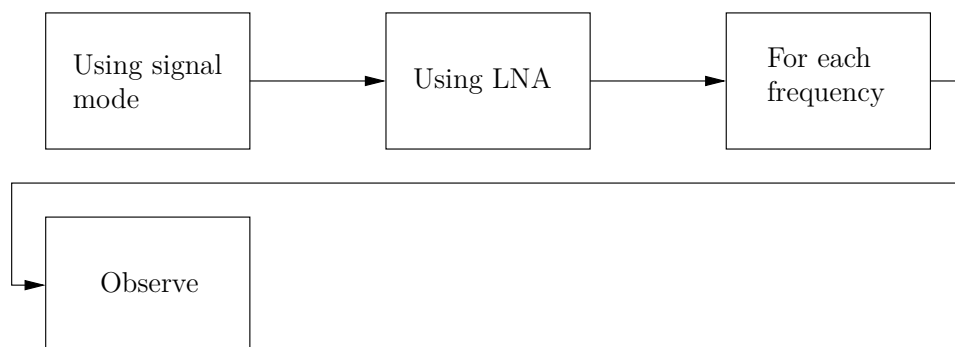


Figure 4.7: The node structure of the 2-hour observation of GPS satellite PRN 30.

Chapter 5

Results

The resulting algorithm implementations, system performance (both hardware and software) and observation results are presented in this chapter. The focus of this chapter is to present results indicating that the system is operating correctly and has features required for system calibration and GNSS satellite tracking.

5.1 Preparing the existing system

The existing main program has been adapted to work with the new program structure. The abstraction level in the main program has been increased and most of the responsibilities have been moved to separate packages. The responsibility of the main program is now to connect the different parts of the GUI to the new software package. An additional feature is that the dependency injection technique has been adopted in the majority of the code, which means that all objects are initialized by the user (e.g. the main program). While no unit tests have been written, this approach enables the possibility of creating unit tests in the future. Unit tests are tests that test the functionality of a unit (such as a function or a class) and are very difficult to write if one does not have the full control over how the unit interacts with other parts of the system, which is what dependency injection does.

5.1.1 Batch measurement program

The batch measurement program is similar to the main SALSA program, only the GUI has been replaced with a configuration file and the observation procedure is created when the program starts. The batch measurement program also has many more features available when creating observation procedures. The responsibility of the batch measurement program is to connect the different parts of the configuration file to the new software package and it can be easily modified to get different observation procedures.

5.1.2 Measurement model

Since the existing system had a number of different options these had to be converted into a node structure to ensure that the node structure was compatible with

the existing system. Next, some additional nodes were created to also include the features that were required by the batch measurement program. Figure 5.1 shows the node structure for two observation procedures and the equivalent implementation. This implies that a change of the order of only a two, three nodes will result in a very different observation procedure.

The node structure is also easily extendable, more flexible and more readable. Extending the structure is as simple as creating a class that implements the observation node interface, and adding the new node is only a matter of inserting it into the hierarchy. In the same manner it is also easy to remove nodes as it is only a matter of removing them from the chain. If the nodes have good names it will also become clearer what will be done during the measurement series (compare the readability of the node structure versus the equivalent code).

5.1.3 Post-processing model

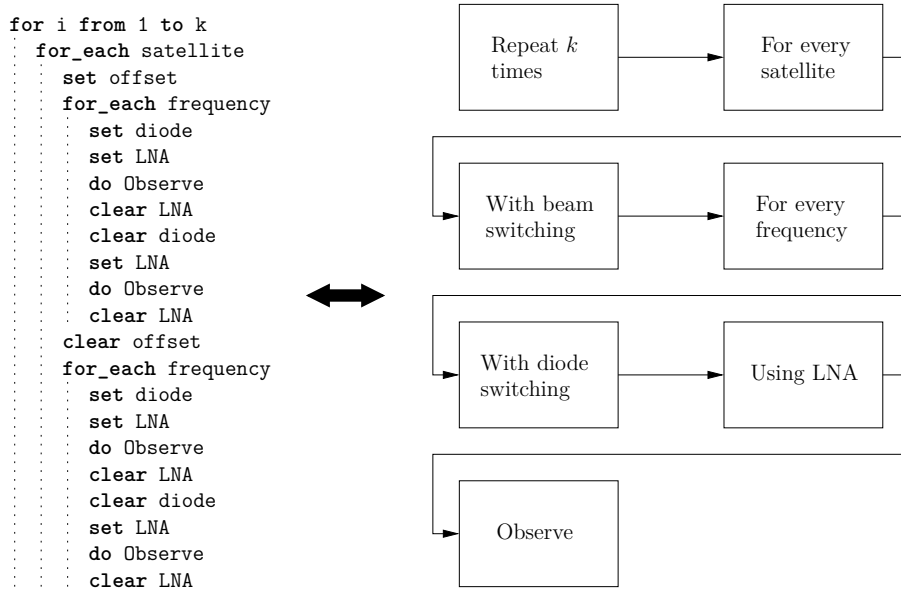
The post-processing model is almost identical to the observation model in its structure. This means that the flexibility of the observation model can also be found in the post-processing model.

