# 1 Abbreviations

| | |
|---|---|
| **ADC** | Analogue-to-Digital Converter |
| **DAC** | Digital-to-Analogue Converter |
| **BPSK** | Binary Phase Shift Keying |
| **CDMA** | Code Division Multiple Access |
| **COTS** | Commercial Off-The-Shelf |
| **DOP** | Dilution of Precision |
| **ECEF** | Earth-Centered, Earth-Fixed |
| **ECI** | Earth-Centered, Inertial |
| **C/A** | Coarse Acquisition |
| **GDB** | GNU Debugger |
| **GLONASS** | GLObalnaya NAvigatsionnaya Sputnikovaya Sistema |
| **GNSS** | Global Navigation Satellite System |
| **GPS** | Global Positioning System |
| **HOW** | Handover Word |
| **ID** | Identifier |
| **IMU** | Inertial Measurement Unit |
| **I/Q** | In-phase, Quadrature |
| **MB** | Megabyte |
| **MTCR** | Missile Technology Control Regime |
| **PRN** | Pseudo-Random Number |
| **RAM** | Random Access Memory |
| **RF** | Radio Frequency |
| **RINEX** | Receiver Independent Exchange Format |
| **RTKLIB** | Real-Time Kinematic Library |
| **SDR** | Software-Defined Radio |
| **TLM** | Telemetry Word |
| **TOW** | Time of Week |
| **UK** | United Kingdom |
| **US** | United States |
| **UTC** | Coordinated Universal Time |
| **WN** | Week Number |
| **XOR** | Exclusive OR |
| **MSB** | Most Significant Bit |

# 2 Introduction

Knowledge of a roving platform's attitude (orientation) in space is beneficial, if not required, in several applications, including aerospace navigation, autonomous vehicle control, robotics, and geospatial mapping. Attitude determination can be achieved via a vast array of techniques, including the subject of this project: using GNSS signals.

A GNSS consists of a constellation of artificial satellites orbiting Earth, supported by a ground segment. Their primary role is to provide navigation data to receivers on the surface, typically via satellite-to-user radio link, so that they might establish their position. Perhaps the most well-known of these is America's Global Positioning System (GPS). The other globally operational systems are GLObalnaya NAvigatsionnaya Sputnikovaya Sistema (GLONASS) (Russian), BeiDou (Chinese) and Galileo (European).

The primary advantages of GNSS-based attitude determination systems over more traditional approaches (such as inertial navigation systems) are:

- **Zero Drift**: Many attitude determination systems rely on an integrated measurement to determine the orientation. This results in an error accumulation over time which must be reset for accurate readings. GNSS systems do not suffer from this issue as they are always directly referenced to the received Radio Frequency (RF) signals.
- **Global Coverage**: GNSS-based systems provide global coverage, allowing attitude determination to be performed anywhere on Earth, unlike regional limitations associated with some traditional approaches.
- **Magnetic Immunity**: Magnetic navigation techniques (e.g. compasses), struggle to give reliable readings in the presence of large metallic structures. This is not an issue for GNSS techniques.

Despite their promise, GNSS-based attitude determination systems, specifically those utilizing carrier phases, are not well represented in the marketplace. A suggested explanation is the until-recent high cost of the hardware required to make the enabling measurements (such as GNSS carrier phase). To address this absence, this project seeks to develop a low-cost system using COTS GNSS modules which could determine the attitude of a receiver array in roll, pitch, and yaw axes.

The three components identified as required to meet the project's goal were a multi-module simulator and data pipeline along with the final attitude determination tool. A diagram demonstrating the theorized inter-operation of these components is provided in Figure 1. The rationale for each component is set out in its associated section of this report.

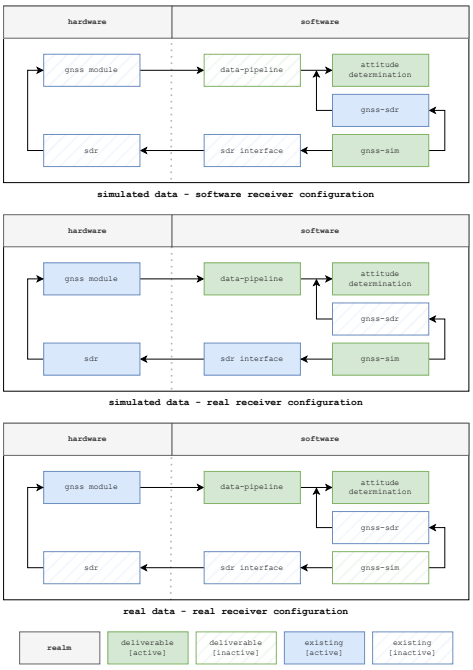


Figure 1: Illustration of how the simulator (`gnss-sim`) and data pipeline would be used in different testing configurations for the attitude determination tool. See base of figure for key.

# 3 GNSS Primer

Several books can, and have [1] [2], been written on the intricacies of each GNSS system. What follows is a lean version to furnish the reader with a sufficient understanding to follow the activities discussed here.

All operational GNSS systems operate on fundamentally the same principles. They consist of a "space segment" (satellites) and a "control segment" (tracking and control stations on the ground). The satellites orbit the earth in several orbital planes at altitudes of approximately 20,000 km, forming a "constellation". Existing constellation sizes range from 24 to 30 satellites.

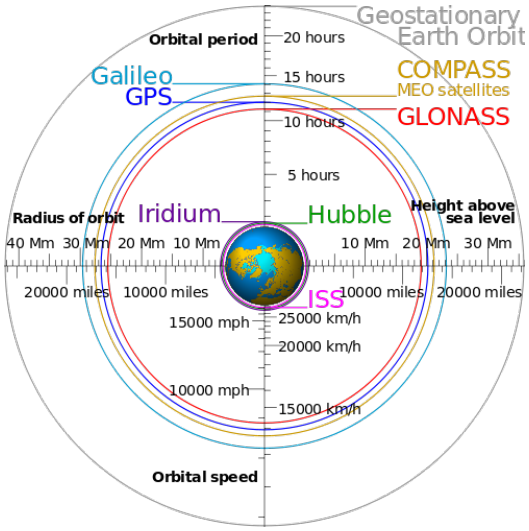


Figure 2: Diagram showing relative satellite orbits around Earth [3].

Figure 3: The approximate geometry of the GPS constellation [4].

Each satellite carries a high stability frequency reference ("clock"), L-band RF transmitter(s), antennas, thrusters for minor orbital corrections, solar panels, and control electronics. The satellite orbital data ("ephemerides") and timing characteristics are determined by the control segment and communicated to the satellites, which re-broadcast these, along with timing and other miscellaneous information for all to hear on the ground in the form of "navigation messages". Each satellite broadcasts it's unique navigation message continuously as it journeys on its orbit. The navigation message contents are updated periodically as new orbital parameters become available, typically every few hours.

Upon reception of at least four of these satellite broadcasts, GNSS receivers on the ground can perform various calculations, supplemented by their understanding of the constellation's dynamics, to determine their own position with reasonable accuracy.

The "various calculations" alluded to mostly hinge on pseudorange and carrier phase measurements. Pseudoranges are determined simply by taking the product of the free space speed of light and the difference in transmitted and received navigation message times to find the approximate line of sight distance between each satellite and the receiver. Carrier-phase tracking supplements the information explicitly provided in received GNSS signals with measurements of said signals' RF carrier phase to improve precision.

# References


[1] H. L. B. Hofmann-Wellenhof and J. Collins, *Global Positioning System: Theory and Practice*, 2005.

[2] G. Xu, *GPS: Theory, Algorithms and Applications*, 2007.

[3] [Online]. Available: https://en.wikipedia.org/wiki/File:Comparison_satellite_navigation_orbits.svg

[4] J. Van Sickle, *GPS and GNSS for Land Surveyors*, 5th ed. CRC Press, 2023. [Online]. Available: https://doi.org/10.1201/9781003405238

# 4 Multi-module Simulator – Toby

## 4.1 Introduction

As stated, the project's goal is to implement an attitude determination system exclusively using GNSS measurements. To assist the development effort and to assess the effectiveness of the final system, accurate ground truth data where the attitude is known is required. This can either be collected from real-world measurements or generated artificially. Manual data collection is encumbered by several key challenges:

- **Required Acquisition of Ground Truth Motion Data**: Knowledge of the authentic attitude at the instant of GNSS data acquisition is essential. Existing works often accomplish this using real-time optical and Inertial Measurement Unit (IMU) data capture [5] [6]. Such practices impose an additional workload to implement which is likely to exceed the capacity available to the project.
- **Spatial Sensitivity**: Wherein long duration GNSS data is gathered to access the system's temporal performance in a specific location, the ensuing challenge arises from the inherent spatial differences. The system's performance in one location may not be representative of its performance in other locations, at varying altitudes, and under disparate relative speeds and directions of motion.
- **Temporal Sensitivity**: The signals required for GNSS receiver operation undergo perpetual flux as their broadcasting satellites traverse the receiving antenna's field of view. Discerning the specific influences contributing to observed errors becomes challenging. Furthermore, this temporal instability makes it impossible to answer questions about the system's functionality across diverse temporal and environmental conditions without real-time exposure to those conditions e.g. "might the system struggle with the weekly GPS clock roll-over?"
- **Signal Unavailability Indoors**: This constraint mandates the collection of data in outdoor environments, even during adverse weather conditions. Requisite outdoor data collection imposes logistical inconveniences, potential demoralization for field operatives, and risk of equipment damage if not adequately mitigated. The mitigation efforts, in turn, necessitate additional investments of time and cost.

By resorting to GNSS simulation to provide test data during development, the discussed impediments can be mitigated; attitude is known at all simulation time steps, simulations can be performed for any duration and at any imagined time and starting location. These benefits drove the decision to utilize such simulation methodologies to support the attitude determination system development, ensuring it is robust to changes in environment.

## 4.2 Simulator Development Justification

Below is an illumination of the typical received GNSS signal power, followed by a presentation of the underlying factors influencing spatial and temporal GNSS sensitivities. It is contended that these insights underscore a key premise of the work; that the challenge of managing real-world GNSS signals complicates the confident validation of systems that rely on them.

### 4.2.1 GNSS Signal Strength in Context

To support further discussion, it is helpful for the reader to appreciate the exceedingly limited signal power typically available to a GNSS receiver's front end by means of comparison to the RF noise floor.

The noise floor's lower-bound for a frequency range can be calculated using the Johnson–Nyquist equation [7]:

$$P_{dBm} = 10\log_{10}(\frac{k_B T \Delta f}{10^{-3}})$$

where $P_{dBm}$ is the noise power (dBm), $k_B$ is the Boltzmann constant ($J \cdot K^{-1}$) and $\Delta f$ is the bandwidth (Hz).

Earth's typical surface temperature is approximately 300 K [8]. With a signal bandwidth of 2.046 MHz (twice the GPS fundamental frequency, to satisfy the Nyquist criterion), the minimum noise seen at the input to a GNSS receiver can reasonably be estimated as follows:

$$P_{dBm} = -173.8 + 10\log_{10}(\Delta f)$$

$$P_{dBm} = -173.8 + 10\log_{10}(2.046 \times 10^6)$$

$$P_{dBm} = -110.7$$

GPS, which will be taken here to adequately represent all active GNSS systems, specifies a signal power of just -130 dBm incident at Earth's surface [9]. As shown, this level is essentially 100 times lower than the anticipated noise power. While using code gain techniques makes reception possible, the difficulties introduced by the reception of low power signals remain.

In defence of this setup, the decision to use remarkably low incident signal powers was made during GPS's design to strike a balance between the ease of reception and the satellite power budget required for signal transmission [10]. The GPS architects were only able to achieve 800 W of solar generation for the first generations of NAVSTAR (GPS) satellites [11], only 50 - 76 W of which could be afforded to signal transmission [12] [13]. Other GNSS's followed suit, presumably with similar reasoning.

For many GNSSs the issue of low transmission power is compounded by the designers' decision to use a spread spectrum encoding scheme which further reduces the transmitted power at each frequency within the signal's bandwidth. In the case of GPS, this decision was taken to make the system more robust against interference and deliberate signal jamming attempts [10], especially relevant considering the development of GPS was driven by US military interests.

### 4.2.2 Sources of GNSS Positioning Sensitivity

The conducted research identified the following as the primary factors influencing GNSS positioning sensitivity:

- Multipath Effects
- Dilution of Precision
- Anthropogenic Effects
- Regulatory Limits

While GNSS signal attenuation is predominantly driven by the vast distance between the satellites and ground receivers [13], weather, foliage, and built-up structures all contribute to further signal attenuation by introduction of multipath fading [10] [14], where copies of the original signal interfere with one another resulting in distortion at the receiver. The effect is illustrated in Figure 4. Multi-path fading is difficult to model analytically as it is subject to a receiver's environment, though simulation techniques do exist [15]. By subjecting the attitude determination system to a range of simulated multipath scenarios, confidence in its performance in the wild can be assured.


Figure 4: GNSS module manufacturer material highlighting the causes of multi-path fading [16].

Satellite elevation angles, their distribution across the sky, and geometric relationships with a receiver, all of which are constantly changing even whilst the receiver is stationary, influence the accuracy of measured GNSS receiver position [17]. This effect is referred to as Dilution of Precision (DOP). A theoretical illustration is provided in Figure 5 though the reader may find Figure 6 a more intuitive rendering of the phenomenon. DOP

is expressed using several DOP values which are derived from a reasonably involved process of linear algebra beyond the scope of this work, suffice to say they are well understood for a stationary receiver [18]. How DOP values evolve in time can be modelled effectively however the impact of receiver motion, especially across the multiple receivers used in the envisioned system, is difficult to access without real-world or simulated data and is highly dependent on receiver location and time of day.

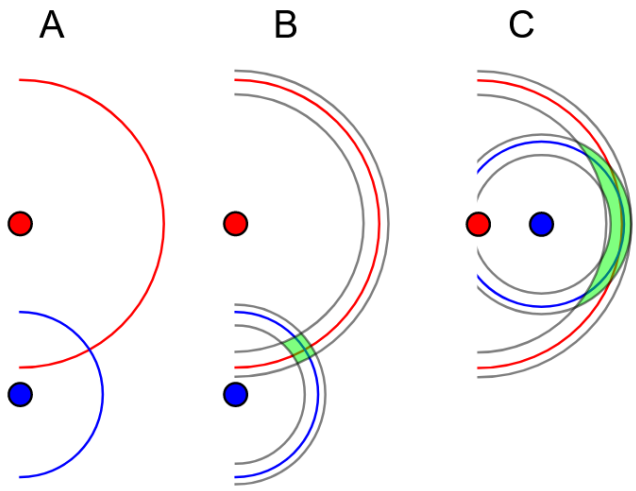


Figure 5: Dilution of precision 2D example shown. A: Measuring exact distances, depicted as intersecting circles, between two points, an observer can determine their position as the intersection of the two circles. B: Introducing error bounds to the original figure results in a region of uncertainty (green). C: Despite consistent measurement error, the region of uncertainty can increase significantly under certain point arrangements [19].

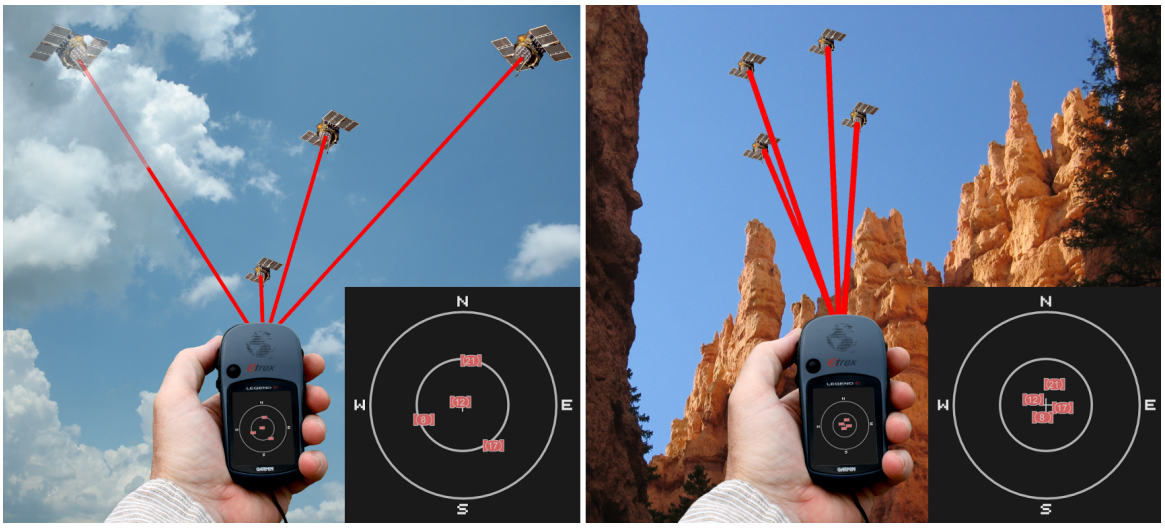


Figure 6: An illustration showing how satellites can be clustered in the sky, resulting in precision dilution. Left, high DOP configuration. Right, low DOP configuration [19].

