University Institute of Lisbon

Department of Information Science and Technology

Assessing spoofing of GPS systems

Rui Filipe Pereira Dias

A Dissertation presented in partial fulfillment of the Requirements for the Degree of

Master in Telecommunications and Computer Engineering

Supervisor

Ph. D. Francisco Cercas, Full Professor

ISCTE-IUL

Co-Supervisor

Ph. D. José Sanguino, Assistant Professor

IST-UL

September, 2019

Resumo

Ultimamente tem havido bastante desenvolvimento de viaturas que se deslocam automaticamente por sinais de radionavegação, como por exemplo drones ou, futuramente, carros autopilotados. No entanto, também é cada vez mais fácil forjar sinais de radionavegação, o que pode vir a ser um problema.

Com o crescimento desta ameaça também tem de haver uma preocupação em preveni-la e o objetivo desta dissertação é estudar formas de mitigar este problema. Para tal, foi usado um receptor de GNSS (Global Navigation Satellite System), u-blox evk-m8t, capaz de devolver dados brutos retirados da leitura dos sinais sem qualquer tipo de processamento. De maneira a analisar os dados foi usado um raspberry pi.

Este problema não é linear, visto que cada spoofer tem a sua especifidade, é necessário prestar atenção às transições comparando dados antigos com recentes.

Como cada cenário é diferente, as variações vão ser observadas de modo a tentar encontrar um padrão de variações. Estas variações serão testadas numa rede neuronal de modo a encontrar sinais falsificados.

Falsificação de sinais como um todo apresenta variações especificas que não deviam lá estar, a variação instável do relógio é o fator mais influenciável.

Este trabalho conseguiu concluir que é possível implementar um algoritmo de calibração que consegue detetar padrões em sinais ilegítimos e distingui-los de sinais legítimos. Os sinais falsificados normalmente são mais incongruentes no que toca a variações de propriedades de sinal e no seu funcionamento como um todo, como por exemplo a posição que seria calculada retirando um satélite da equação. Estes sinais também apresentam variações não previstas no atraso de relógio.

Palavras-chave: Radionavegação, defesa contra spoofing, falsificação, GNSS.

Abstract

Lately, plenty of self navigation vehicles have been developed, as drones, or in the future, self driving cars. However, it has become easier to forge radionavigation signals, which can be a problem.

With the growing risk of this threat, there has to be way to solve it and this thesis goal is to study various ways to mitigate this problem. For this effect, an u-blox evk-m8t GNSS (Global Navigation Satellite System) receiver was used, which is capable of returning raw unprocessed data from radio navigation signals. A raspberry pi was also used to analyze the data.

This is not a linear problem, since each spoofer is unique, it is necessary to pay attention to transitions, comparing old with new data.

Since each scenario is a different scenario, the variations will be observed in order to try to find a variation pattern. These variations will be tested in a neural network in order to find if it is viable to detect forged signals this way.

Spoofing as a whole also has specific variations that should not be there, the unstable clock variation is the most influenceable factor.

This work managed to conclude that it is possible to implement a calibration algorithm that is able to detect patterns in forged signals and distinguish them from legitimate signals. Forged signals, normally, are more incoherent in variations of signal properties and its functioning as a whole, for example, the position that would be calculated by removing a satellite from the equation. These signals also present unpredicted variations in the clock delay.

Keywords: radionavigation, anti spoofing; spoofing, GNSS.

Acknowledgements

I would like to acknowledge my supervisors, Francisco Cercas and José Sanguino, for all the support and guidance in this thesis. I would also like to acknowledge professor Luís Nunes and João Ponte for being always helpful and accessible.

Contents

Resumo **iii**

Abstract **v**

Acknowledgements **vii**

List of Figures **xi**

List of Tables **xiii**

Abbreviations **xiv**

1 Introduction **1**
1.1 Motivation and context . . . 1
1.2 Goals and research questions . . . 2
1.3 Contributions . . . 3
1.4 Dissertation Structure . . . 3

2 Literature Review **5**
2.1 GPS system overview . . . 5
2.1.1 GPS signal . . . 5
2.1.2 Pseudorange detection . . . 7
2.1.3 Navigation Data . . . 7
2.1.4 Satellite position calculation . . . 8
2.1.5 Sagnac effect . . . 10
2.1.6 Position fix . . . 11
2.1.7 Least Squares . . . 12
2.2 Neural Networks . . . 14
2.3 Related work . . . 15

3 Anti-spoofing techniques **19**
3.1 Hardware used . . . 19
3.1.1 U-Blox EVK-M8T . . . 19
3.1.2 Raspberry Pi . . . 20
3.1.3 Ettus N210 . . . 20
3.2 Software used . . . 21

3.2.1 u-center . . . 21
3.2.2 gnss-sdr-sim . . . 21
3.2.3 Neuroph studio . . . 22
3.3 Information transmission . . . 22
3.4 Raspberry pi implementation . . . 25
3.5 Flags to detect spoofed satellites . . . 28
3.5.1 Doppler shift . . . 28
3.5.2 C/N_0 . . . 28
3.5.3 Ephemeris integrity . . . 29
3.5.4 RAIM . . . 29
3.5.5 Expected range . . . 31
3.5.6 Excluding below the horizon satellites . . . 31
3.6 Flags to detect spoofing . . . 32
3.6.1 Predicting the clock bias . . . 32
3.6.2 Position variation . . . 33
3.6.3 Overall . . . 33
3.7 Using Neural Networks . . . 33
3.7.1 Detecting spoofed satellites . . . 33
3.7.2 Detecting spoofing . . . 34

4 Implementation results **35**
4.1 Observation . . . 35
4.1.1 C/N_0 variation . . . 35
4.1.2 Doppler shift . . . 36
4.1.3 RAIM . . . 37
4.1.4 Ephemerides variation . . . 37
4.1.5 Expected range . . . 37
4.1.6 Clock variation . . . 38
4.1.7 Distance between position fixes . . . 40
4.1.8 Difference between position fix and expected position . . . 41
4.1.9 Conclusion . . . 42
4.2 Using Neural Networks . . . 43
4.2.1 Detecting spoofed satellites . . . 44

5 Conclusion **47**
5.1 Future work . . . 48

Appendices **51**

A Code **51**

Bibliography **77**

List of Figures

2.1 An example of trilateration [1] 6
2.2 Legacy GPS satellite signal structure [2] 7
2.3 C/A code correlation [3] 8
2.4 Orbital parameters [2] 9
2.5 Satellite time correction [2] 10
2.6 Sagnac effect [4] 11
2.7 Structure of a Neural Network 14
2.8 Sigmoid function 15

3.1 Block diagram 19
3.2 U-Blox EVK-M8T [5] 20
3.3 Raspberry Pi 1 Model A [6] 20
3.4 Ettus N210 [7] 21
3.5 U-center 22
3.6 Neuroph Studio 23
3.7 UBX-RXM-RAWX Message structure [8] 23
3.8 Poll UBX-AID-EPH structure [8] 23
3.9 UBX-AID-EPH message structure [8] 24
3.10 Subframe two message structure [2] 25
3.11 Fluxogram of the system developed 26
3.12 Developed position calculator 27
3.13 Ephemris integrity 29
3.14 RAIM 30
3.15 Expected range, C/N_0 variation and Doppler shift 31

4.1 Clock bias fix 38
4.2 Clock bias prediction 39
4.3 Clock bias fix in spoofing environment 40
4.4 Clock bias prediction in spoofing environment 40
4.5 Distance shift between position fixes 41
4.6 Difference between position fix and expected position 42
4.7 Difference between position fix and expected position in a spoofing environment 42
4.8 Neural Network to detect spoofed satellites 44
4.9 Test scenario one results 45

List of Tables

4.1 C/N_0 variation 36

4.2 Doppler shift 36

4.3 RAIM posititon drift 37

4.4 Ephemerides variation 37

4.5 Expected range variation 38

4.6 Clock variation offset from range 40

4.7 Distance between position fixes 41

4.8 Difference between position fix and expected position 43

4.9 Conclusion on spoofed satellites detection 43

4.10 Conclusion on detecting spoofing 43

4.11 Neural Network data first scenario 45

Abbreviations

AI	Artificial Intelligence
BPSK	Binary Phase Shift Keying
BSSID	Basic Service Set Identifier
C/A	Coarse Aquisition
CDMA	Code Division Multiplexing Access
$\boldsymbol{C/N_0}$	Carrier to Noise density ratio
DSSS	Direct Spread Spectrum
ECEF	Earth-centered, Earth-fixed
GLONASS	Global'naya Navigatsionnay Sputnikovaya Sistema
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
GSM	Global System for Mobile Communications
IMU	Inertial Measurement Unit
ISCTE	Instituto Superior de Ciências do Trabalho e da Empresa
IST	Instituto Superior Técnico
MEO	Medium Earth Orbit
NORAD	North American Aerospace Defense Command
PRN	Pseudorandom Noise
RAIM	Receiver Autonomous Integrity Monitoring
RX	Reception
SSID	Service Set Identifier
TOW	Time of the week
SDR	Software Defined Radio
Wi-Fi	Wireless Fidelity

Chapter 1

Introduction

1.1 Motivation and context

Presently there are plenty of systems controlled by wireless communications, which, in turn, use radio navigation through satellite as way to determine its position to reach a predetermined location. With the evolution of technology and software defined radios it is easy to hack a wireless system, therefore, there's a need to know how to defend against these threats.

Most wireless systems nowadays, like cell phones or even ships, use GPS to determine its position. This is done by using trilateration of four or more satellites [1]. However, GPS (Global Positioning System) signals have low power and use DSSS (Direct Spread Spectrum) which is based on CDMA (Code Division Multiplexing Access), so it is possible for a remote system to forge these signals with a higher power. This problem could cause a ship to change its course [9], or a cell phone to show a wrong location.

An attacker can forge these signals by using SDR (software defined radios) which are programmable internally or by using software, like GNU radio, which processes the signal in the computer and uses the SDR as transceiver [10].

This project's objective is to avoid a malicious signal emitter from changing the system's predetermined mission. There are many ways in which this can be done, naming some, amplitude discrimination, in which signals with higher power than usual are rejected, angle of arrival, in which an array of antennas is set and if a signal is received with a different phase difference from the expected a forging is detected [11]. Due to the time it takes to determine one's location through only GPS, Apple also maintains a database of hotspots and cell towers to quickly determine its location [12], therefore it is also an effective way to determine the forging of GPS signals.

1.2 Goals and research questions

This thesis goal is to study effective ways to detect spoofing of radio navigation signals.

To accomplish this, a GPS receiver needs to be implemented based on an already existing one. To achieve this goal, different GNSS receivers will be tested in order to conclude which is the most effective one.

The first phase would be studying how GPS signals work and how to use them. The second phase would be testing various GNSS receivers. Finally, the third phase, would be implementing an anti-spoof solution in the GNSS receiver.

Concluding, this thesis final product will be an anti-spoof GPS system and, if possible, it will use other GNSS systems.

That being said, this thesis looks to answer some questions:

- Is it possible to make a spoofing free system?
- Is it possible to use it in an efficient way?
- Will it be useful in the marketplace?
- What is the most effective way to do it?

1.3 Contributions

This dissertation presents the following contributions:

- It reviews the existing approaches;
- It does a study on how effective each measure is;
- It makes a system that analyzes all of the existing approaches and through artificial intelligence it decides whether the signal is legitimate or not;
- It introduces new spoofing countermeasures like predicting the clock variation and fixing a position with this prediction.

The work conducted in this dissertation resulted in one publication:

- R. Dias, F. Cercas, J. Sanguino, J. Ponte, "Assessing spoofing of GPS systems", ConfTele 2019 - 11th Conference on Telecommunications, June, 2019

1.4 Dissertation Structure

This dissertation is composed of five chapters. The first chapter introduces the dissertation theme, motivation and research questions, contributions and a short summary of the dissertation structure.

The second chapter is a revision of theoretical aspects and related work relevant to this dissertation.

The third chapter is about the implementation of the anti-spoofing techniques and how the system was constructed.

The fourth chapter contains the experimentation results of the techniques mentioned in the previous chapter and its analysis.

In the fifth chapter the conclusions of the work are presented, as well as suggestions for future work.

Chapter 2

Literature Review

In this chapter the theoretical basis for this thesis is introduced, namely how GPS systems work.

2.1 GPS system overview

2.1.1 GPS signal

GNSS - Global Navigation Satellite System is the general designation for radionavigation constellations which includes systems as GPS - Global Positioning System, Beidou, GLONASS and Galileo.

GPS constellation has currently 31 satellites which have a MEO - Medium Earth Orbit with a 12 hour orbit. This system has multiple bands, however the main focus of this work will be on L1 band which is centered at 1575.42MHz. In order to fix a position, trilateration is used. Knowing where multiple sources are and how much time the signal takes to arrive, it is possible to set a range of the distance travelled.

Figure 2.1 shows how trilateration would work. Knowing where Foghorn 1, 2 and 3 are, and knowing when they are going to transmit, it is possible to a draw circle of the range the signal has travelled, by crossing the three circles it is possible to fix a position, in this scenario, it is A. However, this assumes the receiver's clock is synchronized with the Foghorn's, and that is not the case, so this problem would require at least four satellites to solve a four variable problem.