The data from the receiver is a very large one-dimensional vector with complex values representing a voltage. For performance reasons the data had to be transformed into a collection of smaller one-dimensional vectors with a size that was favorable to applying an FFT on it, and then stacked after applying an FFT on each of them. In order to maintain a smooth observation process, observations with an integration time greater than 1 s, were split into batches of 1 s and the FFT was applied to each batch in a separate process. This meant that even for a 10-minute observation the FFT would appear to take a fraction of a second.

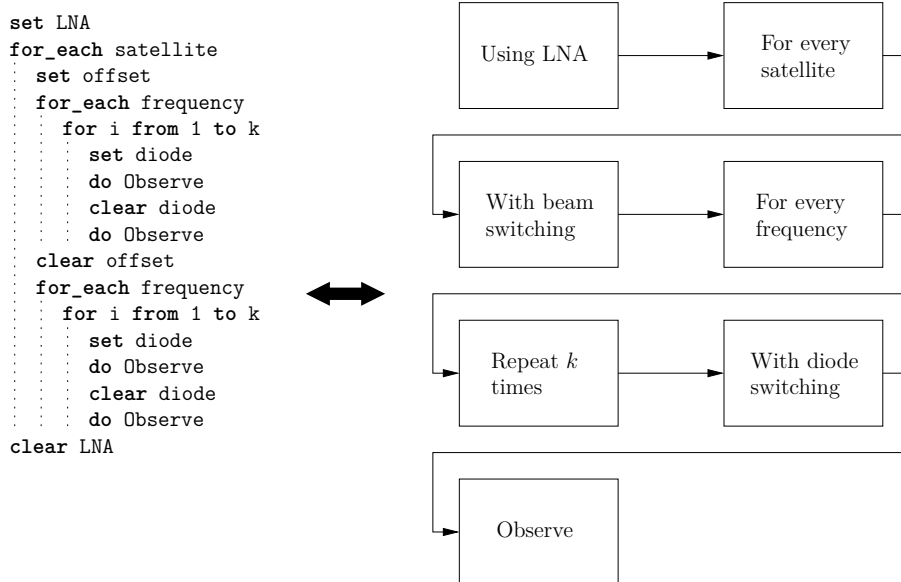
The separate processes were launched using a thread pool, and all resources allocated by the observation were only released when the thread pool terminated. Terminating the thread pool took around 100 ms, which is not negligible when the observation time is in the same order of magnitude or lower. A solution to this could have been leaving it to the garbage collector to terminate the thread pool when the observation object goes out of scope. This turned out to be difficult because the FFT was carried out in a member function of the observation object, thus blocking the garbage collector to delete the object. Instead, the thread pool was moved higher up in the calling hierarchy and was therefore shared between all observations. This meant that data from one up to a few batches would be kept at most. Given that the receiver itself can only handle a single observation at a time, there was no drawback of having a thread pool that was shared between observations.

5.2 Telescope Motion

The outcome of the modifications and repairs of the telescope motion system (both hardware and software) will be presented in this section. Both telescopes have been restored and are operational and the software has been updated on the RIO units that control the telescopes.



(a) An example of a sequence of observation nodes and the equivalent code.



(b) An example of a sequence of observation nodes and the equivalent code.

Figure 5.1: Two node structures and their equivalent procedure.

5.2.1 Hardware repairs

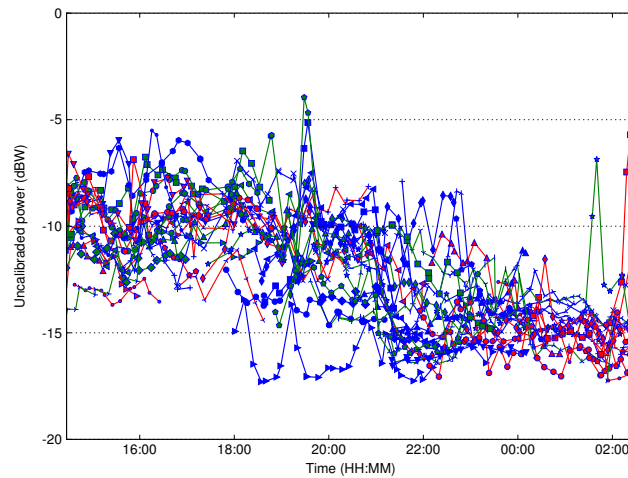
Vale has received a new kill switch circuit and the broken conductor was successfully replaced. Brage has also received a new kill switch circuit and has now been assembled at the observation site next to Vale. When replacing the connector at Brage it was discovered that one of the conductors was damaged. It was the same conductor that had been replaced on Vale, so it was replaced in the same manner.