Selective Availability, the mechanism by which the civilian GPS signals were initially randomly manipulated to reduce accuracy, is no longer active for GPS [20] or any other GNSS system, however deliberate distortions and interruptions to GNSS signals do still occur in some regions of the globe. For instance, the UK's Ministry of Defence carries out occasional GPS blocking exercises which do spill out into civilian areas [21]. Similar unsanctioned jamming activities are also not uncommon [22]. While ensuring either immunity or fail safe conditions in the context of the aforementioned scenarios may be unnecessary for most applications, some uses for the theorized attitude determination system may require a demonstrated resilience to these extreme conditions i.e. defence technologies which are regularly subject to GNSS denial tactics.

Though of no consequence in most applications, the sale of GNSS receivers which continue reporting their position when traveling at speeds above 600 m/s is restricted by Technical Annex, clause 11.A.3 of the Missile Technology Control Regime (MTCR) [23]. The regime was formed by an informal understanding between 35 countries, including the UK, to limit the proliferation of technology useful in missile construction. All

commercial GNSS receivers bought or sold in the signatories' countries must implement the MTCR restrictions. Unfortunately, many GNSS receiver manufacturers go a step further and abide by the now deprecated "CoCom limits" (a similar arrangement, defunct as of 1994) which is ambiguously worded and results in undefined receiver behaviour both near and beyond the stipulated limits [23].

MTCR-related behaviour varies between commercial receiver modules and is impossible to predict analytically as it is scarcely documented. Essentially, the best way to access the performance of COTS GNSS receivers near to, above and after returning from beyond the MTCR/CoCom limits is experimentally. For real-world test scenarios involving speeds in excess of 600 m/s this is likely to be a financial and logistical challenge. As in the previous discussion about selective availability and jamming, these scenarios are likely to have most benefit for military applications however a notable civilian interest comes from amateur rocketry, a topic which the author is familiar with as a member of the Leeds University Rocketry Association. Again, simulation can step in here to expose a receiver to signals which would otherwise be difficult to acquire.

#### 4.2.3 Discussion

The causes, and magnitudes of spatial and temporal GNSS sensitivities were investigated to ensure the validity of a key premise of this work; that real-world received GNSS signals are difficult to control, making systems that depend on them difficult to validate confidently without a simulator. From the research performed, it is clear that the signals are sufficiently sensitive and variant to warrant simulation efforts as a means to assess performance across a diverse range of scenarios.

In the literature, the type of GNSS simulation discussed here is often referred to as "GNSS spoofing", wherein the objective is to trick a GNSS receiver into believing it is where it is not via the application of false RF signals, mimicking those that would plausibly be encountered during the scenario being simulated.

It is important to note that whilst there is nothing in UK law concerning the generation of bogus GNSS signals, their emission in the GNSS-dedicated regions of the RF spectrum is illegal in most all countries on account of the indiscriminate nature of such an action and the dependence of much critical infrastructure on these signals. Accordingly, this work does not advocate for such activities and aims to provide a solution which confines the signals to a coaxial cable fed directly to a commercial receiver's antenna input.

Given the apparent utility of GNSS simulation in product development it should come as no surprise that there already exist several commercial GNSS simulator offerings capable of full GNSS spoofing [24] [25] [26] [27] [28], however the "most affordable GNSS RF recorder and replayer available" starts at over £4,000 [29].

The available products are far outside this project's budget and, perhaps more importantly, the budgets of other small players. The resulting limitation of education and research in this field to well financed organisations prompted a search for low-cost solutions.

It is worth including that even if the project had the budget for such equipment, it would still be preferable to have access to low-cost simulation tools. As all costs incurred during development would necessarily be passed onto the customer, expensive equipment is not conducive to the development of a "low-cost attitude determination system".

A not-unreasonable free GNSS simulation package was found in the form of `gps-sdr-sim` [30]. The software appears to be the only freely available contender to the commercial alternatives and the work of a single individual. While the `gps-sdr-sim` code is obtuse at times, it is capable of generating In-phase, Quadrature (I/Q) data files which can be fed to an Software-Defined Radio (SDR) which generates the appropriate RF signals as if received by an antenna on the ground. Several demonstrations are available online, using `gps-sdr-sim` signals to spoof commercial GNSS modules [31] [32].

Despite its strengths, there are several shortcomings of `gps-sdr-sim` which prevent its immediate application in this instance:

- **Limited to L1 GPS signals**: The existing literature suggests observations from multiple carrier frequencies can be used to significantly augment carrier phase tracking techniques and thus attitude determination abilities of a GNSS-dependant system such the one this report focuses on [5] [6]. An additional benefit of a system that uses multiple carrier frequencies is the on-receiver ionospheric correction this enables [33]. GPS provides multiple signals at different frequencies for this purpose including the L1 (1575 MHz) and L2C (1227 MHz) signals [34].
- **Lack of real-time functionality**: 1.023 MHz is the typical code frequency which GNSSs modulate their navigation data on their carrier signals with [33] [35]. This requires a minimum simulated sample rate 2.043 MSps to prevent aliasing at the receiver. Using a typical interleaved 16-bit I/Q data format [36], each second of generated baseband RF data would require a minimum of 61 MB to store. For reference,

at least one full "frame" of data must typically be transmitted in the GPS context before a receiver is furnished with sufficient information to begin determining its position. This takes 30 seconds, requiring practical simulation files be a minimum of 1.8 GB. This burden quickly becomes unreasonable even for medium duration simulations. Real-time support would solve this issue.

- **Lack of multi-module support**: `gps-sdr-sim` only enables the simulation of a single receiver. This could be overcome by running the tool multiple times with marginally offset input motion sequences, one for each receiver on the simulated platform. The resulting file sizes and potential issues synchronising the outputs make this option unappealing.
- **Poor code quality and documentation**: The documentation is primarily short YouTube videos and brief text files supplemented by the code's limited and nondescript comments. This makes it challenging to extend, debug or even operate the tool. Secondary to the issues it presents for this project, the difficulty associated with interpreting the code results in a missed opportunity to educate many others on how to build such digital radio systems.

While other works have had success modifying `gps-sdr-sim` to add features such as real-time streaming [37], this is rarely made available and is likely to be precariously strung in due to the brittle architecture of the software.

To overcome `gps-sdr-sim`'s limitations, it was decided to create an alternative using the original project as a reference implementation while adding the features required for this project. The agreed requirements were:

- It shall generate time synchronized spoofed GNSS module data for at least three modules.
- Output data format shall be one that can be used, by any means, to synthesis input RF signals compatible with the NEO-M8T GNSS module.
- Must be able to generate data for a platform taking any path at any attitude on earth.
- Real-time data streaming shall be implemented in addition to simple binary I/Q data file generation.
- At least one GNSS shall be supported with a clear upgrade path for adding future constellations.
- The code shall be of high quality so that it may be used as a teaching device.

## 4.3 Verification Tools

Prior to working on the simulator, verification techniques were established for software evaluation and debugging. The anticipated final use case, an SDR interfacing with a commercial GNSS receiver and receiving I/Q data from the simulator, was deemed impractical for initial verification on account of:

- **Anti-spoofing detection**: Many commercial GNSS receivers, including the project's available ublox NEO-M8Ts, incorporate anti-spoofing features whose details are restricted for security reasons [38]. Disabling these features is not possible [39], and efforts to circumvent them would have diverted attention from the project objectives.
- **Opaque receivers**: Proprietary aspects of commercial receivers obscure internal operations. This would have seriously hindered diagnosis and resolution of issues encountered during simulator development.
- **Data-pipeline dependence**: Relying on debug information from off-the-shelf GNSS modules necessitates readiness of the data-pipeline before any testing can commence. Decoupling this aspect from other project components facilitated independent progression of team members.

Other team members were responsible for attitude determination and data-pipeline portions of the project. As real-world testing of the COTS GNSS modules requires the data-pipeline, this aspect is not covered here.

The two main verification tools used during development were GNU Radio and GNSS-SDR. GNU Radio scripts were used to identify issues with early Binary Phase Shift Keying (BPSK) modulation attempts while GNSS-SDR was used to evaluate almost every other facet of the simulator. An introduction to the two tools and how they were used in the context of this project is provided below. They should be especially useful to any student picking up this task in future.

GNU Radio is an open-source software development toolkit offering signal processing blocks for the simulation and signal processing steps involved in SDR applications. While it can interact with SDR hardware in real-time, performing modulation, demodulation, filtering, interleaving and other processes, it can also be used as a simulation environment for experimentation with RF communications techniques. GNU Radio scripts can be written in C++, Python or using the provided graphical node interface.

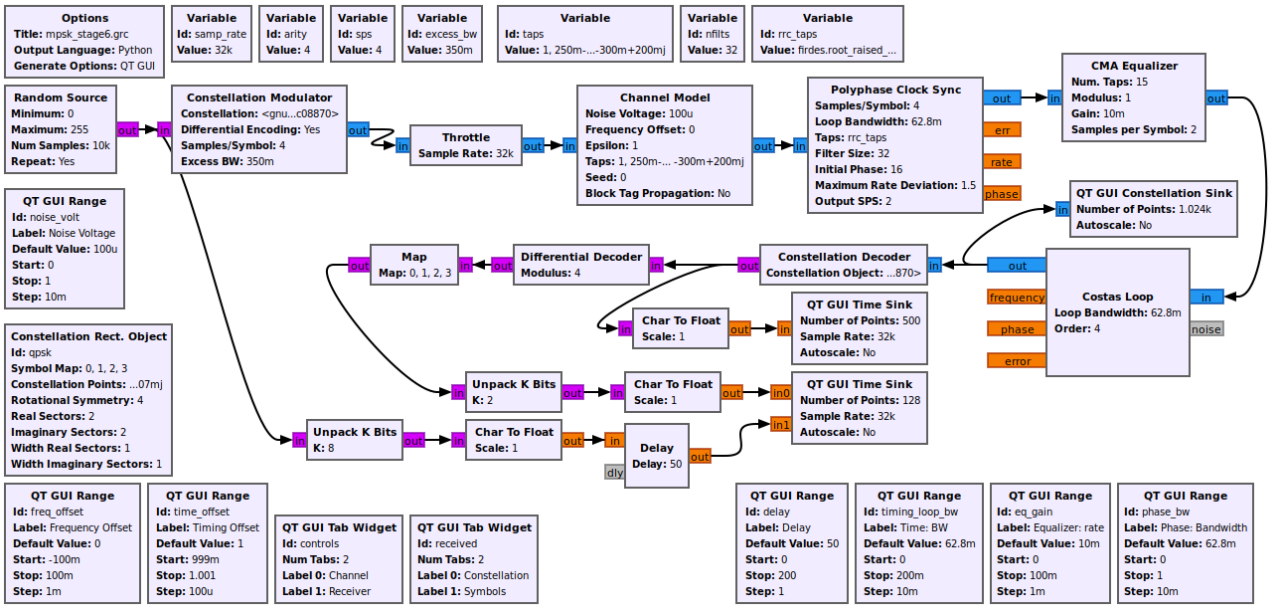


Figure 7: Example GNU Radio script in the graphical editor [40].

GNSS-SDR [41] is an open-source project built atop GNU Radio which implements a full GNSS receiver in C++, using a suitable commercial SDR as an RF front end. It supports the various modes of all operational GNSS constellations and can: work with real-time input streams and generate synchronised output streams, debug each component in the signal tracking chain and determine the receiver's position using this information. GNSS-SDR also accepts synthetic I/Q file(s)/stream(s) as input without requiring the use of SDR hardware.

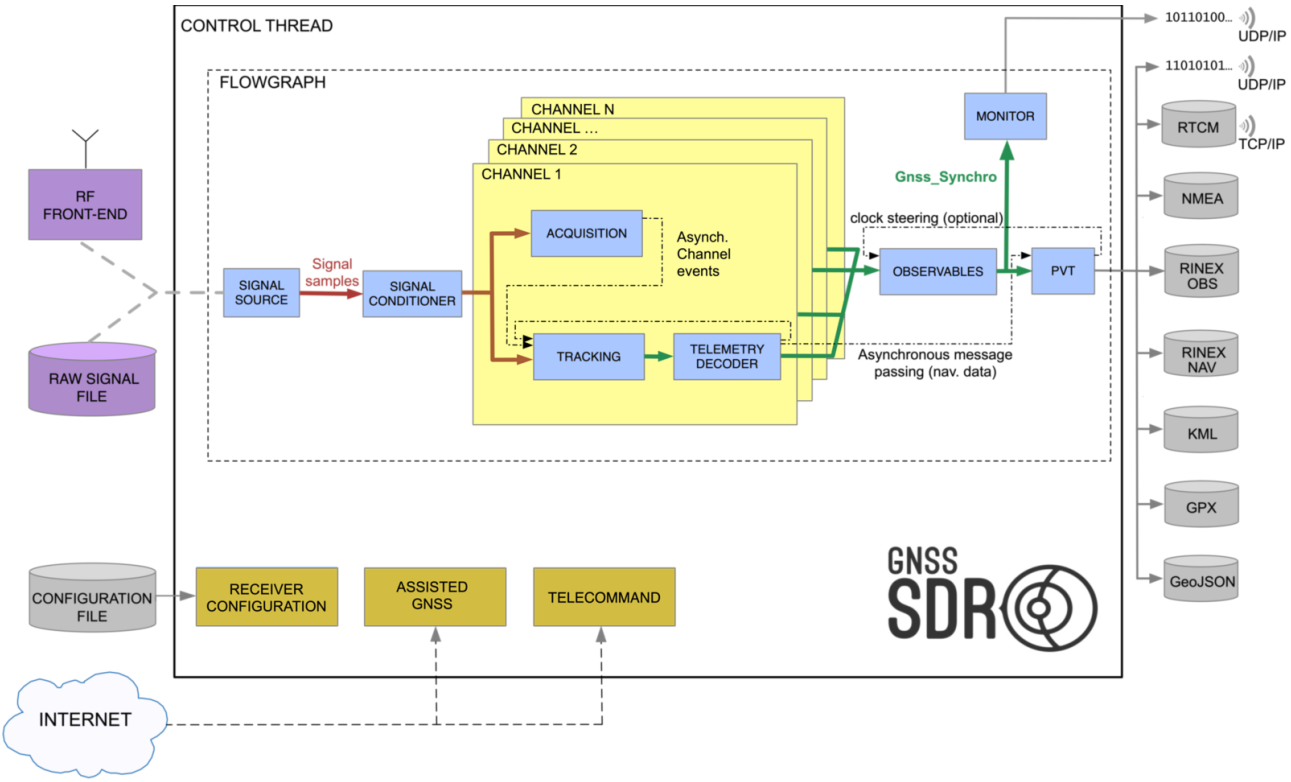


Figure 8: GNSS-SDR internals block-diagram [41].

Figure 9 depicts the simulator's use in operational scenarios compared to how it was evaluated using GNSS-SDR.

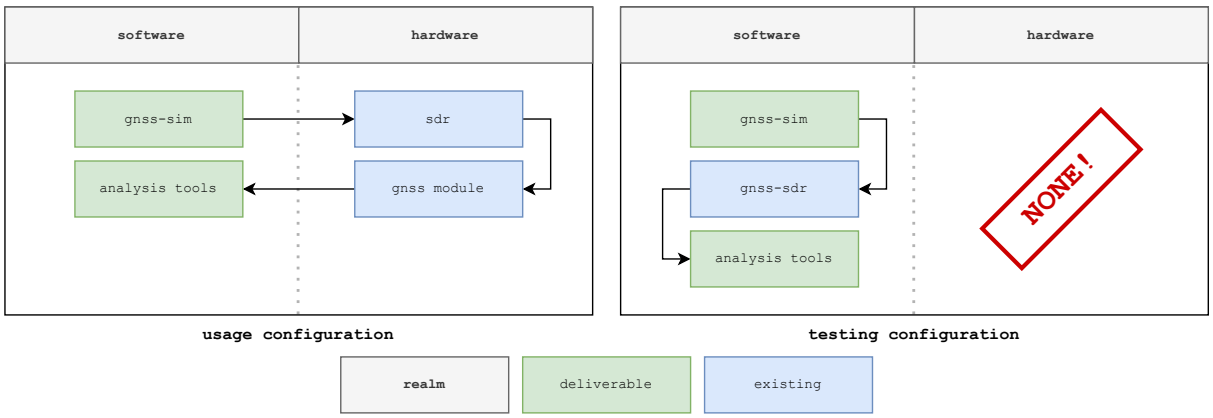


Figure 9: Block diagram showing differences between how the simulator would be used in service versus testing. See base of figure for key.