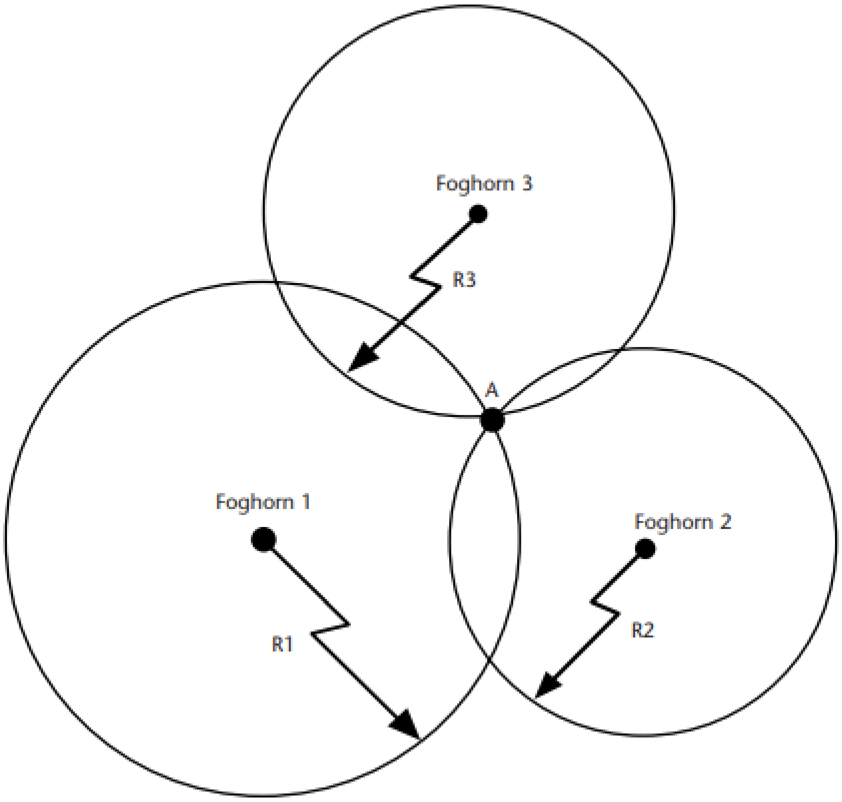


FIGURE 2.1: An example of trilateration [1]

GPS signals use DSSS - Direct Spread Spectrum which is based on CDMA - Code Division Multiplexing Access. Each satellite has a specific PRN - Pseudo-random noise code also known as C/A - Coarse Acquisition which is the civilian access code. This code has a chiprate of 1Mb/s and is xored with data which has a rate of 50b/s. The resulting signal is BPSK - Binary Phase Shift Keying modulated in the L1 carrier, that means that the phase is 180 degrees when there is a bit with a logic value of 1 or 0 degrees when the bit has the logic value of 0. This signal is mixed with an P(Y) encrypted code xored with data carrier with a 90 degrees offset. The P(Y) code is only for military use. This process is illustrated in Figure 2.2 [1].

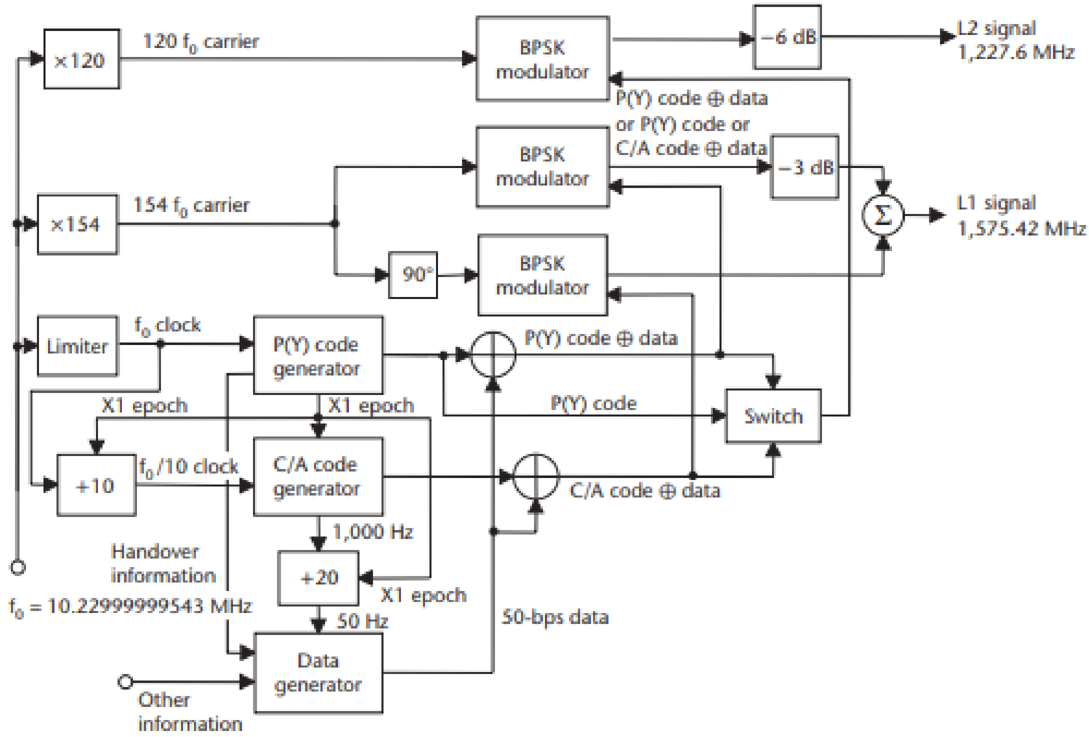


FIGURE 2.2: Legacy GPS satellite signal structure [2]

2.1.2 Pseudorange detection

In order to acquire a lock, the receiver has multiple channels that use signal replicas of the respective PRN code. It does this to achieve a auto-correlation with the incoming signal, when there is a lock there will be a positive or negative peak, depending on the value of the navigation data bit [3]. The local replica rotates until there is a peak, in order to find in which chirp bit it is and to know when the first arrived. Having one milliseconds marks, it is possible to know the propagation delay with the clock bias. Figure 2.3 illustrates this process.

2.1.3 Navigation Data

Like shown in Figure 2.2 the navigation data has a 50b/s bitrate, which is much lower than the chiprate of the code. Navigation data needed to calculate the satellite position is subdivided into three subframes. These subframes contain the following polynomials values. Figure 2.4 shows the needed parameters [2].

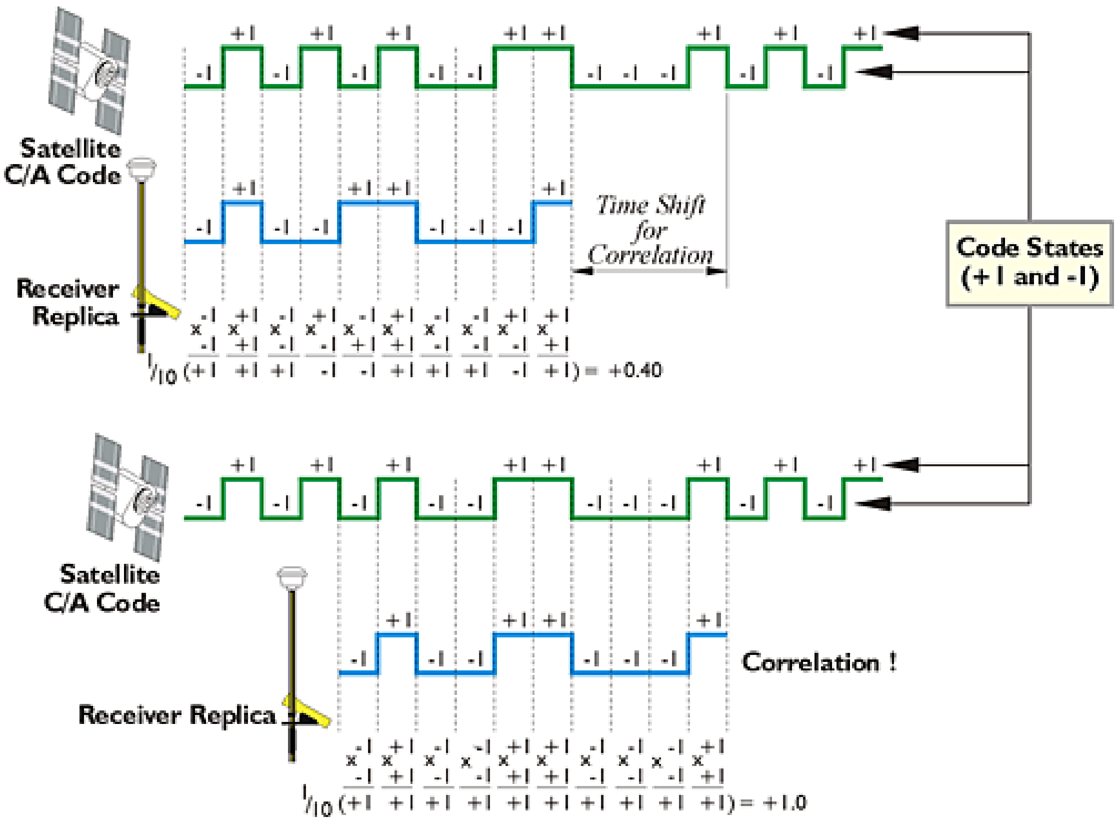


FIGURE 2.3: C/A code correlation [3]

2.1.4 Satellite position calculation

The data referenced in the previous section contains ephemeris parameters which can be used to calculate a satellite's position at a given time, in order to retrieve accurate results the time of transmission should be used. However, the pseudoranges need to be corrected because the satellites are not in total synchronization between them. Ignoring troposphere and ionosphere propagation delay, the time of transmission would be:

$$t = rcvTow - \frac{pseudorange}{c} \tag{2.1}$$

Where *rcvTow* is the time of reception of the signal where the time of travel is subtracted, since the *pseudorange* and *rcvTow* both contain the same clock bias,

M_0	Mean Anomaly at Reference Time
Δn	Mean Motion Difference From Computed Value
e	Eccentricity
$\sqrt{A}$	Square Root of the Semi-Major Axis
Ω_0	Longitude of Ascending Node of Orbit Plane at Weekly Epoch
i_0	Inclination Angle at Reference Time
ω	Argument of Perigee
$\dot{\Omega}$	Rate of Right Ascension
IDOT	Rate of Inclination Angle
C_{uc}	Amplitude of the Cosine Harmonic Correction Term to the Argument of Latitude
C_{us}	Amplitude of the Sine Harmonic Correction Term to the Argument of Latitude
C_{rc}	Amplitude of the Cosine Harmonic Correction Term to the Orbit Radius
C_{rs}	Amplitude of the Sine Harmonic Correction Term to the Orbit Radius
C_{ic}	Amplitude of the Cosine Harmonic Correction Term to the Angle of Inclination
C_{is}	Amplitude of the Sine Harmonic Correction Term to the Angle of Inclination
t_{oe}	Reference Time Ephemeris (reference paragraph 20.3.4.5)
IODE	Issue of Data (Ephemeris)

FIGURE 2.4: Orbital parameters [2]

it gets canceled. To apply the satellite's clock correction the following term needs to be calculated:

$$\Delta t_{sv} = a_{f0} + a_{f1}(t - t_{oc}) + a_{f2}(t - t_{oc})^2 + \Delta t_r \tag{2.2}$$

Where a_{f0}, a_{f1} and a_{f2} are the polynomial coefficients retrieved in ephemeris subframe one, t_{oc} is time of clock referenced in seconds and Δt_r is as follows:

$$\Delta t_r = Fe\sqrt{A}sin(E_k) \tag{2.3}$$

Where $\sqrt{A}$, e and E_k are orbital parameters given in the ephemeris. F is a constant value. E_k is calculated through iteration having already a transmission time, so for a first approximation the equation 2.1 can be used and then Δt_{sv} can be calculated and the new E_k as well.

Figure 2.5 shows how GPS time is corrected. As mentioned before, besides the clock bias of the user, three more things influence the imprecision. The Ephemerides contain the parameters needed in order to determine how much a clock has drifted over a period of time and parameters to determine ionospheric delays. Troposphere corrections require additional models which vary with the weather [2].

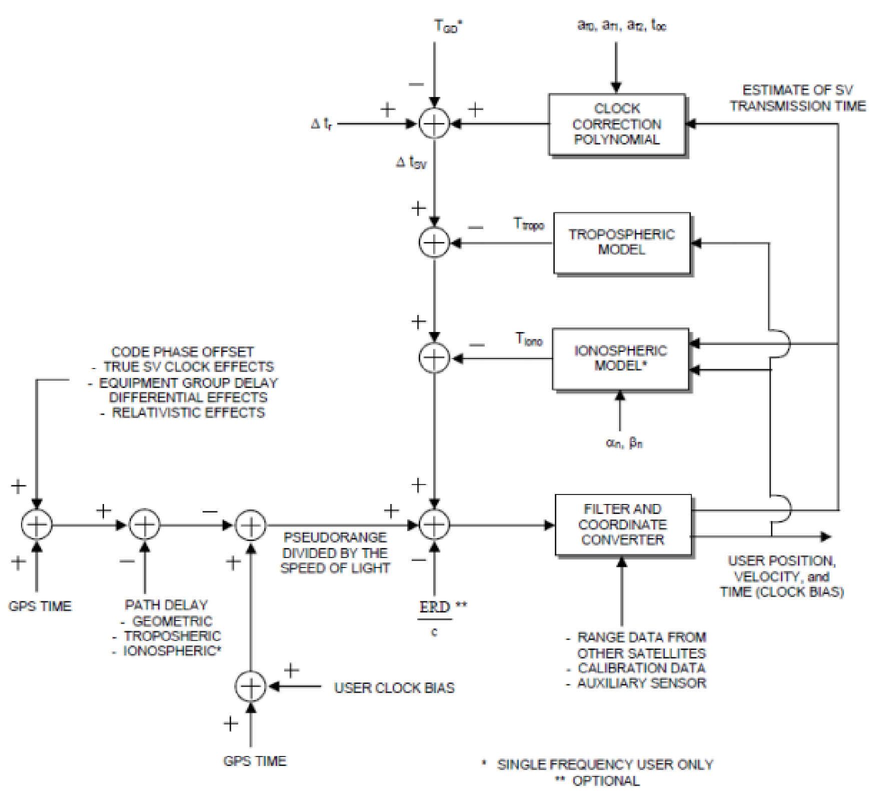


FIGURE 2.5: Satellite time correction [2]

2.1.5 Sagnac effect

The developed position calculator also takes into account the Sagnac effect which gives an error of around 20 meters. This effect works on the earth rotation, when the receiver measures the pseudoranges, the signal that is arriving it is not a direct one, since the earth has moved.