The power to the noise diode has now been properly connected so that it can be turned on from the host machine, but only Brage has a noise diode installed on the

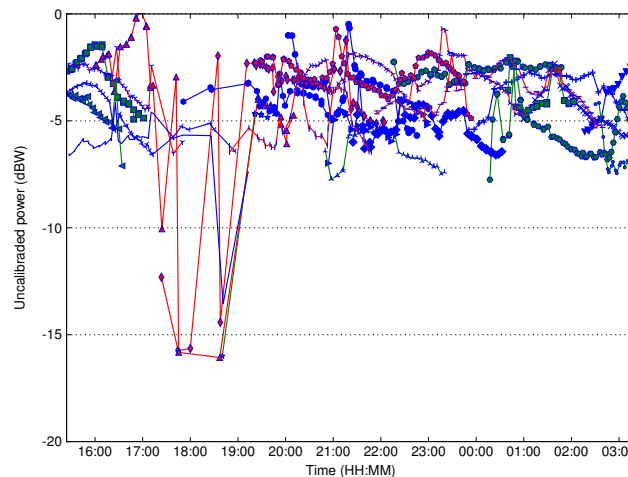
dish.

5.2.2 Software modifications

After installing the new DMC software on both RIO units the drifts disappeared. This is shown in Figure 5.2. The trend that the received power decreases with time (until the pointing is reset) that can be seen in Figure 5.2a (where the old DMC software is used) does not appear to be present in Figure 5.2b (where the new DMC software is used).



(a) Time series of power measurements shown for the first 12 hours from a 60-hour tracking of all visible GPS satellites using the old software. Different line colors and marker shapes represent different satellites.



(b) Time series of power measurements shown for the 12-hour hour tracking of all visible GPS satellites using the new software. Different line colors and marker shapes represent different satellites.

Figure 5.2: A comparison of the performance of the telescope control code using the old software (upper) and the new software (lower).

In addition to resolving the pointing drift, the new DMC software has made the telescope more autonomous in the sense that fewer actions are required by the SALSA host machine in order to operate the telescope.

5.3 Optimal Telescope route

The performance of the algorithm was evaluated by setting up an environment that is similar to the actual environment. This was done by creating satellite objects at random positions. For simplicity, the goal of the simulations was to evaluate how the algorithm performed in a static environment since it is easier to verify. Since the static environment was simulated the generated satellites did not move.

The comparison was made for 1000 pairs of Naive and TSP routings, where each pair used identical seed for the random function. This meant that within each pair, the static and dynamic simulations had identical setup.

5.3.1 Implementation

Algorithm 2 does not specify language and data structure, so a few different solutions with different performance were created. Since the rest of the SALSA code is written in Python, it was the first choice for the implementation. When the Python implementation proved to be too inefficient, the algorithm was implemented in Cython, which is Python with the option to specify types and also converts the Python code into C and compiles it for improved performance. Finally, when Cython was also deemed to be too inefficient, the algorithm was implemented in C++ and Python bindings were generated with SWIG.

The Python implementation was the first to be created and it used the built-in `list` type to store the nodes. When calling a function multiple times, the overhead of the function call becomes significant. Since Algorithm 2 is recursive, this proved to be a devastating problem and the algorithm was deemed to be too slow.

The Cython implementation was the second option to test since it is similar to Python but includes the option of adding type definitions and also is compiled into C code. While porting the algorithm to Cython, it is likely to yield a performance gain, it also comes with the price that the developer now needs to know two languages to maintain the system. This drawback, however, is small since the syntax is very similar.

The Cython implementation proved to be about 1000 times faster than the Python implementation. However, one of the features of Cython (that is is similar to Python), is also its weakness when performance is crucial. Without introducing C-style pointers it is difficult to manage the memory in greater detail (e.g. avoid unnecessary copying), and doing so would mean that the code looks less like Python and more like C. This proved to be a problem when tweaking the implementation and it was also deemed to be too slow.

The C++ implementation was the final implementation to be created. The built-in `list` type from the Python and Cython implementation was replaced with `std::list<int>` from the C++ standard library which gave a performance improvement of about 10 times. Further improvements occurred after simplifying the implementation and avoid unnecessary copying. This modification improved code performance by an additional factor of 1000.

Finally, `std::list` was replaced with `std::vector` which gave an improvement of about 20 %. The `std::list` container is more effective when rotating its elements because each element consists of a pointer to the next and previous element in the node and the data, which means that moving nodes is as simple as changing the pointers. This has the drawback that traversing a list is slower. The `std::vector` container, however, holds contiguous data, so traversing the list is very fast. This has the drawback that rotating a vector is slower because all the elements have to be shifted. When using the depth-first branch-and-bound algorithm it turns out that traversing the list is more important than rotating it. For the brute-force algorithm it is the opposite. This algorithm is also far too slow to be used on any larger collection of nodes, so the `int` was replaced with `uint_fast8_t`, which is an unsigned integer that is at least 8 bits and possibly more efficient than an unsigned 8-bit integer on the current hardware. So in the final implementation C++ and a `std::vector<uint_fast8_t>` container was used to realize the depth-first branch-and-bound algorithm.