## 4.4 Simulator Implementation

The `gnss-sim` GNSS simulator has been built, though not all project objectives were met due to challenges in pseudorange calculations. It operates from the command line, generating GPS L1 navigation messages, simulating satellite positions, and producing I/Q data representative of the signals received at a GNSS front end. While decoding the generated signals using GNSS-SDR has been achieved, inaccuracies in estimated receiver positions are observed, stemming from the aforementioned pseudorange calculation issues.

`gnss-sim` offers both real-time and binary file output capabilities. Presently, it only supports a fixed receiver position, though integrating variable positions should be straightforward. Similarly, it currently supports a single module but can be expanded with ease. While only able to support GPS L1 signals, the code is structured for inclusion of other constellations. The GPS L1 signal was chosen as the first to support due to its ubiquitous documentation with IS-GPS-200 (the GPS Interface Specification) [42] serving as a vital resource during this work. The `gnss-sim` internals should be comprehensible to students with a foundational understanding of digital communications, especially when aided by this report and its accompanying sources. The code repository can be accessed at https://github.com/TobyThomson/gnss-sim.

Use of a high performance programming language was necessary due to the computational demands of the task. C was chosen due to the author's familiarity with the language, it's execution speed and the existing reference implementation in the form of `gps-sdr-sim`. MATLAB was dismissed due to its proprietary nature.

Opting for C also provided an additional advantage: the ability to utilize the open-source Real-Time Kinematic Library (RTKLIB) library [43]. RTKLIB provides a suite of data structures functions and programs tailored for analyzing and manipulating raw GNSS signals, specifically designed for developers of systems such as this. Within `gnss-sim`, RTKLIB is lent on for Receiver Independent Exchange Format (RINEX) file parsing, satellite position estimation, pseudorange calculations, and coordinate system transformations. The library's operations and methodologies are well documented in its manual and various other online resources. The decision not to reimplement RTKLIB's functionality was taken due the basic numerical nature of much of its operations, offering limited educational value to the author and negligible benefits to the wider community. This view is supported by GNSS-SDR's also using RTKLIB.

### 4.4.1 Architecture

The `gnss-sim` repository contains source files, header files, test scripts, a Makefile and `tools/` and `libs/` directories. Presently, `tools/` solely houses the source for `trig-tables-generator.c`, a program to automate the generation of statically-typed trigonometric tables. The external sources required for compilation, such as RTKLIB and the "progressbar" library, are not bundled within the distribution. This decision was made to avoid managing a separate fork of these sources, which could deviate from their upstream counterparts.

The source code is split between three C files: `main.c`, `simulator.c` and `debug.c`. `main.c` is responsible for handling the user input and initiating appropriately configured simulation runs, `simulator.c` contains all the functions necessary to generate the I/Q data seen at the output while `debug.c` is a helper script which contains several functions to assist in debugging.

The program flow is illustrated in Figure 10. Shown is a simplified version of events to orientate the reader. Code snippets and implementation details are provided later.

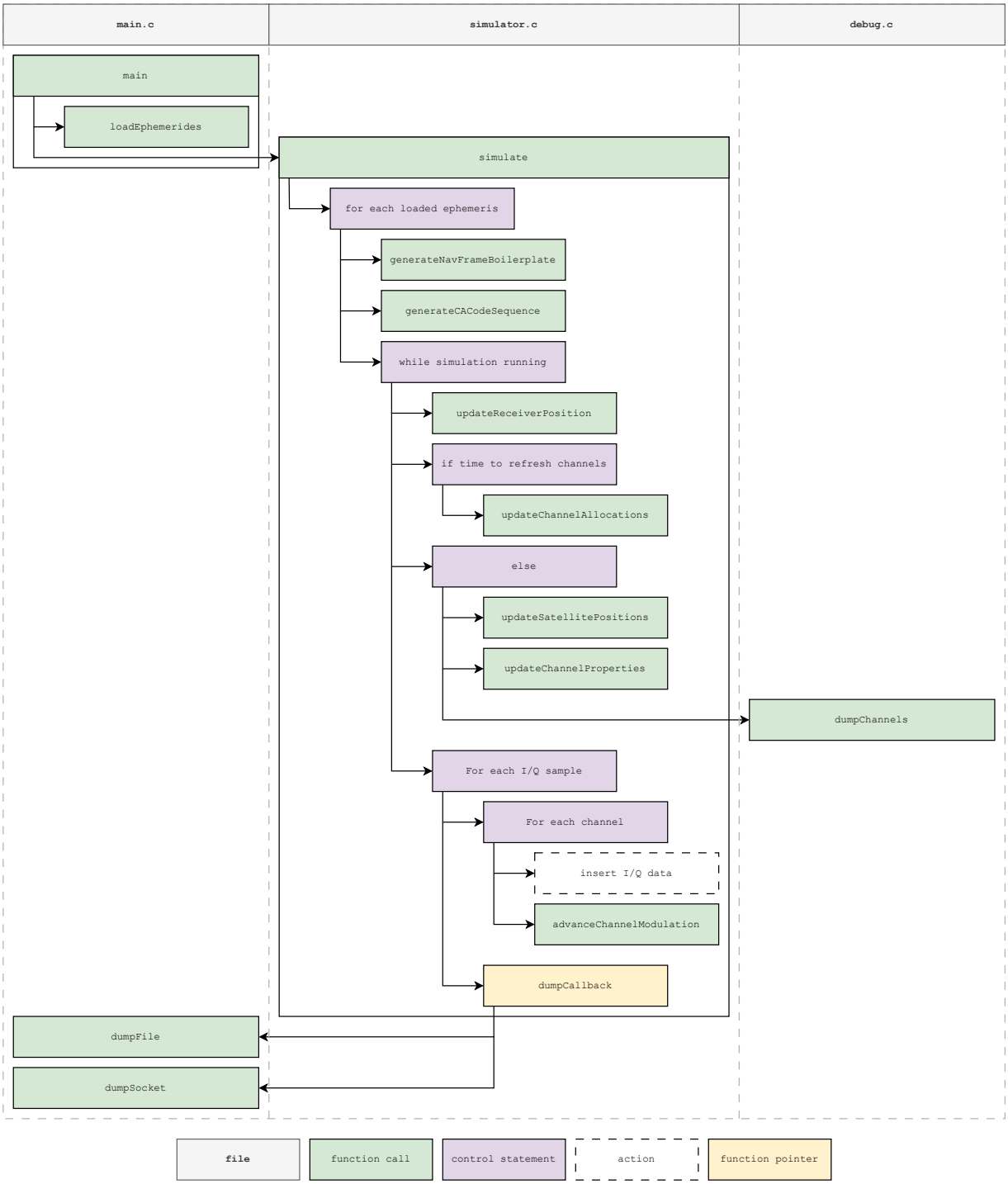


Figure 10: `gnss-sim` block diagram. Execution starts with the main function in `main.c` and progresses down the figure. See base of figure for key.

A breakdown of the `gnss-sim` repository's contents, to the first three levels, is provided below:

```
gnss-sim
├── include
│   ├── debug.h
│   ├── main.h
│   ├── simulator.h
│   └── trig-tables.h
└── libs
    └── progressbar
```

```
    │
    │   └── rtklib-2.4.3
    ├── src
    │   ├── main.c
    │   ├── simulator.c
    │   ├── debug.c
    │   └── makefile
    ├── test
    │   ├── navmessage
    │   ├── rinex-import
    │   ├── tracking
    │   └── zmq
    ├── tools
    │   └── trig-tables-generator.c
    └── README.md
```

### 4.4.2 External Interfaces

`gnss-sim` must be aware of each satellite's position within the simulated environment. This information, derived from "ephemerides", continuously adjusts under the influence of factors including solar pressure, atmospheric drag, and Earth's oblateness. Real-world satellite ephemeris updates are necessitated approximately every two hours to maintain accuracy.

For simulation accuracy, `gnss-sim` must be seeded with historic ephemerides. These are abundant online in the form of RINEX files sourced from ground stations across the globe. Leveraging RTKLIB functions for RINEX file processing facilitates the use of recent ephemerides for all satellite constellations. The RTKLIB functions were used as they support multiple RINEX file versions, an advantage over `gps-sdr-sim` which is only able to work with legacy version 2 files.

The `loadEphemerides` function in `main.c` parses RINEX files to extract the relevant information. See Listing 1. A sort function was added to make selection of ephemeris by satellited Identifier (ID) (Pseudo-Random Number (PRN)) more intuitive. When sorted in ascending order of PRN, a satellite's PRN number can now be used as as the array index. Another feature was added to deal with the repeated entries encountered in RINEX files by removing all bar the most recent of ephemerides for each satellite once they are read in.

```
28 int compareEphemerides(const void *p1, const void *p2) {
29     eph_t *q1 = (eph_t *)p1;
30     eph_t *q2 = (eph_t *)p2;
31 
32     // Compare based on satellite ID and time of ephemeris
33     if (q1->sat != q2->sat) {
34         return q1->sat - q2->sat;
35     }
36 
37     // Sort in descending order of time
38     else {
39         return q2->toe.time - q1->toe.time;
40     }
41 }
42 
43 int loadEphemerides(char* filename, eph_t* ephemerides) {
44     // Only interested in GPS ephemerides for now
45     char* opt = "-SYS=G";
46 
47     // Dummy variable to store navigation data initially
48     nav_t nav = {0};
49 
50     // Open ephemerides file
51     if (!readrnx(filename, NULL, opt, NULL, &nav, NULL)) {
52         printf("Error: Could not open ephemerides file\n");
53         return -1;
54     }
55 
56     // Sort the array based on satellite ID and time of ephemeris
57     qsort(nav.eph, nav.n, sizeof(eph_t), compareEphemerides);
58 
59     // Declare iterators
60     int i, j;
61 
62     // Iterate through the array to keep only the most recent ephemeris for each satellite ID
63     for (i = 0, j = 1; j < nav.n; j++) {
64         if (nav.eph[j].sat != nav.eph[i].sat) {
65             // Move to the next satellite ID
66             i++;
67 
68             // Keep the most recent ephemeris for this satellite ID
69             nav.eph[i] = nav.eph[j];
```

```
70          }
71      }
72
73      // Copy out of nav
74      short count = (i + 1);
75      memcpy(ephemerides, nav.eph, (count * sizeof(eph_t)));
76
77      // Display result!
78      printf("GPS EPHEMERIDES LOADED: %i\n", count);
79
80      // Uncomment for debugging
81      // Dump the 1st ephemeris entry
82      dumpEphemeride(ephemerides, 0);
83
84      return 0;
85  }
```

Listing 1: The function used to import and sort ephemerides. [main.c]

gnss-sim outputs I/Q data compatible with SDRs and GNSS-SDR, both of which require identically formatted binary I/Q data. Using the author's personal LimeSDR Mini as a reference, this data can be provided to the SDR via GNU Radio blocks from either a file source or ZeroMQ socket (part of the widely used ZeroMQ messaging framework) for real-time operation. GNSS-SDR also accepts binary files and ZeroMQ streams as inputs. This drove the decision to implement these two output methods in gnss-sim. As illustrated in Figure 10, the simulate function employs a function pointer to invoke either dumpFile (Listing 2) or dumpSocket (Listing 3) to export the generated data, depending on the argument provided.

```
19  void dumpFile(short* buffer, int length) {
20      fwrite(buffer, sizeof(buffer[0]), length, OutputFile);
21  }
```

Listing 2: The binary file callback handler. [main.c]

Real-time data transfer from gnss-sim to either SDR or GNSS-SDR is facilitated by the czmq library. An initial oversight led to only a quarter of the data transmitted by gnss-sim being received by GNSS-SDR. This was found using GNSS-SDR's ability to dump its received input to a file for inspection. The solution was to multiply the requested transmit value count by the size of each value (i.e. four bytes). Once implemented, a discrepancy still existed between the number of bytes dumped by GNSS-SDR when using the ZeroMQ input versus using the file input; approximately 900 bytes less in the ZeroMQ instance. Using hexdump and diff, the data was found to be missing from the end of the ZeroMQ dump, suspected to be caused by buffering internal to GNSS-SDR. As this only marginally affected simulation runtime, further investigation was deemed unnecessary.

```
23  void dumpSocket(short* buffer, int length) {
24      zsock_send(OutputSocket, "b", buffer, (length * sizeof(short)));
25      zclock_sleep(10);
26  }
```

Listing 3: The ZeroMQ socket callback handler. A 10 ms sleep is inserted after each dump to give the buffer time to ingest the data. [main.c]

### 4.4.3 C/A Code Generation

All active GNSSs bar GLONASS utilize Code Division Multiple Access (CDMA) techniques to manage the radio spectrum allocated to them [44]. Under CDMA, all transmission stations (satellites) transmit on the same frequency and at the same time as one another. The separation of transmissions in the receiver is enabled by XORing the data signal with a unique pseudorandom sequence of 1s and 0s ("chips") at the transmitter. See Figure 11. GPS refers to these sequences as Coarse Acquisition (C/A) codes. GPS also uses military-restricted P (Precision) codes, however these are irrelevant to the civilian signal discussed here. A C/A code's defining properties are its being indistinguishable from random noise and having a fixed repetition interval (1023 chips).

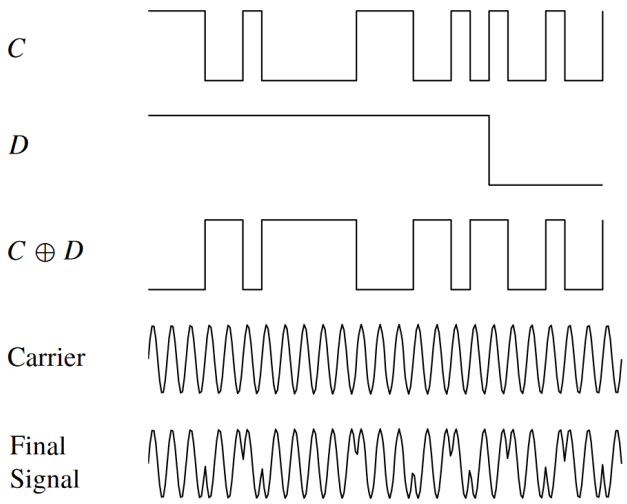


Figure 11: Typical GNSS CDMA signal built from XORing the data (D) with a pseudorandom code (C) and modulating the result onto the carrier [45].

Given all the active C/A codes are included in IS-GPS-200 (Figure 12), by correlating ("sliding") a clone of the desired satellite's C/A code against the RF backdrop, a receiver can sift a transmitted signal from the noise. The C/A code runs much much faster (1.023 MHz) than the data code (50 bit/s) which results in the extreme coding gain required to find these signals so far below the noise floor. An added bonus of this technique is the measure of phase difference between the satellite and receiver's clock that it provides. Figure 13 provides an illustration of the process.

| SV ID No. | GPS PRN Signal No. | Code Phase Selection | | Code Delay Chips | | First 10 Chips Octal* | First 12 Chips Octal |
|---|---|---|---|---|---|---|---|
| | | C/A($G2_i$)** | ($X2_i$) | C/A | P | C/A | P |
| 1 | 1 | 2 ⊕ 6 | 1 | 5 | 1 | 1440 | 4444 |
| 2 | 2 | 3 ⊕ 7 | 2 | 6 | 2 | 1620 | 4000 |
| 3 | 3 | 4 ⊕ 8 | 3 | 7 | 3 | 1710 | 4222 |
| 4 | 4 | 5 ⊕ 9 | 4 | 8 | 4 | 1744 | 4333 |
| 5 | 5 | 1 ⊕ 9 | 5 | 17 | 5 | 1133 | 4377 |
| 6 | 6 | 2 ⊕ 10 | 6 | 18 | 6 | 1455 | 4355 |
| 7 | 7 | 1 ⊕ 8 | 7 | 139 | 7 | 1131 | 4344 |
| 8 | 8 | 2 ⊕ 9 | 8 | 140 | 8 | 1454 | 4340 |
| 9 | 9 | 3 ⊕ 10 | 9 | 141 | 9 | 1626 | 4342 |
| 10 | 10 | 2 ⊕ 3 | 10 | 251 | 10 | 1504 | 4343 |
| 11 | 11 | 3 ⊕ 4 | 11 | 252 | 11 | 1642 | \| |
| 12 | 12 | 5 ⊕ 6 | 12 | 254 | 12 | 1750 | \| |
| 13 | 13 | 6 ⊕ 7 | 13 | 255 | 13 | 1764 | \| |
| 14 | 14 | 7 ⊕ 8 | 14 | 256 | 14 | 1772 | \| |
| 15 | 15 | 8 ⊕ 9 | 15 | 257 | 15 | 1775 | \| |
| 16 | 16 | 9 ⊕ 10 | 16 | 258 | 16 | 1776 | \| |
| 17 | 17 | 1 ⊕ 4 | 17 | 469 | 17 | 1156 | \| |
| 18 | 18 | 2 ⊕ 5 | 18 | 470 | 18 | 1467 | \| |
| 19 | 19 | 3 ⊕ 6 | 19 | 471 | 19 | 1633 | 4343 |

* In the octal notation for the first 10 chips of the C/A code as shown in this column, the first digit (1) represents a "1" for the first chip and the last three digits are the conventional octal representation of the remaining 9 chips. (For example, the first 10 chips of the C/A code for PRN Signal Assembly No. 1 are: 1100100000).