Figure 2.6 illustrates this phenomenon. On the left side the circle is not moving, so the signal in both directions travels the same distance. On the right side, the

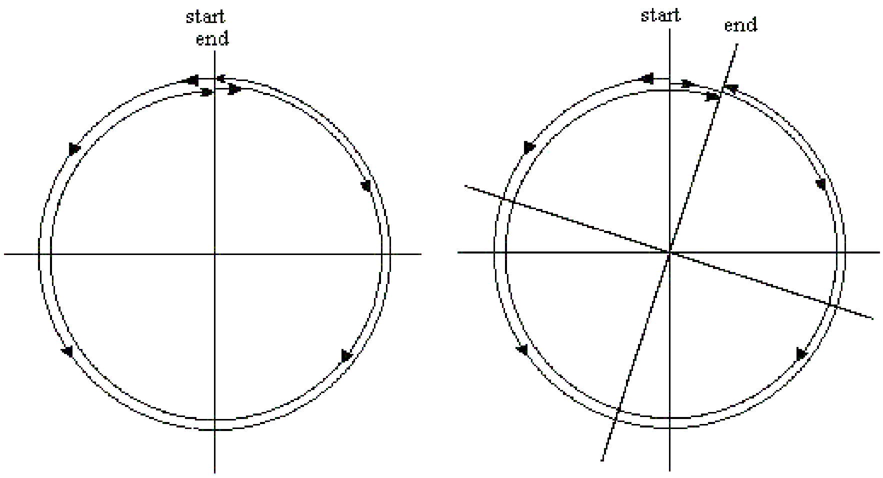


FIGURE 2.6: Sagnac effect [4]

circle has moved, so the signal traveling in the counter clockwise direction travels a smaller distance and the signal traveling in the clockwise direction travels a larger distance [4].

2.1.6 Position fix

In order to fix a position, as mentioned before, four satellites are needed to solve a four equation system. The pseudorange to a satellite can be written as:

$$p = ||s - r|| + c\Delta t \tag{2.4}$$

Where p is the pseudorange, s is the position of the satellite, r the position of the receiver, *//s-r//* is the distance between the satellite and the receiver, c is the speed of light and Δt is the receiver clock bias. The position of the satellite can be calculated using the ephemeris parameters, so this equation has four variables, the coordinates of the receiver, x,y,z and the clock bias. By, stacking four pseudorange measurements, a matrix of equations can be assembled in order to fix a position.

$$\begin{aligned}
p^1 &= \sqrt{(x^1 - x)^2 + (y^1 - y)^2 + (z^1 - z)^2} + c\Delta t \\
p^2 &= \sqrt{(x^2 - x)^2 + (y^2 - y)^2 + (z^2 - z)^2} + c\Delta t \\
p^3 &= \sqrt{(x^3 - x)^2 + (y^3 - y)^2 + (z^3 - z)^2} + c\Delta t \\
p^4 &= \sqrt{(x^4 - x)^2 + (y^4 - y)^2 + (z^4 - z)^2} + c\Delta t
\end{aligned} \tag{2.5}$$

Where x^n, y^n and z^n are the nth satellite's coordinates in ECEF format [1].

2.1.7 Least Squares

Sometimes there are more than four satellites visible, and having only four variables, it is preferable to use as many measurements as possible. This problem can be solved using the least squares algorithm that produces a solution approximation to overdetermined systems in which there are more equations than variables.

$$Z = Hx \tag{2.6}$$

Where Z is a matrix of n lines and one column, n is respective to the number of observations. Matrix x has one column and four lines respective to the position of the receiver and its clock bias. H is an n by four matrix. The x matrix can be isolated.

$$H^{-1}Z = x \tag{2.7}$$

When there are four observations, H will be a four by four matrix and so will its inverse, Z will be a four by one matrix. In this case there won't be any problem multiplying this matrices, because H is a square matrix and therefore it has an inverse, however if there are more than four observations H is not going to have an inverse matrix. However rewriting equation 2.6 the following way, removes this problem.

$$x = (H^T H)^{-1} H^T Z \tag{2.8}$$

Equation 2.8 allows multiple observations, however the equations of the position fix need to be represented in this format. An observation can be written as following.

$$p^j = ||s^j - r|| + c\Delta t \tag{2.9}$$

Where p^j is the pseudorange of the satellite j measured by the receiver, s^j is the position of the satellite j, r is the position of the receiver, c is the speed of light and Δt is the clock bias of the receiver.

The least squares method is iterative and through trial and error tries to find an approximation to the solution. The position of the receiver, r, wants to be known, so by linearizing the equation around r_0 an approximation can be obtained. The first estimation can be any set of values, however this is a linear system and is only valid for the values near r_0, so if the differences between r, the solution, and r_0, the estimation, are too big, then the solution is not considered valid or reliable. If r is near r_0, it means that there is a low error since the solution is close to the point where the approximation was made.

$$p^j - {e_0^j}^T s^j = -{e_0^j}^T r + c\Delta t \tag{2.10}$$

Where p^j is the pseudorange of the satellite j measured by the receiver, ${e_0^j}^T$ is the transposed normalized vector between satellite j and the estimation of the receiver r_0, r is the position of the receiver, c is the speed of light and Δt is the clock bias. This equation can now be stacked and converted to the *Z*=*Hx* format as follows [13].

$$\begin{bmatrix} p^1 - e_0^{1T} s^1 \\ p^2 - e_0^{2T} s^2 \\ p^3 - e_0^{3T} s^3 \\ p^4 - e_0^{4T} s^4 \end{bmatrix} = \begin{bmatrix} -e_0^{1T} & 1 \\ -e_0^{2T} & 1 \\ -e_0^{3T} & 1 \\ -e_0^{4T} & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ c\Delta t \end{bmatrix} \tag{2.11}$$

2.2 Neural Networks

In this project neural networks were tested in order to achieve the desired result since only the variance of parameters are measured, this algorithm would try to find a pattern.

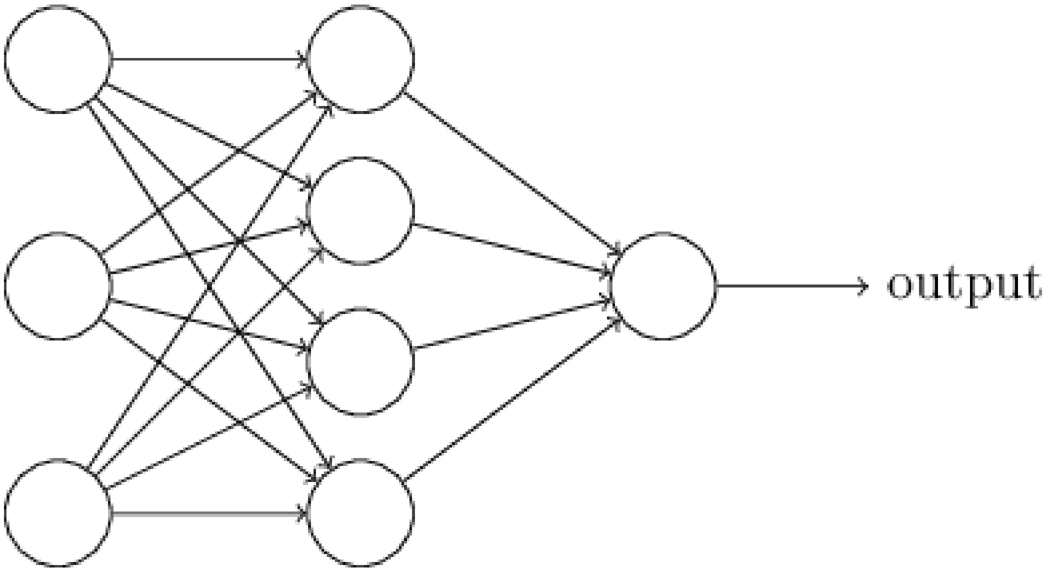


FIGURE 2.7: Structure of a Neural Network

In this scenario there is a neural network with three layers, input, hidden and output, having three, four and one nodes respectively. Each node of the nth layer value depends on the sum of the values from the nodes in the previous layers and multiplied by calculated weights.

$$value = f(\sum_j w_j x_j) \tag{2.12}$$

Where the value is respective to a node in the nth layer, w_j is respective to the weight of the node j of the nth-1 layer and x_j is respective to its value. The resulting sum goes into an activation function to introduce non-linearity between the input and the output, the simplest activation function would be the step function, that is, if the sum is above a given threshold then value would be equal to one, otherwise it would be equal to zero. However, a lot of values would be lost in this scenario, so the commonly used activation method is the Sigmoid function.

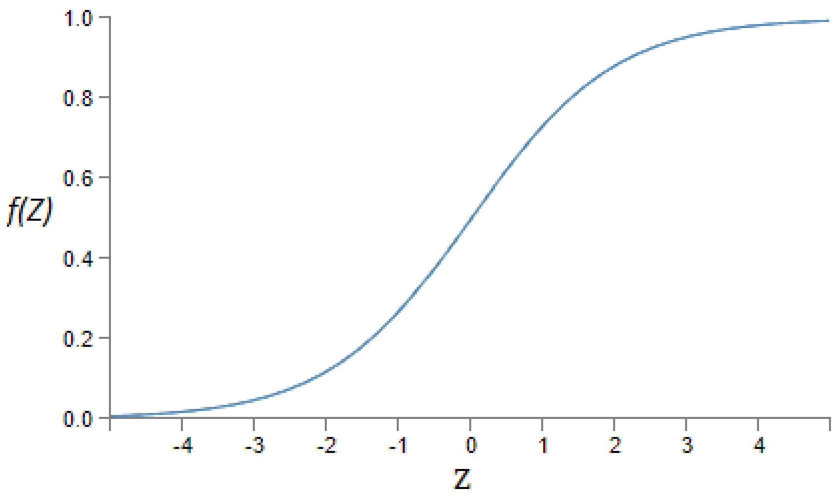


FIGURE 2.8: Sigmoid function

Where Z is the function input and y-axis is the output, so all values are between zero and one [14].

$$f(z) = \frac{1}{1 + e^{-z}} \tag{2.13}$$

2.3 Related work

Todd E. Humphreys et al present anti-spoofing solutions in [4]. In this paper anti-spoofing techniques are discussed and then presented in which way it can fail.

This paper suggests six ways to prevent spoofing, amplitude discrimination, time-of-arrival discrimination, navigation inertial measurement unit (IMU) cross-check, polarization discrimination, angle of arrival discrimination and cryptographic discrimination. The first and second method would only work against the most simple spoofing systems. The third, fourth and fifth methods require additional hardware however they are more effective.

At least 20 ships in the black sea got their course changed according to [2]. Fake signals were sent in a subtle way in order not to change the ship's course abruptly but smoothly. This website alerts to the danger of GPS spoofing and how it is becoming easier, this way self-driving vehicles or autonomous ships could be hijacked.

According to [5] Apple maintains a database of Wi-Fi hotspots and cell towers around one's location in order to calculate its position faster, because using just GPS data could take minutes to get a fix. In this paper, fake SSIDs and BSSIDs are generated in order to test this theory. After a while, the position is changed.

The work in [6] uses a two antenna array separated by 1.46 meters oriented along the true North-South axis to detect spoofing. In this paper the expected carrier phase differences are calculated for each satellite. If the measured delta phase doesn't match the profiled expected value a spoofing signal is identified. The units used in this difference are L1 cycles.

The work in [7] suggests some ways to achieve the desired goal. The first one is to monitor the absolute power of each carrier, that is, ignore signals with a power higher than a given threshold. The third method suggests comparing L1 and L2 frequencies power. The fifth method suggests checking the Doppler shift, by obtaining the receiver's relative speed with respect to the satellite it can be compared to the carrier frequency received.

$$f = f_0 \frac{c + v_r}{c + v_s} \tag{2.14}$$

Where f_0 is the frequency emitted by the transmitter, in this case the satellite, v_r is the velocity of the receiver, v_s is the velocity of the source and c is the velocity of the signal. If the receiver is moving towards the source, v_r is positive and if the source is moving away from the receiver, v_s is positive. In Fig. 1 this effect can be observed. When the source of the waves, the ambulance, is moving towards the observer each successive wave is moving closer to him, decreasing the wavelength and increasing the frequency.

The ninth method suggests comparing known ephemeris data to the one received in order to check for anomalies in the satellite's position. This method would require an internet connection to obtain such data from NORAD which sometimes might not be practical. The tenth method suggests that data relating to power and position should be monitored in order to find abrupt changes. However, a clever attacker might be able to fool the system, like mentioned before, a ship's course was gradually changed having a smooth transition and not raising any flags.

The tenth method suggests that data relating to power and position should be monitored in order to find abrupt changes. However, a clever attacker might be able to fool the system, like I mentioned before, a ship's course was gradually changed having a smooth transition and not raising any flags.

The work at [8] also suggests cryptographic authentication and it's something that's already used in P(Y) code which is a military grade encrypted signal. Implementing this in the civilian C(A) code would require changes to the GPS legacy signal. Also most GPS devices developed until now would not be able to decrypt the signal if changes were made. Although, if made properly, it would be a good defence against spoofing, it's not feasible, at least not for now.

The work at [9] suggests using a M-Estimator based extended Kalman filter which is able to provide an accurate position in the presence of outlying errors due to spoofing. It takes into account the user's position, velocity, clock bias and clock drift to make a prediction based on previous values and compare them to the current received ones. If the error is large, the weight matrix decreases, if the error is small, the weight matrix is not influenced.

The work at [4] suggests using vestigial signal defense. A receiver copies the incoming digitized front-end data into a buffer. After that, the receiver selects one of the various GPS signals being tracked, then it removes the signal from the buffered data. Once this signal has been removed from the buffered data, the receiver performs acquisition for the signal with the same PRN identifier in the buffered data. These steps are repeated over and over and the results are summed until the signal meets a desired C/N0 threshold.

Chapter 3

Anti-spoofing techniques

3.1 Hardware used

The U-Blox EVK-M8T was connect via UART Serial, which is shown in the block diagram below.

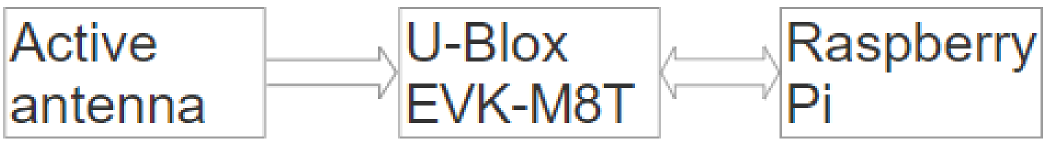


Figure 3.1: Block diagram

3.1.1 U-Blox EVK-M8T

In order to retrieve raw GPS data a GPS receiver is needed, the one used was U-Blox EVK-M8T. This device returns all types of raw information, from sinal properties like Doppler shift and carrier to receiver noise density ratio to signal observations like pseudoranges and ephemerides.

FIGURE 3.2: U-Blox EVK-M8T [5]

3.1.2 Raspberry Pi

Raspberry Pi is a microcomputer which allows processing of the data incoming to it. In this scenario binary data was being received via the RX pin, in order to read the incoming data, a binary parser was developed. This parser would deconstruct the frames and store the respective variables.