5.3.2 Naive routing

The naive routing has a complexity of $\mathcal{O}(n \log_2 n)$ ¹, making it very fast to compute, but the performance of the resulting route can vary from the best-performing to a worst-performing solution.

5.3.3 TSP routing

The TSP routing has a complexity of $\mathcal{O}(n!)$, making it very slow to compute, but the performance of the resulting route is always the most optimal. To improve performance of the scheduling one can partition the route into subroutes, and solve the TSP algorithm for the subroutes. The subroutes can then be joined to create the full route. By partitioning the route, the individual subroutes have a complexity of $\mathcal{O}(n!)$ (where n is the number of nodes in the subpath), and the total route has a complexity of $\mathcal{O}(n)$ (where n is the number of subroutes) on top of that. One can then trade the path performance for faster computation, and the path performance loss depends on how the paths are partitioned.

The chosen partitioning method was to sort the satellites based on their azimuth position at the time of partitioning. From Figure 5.3 it can be concluded that the worst-case computation time for a route with 11, 12 and 13 nodes is about 60 ms, 660 ms and 9500 ms respectively, while the typical computation times for the same number of nodes is 1.8 ms, 8.5 ms and 54 ms respectively. For this reason one should avoid using more than 12 nodes per route.

¹<http://www.cplusplus.com/reference/algorithm/sort/>

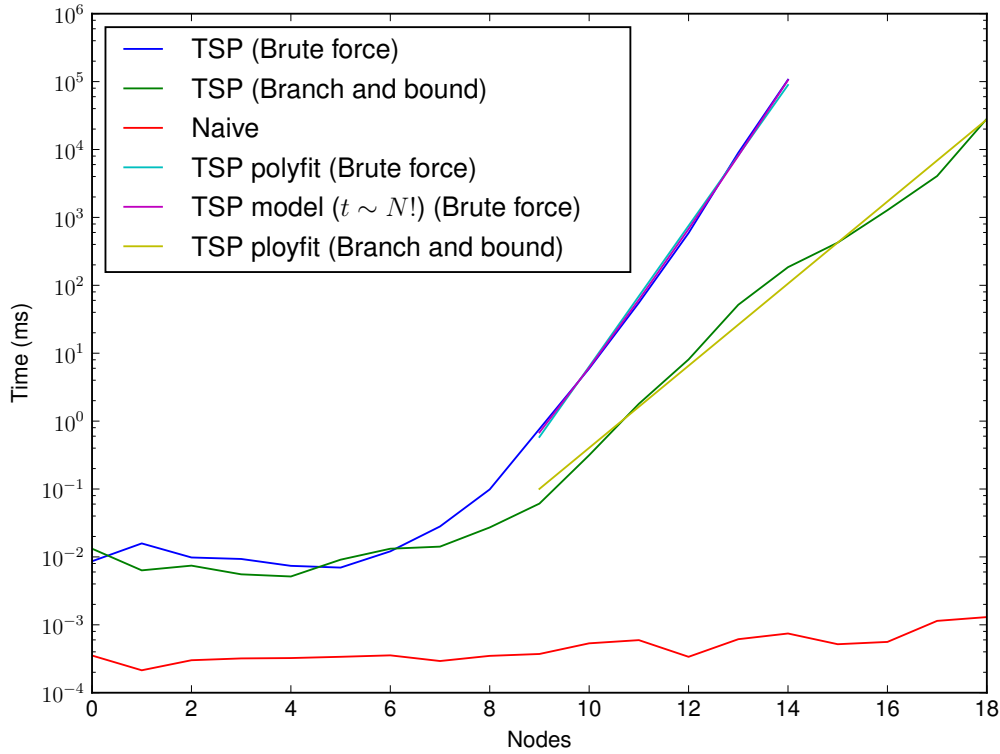


Figure 5.3: Computation times for different pathfinding methods.

The performance was investigated using simulations of satellites at randomly generated positions that did not change during the simulation. With 10 satellites the TSP routing was found to be 1.4 ms with a standard deviation of 1.1 ms. With 40 satellites the TSP routing was found to be 12.2 ms with a standard deviation of 6.0 ms. A total of 100 simulations were combined to produce each of the results.

5.3.4 Naive vs. TSP routing

With 40 satellites the naive routing was found to be 1.6 times the length of TSP with a standard deviation of 0.14. With 10 satellites the naive routing was found to be 1.4 times the length of TSP with a standard deviation of 0.19.