** The two-tap coder utilized here is only an example implementation that generates a limited set of valid C/A codes.

⊕ = "exclusive or"

NOTE #1: The code phase assignments constitute inseparable pairs, each consisting of a specific C/A and a specific P-code phase, as shown above.

Figure 12: GPS L1 C/A code reference table from [42].

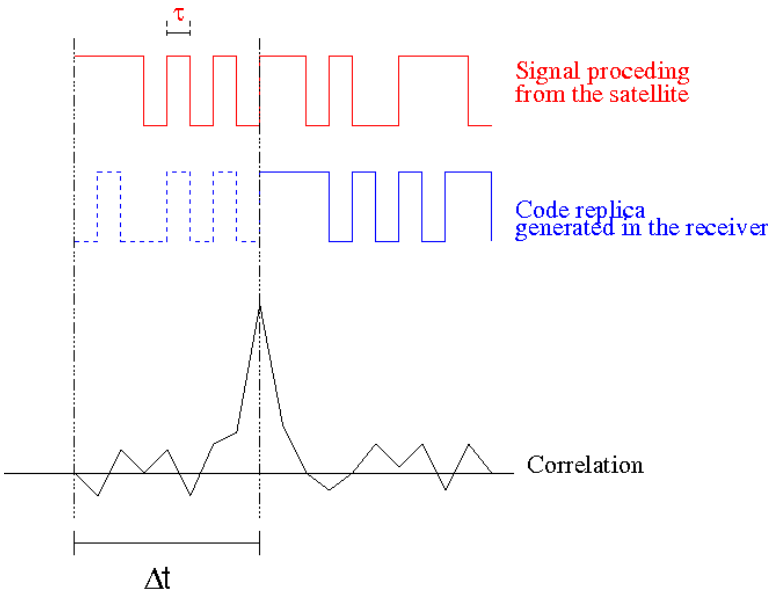


Figure 13: CDMA correlation technique visualized [46].

Each active GPS satellite is assigned a PRN (Pseudo Random Number) from 1 to 32. The number is not itself pseudorandom and instead constitutes an ID used to determine the correct C/A code sequence generation parameters from Figure 12. It is akin to a FM transmitter's channel ID. PRNs are reassigned to a new satellite during the decommissioning of a retired one.

Aboard the GPS satellites, the C/A code is generated using two tapped shift registers with one delayed from the other. The effect of this delay can also be achieved with two additional adjustable taps off the second shift register as depicted in Figure 14. While the two taps method can generate all of the valid GPS C/A codes, it is unable to generate all the 1023 pseudorandom sequences otherwise possible with this setup. For ease of extension to other constellations at a later date, the "delayed chips" method was chosen for `gnss-sim`.

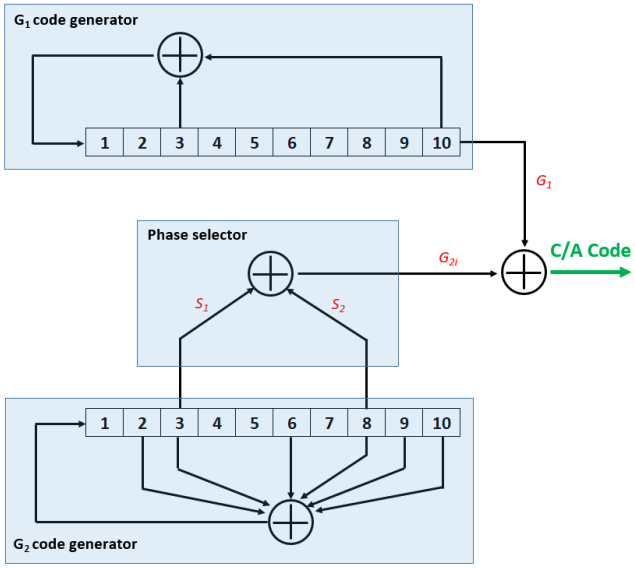


Figure 14: The shift registers used on the GPS satellites to generate the C/A codes. S1 and S2 are the optional adjustable taps if that method is to be used [47].

Listing 4 details the `generateCACodeSequence` function created to determine each simulated satellite's C/A code. Rather than running in real-time, this function is called once for each satellite from `simulate` during the simulation setup. Upon execution, the function generates the entire code sequence and saves it to an array which can be stepped through to advance the C/A chip. This reduces the processor overhead at the expense of a insignificant amount of Random Access Memory (RAM). No major issues were encountered when writing the function and its output was trivial to validate by comparison to the reference outputs provided in Figure 12.

```c
void generateCACodeSequence(char caCodeSequence[CA_CODE_SEQUENCE_LENGTH], char prn) {
    // (Ref: IS-GPS-200N, Table 3-Ia. Code Phase Assignments)
    short delays[] = {  5,   6,   7,   8,  17,  18, 139, 140, 141, 251,
                      252, 254, 255, 256, 257, 258, 469, 470, 471, 472,
                      473, 474, 509, 512, 513, 514, 515, 516, 859, 860,
                      861, 862    };

    char g1[CA_CODE_SEQUENCE_LENGTH];
    char g2i[CA_CODE_SEQUENCE_LENGTH];
    char gi[CA_CODE_SEQUENCE_LENGTH];

    char g1Register[10];
    char g2iRegister[10];

    // i and j declared here to save recreating them for each loop
    short i, j;

    // Initialize the registers
    for (i = 0; i < CA_REGISTER_LENGTH; i++) {
        // By initializing the registers with -1, multiplying their bits later gives us the desired effect of modulo 2 addition (i.e. XOR)!
        g1Register[i] = -1;
        g2iRegister[i] = -1;
    }

    // Generate the g1 and g2i sequences
    for (i = 0; i < CA_CODE_SEQUENCE_LENGTH; i++) {
        // Save the register outputs
        g1[i] = g1Register[9];
        g2i[i] = g2iRegister[9];

        // Calculate the feedback values
        char c1 = g1Register[2] * g1Register[9];
        char c2 = g2iRegister[1] * g2iRegister[2] * g2iRegister[5] * g2iRegister[7] * g2iRegister[8] * g2iRegister[9];

        // Shift each of the bits one position to the right
        for (short j = (CA_REGISTER_LENGTH - 1); j > 0; j--) {
            g1Register[j] = g1Register[j - 1];
            g2iRegister[j] = g2iRegister[j - 1];
        }

        // Shift the feedback values into each of the registers
        g1Register[0] = c1;
        g2iRegister[0] = c2;
    }

    // Calculate the C/A Code Sequence!
    // Start by defining two indicies, i and j.
    // i starts at 0 as it is used to index the G1 code.
    // j is delayed "delay[prn - 1]" chips behind i in the previous cycle.
    for (i = 0, j = (CA_CODE_SEQUENCE_LENGTH - delays[prn - 1]); i < CA_CODE_SEQUENCE_LENGTH; i++, j++) {
        // We wish to perform G1 XOR G2i (delayed).
        // Multiplying G1 by G2i can give 1 or -1. Taking this from 1 we get either 0 or 2. We divide by 2 to get either 0 or 1 (desired).
        // j % CA_CODE_SEQUENCE_LENGTH ensures j will wrap back to zero when it crosses the sequence boundary.
        caCodeSequence[i] = (1 - g1[i] * g2i[j % CA_CODE_SEQUENCE_LENGTH]) / 2;
    }
}
```

Listing 4: The function used to generate the C/A code for each satellite. [simulator.c]

### 4.4.4 Navigation Message Formatting

The GPS navigation message broadcast by each satellite contains the satellite's ephemeris, time, the "almanac" (a rough record of all the constellation's ephemerides), and various other values indicating satellite status. The format is consistent across all GPS satellites, with values adjusted per satellite's unique orbital parameters and conditions.

Until receiving an updated ephemeris from ground control stations, each satellite continuously transmits an almost identical navigation message. The primary difference between successive transmissions lies in the encoded GPS time, expressed through two values: the Week Number (WN), representing weeks since January 6th, 1980, and Time of Week (TOW), denoting the count of six-second intervals since the start of the current week.

For context, it is useful to understand the navigation message structure. Each navigation message consists of 25 "frames", each frame 5 "subframes", each subframe 10 "words" and each word 30 bits. This is captured in Figure 15.

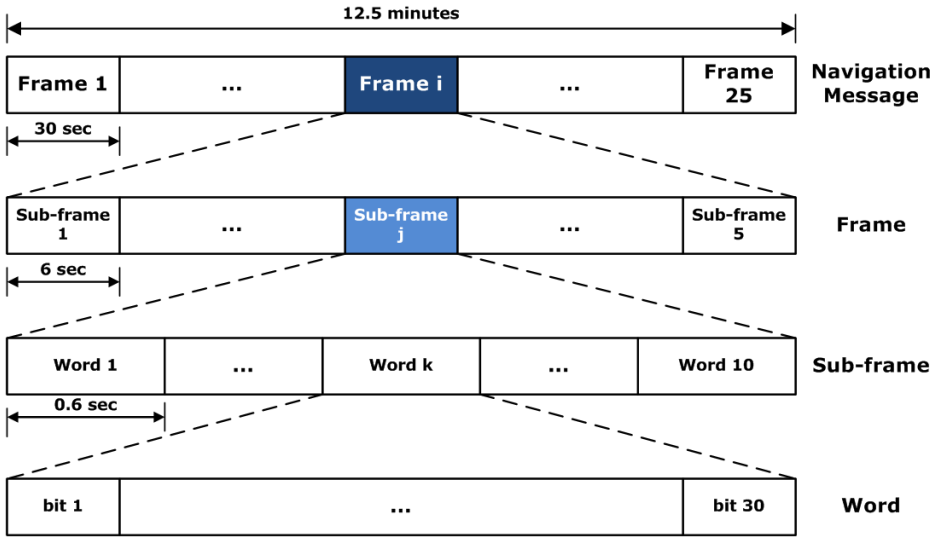


Figure 15: An overview of the navigation message structure [48].

The first three subframes of each frame are identical (with the exception of the timing information) between frames. The fourth and fifth subframes differ between frames as they contain successive installments of the large volume of data that is the almanac, see Figure 16. Every subframe starts with the Telemetry Word (TLM) followed by the Handover Word (HOW). The HOW contains the the TOW count of the subsequent subframe. Since each subframe takes six seconds to transmit, it is evident why the TOW only counts the elapsed six second intervals in the week.

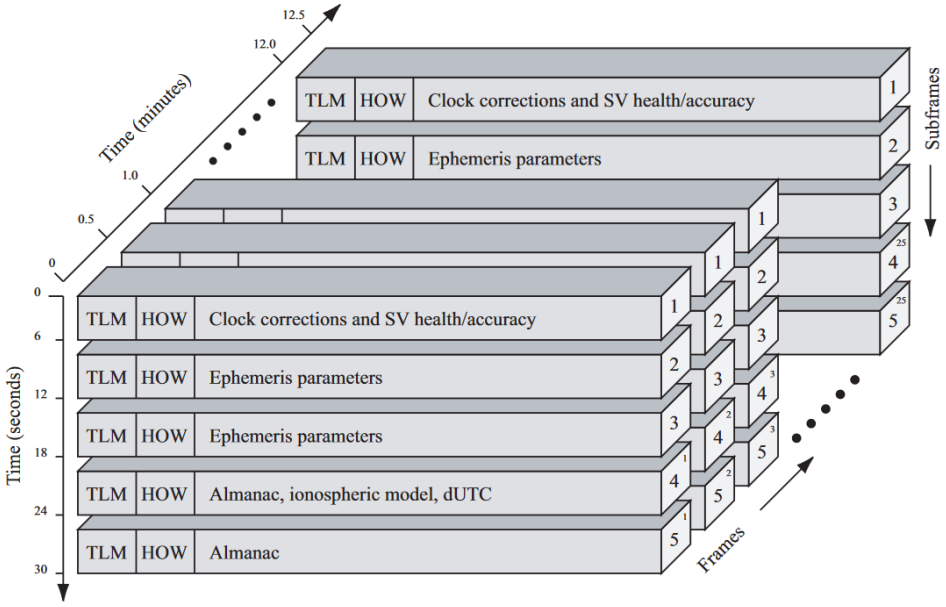


Figure 16: A navigation message diagram highlighting the repetition of subframes 1-3 and changed subframes 4 and 5 (notice incrementing superscript in subframe numbers 4 and 5 between frames) [45].

To avoid unnecessary recreations of simulated navigation messages, `gnss-sim` splits the creation of a satellite's boilerplate navigation message and its population with timing information into two separate functions: `generateNAVFrameBoilerplate` and `generateNAVFrame` respectively. `generateNAVFrameBoilerplate` is called at simulation initialization while `generateNAVFrame` is called after the completed transmission of a previous frame.

Taking inspiration from `gps-sdr-sim`, `generateNAVFrameBoilerplate` leaves all the almanac values in subframes 4 and 5 blank. Experiments with GNSS-SDR revealed that receivers can get by without this information and will overlook its absence given its limited utility outside the initial signal acquisition phase. This simplified

the flow, allowing the same frame structure to be used for all the navigation message installments. Taking this a step further, the ionospheric and Coordinated Universal Time (UTC) correction terms are also omitted in `gnss-sim`. Further experimentation is required to determine the impact on positioning accuracy (presumed minimal).