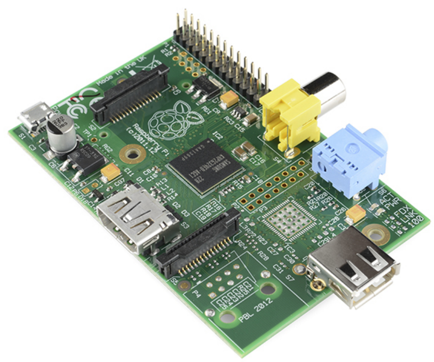


FIGURE 3.3: Raspberry Pi 1 Model A [6]

3.1.3 Ettus N210

Ettus N210 is a software defined radio board which allows the transmission and reception of sinals, aswell as signal processing, through internal programming or using the computer as the processing unit and this device as the transceiver [7].

In this scenario there was a need to have a spoofer in order to retrieve values and find patterns. An open source spoofer was used, using this device as the transmitter. The software used was gps-sdr-sim, which takes as input an ephemerides file and a position, with that information it generates fake signals posing as a genuine satellite [15].

FIGURE 3.4: Ettus N210 [7]

3.2 Software used

3.2.1 u-center

U-center is a visual interface software developed for Windows which allows the user to analyze real time the data being returned from the u-blox device. It also allows the user to configure the device settings, like which messages should it return, which GNSS constellations should it be looking for, refresh rate and many other parameters.

3.2.2 gnss-sdr-sim

Like mentioned before, a spoofer was needed to infer some kind of pattern and distinguish it from the real signals. This program takes as input a position and ephemerides. It generates a binary file based on the specifications needed, and after that the spoofer can be executed through the ettus n210 board.

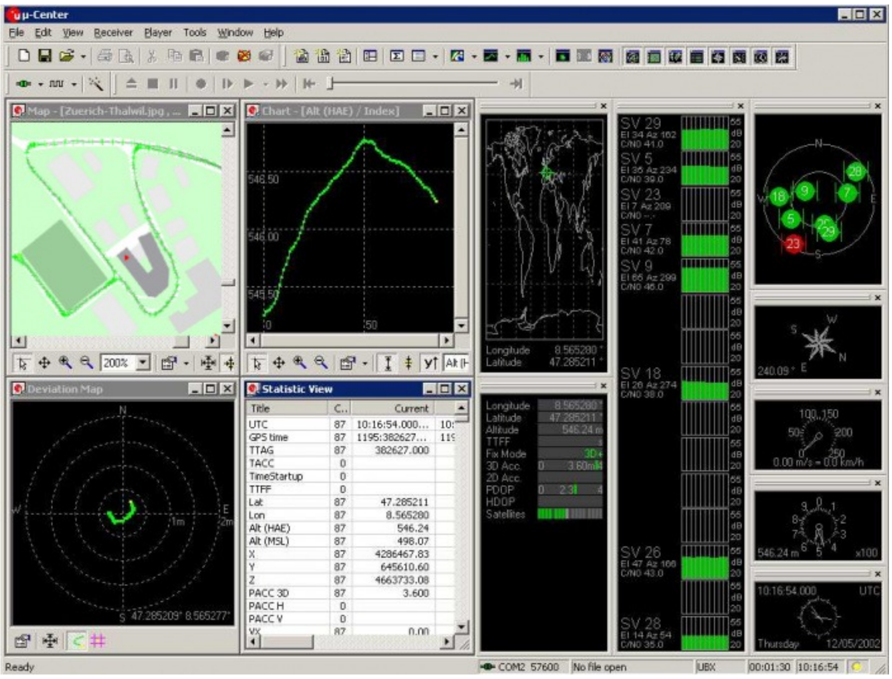


FIGURE 3.5: U-center

3.2.3 Neuroph studio

In order to discover some kind of pattern, a neural network was tested. After retrieving data from the u-blox device, using a developed python script running in the raspberry pi, a neural network was trained. This program trains the algorithm based on a previously given dataset, the number of neurons per layer are adjustable, as well as the number of layers.

3.3 Information transmission

The required information is transmitted via UART from the u-blox device to the raspberry pi. Figure 3.7 shows the structure of the UBX-RXM-RAWX message which contains signal properties like pseudoranges and Doppler shift measurements.

The developed program reads the buffer and checks if the header, class and ID match with the given values. In this case, it was done in a way that allows

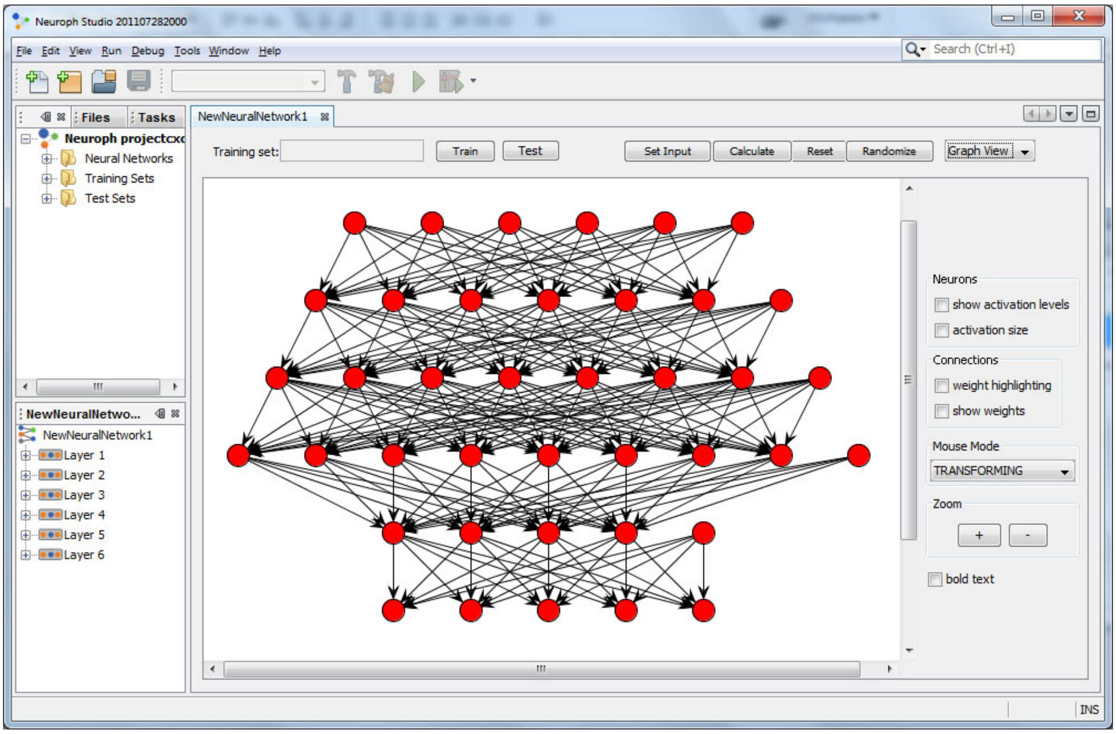


FIGURE 3.6: Neuroph Studio

	Header	Class	ID	Length (Bytes)	Payload	Checksum
Message Structure	0xB5 0x62	0x02	0x15	16 + 32*numMeas	see below	CK_A CK_B

FIGURE 3.7: UBX-RXM-RAWX Message structure [8]

the reading of multiple measures from different satellites through the "numMeas" field which indicates how many measurements there are in a message.

After receiving this information, the ephemeris of a satellite is polled by constructing the message in Figure 3.8.

	Header	Class	ID	Length (Bytes)	Payload	Checksum
Message Structure	0xB5 0x62	0x0B	0x31	1	see below	CK_A CK_B

Payload Contents:

Byte Offset	Number Format	Scaling	Name	Unit	Description
0	U1	-	svid	-	SV ID for which the receiver shall return its Ephemeris Data (Valid Range: 1 .. 32).

FIGURE 3.8: Poll UBX-AID-EPH structure [8]

The data is transmitted in little endian format, which consists in transmitting the least significant bytes first in order to facilitate the storage in the receiver. This way the least significant byte is stored in a lower register address and the most significant byte is stored in a higher register address, the developed program took this in consideration. Only the byte order is litle endian, the bit order is big endian.

Figure 3.9 shows the structure of the UBX-AID-EPH, it has the three subframes of navigation data that contain the parameters to calculate the satellite position.

	Header	Class	ID	Length (Bytes)	Payload	Checksum
Message Structure	0xB5 0x62	0x0B	0x31	(8) or (104)	see below	CK_A CK_B

Payload Contents:

Byte Offset	Number Format	Scaling	Name	Unit	Description
0	U4	-	svid	-	SV ID for which this ephemeris data is (Valid Range: 1 .. 32).
4	U4	-	how	-	Hand-Over Word of first Subframe. This is required if data is sent to the receiver. 0 indicates that no Ephemeris Data is following.
Start of optional block					
8	U4[8]	-	sf1d	-	Subframe 1 Words 3..10 (SF1D0..SF1D7)
40	U4[8]	-	sf2d	-	Subframe 2 Words 3..10 (SF2D0..SF2D7)
72	U4[8]	-	sf3d	-	Subframe 3 Words 3..10 (SF3D0..SF3D7)
End of optional block					

FIGURE 3.9: UBX-AID-EPH message structure [8]

Figure 3.10 shows the parameters that the subframe two has. Each subframe is divided into ten words, however the u-blox device only returns words three to ten. Each word has 24 bits without the parity bits, which are three bytes. Figure 3.9 shows that each subframe transmitted by the u-blox will have 32 bytes. From word three to ten, there are eight words which amount to 24 bytes, the rest are delimiters between words with the 0x00 value. Since the bytes come in litle endian order, the parameters will need some rearrangements, for example, looking at Figure 3.9 at word three, IODE will not be the first byte but the last one.

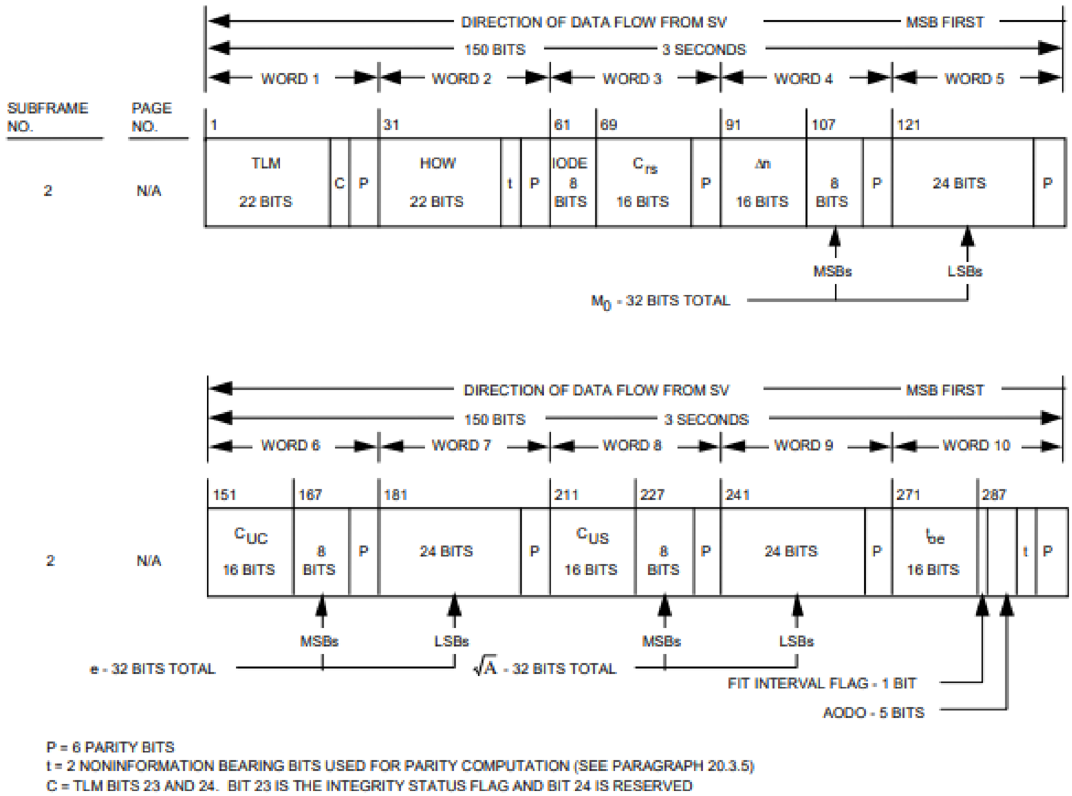


FIGURE 3.10: Subframe two message structure [2]

After unpacking the needed information, it is stored in the system by doing the necessary conversions.

3.4 Raspberry pi implementation

Before implementing any anti-spoofing measures, there needs to be an understanding on how the receiver is working. If it is just returning a position, there is no way to know which corrections where made to it. So, in order to understand exactly what is happening, a GPS position calculator was developed which would do its calculations based on raw data and ephemerides. Clock drift data was used for spoofing detection measures, not being needed to fix a position.