5.3.5 Static vs. dynamic routing

Actual observations have shown that a single lap with about 10 satellites and an observation time per satellite of about 15 s takes about 10 minutes, and 20-25 minutes with about 40 satellites with the same observation time. During this time the satellites move considerably. In particular, GNSS satellites with an orbit of 12 hours will move an angular distance of 30° per hour.

With 40 satellites the static routing was found to be 1.06 times the length of the dynamic with a standard deviation of 0.08. With 10 satellites there was no difference

between static and dynamic routing.

This means that the dynamic routing is better on average even if the satellites do not move.

5.4 Observations

The results in this section are based upon the data output from the post-processing node `SaveObservationDataNode`, which prints observation data to a file in the format presented in Table 5.1. Very little post-processing of the data has been done when displaying the results as it is outside the scope of this project.

Table 5.1: Data format used to store the measurement results.

Time	Name	Freq	Az	Offset	El	Offset	Power	Diode
2018:344:32048	Cas. A	1400	21.78	6.00	30.24	0.00	4.58e-03	1
2018:344:32048	Cas. A	1400	21.78	6.00	30.24	0.00	3.19e-03	0
...	Cas. A	1400	0.00
2018:344:32060	Cas. A	1400	21.80	6.00	30.25	0.00	4.47e-03	1
2018:344:32060	Cas. A	1400	21.80	6.00	30.25	0.00	3.11e-03	0
2018:344:32083	Cas. A	1400	21.86	0.00	30.27	0.00	4.53e-03	1
2018:344:32083	Cas. A	1400	21.86	0.00	30.27	0.00	3.16e-03	0
...	Cas. A	1400	0.00

5.4.1 Cassiopeia A

The results of an observation of Cassiopeia A with the settings described in Section 4.5.1 are shown in Figure 5.4.

Figure 5.4 indicates that SALSA can observe Cassiopeia A, but the received signal is very weak. By applying a weighted average, the signal from Cassiopeia is about $\bar{P}_{\text{Cas. A}} = 41.4 \mu\text{W}$ with a standard deviation of $\sigma_{\text{Cas. A}} = 7.91 \mu\text{W}$ (W refers to the uncalibrated power). Since the uncertainty in the contribution from Cassiopeia is very large, it may not be enough to integrate for 100 ms. This should be further investigated.

5.4.2 Zenith diode switch

The results of a 12-hour diode-switching observation at zenith using frequencies spanning from 1000 MHz to 2000 MHz in steps of 4 MHz are shown in Figure 5.5. In this case, an integration time of 3 s and a bandwidth of 5 MHz was used. About 50 measurements per frequency were made during the 12-hour period (25 pairs of diode on/off).

The error bars are quite small with the exception of around 1200 MHz, 1600 MHz and 1800 MHz. The larger error bars at 1200 MHz and 1600 MHz are likely caused by signals from GNSS satellites that have been picked up in some of the samples. The very large error bars at 1800 MHz are likely caused by RFI.

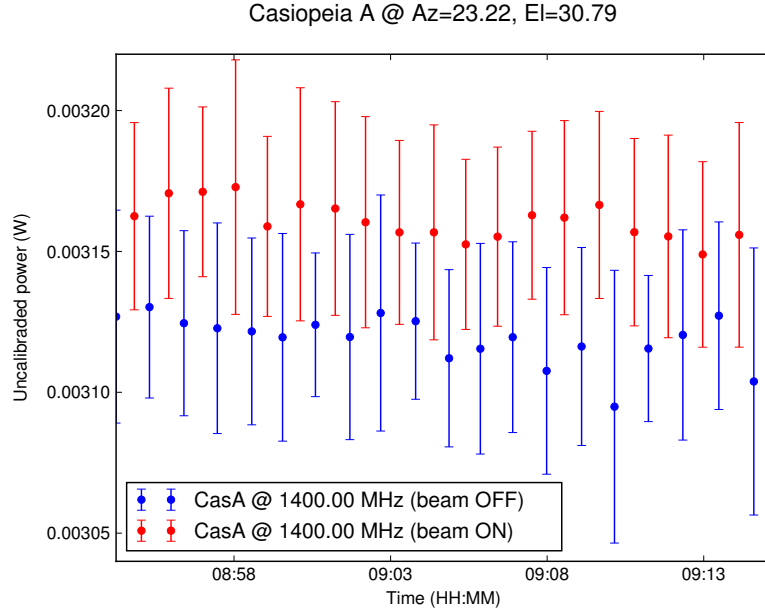


Figure 5.4: Observation of Cassiopeia A using beam-switching mode (20 switches) where the beam offset was 6° in azimuth. 40 observations with an integration time of 100 ms and a bandwidth of 200 kHz was made for each switch (on or off target).

From Figure 5.5 it is clear that the diode's signal is seen between 1100 MHz and 1800 MHz.