A procedure for "scaling" floating point numbers so that they can be expressed using unsigned integer values is specified in IS-GPS-200. This process has been implemented in `scaleDouble`.

```
14 long scaleDouble(double operand, double scaleFactorIndex) {
15     double scaleFactor = powf(2, scaleFactorIndex);
16     return (long)(operand / scaleFactor);
17 }
```

Listing 5: The scaleDouble function used to express doubles using unsigned longs in the fashion described by IS-GPS-200. `[simulator.c]`

To establish if the navigation messages were formatted correctly, a Bash test script (Listing 6) was written to compare the ephemeris values read into `gnss-sim` via RINEX file to the values decoded by GNSS-SDR. For a time, certain values we inexplicably off by a factor of $\pi$. Careful inspection of the RINEX specification and IS-GPS-200 revealed that RINEX files store angular quantities in radians while the GPS navigation messages should transmit the values in "semicircles" i.e. multiples of pi radians. The rationale could not be found in the GPS documentation however it is theorised that this approach is used to improve the numerical precision that can be achieved with a limited number of bits in the message. The section of `generateNAVFrameBoilerplate` responsible for scaling the ephemeris values and assigning these to the navigation message is shown in Listing 7.

```
1  #!/bin/bash
2
3  # Run the simulator
4  echo "GENERATING SIMULATION DATA..."
5  ../../build/gnss-sim -e ABMF00GLP_R_20240490000_01D_MN.rnx -o file-output.bin
6
7  # Run GNSS-SDR
8  echo "RUNNING GNSS-SDR..."
9  gnss-sdr --config_file=navmessage.conf
10
11 # Update user
12 echo ""
13 echo "FORMATTING AND COMPARING OUTPUTS..."
14
15 # Extract the "<second>" region from the XML file
16 awk '/<second/,/<\/second>/' gps_ephemeris.xml | grep -vE '(satClkDrift|dtr|IODE_SF2|IODE_SF3|code_on_L2|L2_P_data_flag|AODO|fit_interval_flag|spare1|
       spare2|integrity_status_flag|alert_flag|antispoofing_flag|tow)' > extracted-region.xml
17
18 # Convert the extracted XML region to the text format
19 awk -F'[><]' '/<second/{p=1;next}/<\/second/{p=0}p' extracted-region.xml | awk -F'[><]' '{print $2 ": " $3}' > converted-extracted-region.txt
20
21 # Perform the diff
22 diff -y  gnss-sim-ephemeris.txt converted-extracted-region.txt
23
24 # Clean up temporary files
25 rm extracted-region.xml converted-extracted-region.txt
```

Listing 6: One of the several Bash scripts used to confirm correct simulator behaviour. See the `gnss-sim/test/` directory for more. `[test.sh]`

```
35     // *** EPHEMERIS PARAMETERS ***
36     // NOTES:
37     // 1. Some values divided by PI to convert from radians to semicircles. Navmessages use
38     //    the semicircle unit however the RINEX format (which the supplied ephemeris is in)
39     //    stipulates radians be used instead.
40     // 2. (unsigned) cast required to resolve odd behaviour where compiler was treating
41     //    unsigned long as long during later right-shift operations.
42     unsigned long iode       = (unsigned)(ephemeris->iode                                    & BITMASK(8));
43     unsigned long crs        = (unsigned)(scaleDouble(ephemeris->crs,                  -5) & BITMASK(16));
44     unsigned long deltaN     = (unsigned)(scaleDouble((ephemeris->deln / PI),         -43) & BITMASK(16));
45     unsigned long m0         = (unsigned)(scaleDouble((ephemeris->M0 / PI),           -31) & BITMASK(32));
46     unsigned long cuc        = (unsigned)(scaleDouble(ephemeris->cuc,                 -29) & BITMASK(16));
47     unsigned long e          = (unsigned)(scaleDouble(ephemeris->e,                   -33) & BITMASK(32));
48     unsigned long cus        = (unsigned)(scaleDouble(ephemeris->cus,                 -29) & BITMASK(16));
49     unsigned long sqrtA      = (unsigned)(scaleDouble(sqrt(ephemeris->A),             -19) & BITMASK(32));
50     unsigned long toe        = (unsigned)(scaleDouble(time2gpst(ephemeris->toe, NULL),  4) & BITMASK(16));
51     unsigned long cic        = (unsigned)(scaleDouble(ephemeris->cic,                 -29) & BITMASK(16));
52     unsigned long omega0     = (unsigned)(scaleDouble((ephemeris->OMG0 / PI),         -31) & BITMASK(32));
53     unsigned long cis        = (unsigned)(scaleDouble(ephemeris->cis,                 -29) & BITMASK(16));
54     unsigned long i0         = (unsigned)(scaleDouble((ephemeris->i0 / PI),           -31) & BITMASK(32));
55     unsigned long crc        = (unsigned)(scaleDouble(ephemeris->crc,                  -5) & BITMASK(16));
56     unsigned long omega      = (unsigned)(scaleDouble((ephemeris->omg / PI),          -31) & BITMASK(32));
57     unsigned long omegaDot   = (unsigned)(scaleDouble((ephemeris->OMGd / PI),         -43) & BITMASK(24));
58     unsigned long idot       = (unsigned)(scaleDouble((ephemeris->idot / PI),         -43) & BITMASK(14));
```

Listing 7: Pictured, conversion and assignment of the ephemeris values for insertion into the navigation message later. Notice the conversions from radians to semicircles. `[simulator.c]`

As discussed, the TLM and HOW are required in each subframe. This is achieved in `generateNAVFrameBoilerplate` with a for loop to save the repeated instructions from cluttering the function.

```
92      // The first two words (TLM and HOW) are almost identical between subframes so generate them all at once
93      for (unsigned long subframe = 0; subframe < SUBFRAME_COUNT; subframe++) {
94          // NOTES:
95          // 1. Using legacy ISF until someone suggests otherwise
96          frame[subframe][0] = (TLM_WORD_PREAMBLE << 22) | (TLM_WORD_TLM_MESSAGE << 8) | (TLM_WORD_ISF_LEGACY
                << 7);
97
98          // NOTES:
99          // 1. TOW-Count Message populated later
100         // 2. Assuming User Range Accuracy (URA) is normal (HOW_WORD_ALERT_NOMINAL)
101         // 3. Bits 23 and 24 generated later during parity bit computation (Ref: IS-GPS-200N, Figure 20-2.
                TLM and HOW Formats)
102         frame[subframe][1] = (HOW_WORD_ALERT_NOMINAL << 12) | (HOW_WORD_ANTI_SPOOF_OFF << 11) | ((subframe +
                1) << 8);
103     }
```

Listing 8: Insertion of the TLM and HOW (sans TOW which is added later) into each subframe. `[simulator.c]`

`unsigned long`s are used to store the contents of each word in the navigation message. These are 32 bits wide, while the word contents never exceeds 30 bits. To improve legibility, the two excess bits are used by `gnss-sim` in the `computeParity` function which requires they both be 0 before entering this routine. Explicit clearing of the two Most Significant Bits (MSBs) of each word was added (Listing 9) after issues were encountered with checksum computation in cases where the data overflowed the 30-bit word boundary.

```
174     for (int subframe = 0; subframe < 5; subframe++) {
175         for (int word = 0; word < 10; word++) {
176             frame[subframe][word] &= BITMASK(30);
177         }
178     }
```

Listing 9: Critical masking to prevent the two MSBs of any of the data words from being occupied as they are used during checksum computation. `[simulator.c]`

The full `generateNAVFrame` function is shown in Listing 10. Despite being clear in IS-GPS-200, two bugs were introduced here due to an incorrect interpretation of the TOW count as being in seconds and a misunderstanding that each subframe's TOW value must equal the TOW at the start of the following subframe. These were made apparent by GNSS-SDR-reported pseudorange values in the order of 10,000 times their expected range.

```
282 void generateNAVFrame(gtime_t initalTime, unsigned long* previousWord, unsigned long frame[SUBFRAME_COUNT][
        WORD_COUNT], bool init) {
283     int wn;
284     double tow_s = time2gpst(initalTime, &wn);
285
286     // NOTES:
287     // 1. Convert tow_s to TOW by dividing by TOW epoch duration
288     // 2. See IS-GPS-200N: 20.3.3.2 Handover Word (HOW) and IS-GPS-200N: 3.3.4 GPS Time and SV Z-Count
289     unsigned long tow_epochs = (unsigned)(tow_s / SUBFRAME_DURATION_S);
290     unsigned long navmessageWn;
291
292     // The last word of the previous frame is required to generate the parity bits for this frame
293     // We must generate the last word of the previous frame if one is not available (occurs on init)
294     if (init) {
295         for (short word = 0; word < WORD_COUNT; word++) {
296             // Insert TOW into the HOW (2nd word of subframe)
297                         if (word == 1) {
298                 frame[4][word] |= (tow_epochs << 13);
299             }
300
301             // solveNibs = "true" when looking at either the 2nd or 10th word.
302             computeParity(&frame[4][word], previousWord, ((word == 1) || (word == 9)));
303         }
304     }
305
306     for (short subframe = 0; subframe < SUBFRAME_COUNT; subframe++) {
307         // Increment the TOW between subframes
308         updateTOW(&tow_epochs, &wn, SUBFRAME_DURATION_S);
309
310         // Mask TOW and WN for safety
311         tow_epochs = tow_epochs & BITMASK(17);
312         navmessageWn = wn & BITMASK(10);
313
314         for (short word = 0; word < WORD_COUNT; word++) {
315             // Add Transmission Week Number to the third word of subframe one.
316                         if ((subframe == 0) && (word == 2)) {
317                 frame[subframe][word] |= (navmessageWn << 20);
318             }
319
320                         // Insert TOW into the HOW (2nd word of subframe).
```

```
                        if (word == 1) {
                frame[subframe][word] |= (tow_epochs << 13);
            }

            // solveNibs = "true" when looking at either the 2nd or 10th word.
            computeParity(&frame[subframe][word], previousWord, ((word == 1) || (word == 9)));
        }
    }
}
```

Listing 10: The full `generateNAVFrame` function. `[simulator.c]`

### 4.4.5 Parity Bit Computation

Parity bits provide provide a means of error checking to the navigation messages. IS-GPS-200 provides the algorithm used on the satellites (Figure 17). This is implemented in `gnss-sim`'s `computeParity` function using several techniques from `gps-sdr-sim`, notably a compact multi-stage XOR operation. A detailed explanation of how this works is laid out in the code's comments, see Listing 11.

$$
\begin{aligned}
D_1 &= d_1 \oplus D_{30}^{\star} \\
D_2 &= d_2 \oplus D_{30}^{\star} \\
D_3 &= d_3 \oplus D_{30}^{\star} \\
&\bullet \quad\quad \bullet \\
&\bullet \quad\quad \bullet \\
&\bullet \quad\quad \bullet \\
&\bullet \quad\quad \bullet \\
D_{24} &= d_{24} \oplus D_{30}^{\star} \\
D_{25} &= D_{29}^{\star} \oplus d_1 \oplus d_2 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_{10} \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{17} \oplus d_{18} \oplus d_{20} \oplus d_{23} \\
D_{26} &= D_{30}^{\star} \oplus d_2 \oplus d_3 \oplus d_4 \oplus d_6 \oplus d_7 \oplus d_{11} \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{18} \oplus d_{19} \oplus d_{21} \oplus d_{24} \\
D_{27} &= D_{29}^{\star} \oplus d_1 \oplus d_3 \oplus d_4 \oplus d_5 \oplus d_7 \oplus d_8 \oplus d_{12} \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{19} \oplus d_{20} \oplus d_{22} \\
D_{28} &= D_{30}^{\star} \oplus d_2 \oplus d_4 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{13} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{20} \oplus d_{21} \oplus d_{23} \\
D_{29} &= D_{30}^{\star} \oplus d_1 \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_7 \oplus d_9 \oplus d_{10} \oplus d_{14} \oplus d_{15} \oplus d_{16} \oplus d_{17} \oplus d_{18} \oplus d_{21} \oplus d_{22} \oplus d_{24} \\
D_{30} &= D_{29}^{\star} \oplus d_3 \oplus d_5 \oplus d_6 \oplus d_8 \oplus d_9 \oplus d_{10} \oplus d_{11} \oplus d_{13} \oplus d_{15} \oplus d_{19} \oplus d_{22} \oplus d_{23} \oplus d_{24}
\end{aligned}
$$

Where

$d_1, d_2, ..., d_{24}$ are the source data bits;

the symbol $\star$ is used to identify the last 2 bits of the previous word of the subframe;

$D_{25}, D_{26}, ..., D_{30}$ are the computed parity bits;

$D_1, D_2, ..., D_{29}, D_{30}$ are the bits transmitted by the SV;

$\oplus$ is the "modulo-2" or "exclusive-or" operation.

Figure 17: "Table 20-XIV. Parity Encoding Equations" from [42].

```
void computeParity(unsigned long* word, unsigned long* previousWord, unsigned short solveNibs) {
    // NOTES:
    // 1. This function is quite heavy! Don't worry about understanding it perfectly.
    // 2. "IS-GPS-200N, Table 20-XIV. Parity Encoding Equations" is essential reading if you do want to
         understand what's going on here.
    // 3. Variable names the same as "IS-GPS-200N, Table 20-XIV. Parity Encoding Equations" to make it
         easier to follow along

    // Prepend 2 LSBs of the previous transmitted word to this word to form "source"
    unsigned long source = *word | ((*previousWord << 30) & 0xC0000000UL);

    // Masking source so we have just b1 --> b24
    unsigned long d = source & 0x3FFFFFC0UL;

    // Accessing D29 and D30 from previous word.
    unsigned long d29Star = (source >> 31) & 0x1UL;
    unsigned long d30Star = (source >> 30) & 0x1UL;

    // The following graciously provided by gps-sdr-sim. Values can be calculated from "IS-GPS-200N, Table
         20-XIV. Parity Encoding Equations"
        unsigned long parityBitMasks[6] = {
                0x3B1F3480UL,
        0x1D8F9A40UL,
        0x2EC7CD00UL,
                0x1763E680UL,
        0x2BB1F340UL,
        0x0B7A89C0UL
    };

    // If instructed, solve non-information bearing bits 23 and 24 to preserve parity check.
    // This is required for words 2 and 10 of each subframe. See comments in "generateNavmessageBoilerplate
         ".
    // See later comments for an explanation of what's going on here.
    if (solveNibs) {
```

```
220         if ((d30Star + countSetBits(parityBitMasks[4] & d)) & 1UL) {
221             d ^= (1UL << 6);
222         }
223
224         if ((d29Star + countSetBits(parityBitMasks[5] & d)) & 1UL) {
225             d ^= (1UL << 7);
226         }
227         }
228
229     unsigned long D = d;
230
231     // The following XORs b1 --> b24 with d30Star (word XOR D30Star will just be word when D30Star is 0)
232     if (d30Star) {
233         D ^= 0x3FFFFFC0UL;
234     }
235
236     // The following lines compute and insert the 6 parity bits at the end of the word.
237     // STEPS:
238     // 1. Mask b so we're just looking at the relevant message bits (word & parityBitMasks[i])
239     // 2. Count total "set" (1) bits between masked word and DxStar (dxStar + countSetBits(y))
240     // 3. Modulo 2 the previous step's result (& 1UL). "& 1UL" = "% 2"! It's a performance thing (Ref: https
            ://mziccard.me/2015/05/08/modulo-and-division-vs-bitwise-operations/)
241     // 4. Shift the computer parity bit into the correct location in word
242     // For why we use steps 2 and 3, see here: https://stackoverflow.com/questions/32747229/fastest-way-to-
            xor-all-bits-from-value-based-on-bitmask
243     D |= ((d29Star + countSetBits(d & parityBitMasks[0])) & 1UL) << 5;
244     D |= ((d30Star + countSetBits(d & parityBitMasks[1])) & 1UL) << 4;
245     D |= ((d29Star + countSetBits(d & parityBitMasks[2])) & 1UL) << 3;
246     D |= ((d30Star + countSetBits(d & parityBitMasks[3])) & 1UL) << 2;
247     D |= ((d30Star + countSetBits(d & parityBitMasks[4])) & 1UL) << 1;
248     D |= ((d29Star + countSetBits(d & parityBitMasks[5])) & 1UL);
249
250     D &= 0x3FFFFFFFUL;
251
252     *previousWord = D;
253     *word = D;
254 }
```

Listing 11: The full `computeParity` function. `[simulator.c]`

Consulting the GNSS-SDR Telemetry Decoder logs revealed an issue with initial parity bit calculations. Isolating the cause proved challenging due to inconsistent parity check results across subframes. Using GNU Debugger (GDB), GNSS-SDR's internals were inspected during the reassembly of simulated navigation messages to uncover improper masking of the two MSBs of each word within the navigation message, some words happened to overflow while others did not, leading to the inconsistent behaviour between subframes. Now this is addressed in `generateNAVFrameBoilerplate`, GNSS-SDR's parity checks all pass.

### 4.4.6 I/Q Data Generation

This section explains how `gnss-sim` translates generated digital navigation signals into a modulation-compatible format. Pertinent GPS L1 signal parameters include a carrier frequency of 1575.42 MHz, a data signal frequency of 1.023 MHz (representing the C/A code XORed with the navigation message), and a bandwidth of approximately 24 MHz. GPS employs BPSK for encoding digital bits onto the carrier.

"I/Q" notation denotes base band signals in SDRs, describing both receiving and transmitting ends. Typically, an interleaved format is adopted, featuring pairs of I and Q values immediately succeeding each other in the dataset. Each pair constitutes a sample, representing a vector where the magnitude equals the signal's magnitude, and the phase reflects the signal's phase at that instance. The nomenclature "I/Q" originates from the orthogonal relationship of the components, with "I" representing the component in-phase and "Q" representing the component in quadrature with the carrier signal during modulation.

In SDR transmission, I/Q data feeds an I/Q modulator, such as the one depicted in Figure 18, employing a pair of Digital-to-Analogue Converters (DACs) to modulate a fixed carrier, thereby facilitating phase, amplitude, and frequency modulation [49]. Frequency modulation follows from its definition as the rate of phase change over time. During SDR reception, the received signal feeds an I/Q demodulator, where Analogue-to-Digital Converters (ADCs) extract the I and Q values to form each sample.