The fluxogram in Figure 3.11 explains the logic behind this implementation. For every one minute that passes, there is a verification on the number of satellites

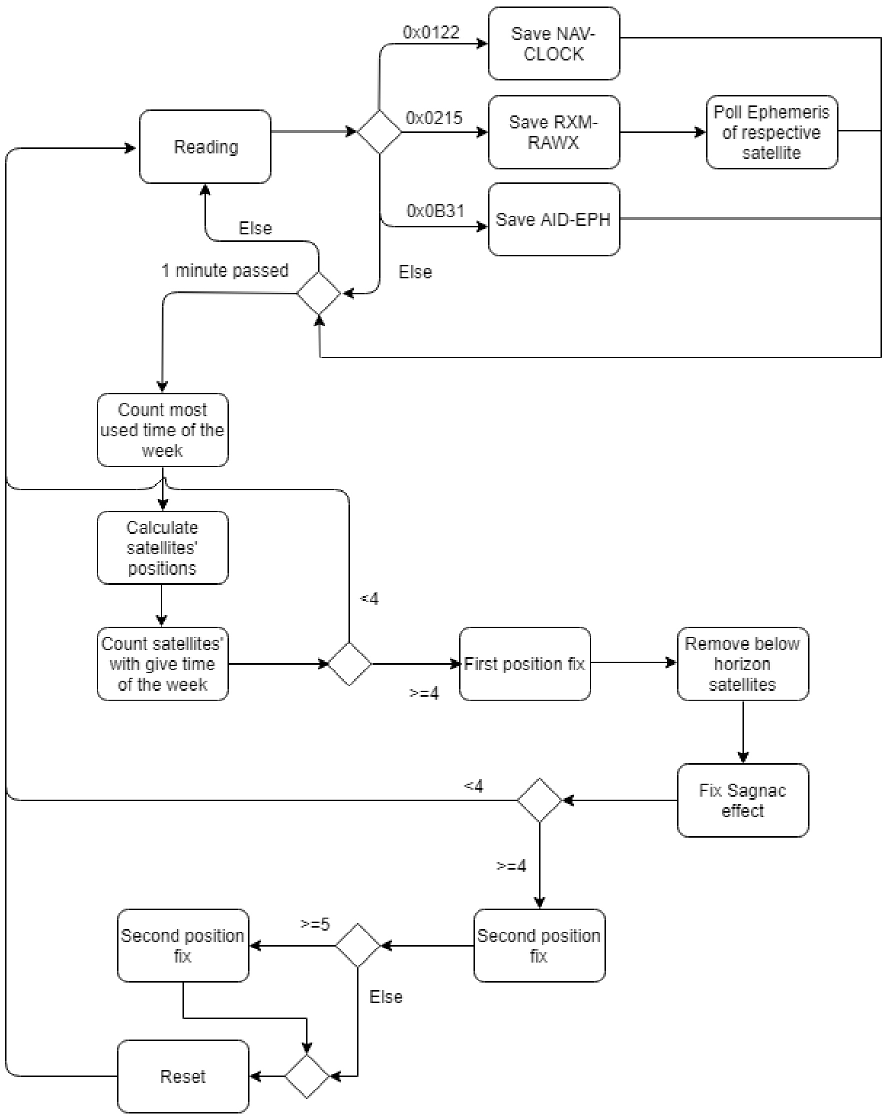


Figure 3.11: Fluxogram of the system developed

and if it is possible to get a fix. It was done this way in order to give a chance for the receiver to transmit as many data as possible. As mentioned before, the NAV-CLOCK is not relevant to fix a position. When RXM-RAWX data, respective to a satellite, is received, the raspberry pi immediately polls for the respective ephemeris. RAWX data is received multiple times in order to check for variation on signal properties like pseudoranges, Doppler effect and others. If the raspeberry pi already has a given ephemeris, it won't poll it again, not until it is reseted.

After the one minute mark, there is a counting process in order to find which time of the week is in majority. In this scenario, pseudoranges are associated to a given received time of the week which indicates at which instant this measurement was received. After calculating the position of the satellites and excluding the ones which have data relative to different instants, there is a recount. If, after this exclusion process, four satellites are still available, the program attempts the first position fix, otherwise it returns to the reading activity.

Two position fixes are needed in order to exclude below the horizon satellites which might be affecting the position calculation through multipath transmissions and to fix the Sagnac effect. After that, there is a recount, if there are not at least four satellites, the program returns to the reading activity in order to find more satellites. Removing below the horizon satellites in this process not only excludes multipath problems, but also spoofed signals which should not be visible.

After fixing the second position, epheremides are erased in the reset activity.

```
-----POSFIX:-----
Latitude: 38.7489337277
Longitude: -9.1531008007
Altitude: 154.335913626
Delta_clock: -0.000604451045849
Tow: 322265.999322
-----------------
```

FIGURE 3.12: Developed position calculator

Sometimes two satellites will be near each other and the H^TH matrix will be singular, that is, non invertible. To solve this problem, this program adds noise to the matrix until the determinant is different than zero, thus making the matrix invertible. The other way to solve this is to remove one of the satellites in conflict. A matrix is non invertible when the determinant is zero. This program iterates a while loop until the determinant is different than zero adding a four by four matrix of noise containing the value 0.00001. Both the H^TH matrix and the noise matrix are four by four.

3.5 Flags to detect spoofed satellites

There is no straight forward way to detect spoofing or satellite's that are not real, it is all about paying attention to transitions and finding the odd variations. In order to know exactly what is happening, the algorithm to fix a position was programmed. It collects ephemerides and signal related information in order to this. It also collects clock drift values in order to predict positions which is talked about further ahead. This section presents techniques to detect forged signals from specific signals.

3.5.1 Doppler shift

From equation 1.1, the following can be deduced.

$$\Delta f = f_0 \frac{\Delta v}{c} \tag{3.1}$$

Where f_0 is the GPS L1 band frequency, 1575.42 MHz, c is the speed of light. By collecting pseudorange data in two instants, a satellite's speed relative to the receiver, Δv can be infered, by subtracting the pseudoranges and dividing them by the time difference. This value can be compared against the measured Doppler shift in the integration stage. See Figure 3.15.

3.5.2 C/N_0

Carrier to noise density ratio, also know as the ratio of carrier power and the noise power per bandwith unit can also be used to determine strange variations. Usually spoofed signals have high power, so if one signal has an abrupt transition, it should be suspected. See Figure 3.15.

3.5.3 Ephemeris integrity

In the receiver implemented, for each position fix, an ephemeris for each satellite is polled. That way for every new calculations there are always new ephemeris. By storing the old ephemerides and comparing them against the new ones when calculating the satellite's position, both positions can be compared, in order to find abrupt changes.

```
------------------
--Sat Pos 21: New Eph--
****
X: 10465999.9264136
Y: 11933311.5408138
Z: 22100898.5375861
---------
--Sat Pos 21: Old Eph--
****
X: 10465999.5123386
Y: 11933311.8873755
Z: 22100898.5498615
---------
--Sat Pos 16: New Eph--
****
X: 18828082.1062352
Y: -3678077.62994914
Z: 18371228.1755685
---------
--Sat Pos 16: Old Eph--
****
X: 18828082.1770028
Y: -3678077.58751673
Z: 18371228.1136684
```

FIGURE 3.13: Ephemris integrity

3.5.4 RAIM

Receiver autonomous integrity monitoring (RAIM) must be used when there are at least five satellite's visible [8]. This algorithm creates subsets of all possible combinations between the set of visible satellites and performs a consistency check.

After fixing a position with all the available satellites, RAIM can be used to recalculate the receiver's position without a given satellite, if there is one that is far away from the overall position, then that satellite should be excluded [16].

```
----------------
|-----ARRAY COUNT----|
counter: 5
svId: 21
svId: 26
svId: 20
svId: 10
svId: 16
|-------------------|
---Raim started---
---Removed sv: 16
-----POSFIX:-----
Latitude: 38.7489337277
Longitude: -9.1531008007
Altitude: 154.335913626
Delta_clock: -0.000604451045849
Tow: 322265.999322
----------------
---Removed sv: 10
-----POSFIX:-----
Latitude: 38.7486751249
Longitude: -9.15311265895
Altitude: 53.0420738328
Delta_clock: -0.000604676887531
Tow: 322265.999322
----------------
---Removed sv: 20
-----POSFIX:-----
Latitude: 38.7485419915
Longitude: -9.15555904753
Altitude: -219.611410109
Delta_clock: -0.00060561911667
Tow: 322265.999322
----------------
---Removed sv: 26
-----POSFIX:-----
Latitude: 38.7489102995
Longitude: -9.15275458196
Altitude: 122.883550173
Delta_clock: -0.000604443252279
Tow: 322265.999322
----------------
---Removed sv: 21
-----POSFIX:-----
Latitude: 38.7481446034
Longitude: -9.15015582686
Altitude: 559.723189835
Delta_clock: -0.000602936352347
Tow: 322265.999322
----------------
```

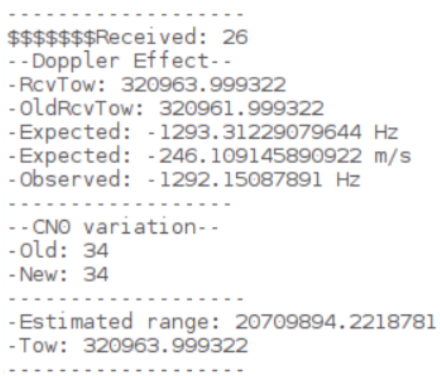
Figure 3.14: RAIM

3.5.5 Expected range

After fixing a position it is possible to retrieve a clock bias and know how it drifts since the receiver returns that parameter. The orbits of GPS satellites usually are around 20,000 Km, so by subtracting the clock bias times the speed of light to the pseudorange, the expected range should be around that value.

The tested spoofer usually had very high pseudoranges, after all it wasn't synchronized to GPS time, which would imply a large clock bias for it to make sense. So, if the clock bias is set and fixed to a small number, the expected range should be near the observed one.

After the calculation of the receiver's position, it is also possible to measure the range between the receiver and the satellite and compare it with the pseudorange of the satellite minus the clock bias times the speed of light.



```
------------------
$$$$$$$Received: 26
--Doppler Effect--
-RcvTow: 320963.999322
-OldRcvTow: 320961.999322
-Expected: -1293.31229079644 Hz
-Expected: -246.109145890922 m/s
-Observed: -1292.15087891 Hz
-----------------
--CN0 variation--
-Old: 34
-New: 34
------------------
-Estimated range: 20709894.2218781
-Tow: 320963.999322
------------------
```

FIGURE 3.15: Expected range, C/N_0 variation and Doppler shift

3.5.6 Excluding below the horizon satellites

After fixing a position with the available satellites, it is possible to determine the elevation of each one to the receivers position. If a satellite has an elevation below zero, it means it shouldn't be there.

The spoofer tested, gps-sdr-sim, didn't take into account this effect so the elevation can be calculated for each satellite in the new position or the old position depending on how long before it was. The satellites used for each calculation should also be cross verified, in order to find some that suddenly disappeared or appeared with different properties.

3.6 Flags to detect spoofing

3.6.1 Predicting the clock bias

The u-blox device returns clock drift parameters, so it is possible to know how it will change overtime. After fixing a position, the clock bias is stored. When a new fix is needed, the following equation is used:

$$\Delta t = \Delta t_0 + \delta(TOW - TOW_0) \tag{3.2}$$

Where Δt is the expected new clock bias, Δt_0 is the clock bias calculated from the previous position fix, TOW is the current GPS time of the week, TOW_0 is the last position fix GPS time of the week and δ is the clock drift.

The clock drift is how much the clock bias gets delayed per second, multiplying that for the time that passed it is possible to know how much it delayed. Using this method only three measurements are needed which is the receiver's position, since the clock bias is already known. This position is compared against the normal position fix.

Having the distance between both positions, it can be divided by the speed of light and added or subtracted to the expected clock bias, generally the calculated clock bias is inside this range. Since the used spoofer, gps-sdr-sim, is not synchronized with GPS time, the clock bias will change abruptly in unexpected ways.

3.6.2 Position variation

Usually spoofers change the position in a gradual way, so this method would not be as effective. However it is something to always consider, it is not possible for someone to travel large distances in one instant.

3.6.3 Overall

Using the mentioned methods one can implement multiple variations. Using the method of predicting the clock bias, it is useful to compare the position fixed using this method and the normal position fix. Applying RAIM on both, removing a given satellite per combination, it is possible to see which satellites are contributing most to the position bias.

3.7 Using Neural Networks

Since this is not a very complex problem with large amounts of data and variables, only ten nodes were used in the hidden layer with one node in the output that returns a value between zero and one. Being one a spoofing detection. The inputs will vary for each test scenario.

In order to optimize the functioning of this neural network, every data was normalized to the range between zero and one. For every input the maximum and minimum were retrieved and then the conversion was made.

$$NormalizedValue = \frac{value - minimum}{maximum - minimum} \tag{3.3}$$

3.7.1 Detecting spoofed satellites

For this problem five inputs were considered. Doppler shift variation, CN_0 variation, RAIM position difference without the given satellite to the global solution, variation of the ephemeredis given position and difference between the range between the satellite and the receiver and the pseudorange minus the clock bias. CN_0 variation is respective to the variation between two readings of this parameter. After fixing a position there's also a way to know how much a specific satellite is off the global position by using RAIM, and it is also possible to use the calculated clock bias and check if the pseudorange minus the clock bias times the speed of light is the same as the distance between the receiver and the satellite. Saving the ephemeredis from the previous position fix, it is also possible to compare the satellite position they return against the new ephemerides.

Solving this problem requires a special attention to variations, looking for changes that shouldn't happen. By having the position fix time span only the biggest variations are considered. Doppler shift variation is the difference between the predicted one and the observed.

3.7.2 Detecting spoofing

In order to detect spoofing, one should check for the variation between the clock prediction and the calculated one from the position fix. Variation in position from one iteration to another is also important, however they must be close in time. The variation between the position fix and the position fix with the expected clock bias is also another input.

Chapter 4

Implementation results

4.1 Observation

It is easier to detect spoofing when there is a variation from a non spoofing environment to spoofing one, however it is also possible to find discrepancies in a forged environment. In order to assess the results, the following formula will be applied.

$$Deviation = \frac{|Reference - Value|}{Reference} \tag{4.1}$$

The *Deviation* of a given *Value* relative to a *Reference* value.

4.1.1 C/N_0 variation

The easiest way one would figure how to spot a forged signal would be to look at the signal power, however that is not straightforward and only the simplest spoofers would be detected with this method. The spoofer used allows an adjustment in power, however a transition from a real signal to a spoofed one would be easily detected.

Real signals have a bigger C/N_0 variation, since the sources are further away and susceptible to all kind of phenomenons. A spoofer with a direct line to a receiver, usually does not vary much. Table 4.1 shows the observed variations.

TABLE 4.1: C/N_0 variation

	Spoofed Signal(%)	Real Signal(%)
Minimum	0	2.32
Maximum	2.27	43.75
Average	1.07	10.59

4.1.2 Doppler shift

U-blox receiver retrieves the measured Doppler frequency shift at the integration stage, since this effect is generated through the movement between the receiver and the transmitter, it can be predicted through the variation of the pseudoranges. This variation should not be measured on a big time span, because of the clock drift. For a time span of one second, the clock drift of this receiver is around 0.180 microseconds.

Real signals should be uniform and have close values between the predicted and observed Doppler shift, since the variation of the pseudoranges is an indicator on how the satellite is moving according to the receiver. For forged signals, it is an harder task, since they are on a fixed position and have to simulate the variation of the pseudoranges in order to match the transmitted frequency. Table 4.2 shows the variation from the observed and expected Doppler frequency shift for real and forged signals.