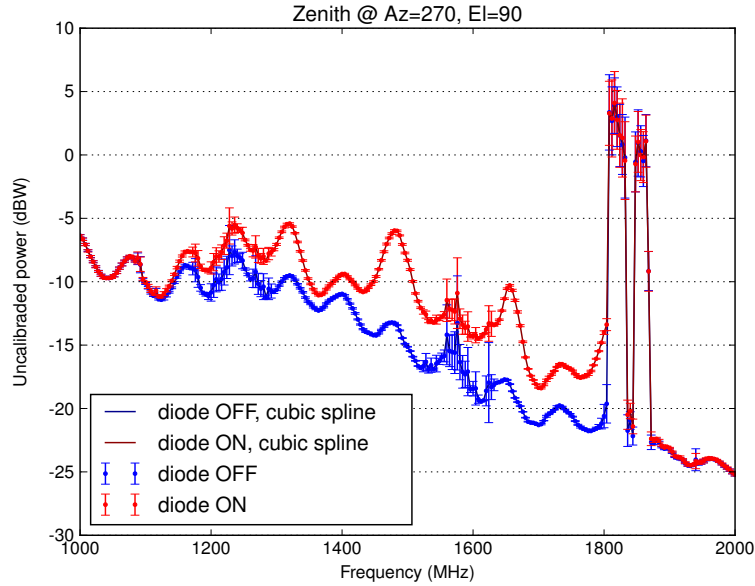
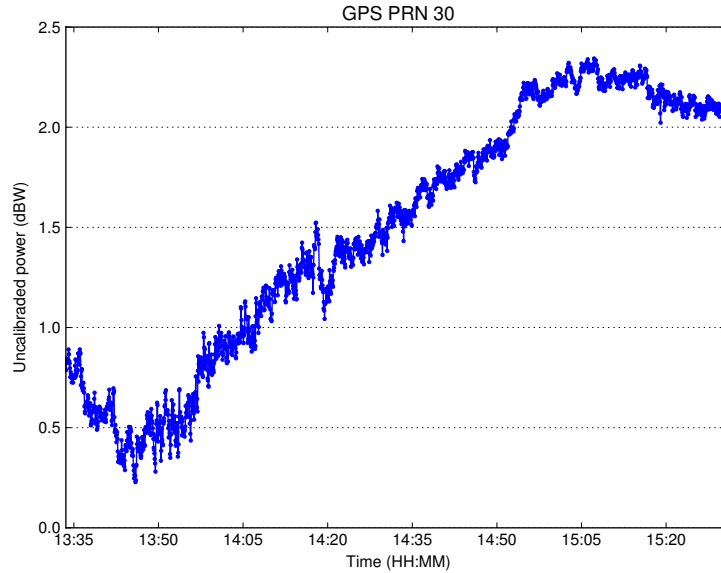


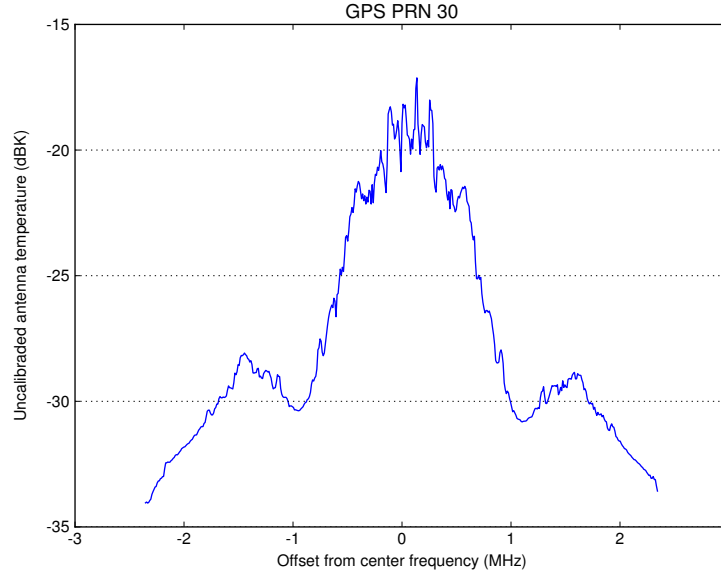
Figure 5.5: A 12-hour observation at zenith (telescope points at 90° in elevation) using frequencies 1000 MHz through 2000 MHz in steps of 4 MHz and an integration time of 3 seconds with a bandwidth of 5 MHz.

5.4.3 GNSS satellites

The time series of a 2-hour measurement of GPS PRN 30 satellite as described in Section 4.5.3 is shown in Figure 5.6a. It is clear from Figure 5.6a that the variations are small (about 0.2-0.3 dB) and that the received power decreases with elevation angle.



(a) Time series of a 2-hour measurement of GPS PRN 30 satellite.



(b) An example of the signal spectrum of PRN 30 from one of the observations carried out during the 2-hour tracking.