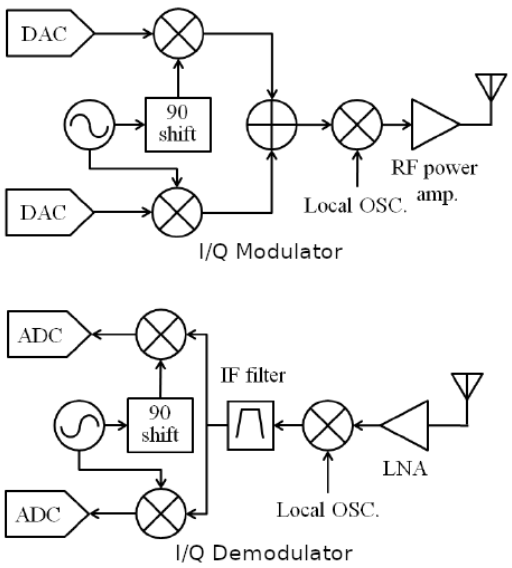


Figure 18: Generic I/Q modulator and demodulator block diagrams [50].

Listing 12 is an extract from the `simulate` function. Its purpose is to determine the signal contribution from each channel associated to a satellite in view of the simulated receiver. All the I and Q values are accumulated to determine the net contribution across all channels i.e. the summation of all the signals. The I and Q values for each channel are found by XORing the current navigation message bit and C/A code chip then multiplying this by the sine/cosine of the carrier signal's phase.

```
685         // Fill sample window IQ buffer
686         for (int i = 0; i < IQ_BUFFER_SIZE; i += 2) {
687             short iAccumulated = 0;
688             short qAccumulated = 0;
689             int previousNavBitPointer = 0;
690             int carrierPhaseIndex = 0;
691 
692             for (char channel = 0; channel < CHANNEL_COUNT; channel++) {
693                 // Map the channel's carrier phase to an index in the carrier phase look-up table
694                 // NOTE: Using look-up table to save processing time otherwise spent computing trig
                        functions
695                 carrierPhaseIndex = (int)(channels[channel].carrierPhase_cycles * (TRIG_TABLE_SIZE - 1));
696 
697                 // Write I followed by Q value to the buffer
698                 // NOTES:
699                 // 1. "((x * 2) - 1)" maps a 0/1 value to a -1/1 value. Multiplying these remapped terms
                        allows us to XOR them
700                 // 2. Multiplication by sine and cosine used to introduce carrier phase differences
701                 iAccumulated += ((channels[channel].codeChip * 2) - 1) * ((channels[channel].navBit * 2) -
                        1) * cosTable[carrierPhaseIndex];
702                 qAccumulated += ((channels[channel].codeChip * 2) - 1) * ((channels[channel].navBit * 2) -
                        1) * sinTable[carrierPhaseIndex];
703 
704                 // Advance the channel modulation to the appropriate starting bit and chip. May result in no
                        change yet
705                 advanceChannelModulation(&channels[channel], simulationTime, SAMPLE_INTERVAL_S, false);
706             }
707 
708             iqBuffer[i] = iAccumulated;
709             iqBuffer[i + 1] = qAccumulated;
710 
711             simulationTime = timeadd(simulationTime, SAMPLE_INTERVAL_S);
712         }
713 
714         dumpCallback(iqBuffer, IQ_BUFFER_SIZE);
```

Listing 12: This brief snippet forms the heart of the `simulate` function, generating the final I/Q values. `[simulator.c]`

The simulation's sample rate can be configured in `simulator.h`. It should be set to a minimum of 2.5 Msps to ensure the 1.023 MHz fundamental frequency of the data signal is captured. Testing with GNSS-SDR at higher sample rates did not indicate an improvement in signal reception.

A GNU Radio script was employed to visualize vector and time series plots of the `gnss-sim` I/Q output for verification purposes (Figure 19). Although the I/Q modulation logic exhibited no anomalies during initial development, this methodology later pin-pointed multiple defects within the `advanceChannelModulation` func-

tion, responsible for updating the navigation message bit and C/A chip of the channel.

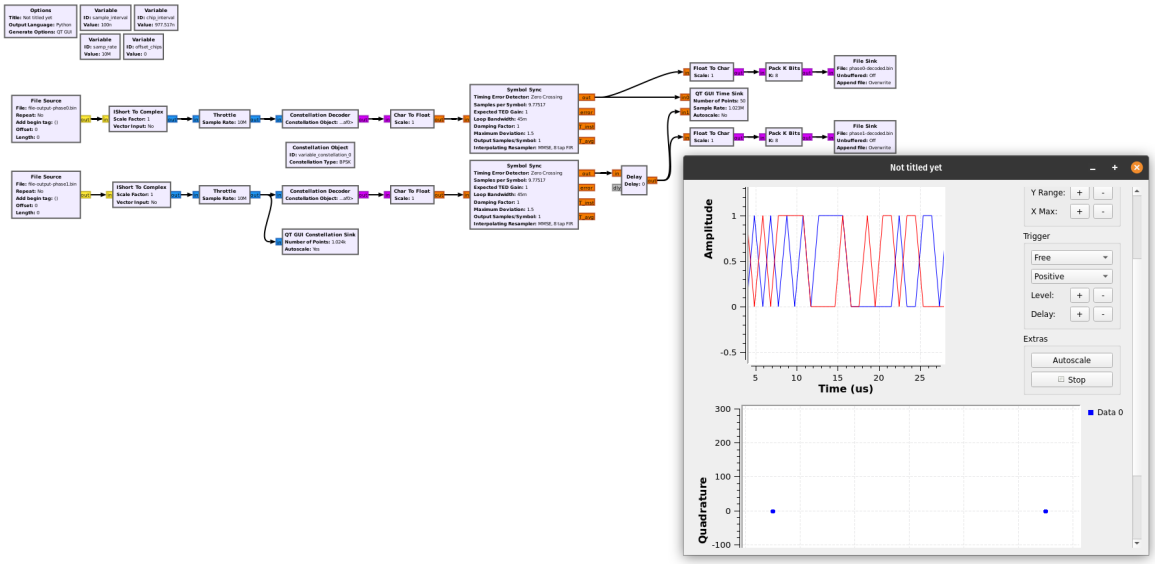


Figure 19: Custom GNU Radio tool being used to verify I/Q data generation working.

### 4.4.7 Coordinate Systems

The Earth-Centered, Earth-Fixed (ECEF) reference frame is used within `gnss-sim` for all the calculations in three-dimensional space. ECEF employs a left-handed Cartesian coordinate system, with its origin at the Earth's center of mass. The Z-axis runs from the North to the South Pole, the X-axis lies on the equatorial plane passing through Greenwich, and the Y-axis follows from the left-handed nature of the system. Figure 20 provides an illustration.

Working with geocentric coordinates (referenced to planet's core) within ECEF, as `gnss-sim` does, is common in GNSS applications. This allows for straightforward computation of distances between satellites and receivers by simply taking the vector difference between them. Geodetic coordinates (referenced to the planet's surface) are uncommon in calculations due to their inelegant definitions however are popular for interchange, a common example is latitude, longitude and height coordinates.

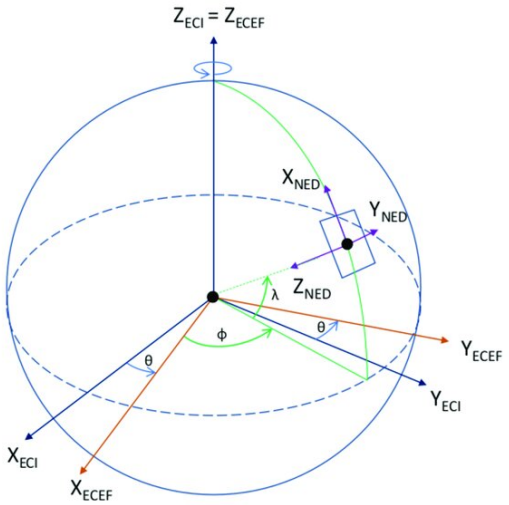


Figure 20: ECEF and Earth-Centered, Inertial (ECI) definitions [51].

The ECI (Earth Centered, Inertial) reference frame is a rotated replica of the ECEF frame around the Z axis, aligning the X axis parallel to an axis fixed relative to the sun. Satellites orbit Earth in the ECI frame, decoupled from Earth's rotational motion. The relative motion between the ECEF and ECI frames gives rise to the complex ground tracks and geometric range patterns between satellites and receivers. This is captured well in Figure 21. The RTKLIB functions used to calculate simulated satellite positions based on their ephemerides

perform the conversion between ECI and ECEF frames automatically during this process. The quality of this operation is such that no development effort was spent solving coordinate system transformation issues.

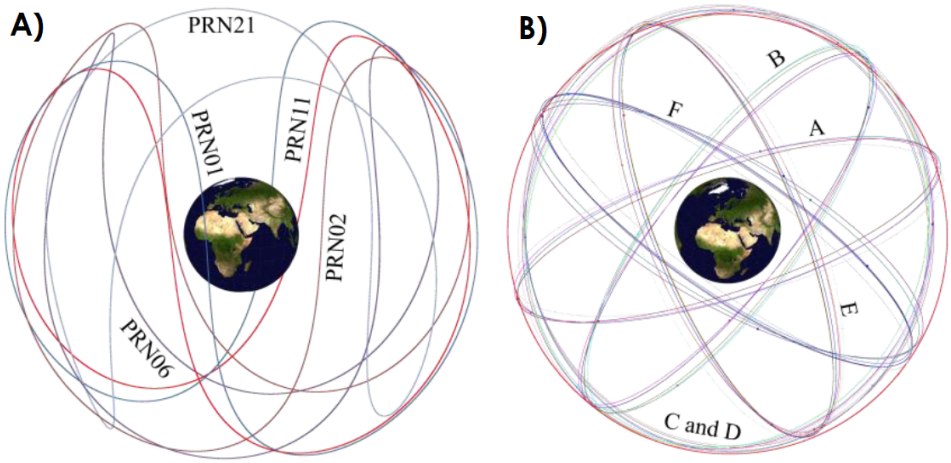


Figure 21: A: GPS satellite orbits drawn in the ECEF system. B: The same orbits drawn in the ECI system [52].

The receiver position within `gnss-sim` is defined by its latitude, longitude, and height. The rationale behind use of these geodetic dimensions is a suspected familiarly of these with the software's users who will need to input these values in the final version. Updating and converting the receiver position to geocentric coordinates is the role of the `updateRecieverPosition` function. It has been written such that dynamic receiver motion can be added in future, see Listing 13.

```
489 void updateRecieverPosition(double* receiverPosition_llh, double* receiverPosition_ecef) {
490     double receiverPositionCopy_llh[3];
491     memcpy(receiverPositionCopy_llh, receiverPosition_llh, (sizeof(receiverPosition_llh) * 3));
492 
493     // Convert latitude and longitude in degrees to radians
494     receiverPositionCopy_llh[0] *= (PI / 180);
495     receiverPositionCopy_llh[1] *= (PI / 180);
496 
497     // Convert llh to ecef frame
498     pos2ecef(receiverPositionCopy_llh, receiverPosition_ecef);
499 }
```

Listing 13: The `updateRecieverPosition` function. `[simulator.c]`

The first incarnation of `updateRecieverPosition` mistakenly overwrote the `recieverPosition_llh` values during the conversion from degrees to radians. This resulted in repeated calls to the function tending to latitude and longitude values of zero. The use of `memcpy` addressed this.

### 4.4.8 Channel Allocation

As of the time of writing, there are 32 operational GPS satellites, few of which will be simultaneously visible to a receiver at one time. A method is required to determine which satellites are visible and should be assigned to the limited number of receiver channels. `gnss-sim` ranks all the satellites by their distance from the receiver's sky center using their elevation angles. Those closest to the center are allocated channels. This is performed by the `updateChannelAllocations` function (Listing 14).

The adopted approach benefits from its ease of interpretation but inevitably results in poor dilution of precision. The trade off was deemed acceptable for an initial implementation. Future work could explore the implication of this method using DOP values derived from GNSS-SDR calculations.

```
449 void updateChannelAllocations(gtime_t simulationTime, Channel channels[CHANNEL_COUNT], SV** svs, short
        svCount, double* receiverPosition_ecef, double timeStep_s) {
450     // Reset all the satellite pseudorange rates. Critical to prevent **MASSIVE** pseudorange rate errors
451     for (int i = 0; i < svCount; i++) {
452         svs[i]->psuedorangeRate_ms = 0;
453     }
454 
455     // Determine all sv positions
456     updateSatellitePositions(simulationTime, svs, svCount, receiverPosition_ecef, timeStep_s);
457 
458     // Rank satellites by distance from the receiver's sky center
459     qsort(svs, svCount, sizeof(SV*), compareSatelliteVisibility);
```

```
460
461     // Assign satellites to channels by most to least visible
462     for (int i = 0; i < CHANNEL_COUNT; i++) {
463         // Assign the next most visible satellite in the svs list
464         channels[i].sv = svs[i];
465
466         // Generate initial navframe
467         generateNAVFrame(simulationTime, &channels[i].previousWord, channels[i].sv->navFrame, true);
468
469         // Determine the code and carrier frequencies and phases
470         updateChannelProperties(&channels[i], 1);
471
472         // Calculate the difference in expected transmission delay
473         double codePhase_s = (channels[i].sv->psuedorange_m / LIGHTSPEED);
474
475         // Determine time in week (used in conjunction with code phase to set the intial navbit and code
                chip)
476         int wn;
477         double tow = time2gpst(simulationTime, NULL);
478
479         // Reset the code chip and navbit pointers
480         channels[i].codeChipPointer = 0;
481         channels[i].navBitPointer = 0;
482
483         // Advance the channel modulation to the appropriate starting bit and chip
484         advanceChannelModulation(&channels[i], simulationTime, (tow + codePhase_s), true);
485     }
486 }
```

Listing 14: The updateChannelAllocations function. `[simulator.c]`

To minimize processing overhead, channel allocation is conducted periodically while satellite positions are updated at a rate of 10 Hz (simulation time) to accommodate their continuous movement. A Python script was written to parse RINEX files retrieved from GNSS monitoring stations to determine a reasonable frequency for updating channel allocations. Its output is shown in Figure 22. Based off this, it was decided that intervals between calls to `updateChannelAllocations` on the order of several minutes (simulation time) are acceptable.

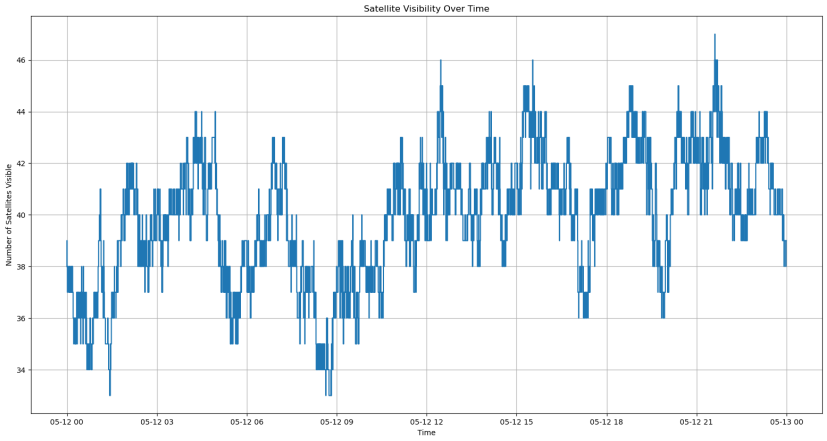


Figure 22: Graph showing visible GNSS satellite count (GPS, GLONASS, BeiDou and Galileo) for a single monitoring station.

### 4.4.9 Valid Observable Ranges

"Observables" are the raw measurements collected by GPS receivers from the signals transmitted by the satellites. These include the differences in pseudorange, carrier phase and Doppler shift introduced by the signals' transmission though space. Combining these observations with the data present in the navigation messages, is how receivers determine their position.

A valid range for each observable was determined by approximating the typical scenario's geometry. This facilitated fast and rough assessment of the observables returned by GNSS-SDR when analyzing the output from `gnss-sim`. This was motivated by a certain confidence in the "integration hell" that was to come as several quantities all began to impact the final signal in ways that are difficult to pick apart at the output of `gnss-sim`.

Figure 23 is a sketch of the assumed flight path of a satellite over a receiving station. For the sake of simplicity, it is assumed that the satellite is visible to the receiver for the entire duration it is above the receiver's horizon.