TABLE 4.2: Doppler shift

	Spoofed Signal(%)	Real Signal(%)
Minimum	6.47	0.066
Maximum	508436023.9	6.5
Average	120364027.4	1.61

4.1.3 RAIM

After fixing a position, and having more than four satellites, it is possible to compare how much a position fix drifts from the one with the exclusion of the respective satellite. In an non spoof environment, the absence of a satellite should not influence the calculation of the position in more than a couple hundred meters.

Table 4.3 shows how much in average the receiver changes its position if one given satellite is removed.

Table 4.3: RAIM position drift

	Spoofed Signal(m)	Real Signal(m)
Minimum	8969.4	5.47
Maximum	254893.1	345.24
Average	85372.74	85.42

4.1.4 Ephemerides variation

It is always useful to save the last used ephemeris and compare the satellite position using both the new and old ephemeris. Most spoofers won't change the ephemeris, so in this test scenario a conclusion can't be inferred.

Table 4.4: Ephemerides variation

	Spoofed Signal(m)	Real Signal(m)
Minimum	0.06	0.11
Maximum	0.60	2.40
Average	0.22	0.61

4.1.5 Expected range

After fixing a position it is always useful to check the range at which the satellite is from the receiver, since the position of the receiver and the satellite are known. From the position fix the clock bias is also determined, so by subtracting the clock bias times the speed of light from the pseudorange the expected range can be obtained. Both ranges can be compared. Table 4.5 shows the variation of the expected range from the determined one using the positions of the receiver and the satellite.

TABLE 4.5: Expected range variation

	Spoofed Signal(%)	Real Signal(%)
Minimum	0.067	0.017
Maximum	31.36	12.98
Average	15.74	6.96

4.1.6 Clock variation

The clock of a GPS receiver usually corrects its bias when it is near one milliseconds. So, unless the bias is near that value, it can be predicted through the clock drift. When spoofing starts the clock bias will have a great value, so if it goes from microseconds to seconds, it should be suspicious. Also, knowing how the clock drifts, even if the spoofer is synchronized, if the clock bias is not near the expected one, then spoofing should be suspected.

After the first fix, the clock bias can be determined, this can be observed in the Figure 4.1.

```
-----POSFIX:-----
Latitude: 38.7493576803
Longitude: -9.15357089746
Altitude: 149.993197005
Delta_clock: -0.000828017454661
Tow: 225754.999
----------------
|-----ARRAY COUNT----|
counter: 4
svId: 10
svId: 27
svId: 26
svId: 20
|-------------------|
--Delta_Clock stored: -0.000828017454661
--Tow stored: 225754.999
```

FIGURE 4.1: Clock bias fix

Knowing how the clock drifts it is possible to obtain an estimation. In this scenario a lower and an upper range were set, based on the clock bias plus the drift and the position difference, as shown in the following equation. In this work, the position difference was considered as a deviation in the clock as well, this assumes a static position.

$$\begin{aligned} \Delta t_{min} &= \Delta t_0 + \delta(TOW - TOW_0) + distanceTimeShift \\ \Delta t_{max} &= \Delta t_0 + \delta(TOW - TOW_0) - distanceTimeShift \end{aligned} \tag{4.2}$$

Where Δt_0 is the previously calculated clock bias, $\delta(TOW - TOW_0)$ is the clock drift times the time difference between calculations and *distanceTimeShift* is the distance between both position fixes divided by the speed of light.

```
|-------------------|
Expected min delta_clock: -0.000837286006857
Expected max delta_clock: -0.000837432902466
Inside range: VALID
-----POSFIX:-----
Latitude: 38.7494368479
Longitude: -9.1536761514
Altitude: 131.996614979
Delta_clock: -0.000837430392529
Tow: 225808.999
```

FIGURE 4.2: Clock bias prediction

In this scenario the calculated clock bias is inside the expected range, like shown in Figure 4.2. Since the clock bias drifted the way it was supposed to, this position fix can be considered legitimate.

Figure 4.3 shows a spoofing scenario, where the receiver was given enough iterations to adjust its clock bias to these signals. A random location was chosen for this test.

Figure 4.4 shows the prediction range of the clock bias and the calculated value. It is possible to observe the clock bias is not close to the estimated range, in contrary to the previous scenario in which it in the estimated range. The bias was off range by $5.800730359 * 10^{-6}$ seconds, which can amount to an error of,

```
-----POSFIX:-----
Latitude: 1.36234055704
Longitude: 103.992751154
Altitude: 12.643442112
Delta_clock: 0.000678128504119
Tow: 172964.001
----------------
```

FIGURE 4.3: Clock bias fix in spoofing environment

roughly, 1739.015 meters. This approximation was done by multiplying the given time for the speed of light. However the calculated positions are really close, so it should be suspicious how one parameter predicts one thing and the other another thing.

```
Expected min delta_clock: 0.000668453702049
Expected max delta_clock: 0.000668381306189
Outside range: INVALID
-----POSFIX:-----
Latitude: 1.36230797454
Longitude: 103.992813941
Altitude: 20.1239197794
Delta_clock: 0.00066258057583
Tow: 173081.001
----------------
```

FIGURE 4.4: Clock bias prediction in spoofing environment

TABLE 4.6: Clock variation offset from range

	Spoofed Signal(seconds)	Real Signal(seconds)
Distance	$5.800730359 * 10^{-6}$	0

4.1.7 Distance between position fixes

This method is only useful when the receiver is assumed as static or in a slow movement, since it is normal for some vehicles to change its position abruptly. However if records of previous position fixes are kept, it is possible to determine the position of the receiver, its average velocity and the direction in which it is going towards to. Figure 4.5 shows the new position fix, which would make the distance between both position fixes 26.153394 meters, which is an acceptable value since this algorithm does not have all the corrections of the pseudoranges.

```
---------
|-----ARRAY COUNT----|
counter: 4
svId: 10
svId: 27
svId: 26
svId: 20
|-------------------|
-----POSFIX:-----
Latitude: 38.7494405117
Longitude: -9.15338065684
Altitude: 128.927674937
Delta_clock: -0.000837439024571
Tow: 225808.999
----------------
```

FIGURE 4.5: Distance shift between position fixes

In the spoofing scenario, both fixes are shown in Figure 4.3 and Figure 4.4. The distance between them is 10.723805 meters. This is a rudimentary position calculator, so this conclusion might not be as accurate as intended, however the precision of position is better in a spoofed scenario.

TABLE 4.7: Distance between position fixes

	Spoofed Signal(meters)	Real Signal(meters)
Distance	10.72380	26.153394

4.1.8 Difference between position fix and expected position

Using the clock prediction and only calculating the variables respective to the position of the receiver, it is possible to compare it against the position fix determining the four variables.

Figure 4.6 compares both algorithms. It can be observed that the clock bias calculated was close to the one predicted. The distance between both positions is 46.21688 meters, so it is possible to infer that there was no tampering of the data.

Figure 4.7 shows this difference in a spoofing environment. The euclidean distance between both positions is 2674.057778 meters, which indicates that this position might not be legitimate.

```
-----POSFIX:-----
Latitude: 38.7492552184
Longitude: -9.15341222077
Altitude: 170.436843179
Delta_clock: 6.51291670043e-05
Tow: 242613.999739
----------------
|-----ARRAY COUNT----|
counter: 5
svId: 22
svId: 3
svId: 1
svId: 11
svId: 23
|-------------------|
-----POSFIX3SAT:-----
Latitude: 38.7493037439
Longitude: -9.15356918428
Altitude: 123.115832611
Delta_clock: 6.49834628249e-05
----------------
```

FIGURE 4.6: Difference between position fix and expected position

```
-----POSFIX:-----
Latitude: 1.36230797454
Longitude: 103.992813941
Altitude: 20.1239197794
Delta_clock: 0.00066258057583
Tow: 173081.001
----------------
|-----ARRAY COUNT----|
counter: 4
svId: 32
svId: 11
svId: 27
svId: 31
|-------------------|
-----POSFIX3SAT:-----
Latitude: 1.35273449716
Longitude: 104.004620413
Altitude: 2094.16753827
Delta_clock: 0.000668417504119
----------------
**********************
```

FIGURE 4.7: Difference between position fix and expected position in a spoofing environment

4.1.9 Conclusion

These methods have a different approach than the usual ones, since they focus more on how the values should vary and not so much on how they should be. Any spoofer can change the values to what they want to, however it is harder to imitate a behaviour. Table 4.9 shows how this parameters should change for a spoofed and

Table 4.8: Difference between position fix and expected position

	Spoofed Signal(meters)	Real Signal(meters)
Distance	2674.05777	46.21688

a real signal, being lower a smoother variation and higher an abrupter variation.

Table 4.9: Conclusion on spoofed satellites detection

	Spoofed Signal	Real Signal
C/N_0	Lower	Higher
Doppler shift variation	Higher	Lower
RAIM variation	Higher	Lower
Ephemerides variation	Lower	Higher
Expected range variation	Higher	Lower

When it comes to detecting spoofing the Table 4.10 shows the respective conclusions. In a spoofing scenario it is expected for the clock variation to be further away from the expected range, where the real signal should be inside it. Since the spoofer acts closer to the receiver than the satellites, it is expected for variations in position to be lower because there are less variables that can influence this factor like different satellites used for a position fix or multipath propagation. The difference between position fix and expected position should follow the same pattern as the clock variation from the expected range since both work on an expected clock bias, therefore the spoofed signal should have a larger distance difference.

Table 4.10: Conclusion on on detecting spoofing

	Spoofed Signal	Real Signal
Clock variation from the expected range	Higher	Lower
Distance between position fixes	Lower	Higher
Difference between position fix and expected position	Higher	Lower

4.2 Using Neural Networks

In order to develop this model, data will be retrieved from scenarios where there is only spoofing and scenarios where there is no spoofing.

4.2.1 Detecting spoofed satellites

Figure 4.8 shows the implemented neural network for this scenario. In 1 is respective to the Doppler shift variation, In 2 is respective to C/N_0 variation, In 3 to RAIM position difference without the given satellite to the global solution, In 4 to the variation of the ephemeredis given position and In 5 to the difference between the range between the satellite and the receiver and the pseudorange minus the clock bias. Out 1 is respective to the detection, logical value of one, or no detection, logical value of zero, of a spoofed satellite signal.

Figure 4.8 already shows an example of data from a spoofed satellite. In1 is not zero, since neuroph studio does an approximation, but close to zero which is not frequent in spoofed signals. In2 has a value near zero, which is more usual for a spoofed signal than for a real signal, unless the spoofer intentionally changes the power of the signal. In3 and In5 are the highest values. In4, like mentioned before, is lower in spoofed signals. Even though In1 was an exception to the rule, the neural network was capable of detecting it was a forged signal.

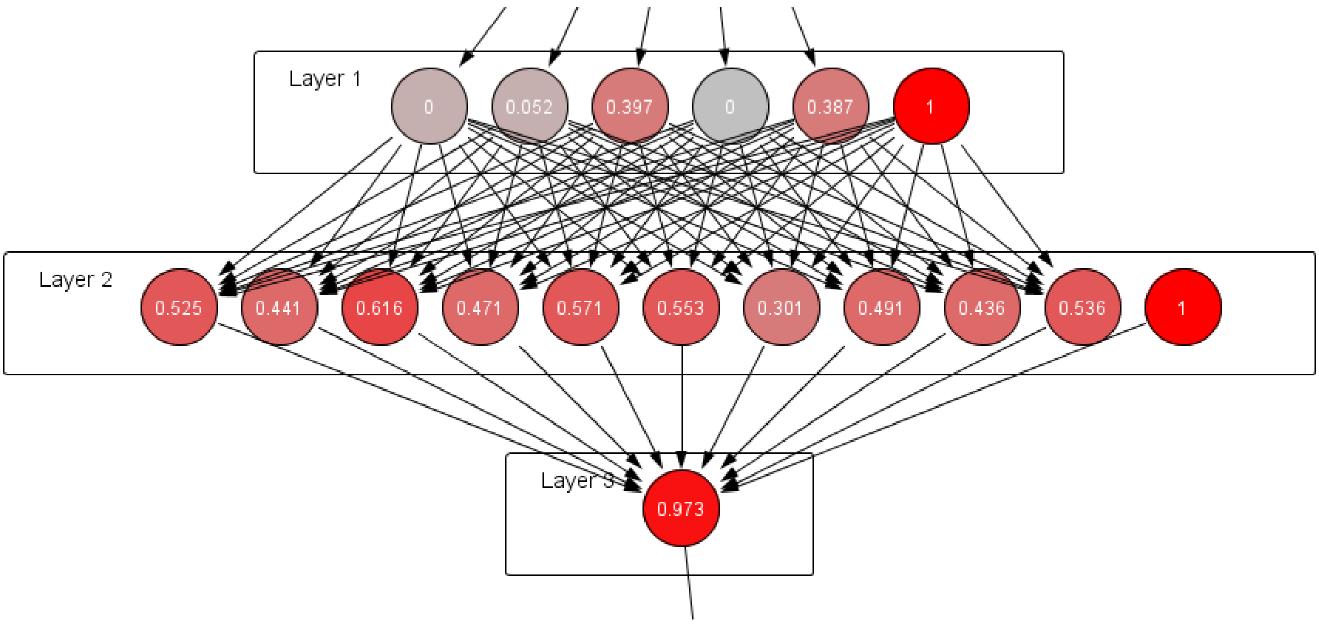


FIGURE 4.8: Neural Network to detect spoofed satellites

In order to train the neural network, data is needed, from either spoofed signals or real signals, so to facilitate the visualization of the solution, a neural network with 19 samples will be trained. This 19 samples will be split, 70 percent for training and 30 percent for testing. In order to infer the accuracy of the model, some samples must be used only for testing. For a final product more samples would be needed.