Figure 5.6: Time series of power measurements shown for a 2-hour measurement of GPS PRN 30 satellite at L1 for an integration time of 3 s using signal mode (top) and an example of the recorded spectrum for one of the measurements (bottom).

Chapter 6

Conclusions and Outlook

While the original goal (to calibrate SALSA) has not been met, the systems have been repaired and improved on a lower level (e.g. telescope hardware and its motion control software), which I believe will allow other people to use SALSA for new types of observations, and with more accurate and reliable results. One could also argue that one part of calibrating the systems is to make sure that the telescopes are pointing correctly and that the observation results are reliable.

Many decisions have been made during the project, many of which have been to either go deeper into the system and fix the issues, or find a way to bypass the issue (by e.g. resetting the telescope every 5 minutes) and move on with the development of the project. In each of these situations I chose to go deeper into the system and attempt to solve the problem.

As I studied the lower levels of the system in greater detail, I discovered even more issues in many cases. If I had chosen to find a way to bypass these issues I would most likely have come to the conclusion that the obtained results can not be used for satellite tracking. An example of such a scenario is that after fixing the pointing drift I discovered that Vale had a noisy front-end (a measurement precision of about 2 dB when using an integration time of 5 seconds). If I had not taken the time to repair Brage I would not have discovered that Brage did not have such issues. The measurement precision when observing a satellite with Brage is typically less than 0.2 dB.

The ultimate goal of this project is to provide the scientific community with the transmit power and antenna gain pattern of GNSS satellites in a standardized format (e.g. SINEX). Due to the complexity of this task however, this goal has not been met in this project.

6.1 Calibration

Since no calibrations have been done the obvious improvement is to do the actual calibration. The results from observations of Cassiopeia A using position switching can be used to find out how much power is received by subtracting the signal when

not observing Cassiopeia A from the signal when observing, which yields the contribution from Cassiopeia A. Since the flux from Cassiopeia A is known from the model (see Section 2.4) one can compute the antenna temperature that the SALSA telescope should have while observing Cassiopeia A. This can be used in order to determine the calibration factor that converts the measured voltage to proper units of either Kelvin or Watt.

Another observation that could be made is to alternate between placing a hot ($\sim 270\text{ K}$) and a cold (liquid nitrogen 77 K) load right under the antenna horn. Observing a hot and a cold load with known temperatures can be used similarly as in the case of observations of Cassiopeia A, in the sense that a known temperature difference can be used in order to determine the calibration factor.

In addition to calibrating the system it is also useful to calibrate the noise diode. If it is stable and its temperature known, it can be used to quickly calculate the system noise temperature and correct for gain variations during the measurement.

The results from observing in zenith and using the noise diode at many frequencies and during many hours have given us information about the stability of the system and how the system gain varies with frequency.

Once a calibration procedure is created, it can be automated by making a copy of the config file and the batch script and schedule calibration observations at regular intervals.

An important feature that needs to be implemented when observing at low altitudes ($< 30^\circ$ elevation) is to create an elevation mask that suggests the minimum elevation angle that can be observed for every azimuth angle. This is necessary because the terrain and structures surrounding the SALSA telescopes (e.g. other telescopes and nearby buildings) may cover the sky at low altitudes. This problem has been indeed encountered during satellite tracking sessions.

6.2 Pathfinding

The pathfinding is one of the less important parts of the code, especially since only a few satellites will be tracked in most cases. I believe that the current C++ implementation is about as fast as it can get using the depth-first branch-and-bound algorithm. An experimental C implementation using a linked-list structure has proven to be about 10 % faster, but it would not allow us to add another node and the code would be harder to maintain. The improvement likely comes from the implementation of `rotate` that is more suited for this application. Further improvements would be to parallelize the computation or use heuristic and approximation algorithms instead. A better way to improve the performance is probably to find a better way of partitioning satellites to yield a path that is more likely to be closer to the exact solution (such as choosing the first partition to be a circle with constant distance from the telescope), or to find a third-party library that does this task better. In the future it would also be useful to have a feature that removes a satellite from the path if there is another satellite in the beam.

6.3 Software improvements

One of the most important features to add is to be able to set up an observation without having to change the order in the Python script. This will be useful to be able to create different observation procedures that can be reused. The batch measurement program should also have a GUI that allows users to customize their batch measurements. The node structure could be further improved to cover all settings of the observation. This will likely make the program more flexible.

The main program could be a front-end to the batch measurement program. It should be possible to make a copy of a configuration template and specify which configuration files to use, but this has not been implemented because of security issues.

The 200 lines and 40 columns as well as the lack of support for user-defined functions in the DMC scripting language meant that the code had to be compressed to a point where it is difficult to read. The code size constraint has also made the interface larger than it has to be. If it was possible to have more code running on the RIO then most of the telescope-related code on the SALSA host machine could be moved to the RIO. This would increase the abstraction level as the host does not need to convert between cogs and degrees or decide if it is faster to flip over the telescope when moving to new positions – it is handled by the RIO. In addition to functionalities such as resetting the pointing and checking if the telescope is stuck, the only interface to the RIO would then be to set a target position (in degrees) and get the current position of the telescope (in degrees). It is therefore worth considering to replace the RIO with a more powerful unit that does not have these limitations and can be programmed in Python or C++.