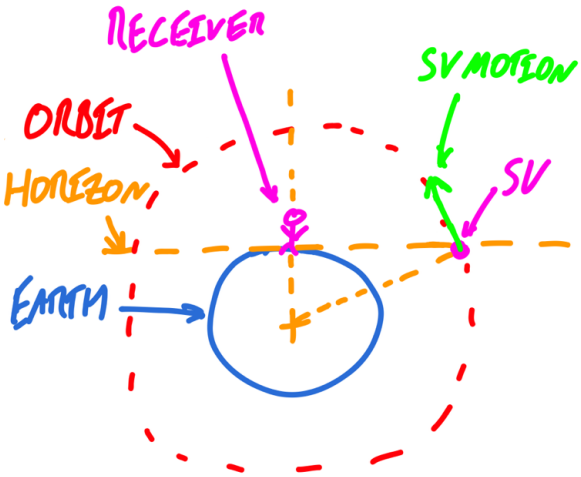


Figure 23: The geometry of a single satellite orbit.

First the typical satellite orbital velocity, $V_T$ was calculated.

$$V_T = \sqrt{\frac{GM}{r}}$$

Using values of $6.67 \times 10^{-11} m^3\ kg \cdot s^2$ for $G$, $5.97 \times 10^{24} kg$ for $M$ and $26.6 \times 10^6 m$ for $r$, we arrive at a $V_T$ value of $3.91 \times 10^3 m/s$.

Next, the visible region's dimensions are established using circle segment formulae combined with knowledge of the orbit's altitude.

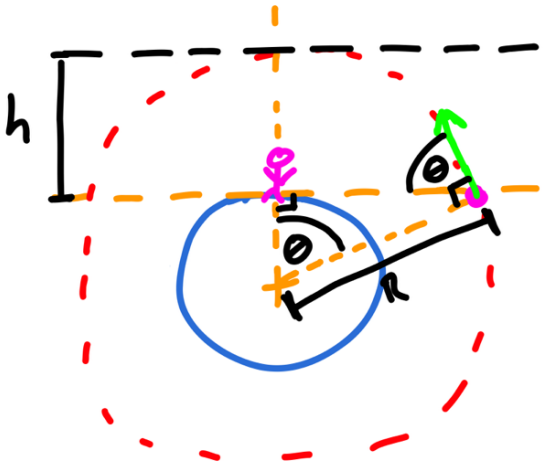


Figure 24: Dimensions assigned to orbit geometry. Notice how $\theta$ is the same at the center and at the tangent.

$$\theta = \arcsin(\frac{\sqrt{2Rh - h^2}}{R})$$

Using values of $26.6 \times 10^6 m$ for $R$ and $20.3 \times 10^6 m$ for $h$, we arrive at a maximum $\theta$ value of $76°$ when the satellite is at the horizon.

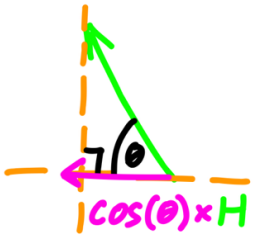


Figure 25: As shown, the pseudorange range rate (meters per second) at the horizon can be found as $\cos(\theta) \times H$, where $H$ is equal to the satellite's orbital velocity.

Taking the cosine of $\theta$ and multiplying it by $V_T$, we get the component of $V_T$ pointing towards the receiver at the horizon i.e. the pseudorange range rate in meters per second. We arrive at a value of approximately $950m/s$. Therefore the range for this value must be $\pm 950m/s$ as the satellite moves away from the receiver once is passes the sky center.

Below is a convenient expression for Doppler shift common in the geospatial community

$$\dot{p} = -\lambda f_D \tag{1}$$

where $\dot{p}$ is the pseudorange range rate (m/s), $\lambda$ is the signal's wavelength (m) and $f_D$ is the signal's Doppler shift (Hz). Using this expression, we can calculate the expected Doppler shift of a GNSS's carrier *and* code signal. The maximum pseudorange range rate calculated earlier ($950m/s$) was combined with the carrier ($0.2m$) and C/A code ($300m$) wavelengths to find the range of expected Doppler shifts.

The extents of the pseudorange value was found using simple trigonometry of Figure 25. The code phases were found by multiplying these pseudorange values by the speed of light. The calculated valid observable ranges are presented in Table 1.

| Quantity | Approximate Minimum | Approximate Maximum |
|---|---|---|
| Pseudorange Range (km) | 19,500 | 25,500 |
| Pseudorange Range Rate (m/s) | -950 | 950 |
| Code Phase (ms) | 65 | 85 |
| Code Doppler Shift (Hz) | -3 | 3 |
| Carrier Doppler Shift (kHz) | -5 | 5 |

Table 1: Calculated valid observable ranges.

#### 4.4.10 Observables Calculation

How the navigation messages are formatted and encoded is described previously. Discussed here is how that information is delivered to the simulated RF front end in `gnss-sim`.

Key to understanding this section is the appreciation that the 10.23 MHz clock source aboard each GPS satellite is extremely well synchronized with its contemporaries, which also have an identical understanding of the time in week. All the signals generated by the GPS satellite are referenced to its 10.23 MHz standard. This includes the 1546.72 MHz L1 carrier, 1.023 MHz C/A code, and 50 bps navigation message. This, combined with the synchronization between satellites, results in the transmission from each GPS satellite being in-phase with the transmission of all the other satellites. This marvel is one of GPS's lesser appreciated features and is fundamental to its use as a navigation service.

Practical implications aside, a GNSS receiver situated at the center of the Earth would receive all the constellation's transmissions exactly in phase with one another. In any other location, the differences in range between satellite-receiver pairs result in a phase offset between the signals at the receiver, which is how it determines where it is (combined with knowledge of the satellite positions derived from the contents of the navigation message).

Worth noting that while the majority of the time delay between signals is driven by the distance between stations, appreciable contributions arise from ionospheric effects, clock synchronization errors, and receiver signal delays. The severity of these effects is cataloged in Figure 26.

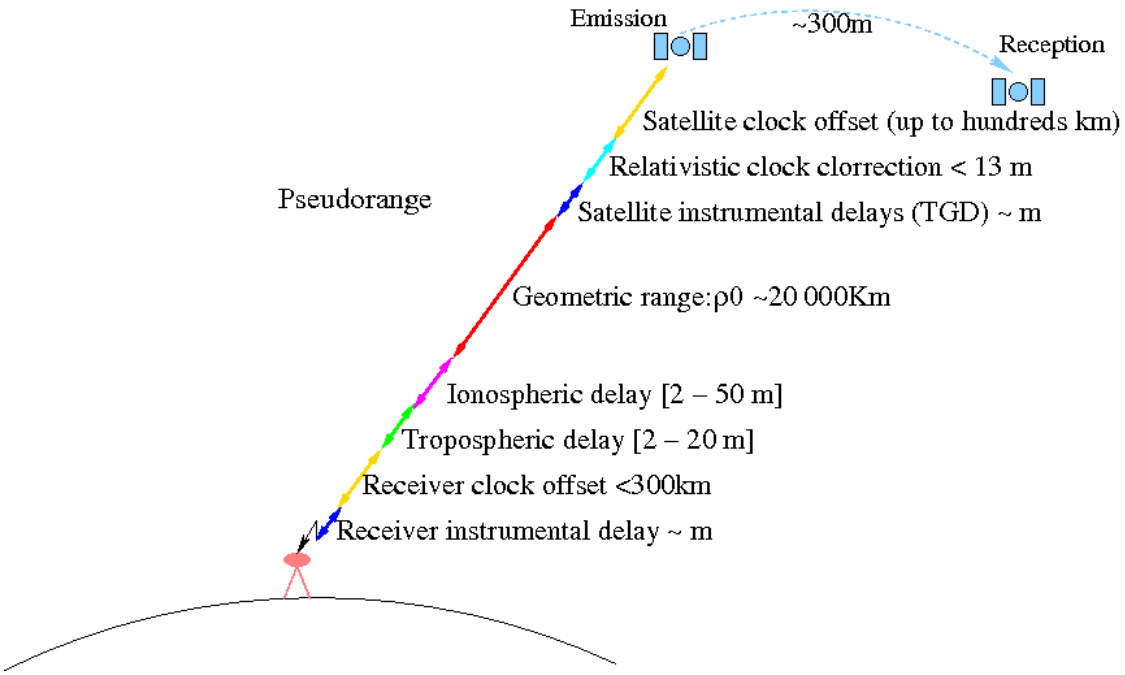


Figure 26: A breakdown of the contributors to the pseudorange measurement [53].

The additional factors affecting the signal propagation times cause the geospatial community to refer to distances inferred from time differences alone as "pseudoranges". `gnss-sim` does not presently model any of these additional effects. It is suggested they be added in future versions.

#### 4.4.10.1 Code Phase

Referring to Table 1, the maximum expected time delay between any two received satellite signals is approximately 20 ms for a receiver on Earth's surface, equivalent to the duration of a single navigation message bit. To enhance measurement resolution, differences in distance along the C/A code are used to measure time delay at the receiver. The C/A code's time period is just under 1 µs, thereby improving precision over the navigation message by a factor of 20,000. This discrepancy in C/A code distance between signals is termed the "code phase," as illustrated in Figure 27.

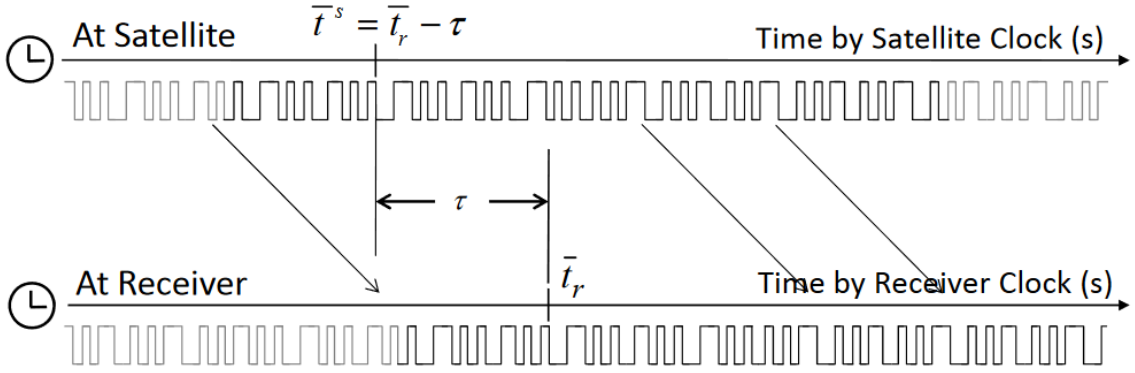


Figure 27: Illustration from the GNSS-SDR manual detailing pseudorange measurement methodology [43].

`gnss-sim` employs the `advanceChannelModulation` function (refer to Listing 15) to adjust each simulated signal's timing according to its transmitting satellite's distance and signal frequency.

```
503 void advanceChannelModulation(Channel* channel, gtime_t simulationTime, double totalIncrement_s, bool init)
    {
504     // Update the carrier phase
505     channel->carrierPhase_cycles += channel->carrierDopplerShift_Hz * totalIncrement_s;
506
507     // TODO: Implement a more perfromant version of this
```

```
508     // Clamp the carrier phase between 0 and 1
509     if (channel->carrierPhase_cycles >= 1.0) {
510         channel->carrierPhase_cycles -= floor(channel->carrierPhase_cycles);
511     }
512
513     else if (channel->carrierPhase_cycles <= 0.0) {
514         channel->carrierPhase_cycles += fabs(ceil(channel->carrierPhase_cycles)) + 1;
515     }
516
517     // Convert increment in seconds to chips, factoring in channel's code freqeuncy (affected by Doppler)
518     double totalIncrement_chips = (totalIncrement_s * channel->codeFrequency_Hz);
519
520     // Peck away at totalIncrement_chips in such away that we don't miss an instance when the navBitPointer
          should be incremented
521     while (totalIncrement_chips > 0) {
522         // Determine the maximum number of chips we can increment by this iteration
523         double partialIncrement_chips = fmin(totalIncrement_chips, CA_CODE_SEQUENCE_LENGTH);
524
525         // Update the number of remaining chips for the following iterations
526         totalIncrement_chips -= partialIncrement_chips;
527
528         // Update the code chip pointer and code chip
529         channel->codeChipPointer += partialIncrement_chips;
530         channel->codeChip = channel->sv->caCodeSequence[(int)fmod(channel->codeChipPointer,
              CA_CODE_SEQUENCE_LENGTH)];
531
532         // Decide if the code chip pointer needs wrapping round
533         if (channel->codeChipPointer >= (CA_CODE_SEQUENCE_LENGTH * CA_CYCLES_PER_NAV_BIT)) {
534             channel->codeChipPointer -= (CA_CODE_SEQUENCE_LENGTH * CA_CYCLES_PER_NAV_BIT);
535             channel->navBitPointer++;
536
537             // If we've gone past the last bit of this NAV frame...
538             if ((channel->navBitPointer % (SUBFRAME_COUNT * WORD_COUNT * WORD_BIT_COUNT)) == 0) {
539                 channel->navBitPointer = 0;
540
541                 // This check allows us to initilize the simulation more than a frame's worth of time (6 s)
                        into the week
542                 if (!init) {
543                     // Generate the next NAV frame!
544                     memcpy(channel->sv->navFrame, channel->sv->navFrameBoilerPlate, sizeof(channel->sv->
                            navFrameBoilerPlate));
545                     generateNAVFrame(simulationTime, &(channel->previousWord), channel->sv->navFrame, false)
                            ;
546                 }
547             }
548
549             // Get the next NAV frame bit
550             short subframe = channel->navBitPointer / (WORD_COUNT * WORD_BIT_COUNT);
551             short word = (channel->navBitPointer % (WORD_COUNT * WORD_BIT_COUNT)) / WORD_BIT_COUNT;
552             short bit = channel->navBitPointer % WORD_BIT_COUNT;
553
554             channel->navBit = (channel->sv->navFrame[subframe][word] >> (29 - bit)) & 0x1;
555         }
556     }
557 }
```

Listing 15: The `advanceChannelModulation` function.

For a period, GNSS-SDR received navigation messages from `gnss-sim` seconds apart. Although no warnings were issued, the author recognized an error given offset in received navigation messages being calculated previosuly to be no more than 20 ms. Several errors were identified in `advanceChannelModulation`, prompting its revision to rectify these issues. Navigation messages are now decoded by GNSS-SDR within 20 ms of each other.

#### 4.4.10.2 Carrier Phase

The carrier frequency is over 1,500 times higher than the C/A code frequency. Comparing the carrier frequency between received signals can improve measurement precision by the same order, as illustrated in Figure 28.

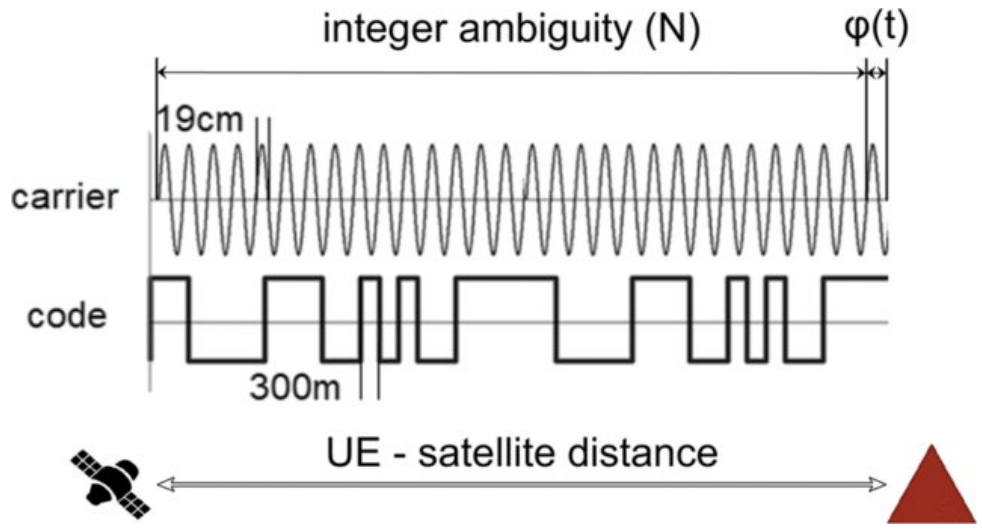


Figure 28: Comparison of precision between GPS C/A measurements and carrier measurements [54].