Table 4.11: Neural Network data first scenario

In1	In2	In3	In4	In5	Out1
5.51515E-11	0.061776062	1.6096E-05	1	0	0
3.02558E-09	0.214285714	0.000352148	0.226124764	0.203199766	0
8.78705E-09	0.142857143	0.000129397	0.221975152	0.357443548	0
8.37785E-10	0.053156146	0.000140697	0.203346702	0.221049285	0
1.33194E-09	0.360902256	9.25617E-05	0.027666185	0.001572	0
0	0.057142857	0.001154503	0.231659561	0.150066639	0
1.25857E-08	0.207792208	0	0.226787736	0.300176768	0
4.59983E-10	0.061776062	5.92986E-05	0.197285185	0.270280084	0
1.8343E-09	0.147465438	7.2611E-05	0.022996656	0.337203507	0
2.66276E-09	0.152380952	0.000566526	0.236837984	0.092319348	0
4.55887E-09	0.065306122	9.77733E-05	0.232301038	0.227161397	0
2.0633E-09	0.623376623	6.28108E-05	0.19200164	0.304327038	0
1.33597E-09	1	0.001332994	0.022531179	0.41336838	0
0.080698679	0.050793651	0.148139239	0.022972461	0.001573843	1
1	0	0.215819986	0.038979065	0.04630737	1
0.102246424	0	1	0.231459005	0.768545467	1
0.237458039	0.043956044	0.035168316	0.030313963	0.807583303	1
1.27963E-08	0	0.213316776	0.082331323	1	1
6.27836E-08	0.051948052	0.397082776	0	0.386595343	1

Table 4.11 already has the data normalized to the interval between zero and one, the value one is the maximum and zero, the minimum. Randomly 70 percent will be used for training and 30 percent for testing.

```
Input: 0; 0; 0.2133; 0.0823; 1;  Output: 0.9966; Desired output: 1;  Error: -0.0034;
Input: 0; 0.0532; 0.0001; 0.2033; 0.221;  Output: 0.0045; Desired output: 0;  Error: 0.0045;
Input: 0; 0.0519; 0.3971; 0; 0.3866;  Output: 0.9727; Desired output: 1;  Error: -0.0273;
Input: 0; 0.0618; 0; 1; 0;  Output: 0; Desired output: 0;  Error: 0;
Input: 0; 0.2078; 0; 0.2268; 0.3002;  Output: 0; Desired output: 0;  Error: 0;
```

Figure 4.9: Test scenario one results

Figure 4.9 shows the test results for the mentioned scenario. There is a very low error, so this experiment was successful, some inputs were so low that the program automatically rounded them to zero. As mentioned before, the dataset is too small in order to make a satisfying product, hence it is only for the simplification of the solution.

Chapter 5

Conclusion

The objective of this work was to study effective anti-spoofing measures due to the emerging self-driving vehicles that use GPS as a navigation system. The spoofer tested was gps-sdr-sim and this spoofer had some particularities that might make it distinguishable from real signals.

The biggest characteristic of this type of spoofers is that its clock is not synchronized, so the clock bias obtained after a position fix will be big, making an abrupt transition. The pseudoranges also will have big values. Even if there is synchronization in the spoofer, it will not know the clock bias of the receiver. The receiver, knowing how its clock drifts, can predict how much the clock bias is going to be.

Some parameters have values that do not change that much, however it is still a significant change that the neural network can predict. The objective of this work was not to try to find which values the parameters should have, because they are easily changed, but how the variation happens, how the spoofer thinks per say. It is possible for a neural network to find a pattern in this data, as long as it is well trained and labeled.

5.1 Future work

Given these flags and the methods studied, a robust system can be built using thousands of samples in different scenarios using different spoofers and without spoofing. Implementing an AI algorithm capable of analyzing the data and returning an answer quickly in order to deal with the forged signals and possibly ignore them. There is not straightforward solution to this problem, however there is a pattern among spoofers and real signals, that pattern can be trained with the neural network. It is also worth to look at other GNSS systems in order to use all available information to determine a position. The easiest way would be to detect spoofing in GPS and change to another constellation, however it would be interesting to make a system that uses satellites from diffrent constellations.

This work can also be continued with the help of sensors that indicate the velocity, acceleration and direction of the receiver. Wi-Fi routers and GSM towers can also be used as a reference for positioning. The receiver can calculate a pattern for the way it is moving, by saving previous positions, it is possible to determine in which direction it is going to and the velocity of it. An AI algorithm can determine if a receiver was supposed to move in a certain direction with a certain velocity.

Appendices

Appendix A

Code

```
from __future__ import division
from decimal import *
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import math
import serial
import binascii
import struct
import pyproj
import numpy
import time
import itertools

port = serial.Serial("/dev/ttyAMA0", baudrate=9600, timeout=3)

#poll_eph = "\xB5\x62\x0B\x31\x00\x00\x3C\xBF"

ecef = pyproj.Proj(proj='geocent', ellps='WGS84', datum='WGS84')
lla = pyproj.Proj(proj='latlong', ellps='WGS84', datum='WGS84')

tolerance = 1*(10**-12)
miu = 3.986005*(10**14)
omega_e = 7.2921151467*(10**-5)
c = 299792458
F = -4.442807633*(10**(-10))
```

```
rt = 6371*(10**3)
l1freq = 1575.42*(10**(6))
lastRcvTow = 0

storedpx = 0
storedpy = 0
storedpz = 0

start = time.time()
delta_clock = 0
delta_clock_set = False
delta_clock_tow = 0

clock_variance = 0

drift_clock = 0
drift_clock_set = False

clock_biasread = 0

class sv:
  def __init__(self, id, pr, rcvTow):
    self.id = id
    self.pr = pr
    self.rcvTow = rcvTow

svList = []

def raim(lengthL, svPos):
  if(lengthL>4):
    print("---Raim started---")
    it = lengthL-1
    for subset in itertools.combinations(svPos,it):
      for svCheck in svPos:
        found = False
        for svCheck2 in subset:
          if(svCheck.id == svCheck2.id):
            found = True
```

```
        if(found == False):
          print("---Removed sv: %s" % svCheck.id)
          sv_remv = svCheck
      lat,lon,ecefrx, ecefry, ecefrz = getFix(subset,0)
      sv_remv.delta_raim = math.sqrt((storedpx-ecefrx)**2 + (storedpy-ecefry)**2 + (storedpz-ecefrz)**2)
def printSvData(sat):
  global delta_clock
  for svcnt in sat:
    print("********************")
    print("*sv_id: %s*" % svcnt.id)
    print("*delta_doppler: %s*" % svcnt.variance)
    print("*delta_cn0: %s*" % svcnt.variancecn0)
    if hasattr(svcnt,'delta_raim'):
      print("*delta_raim: %s*" % svcnt.delta_raim)
    if hasattr(svcnt,'variance_ef'):
      variance_ef = (1.5727-svcnt.variance_ef)/1.5727
      print("*delta_eph: %s*" % svcnt.variance_ef)
    rangeFrompr = svcnt.pr - Decimal(c*delta_clock)
    if hasattr(svcnt,'varianceR'):
      variance_range = math.fabs((Decimal(svcnt.varianceR)-rangeFrompr)/Decimal(svcnt.varianceR))
      print("*variance_range: %s*" % variance_range)
    print("********************")

def fixSagnac(sv,ecefx, ecefy, ecefz):
  print("---Sagnac---")
  for svi in sv:
    Delta_fim=0.000001


    delta_x = Decimal(ecefx) - svi.X
    delta_y = Decimal(ecefy) - svi.Y
    delta_z = Decimal(ecefz) - svi.Z
    dist_rcv = Decimal(math.sqrt((delta_x)**2+(delta_y)**2+(delta_z)**2))
    delta_t = dist_rcv/Decimal(c)

```

```
 Dist_fim=dist_rcv
 Dist_inicio=0

 while(math.fabs(Dist_fim-Dist_inicio)>Delta_fim):
   Delta_rad=Decimal(omega_e-svi.omega_dot)*(delta_t)

   Dist_inicio=Dist_fim
   A = Decimal(pow(svi.sqrt_A,2))
   n = Decimal(math.sqrt(Decimal(miu)/(A**3))) + Decimal(svi.delta_n)

   sentTow = Decimal(svi.rcvTow-delta_clock) - Decimal(delta_t)
   tk = sentTow - Decimal(svi.toe)

   if(tk>302400):
     tk = tk - 604800
   elif(tk<-302400):
     tk = tk + 604800

   M = Decimal(svi.M0) + n*tk
   delta_E = 1
   E = M
   while(math.fabs(delta_E)> tolerance):
     delta_E = (M - (E-Decimal(svi.e*math.sin(E))))/(1-Decimal(svi.e*math.cos(E)))
     E = E+delta_E
   sVk = Decimal(Decimal(math.sqrt(1-Decimal(pow(svi.e,2))))*Decimal(math.sin(E)))/(1-Decimal(svi.e*math.cos(E)))
   cVk = Decimal(Decimal(math.cos(E)-svi.e)/(1-Decimal(svi.e*math.cos(E))))
   true_anomaly = Decimal(math.atan2(sVk,cVk))

   if(true_anomaly<0):
     true_anomaly = true_anomaly + Decimal(2*math.pi)

   arg_latitude = true_anomaly + Decimal(svi.omega)