Another improvement would be to port the code base from Python 2 to Python 3 to access new language features, reduce obscure and annoying bugs and improve the performance.

6.4 Hardware improvements

A better cable managing solution at the telescope appears necessary. Cables that are stretched and bent are likely to be damaged, which has happened a few times. One solution could be to replace the current connector with a slip ring¹. It would also be beneficial to replace the relative encoder with an absolute encoder. This would guarantee that the position counting is correct. Vale should also have its front-end (monopole and/or LNA close to the horn) replaced. In addition, a noise diode should also be added.

Since there have been problems with broken conductors in the cable running from the control room to the telescopes, all outdoor cables should be replaced in the near future to ensure smooth operation. These cables should preferably be more suited to the outdoor environment.

The RIO should probably be replaced with a more modern and powerful system, such as a Raspberry PI² or equivalent.

¹https://en.wikipedia.org/wiki/Slip_ring

²<https://www.raspberrypi.org/>

Bibliography

- [1] “GNSS satellite transmit power and its impact on orbit determination.” Steigenberger, P., Thoelet, S. & Montenbruck, O. J Geod (2018) 92: 609. <https://doi.org/10.1007/s00190-017-1082-2>
- [2] “Enhanced solar radiation pressure modeling for Galileo satellites.” Montenbruck, O., Steigenberger, P. & Hugentobler, U. J Geod (2015) 89: 283. <https://doi.org/10.1007/s00190-014-0774-0>
- [3] “SALSA web page.” Internet:<https://vale.oso.chalmers.se/salsa> [2018-12-12]
- [4] “USRP N210.” Internet: <https://www.ettus.com/product/details/UN210-KIT> [2018-12-07]
- [5] “DBSRX2 800-2300 MHz Rx.” Internet: <https://www.ettus.com/product/details/DBSRX2> [2018-12-15]
- [6] “RIO-4720x.” Internet: <http://www.galil.com/plcs/remote-io/rio-4720x> [2018-12-07]
- [7] “Johnson–Nyquist noise.” Internet: https://en.wikipedia.org/wiki/Johnson%E2%80%93Nyquist_noise, 2018-10-26 [2018-12-13]
- [8] “Astronomical radio source.” Internet: https://en.wikipedia.org/wiki/Astronomical_radio_source, 2018-10-27 [2018-12-13]
- [9] “Cassiopeia A” Internet: https://en.wikipedia.org/wiki/Cassiopeia_A, 2018-09-10 [2018-12-13]
- [10] Baars J, Genzel R, Pauliny-Toth I, Witzel A. (1977, 09). “The absolute spectrum of CAS A – An accurate flux density scale and a set of secondary calibrators.” *Astronomy and Astrophysics*. [On-line]. 61, pp. 99-106, Available: https://www.researchgate.net/publication/234366556_The_absolute_spectrum_of_CAS_A-An_accurate_flux_density_scale_and_a_set_of_secondary_calibrators [2018-12-13]
- [11] Peter J.G. Teunissen, Oliver Montenbruck “*Springer Handbook of Global Navigation Satellite Systems*.” Springer, 2017
- [12] “GPS Signal Plan.” Internet: https://gssc.esa.int/navipedia/index.php/GPS_Signal_Plan, 2014-01-10 [2018-12-13]
- [13] “Galileo Signal Plan.” Internet: https://gssc.esa.int/navipedia/index.php/GALILEO_Signal_Plan, 2014-01-10 [2018-12-13]
- [14] “GLONASS Signal Plan.” Internet: https://gssc.esa.int/navipedia/index.php/GLONASS_Signal_Plan, 2012-06-19 [2018-12-13]
- [15] “Understanding SRT Sensitivity.” Internet: <https://www.haystack.mit.edu/edu/undergrad/srt/SRT%20Projects/ActivitySRTsensitivity.html>, 10-08-1999 [2018-12-13]

- [16] “P.676 : Attenuation by atmospheric gases.” Internet: <https://www.itu.int/rec/R-REC-P.676/>, 2018-04-26 [2018-12-13]
- [17] “RIO Firmware Command Reference.” Internet: <http://www.galil.com/download/comref/com47xxx/doc.pdf>, 09/25/18 [2018-12-07]
- [18] “GalilTools.” Internet: <http://www.galil.com/downloads/software/galiltools> [2018-12-07]
- [19] Alan E.E. Rogers. “Position switching on the moon and other weak continuum sources.” Internet: <https://www.haystack.mit.edu/edu/undergrad/srt/SRT%20Memos/005.pdf> [2018-12-05]