Carrier phase measurements are utilized in survey-grade GNSS receivers requiring accuracy to decimeters and beyond. However, their application in most other devices is limited due to the inability to distinguish between carrier phase cycles. Measuring the carrier phase alone does not provide an absolute distance between the satellite and the receiver because an unknown number of integer cycles has occurred between them in addition to the measured partial cycle. This necessitates an "integer ambiguity resolution" process, driving up receiver hardware requirements and costs.

In `gnss-sim`, carrier phases are updated each time `advanceChannelModulation` is called. However, the carrier phase initialization process (Listing 16), which simulates the correct number of integer cycles, requires further testing before it can be enabled. The resulting errors are capped at the carrier wavelength (200 mm).

```
559 void updateChannelProperties(Channel* channels, int channelCount) {
560     for (int channel = 0; channel < channelCount; channel++) {
561         // Calculate the carrier Doppler shift
562         // NOTE: See here for explanation: https://gnss-sdr.org/docs/sp-blocks/observables/#pseudorange-rate
                -measurement
563         channels[channel].carrierDopplerShift_Hz = -channels[channel].sv->psuedorangeRate_ms /
                CARRIER_WAVELENGTH_M;
564         channels[channel].codeDopplerShift_Hz = -channels[channel].sv->psuedorangeRate_ms /
                CA_CODE_WAVELENGTH_M;
565
566         // Calculate the C/A Code Doppler shift
567         // NOTE: This is only performed for the fundermental freqeuncy as the expected modulation bandwidth
568         //       is ~24 MHz and the Doppler spread over this range should be limited
569         channels[channel].codeFrequency_Hz = CA_CODE_FREQUENCY_HZ + channels[channel].codeDopplerShift_Hz;
570
571         // TODO: Uncomment these lines to enable carrier phase measurements
572         // Calculate carrier cycles between reciever and satellite
573         // double carrierCycles = (channels[channel].sv->psuedorange_m / CARRIER_WAVELENGTH_M);
574
575         // Find where we are within one cycle of the carrier phase
576         // channels[channel].carrierPhase_cycles = 0; //(carrierCycles - floor(carrierCycles));
577     }
578 }
```

Listing 16: The `updateChannelProperties` function.

The calculated carrier phase is applied to the simulated signal in `simulate`, with the relevant section provided in Listing 12. Pre-computed trigonometric tables for sine and cosine are used to enhance performance.

`tr-table-generator.c` in the `tools/` directory was developed to automatically generate trigonometric tables, enabling experiments on the effect of table resolution. However, time constraints prevented its characterization for this report. An example generated table is shown in Listing 17.

```
1 #ifndef TRIG_TABLES_H
2 #define TRIG_TABLES_H
3
4 const int sinTable[8] = {
5         0, 176, 250, 176, 0, -176, -250, -176, };
6
7 const int cosTable[8] = {
8         250, 176, 0, -176, -250, -176, 0, 176, };
9
```

```
#endif
```

Listing 17: Minimal example of trigonometric table generator output. Actual tables are larger.

#### 4.4.10.3 Doppler Shift

Signals emanating from a source with non-zero motion relative to a receiver will experience Doppler shift. Their wavelength increases when the source travels away from the receiver and decreases as they travel towards it. From the expected Doppler shift ranges in Table 1, it is evident the effect has a minimal impact on the data signal, though it is significant for the carrier.

`gnss-sim` uses Equation 1 in `updateChannelProperties` (Listing 16) to calculate carrier and code phase Doppler shifts, which are applied to the signals in `advanceChannelModulation` (Listing 15).

During testing, GNSS-SDR reported measured Doppler shifts in the orders of magnitude greater than 5 kHz. The `dumpChannelData` function was introduced to `debug.c` to log channel data for investigation.

Using the channel data logs and several `gnuplot` scripts, a large jump in pseudorange and pseudorange rate at the start of the simulation was found to correlate with each channel's excessive Doppler shift values. See the initial pseudorange rate Figure 29 plot below.

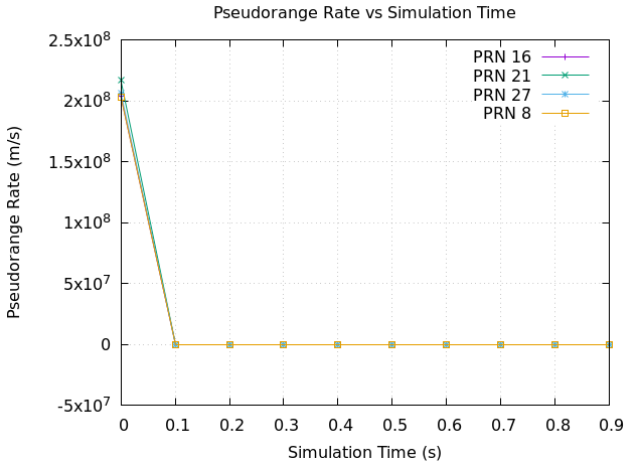


Figure 29: Pseudorange rate plot showing large jump on initialization. Notice the large spike at time 0 causing the axis auto scaling to grow substantially.

Upon realizing the issue was caused by a jump in a satellite's pseudorange value from the initialization value of 0 to its first legitimate value at the beginning of the simulation (which caused an impossible jump in pseudorange rate), a check was added to `updateSatellitePositions` for this condition, which resolved the matter. See Listing 18.

```
        // Critical check to prevent **MASSIVE** initial pseudorange rates which **WILL** completly mess up
             channel navbit and code chip pointers otherwise!
        if (svs[sv]->psuedorange_m == 0) {
            // Use the imagined previous time step
            gtime_t previousSimulationTime = timeadd(simulationTime, -timeStep_s);

            // Find previous satellite position in the ECEF frame
            eph2pos(previousSimulationTime, &svs[sv]->ephemeris, svs[sv]->position_ecef, &svs[sv]->
                clockBias_s, &svs[sv]->variance);

            // Find the previous satellite-receiver pseudorange
            svs[sv]->psuedorange_m = geodist(svs[sv]->position_ecef, receiverPosition_ecef,
                lineOfSightVector_ecef);
        }

        // Update satellite position in the ECEF frame (now using the current simulation time)
        eph2pos(simulationTime, &svs[sv]->ephemeris, svs[sv]->position_ecef, &svs[sv]->clockBias_s, &svs[sv
            ]->variance);

        // Calculate the current psuedorange
        psuedorange_m = geodist(svs[sv]->position_ecef, receiverPosition_ecef, lineOfSightVector_ecef);

        // Calculate the new psuedorange rate
        psuedorangeRate_ms = ((psuedorange_m - svs[sv]->psuedorange_m) / timeStep_s);
```

Listing 18: Section of code with the added pseudorange check from `updateSatellitePositions`.

A corrected pseudorange rate plot is shown in Figure 30.

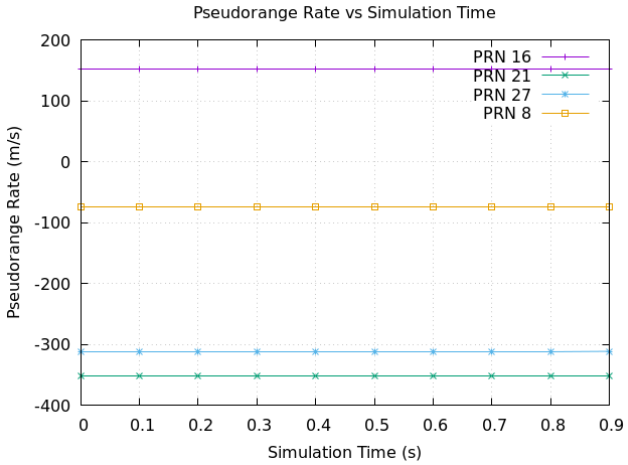


Figure 30: Pseudorange rate plot after correcting the large jump on initialization.

## 4.5 Future Work

The next phase of development should focus on resolving the current issues with the observables calculations driving the incorrect pseudorange values in GNSS-SDR. This necessitates a thorough investigation into the underlying issues causing erroneous position solving.

Furthermore, efforts should be directed towards enhancing the software's features. This includes enabling simultaneous operation with multiple modules, enhancing configuration options through additional command-line arguments, incorporating ionospheric correction into computed ranges, receiver position mobility and multiple constellation support.

Performance optimization is also important, particularly addressing the inefficiencies in the `advanceChannelModulation` function. While the software currently meets real-time requirements, optimizations are advisable to accommodate future expansions, such as the inclusion of additional satellite constellations.

Future enhancements will also encompass improvements in carrier phase initialization, attenuation modeling of satellite signals based on pseudorange, and consideration of antenna pattern-related attenuation. Finally, the implementation of multipath effects could be of academic interest.

## 4.6 Personal Reflection

Undertaking this project has been both stimulating and demanding. There's a certain satisfaction in tackling challenging tasks, but I've encountered some roadblocks along the way that have proven quite frustrating. Despite being on the verge of a breakthrough, I've struggled to resolve the issues preventing an initial position solution in GNSS-SDR from working.

I've come to realize that building something while simultaneously learning about it is a time-consuming process. Many of the obstacles I faced stemmed from gaps in my understanding of how GPS works. While my tool isn't quite perfect yet, I believe its underlying architecture holds promise. With further refinement, I believe it could become a strong competitor to `gps-sdr-sim`, thanks to its flexible design that allows for easy adaptation to other satellite constellations.

I am disappointed in the absence of my peers throughout the project. I believe that sharing insights and pooling our expertise could have smoothed out some of the technical difficulties we faced. It would at least have been beneficial to have a hand writing the final project report.

# References


[5] A. Raskaliyev, S. Patel, T. Sobh, and A. Ibrayev, "GNSS-Based Attitude Determination Techniques—A Comprehensive Literature Survey," *IEEE Access*, vol. PP, pp. 1–1, 01 2020.

[6] M. Gowda, J. Manweiler, A. Dhekne, R. R. Choudhury, and J. D. Weisz, "Tracking drone orientation with multiple GPS receivers." Association for Computing Machinery, 2016, p. 280–293.

[7] J. B. JOHNSON, "Thermal agitation of electricity in conductors," *Nature*, vol. 119, no. 2984, pp. 50–51, Jan 1927. [Online]. Available: https://doi.org/10.1038/119050c0

[8] [Online]. Available: https://science.nasa.gov/resource/solar-system-temperatures/

[9] "Global Positioning System (GPS) Signal Specification," U.S. Department of Defense, Tech. Rep., 1995. [Online]. Available: https://www.gps.gov/technical/ps/1995-SPS-signal-specification.pdf

[10] "GPS NAVSTAR User's Overview," U.S. Department of Defense, Tech. Rep., 1991. [Online]. Available: https://www.navcen.uscg.gov/sites/default/files/pdf/gps/GPSNAVSTARUsersOverviewMarch1991.pdf

[11] [Online]. Available: https://gssc.esa.int/navipedia/index.php/GPS_Space_Segment

[12] P. Steigenberger, S. Thölert, and O. Montenbruck, "GPS and GLONASS Satellite Transmit Power: Update for IGS repro3," 07 2019.

[13] ——, "GNSS Satellite Transmit Power and its Impact on Orbit Determination," *Journal of Geodesy*, vol. 92, 11 2017.

[14] A. Hussain, F. Akhtar, Z. Khand, A. Ali, and Z. Shaukat, "Complexity and Limitations of GNSS Signal Reception in Highly Obstructed Enviroments," *Engineering, Technology Applied Science Research*, vol. 11, pp. 6864–6868, 04 2021.

[15] Y. Lee and B. Park, "Nonlinear regression-based gnss multipath modelling in deep urban area," *Mathematics*, vol. 10, no. 3, 2022. [Online]. Available: https://www.mdpi.com/2227-7390/10/3/412

[16] [Online]. Available: https://www.u-blox.com/en/technologies/multipath-mitigation

[17] R. B. Langley, "Dilution of Precision," *"GPS World"*, 1999.

[18] R. Yarlagadda, N. Al-Dhahir, and J. Hershey, "Gps gdop metric," *Radar, Sonar and Navigation, IEE Proceedings*, vol. 147, pp. 259 – 264, 11 2000.

[19] [Online]. Available: https://en.wikipedia.org/wiki/Dilution_of_precision_%28navigation%29

[20] [Online]. Available: https://www.gps.gov/systems/gps/modernization/sa/faq/

[21] [Online]. Available: https://www.ofcom.org.uk/spectrum/information/gps-jamming-exercises

[22] [Online]. Available: https://www.gpsworld.com/increasing-gnss-interference-uk-and-eu-warn-aviation/

[23] [Online]. Available: https://space.stackexchange.com/questions/14687/current-situation-with-cocom-regulations-and-gps-receivers-for-balloons-and-cube/

[24] [Online]. Available: https://www.labsat.co.uk/index.php/en/products/labsat-3-wideband

[25] [Online]. Available: https://www.spirent.com/products/gnss-simulator-gss9000

[26] [Online]. Available: https://www.rohde-schwarz.com/uk/products/test-and-measurement/digital-standards/gnss-simulation_63493-1124871.html

[27] [Online]. Available: https://satsearch.co/products/gnssmart-gs-sim200-gnss-simulator-gnss-signal-simulation

[28] [Online]. Available: https://www.viavisolutions.com/en-uk/products/gpsg-1000-portable-satellite-simulator

[29] [Online]. Available: https://www.labsat.co.uk/index.php/en/applications/research-and-education

[30] [Online]. Available: https://github.com/osqzss/gps-sdr-sim

[31] [Online]. Available: https://www.youtube.com/watch?v=uEVsyBWtSVU

[32] [Online]. Available: https://www.youtube.com/watch?v=HfceySbuiBw

[33] [Online]. Available: https://gssc.esa.int/navipedia/index.php/GPS_Signal_Plan

[34] [Online]. Available: https://www.gps.gov/systems/gps/modernization/civilsignals/

[35] [Online]. Available: https://gssc.esa.int/navipedia/index.php/Galileo_Signal_Plan

[36] [Online]. Available: https://k3xec.com/packrat-processing-iq/

[37] W. Kim and J. Seo, "Low-Cost GNSS Simulators With Wireless Clock Synchronization for Indoor Positioning," *IEEE Access*, vol. 11, pp. 55 861–55 874, 2023.

[38] u-blox, *"NEO/LEA-M8T - Data sheet"*, 2023. [Online]. Available: https://content.u-blox.com/sites/default/files/documents/NEO-LEA-M8T-FW3_DataSheet_UBX-15025193.pdf

[39] [Online]. Available: https://portal.u-blox.com/s/question/0D52p00009lxHZqCAM/disabling-neom8t-antijamming-and-antispoofing

[40] [Online]. Available: https://wiki.myriadrf.org/Gr-limesdr_Plugin_for_GNURadio

[41] [Online]. Available: https://gnss-sdr.org/

[42] "NAVSTAR GPS Space Segment/Navigation User Segment Interfaces," U.S. Department of Defense, Tech. Rep., 2022. [Online]. Available: https://www.gps.gov/technical/icwg/IS-GPS-200N.pdf

[43] [Online]. Available: http://www.rtklib.com/prog/manual_2.4.2.pdf

[44] [Online]. Available: https://gssc.esa.int/navipedia/index.php/FDMA_vs._CDMA

[45] K. Borre, D. Akos, N. Bertelsen, P. Rinder, and S. Jensen, *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*, 01 2007.

[46] [Online]. Available: https://gssc.esa.int/navipedia/index.php?title=GNSS_Measurements_Modelling

[47] [Online]. Available: https://www.wasyresearch.com/generating-gps-l1-c-a-pseudo-random-noise-prn-code-with-matlab-

[48] [Online]. Available: https://gssc.esa.int/navipedia/index.php/GPS_Navigation_Message

[49] [Online]. Available: "https://www.youtube.com/watch?v=h_7d-m1ehoY"

[50] J.-H. Cho and S. Woo, *Communication Strategies for Various Types of Swallowable Telemetry Capsules*, 10 2011.

[51] A. Finance, C. Dufour, T. Boutéraon, A. Sarkissian, A. Mangin, P. Keckhut, and M. Meftah, "In-orbit attitude determination of the uvsq-sat cubesat using triad and mekf methods," *Sensors*, vol. 21, p. 7361, 11 2021.

[52] C. Specht and P. Dabrowski, "Runaway prn11 gps satellite," 04 2017.

[53] [Online]. Available: https://gssc.esa.int/navipedia/index.php?title=GNSS_Measurements_Modelling

[54] M. Hoffmann, P. Kryszkiewicz, and G. Koudouridis, "Modeling of real time kinematics localization error for use in 5g networks," *EURASIP Journal on Wireless Communications and Networking*, vol. 2020, 01 2020.