```

```
    delta_u = (Decimal(svi.Cuc) * Decimal(math.cos(2*arg_latitude))+Decimal(svi.Cus)* Decimal(math.sin(2*arg_latitude)))
    u = Decimal(arg_latitude + delta_u)

    delta_i = (Decimal(svi.Cic) * Decimal(math.cos(2*arg_latitude)) +Decimal(svi.Cis) * Decimal(math.sin(2*arg_latitude)))
    i = Decimal(Decimal(svi.i0) + delta_i + tk*(Decimal(svi.idot)))

    delta_r = (Decimal(svi.Crs) * Decimal(math.sin(2*arg_latitude)) + Decimal(svi.Crc) * Decimal(math.cos(2*arg_latitude)))

    r = A*(1-Decimal(svi.e*math.cos(E))) + delta_r

    omega = Decimal(svi.omega0) + Decimal(svi.omega_dot - omega_e)*tk - Decimal(omega_e*svi.toe)


    Xkl=Decimal(r*Decimal(math.cos(u)))
    Ykl=Decimal(r*Decimal(math.sin(u)))

    X=Decimal(Xkl*Decimal(math.cos(omega))-Ykl*Decimal(math.cos(i)*math.sin(omega)))
    Y=Decimal(Xkl*Decimal(math.sin(omega))+Ykl*Decimal(math.cos(i)*math.cos(omega)))
    Z=Decimal(Ykl*Decimal(math.sin(i)))
    XYZ = numpy.matrix([[X],[Y],[Z]])

    if(delta_t>0):
      Mat_trans11=Decimal(math.cos(Decimal(omega_e-svi.omega_dot)*(delta_t)))
      Mat_trans12=Decimal(math.sin(Decimal(omega_e-svi.omega_dot)*(delta_t)))
      Mat_trans13=0
```

```
        Mat_trans21=Decimal(-math.sin(Decimal(omega_e-svi.omega_dot)*(delta_t)))
        Mat_trans22=Decimal(math.cos(Decimal(omega_e-svi.omega_dot)*(delta_t)))
        Mat_trans23=0
        Mat_trans31=0
        Mat_trans32=0
        Mat_trans33=1
        Mat_trans = numpy.matrix([[Mat_trans11,Mat_trans12,Mat_trans13],[Mat_trans21,Mat_trans22,Mat_trans23],[Mat_trans31,Mat_trans32,Mat_trans33]])
        XYZ = Mat_trans.dot(XYZ)
      delta_x = (svi.X) - XYZ.item(0,0)
      delta_y = (svi.Y) - XYZ.item(1,0)
      delta_z = (svi.Z) - XYZ.item(2,0)
      dist_prev = math.sqrt((delta_x)**2+(delta_y)**2+(delta_z)**2)

      svi.X = (XYZ.item(0,0))
      svi.Y = (XYZ.item(1,0))
      svi.Z = (XYZ.item(2,0))

      delta_x = Decimal(ecefx) - (svi.X)
      delta_y = Decimal(ecefy) - (svi.Y)
      delta_z = Decimal(ecefz) - (svi.Z)
      dist_rcv = Decimal(math.sqrt((delta_x)**2+(delta_y)**2+(delta_z)**2))
      delta_t = dist_rcv/Decimal(c)

      Dist_fim=dist_rcv

def checkHealth(tupSv):
  for svi in tupSv:
    if(svi.health==1):
      print("Sv: %s, not healthy" % svi.id)
      svi.pos = 0

def checkElev(sv,lat,lon):
```

```
  for svi in sv:
    lat = math.radians(lat)
    lon = math.radians(lon)
    phi = math.radians(svi.lat)
    teta_L = math.radians(svi.lon)
    L = teta_L-lon
    r = rt + svi.alt
    p1 = (math.cos(phi)*math.cos(L)*math.cos(lat)+math.sin(lat)*
   math.sin(phi))
    p2 = r*math.sqrt(1-(p1**2))
    p3 = math.sqrt((rt**2)+(r**2)-(2*rt*r*p1))
    print("----Range for sat %s: %s----" % (svi.id,p3))
    svi.varianceR = p3
    E = math.acos(p2/p3)
    E = math.degrees(E)
    if(E<0):
      svi.pos=0
    print("-------------------")
    print("->Sat: %s" % svi.id)
    print("->Elev: %s" % E)
    print("-------------------")

def svPosCount():
  toCalc = []
  counter = 0
  for svi in svList:
    if(svi.pos==1):
      counter = counter+1
      toCalc.append(svi)
  print("|-----ARRAY COUNT----|")
  print("counter: %s" % counter)
  for stest in toCalc:
    print("svId: %s" % stest.id)
  print("|-------------------|")
  return (counter,toCalc)

def MostTowCount():
  toCalc = []
```

```
  for svi in svList:
    if(svi.pos==1):
      toCalc.append(svi)
      print("|-----ARRAY COUNT TOW----|")
      print("| svi: %s tow: %s |" % (svi.id,svi.rcvTow))
      print("|--------------------|")
  count = 0
  Tow = 0
  for tCsV in toCalc:
    countaux=0
    for tCsV2 in toCalc:
      if(tCsV.rcvTow == tCsV2.rcvTow):
        countaux= countaux+1
    if(countaux>count):
      count=countaux
      Tow = tCsV.rcvTow
  return Tow


def getFix3SAT(svPos, clock):
  r0 = numpy.matrix([[Decimal(0)],[Decimal(0)],[Decimal(0)]])
  errorC = 1000
  clock_d = clock * c
  it = 0
  while errorC>0.001 and it<=20:
    r0 = numpy.matrix([[Decimal(r0.item(0,0))],[Decimal(r0.item(1,0))],[Decimal(r0.item(2,0))]])
    Z = numpy.zeros(shape=(len(svPos),1))
    linc = 0
    H = numpy.zeros(shape=(len(svPos),3))
    for svr in svPos:
      vetor_sj_r = numpy.subtract([[svr.X],[svr.Y],[svr.Z]],r0)
      mv_sj_r = numpy.linalg.norm(vetor_sj_r)
      unit_vetorT = (vetor_sj_r /  mv_sj_r).T
      Zd2 = unit_vetorT.dot([[svr.X],[svr.Y],[svr.Z]])
      Z2 = Decimal(svr.pr)- Decimal(clock_d) - Zd2
      H2 = -unit_vetorT
      H2 = numpy.asarray(H2).reshape(-1)
```

```
      Z[linc] = [Z2.item(0,0)]
      H[linc] = H2
      linc = linc+1
    p1 = (H.T).dot(Z)
    p2 = (H.T).dot(H)
    while numpy.linalg.det(p2)==0:
      noise = numpy.full((3,3),0.00001)
      p2 = p2 + noise
    p3 = numpy.linalg.inv(p2)
    x = p3.dot(p1)
    errorC = math.sqrt((r0.item(0,0)-Decimal(x.item(0,0)))**2+(r0.item(1,0)-Decimal(x.item(1,0)))**2+(r0.item(2,0)-Decimal(x.item(2,0)))**2)
    r0 = x[:,:]
    it = it+1
  lon, lat, alt = pyproj.transform(ecef, lla, x.item(0,0), x.item(1,0), x.item(2,0), radians=False)
  print('-----POSFIX3SAT:-----')
  print("Latitude: %s" % lat)
  print("Longitude: %s" % lon)
  print("Altitude: %s" % alt)
  print("Delta_clock: %s" % clock)
  print('-----------------')

def getFix(svPos, sagnac):
  global clock_variance
  global delta_clock
  global delta_clock_set
  global delta_clock_tow
  global storedpx
  global storedpy
  global storedpz
  global drift_clock
  global drift_clock_set
  r0 = numpy.matrix([[Decimal(0)],[Decimal(0)],[Decimal(0)]])
  errorC = 1000
  it = 0
  while errorC>0.001 and it<=20:
```

```
    r0 = numpy.matrix([[Decimal(r0.item(0,0))],[Decimal(r0.item(1,0))],[Decimal(r0.item(2,0))]])
    Z = numpy.zeros(shape=(len(svPos),1))
    linc = 0
    H = numpy.zeros(shape=(len(svPos),4))
    for svr in svPos:
      vetor_sj_r = numpy.subtract([[svr.X],[svr.Y],[svr.Z]],r0)
      mv_sj_r = numpy.linalg.norm(vetor_sj_r)
      unit_vetorT = (vetor_sj_r /  mv_sj_r).T
      Zd2 = unit_vetorT.dot([[svr.X],[svr.Y],[svr.Z]])
      Z2 = Decimal(svr.pr) - Zd2
      H2 = numpy.insert(-unit_vetorT, 3, 1, axis=1)
      H2 = numpy.asarray(H2).reshape(-1)
      Z[linc] = [Z2.item(0,0)]
      H[linc] = H2
      linc = linc+1
    p1 = (H.T).dot(Z)
    p2 = (H.T).dot(H)
    while numpy.linalg.det(p2)==0:
      noise = numpy.full((4,4),0.00001)
      p2 = p2 + noise
      print("singular")
    p3 = numpy.linalg.inv(p2)
    x = p3.dot(p1)
    errorC = math.sqrt((r0.item(0,0)-Decimal(x.item(0,0)))**2+(r0.item(1,0)-Decimal(x.item(1,0)))**2+(r0.item(2,0)-Decimal(x.item(2,0)))**2)
    r0 = x[:-1,:]
    it=it+1
  lon, lat, alt = pyproj.transform(ecef, lla, x.item(0,0), x.item(1,0), x.item(2,0), radians=False)
  delta_clockaux = x.item(3,0)/c
  if sagnac==1:
    if(delta_clock_set==True):
      #[-0.179,-0.185] us/s drift relogio
      distance_timeshift = math.sqrt((storedpx-x.item(0,0))**2 + (storedpy-x.item(1,0))**2 + (storedpz-x.item(2,0))**2) / c
```

```
      delta_tc_min = (lastRcvTow-delta_clock_tow)*(drift_clock)+delta_clock+distance_timeshift
      delta_tc_max = (lastRcvTow-delta_clock_tow)*(drift_clock)+delta_clock-distance_timeshift
      print("Expected min delta_clock: %s" % delta_tc_min)
      print("Expected max delta_clock: %s" % delta_tc_max)
      if(math.fabs(delta_clockaux)>math.fabs(delta_tc_min) and math.fabs(delta_clockaux)<math.fabs(delta_tc_max)):
        print("Inside range: VALID")
        clock_variance = 0
      else:
        print("Outside range: INVALID")
        if(delta_clockaux>delta_tc_min):
          clock_variance = (delta_clockaux-delta_tc_min)/(delta_tc_min-delta_tc_max)
        else:
          clock_variance = (delta_tc_max-delta_clockaux)/(delta_tc_min-delta_tc_max)
    delta_clock = x.item(3,0)/c
    delta_clock_tow = lastRcvTow
    delta_clock_set = True
    drift_clock_set = False
    storedpx = x.item(0,0)
    storedpy = x.item(1,0)
    storedpz = x.item(2,0)
  print('-----POSFIX:-----')
  print("Latitude: %s" % lat)
  print("Longitude: %s" % lon)
  print("Altitude: %s" % alt)
  print("Delta_clock: %s" % delta_clockaux)
  print("Tow: %s" % lastRcvTow)
  print('----------------')

  return(lat,lon,x.item(0,0), x.item(1,0), x.item(2,0))

def pollEphSv(svEphId):
  CK_A = 0x00
  CK_B = 0x00
```

```
  CK_A = CK_A + 0x0B
  CK_B = CK_B + CK_A

  CK_A = CK_A + 0x31
  CK_B = CK_B + CK_A

  CK_A = CK_A + 0x01
  CK_B = CK_B + CK_A

  CK_A = CK_A + 0x00
  CK_B = CK_B + CK_A

  CK_A = CK_A + svEphId
  CK_B = CK_B + CK_A

  sum1 = CK_A & 0xff
  sum2 = CK_B & 0xff

  x = 'B5' + '62' + '0B' + '31' + '01' + '00' + format(svEphId,'02
    x') + format(sum1,'02x') + format(sum2,'02x')
  y = x.decode("hex")
  return y

def checkSvList(id):
  if(len(svList)==0):
    return (False,0)
  else:
    for svi in svList:
      if(svi.id == id):
        return (True,svi)
  return (False,0)

def sat_pos(sva):
  print("****")
  A = pow(sva.sqrt_A,2)
  n = math.sqrt(miu/(A**3)) + sva.delta_n

```

```
sentTow = Decimal(Decimal(sva.rcvTow) - Decimal(sva.pr/c))
delta_t = sentTow - sva.toe

if(delta_t>302400):
  delta_t = delta_t - 604800
elif(delta_t<-302400):
  delta_t = delta_t + 604800

M = Decimal(Decimal(sva.M0) + Decimal(n)*delta_t)
delta_E = 1
E = M
while(math.fabs(delta_E)> tolerance):
  delta_E = Decimal(M - (E-Decimal(sva.e)*Decimal(math.sin(E))))
  /(1-Decimal(sva.e)*Decimal(math.cos(E)))
  E = E+delta_E

#SV time correction

delta_tsv = sentTow - sva.toc
if(delta_tsv>302400):
  delta_tsv = delta_tsv - 604800
elif(delta_tsv<-302400):
  delta_tsv = delta_tsv + 604800
delta_tr = Decimal(F*sva.e*sva.sqrt_A*math.sin(E))
delta_tsv = Decimal(Decimal(sva.af0) + Decimal(sva.af1)*
 delta_tsv + Decimal(sva.af2)*(delta_tsv**2) + delta_tr -
 Decimal(sva.tgd))

sva.tcorr = delta_tsv

#///Fixing pseudorange///
fixedpr = Decimal(Decimal(sva.pr) + (delta_tsv*Decimal(c)))
sva.pr = fixedpr



sentTow = Decimal(Decimal(sva.rcvTow) - Decimal(sva.pr/c))

```

```
delta_t = sentTow - sva.toe

if(delta_t>302400):
  delta_t = delta_t - 604800
elif(delta_t<-302400):
  delta_t = delta_t + 604800

M = Decimal(Decimal(sva.M0) + Decimal(n)*delta_t)
delta_E = 1
E = M
while(math.fabs(delta_E)> tolerance):
  delta_E = Decimal(M - (E-Decimal(sva.e)*Decimal(math.sin(E))))
 /(1-Decimal(sva.e)*Decimal(math.cos(E)))
  E = E+delta_E

sVk = Decimal(math.sqrt(1-pow(sva.e,2))*math.sin(E))/Decimal(1-
 sva.e*math.cos(E))
cVk = Decimal(math.cos(E)-sva.e)/Decimal(1-sva.e*math.cos(E))
true_anomaly = Decimal(math.atan2(sVk,cVk))

if(true_anomaly<0):
  true_anomaly = true_anomaly + Decimal(2*math.pi)

arg_latitude = true_anomaly + Decimal(sva.omega)

delta_u = Decimal(sva.Cuc * math.cos(2*arg_latitude)+sva.Cus*
 math.sin(2*arg_latitude))
u = arg_latitude + delta_u

delta_i = Decimal(sva.Cic * math.cos(2*arg_latitude) + sva.Cis *
  math.sin(2*arg_latitude))
i = Decimal(sva.i0) + delta_i + delta_t*Decimal(sva.idot)

delta_r = Decimal(sva.Crs * math.sin(2*arg_latitude) + sva.Crc *
  math.cos(2*arg_latitude))

r = Decimal(Decimal(A*(1-sva.e*math.cos(E))) + delta_r)

```

```
omega = Decimal(sva.omega0) + Decimal(sva.omega_dot - omega_e)*
    delta_t - Decimal(omega_e*sva.toe)


Xkl=Decimal(r*Decimal(math.cos(u)))
Ykl=Decimal(r*Decimal(math.sin(u)))

#POSITION ECEF FORMAT
sva.X=Decimal(Xkl*Decimal(math.cos(omega))-Ykl*Decimal(math.cos(
    i)*math.sin(omega)))
sva.Y=Decimal(Xkl*Decimal(math.sin(omega))+Ykl*Decimal(math.cos(
    i)*math.cos(omega)))
sva.Z=Decimal(Ykl*Decimal(math.sin(i)))
sva.lon, sva.lat, sva.alt = pyproj.transform(ecef, lla, sva.X,
    sva.Y, sva.Z, radians=False)
'''
print("------SAT POS CALCULUS------")
print("E: %s" % E)
print("sVk: %s" % sVk)
print("cVk: %s" % cVk)
print("true_anomaly: %s" % true_anomaly)
print("arg_latitude: %s" % arg_latitude)
print("delta_u: %s" % delta_u)
print("u: %s" % u)
print("delta_i: %s" % delta_i)
print("i: %s" % i)
print("r: %s" % r)
print("omega: %s" % omega)
print("Xkl: %s" % Xkl)
print("Ykl: %s" % Ykl)
print("sva.X: %s" % sva.X)
print("sva.Y: %s" % sva.Y)
print("sva.Z: %s" % sva.Z)
print("tcorr: %s" % sva.tcorr)
print("--------------------------")
#CONVERTION TO LAT/LON
print("-----------")
print("svId: %s" % sva.id)
```

```
  print(sva.lon,sva.lat,sva.alt)
  print("X: %s" % sva.X)
  print("Y: %s" % sva.Y)
  print("Z: %s" % sva.Z)
  print("Pseudorange: %s" % sva.pr)
  print("Rcv Tow: %s" % (sva.rcvTow))
  print("------------")
  '''
  print("X: %s" % sva.X)
  print("Y: %s" % sva.Y)
  print("Z: %s" % sva.Z)
  #print_sv(sva)
  #clockDelay(X,Y,Z,sva)

def clockDelay(X,Y,Z,sva):
  rg = math.sqrt(((X-ecefx)**2)+((Y-ecefy)**2)+((Z-ecefz)**2))
  delta_clock = (sva.pr-rg)/c
  print("------------")
  print("delta_clock: %s" % (delta_clock))
  print("------------")

def print_sv(svt):
  print("Sv id: %s" % (svt.id))
  print("Pseudo range: %s" % (svt.pr))
  print("Rcv Tow: %s" % (svt.rcvTow))
  print("Crs: %s" % (svt.Crs))
  print("delta_n: %s" % (svt.delta_n))
  print("M0: %s" % (svt.M0))
  print("Cuc: %s" % (svt.Cuc))
  print("e: %s" % (svt.e))
  print("Cus: %s" % (svt.Cus))
  print("sqrt_A: %s" % (svt.sqrt_A))
  print("toe: %s" % (svt.toe))
  print("cic: %s" % (svt.Cic))
  print("cis: %s" % (svt.Cis))
  print("omega0: %s" % (svt.omega0))
  print("i0: %s" % (svt.i0))
  print("crc: %s" % (svt.Crc))
```

```
    print("omega: %s" % (svt.omega))
    print("omega_dot: %s" % (svt.omega_dot))
    print("idot: %s" % (svt.idot))
    print("af0: %s" % (svt.af0))
    print("af1: %s" % (svt.af1))
    print("af2: %s" % (svt.af2))
    print("tgd: %s" % (svt.tgd))
    print("toc: %s" % (svt.toc))


def handle_efsf1(sf1,svp):
    tocr = sf1[20:22]
    toc = struct.unpack('=H',tocr)[0]
    svp.toc = toc*(2**4)

    af0r1r = sf1[30:31]
    af0r1 = format(int(af0r1r.encode('hex'),16),'008b')

    af0r2r = sf1[29:30]
    af0r2 = format(int(af0r2r.encode('hex'),16),'008b')

    af0r3r = sf1[28:29]
    af0r3 = format(int(af0r3r.encode('hex'),16),'008b')[:6]

    af1r1r = sf1[24:25]
    af1r1 = format(int(af1r1r.encode('hex'),16),'008b')

    af1r2r = sf1[25:26]
    af1r2 = format(int(af1r2r.encode('hex'),16),'008b')

    af0aux = af0r1 + af0r2 + af0r3
    af1aux = af1r2 + af1r1

    af2r1 = sf1[26:27]
    af2aux = format(int(af2r1.encode('hex'),16),'008b')

    tgdr = sf1[16:17]
    tgdr2 = format(int(tgdr.encode('hex'),16),'008b')
```

```

    af0 = twos_comp(int(af0aux,2),22)
    af1 = twos_comp(int(af1aux,2),16)
    af2 = twos_comp(int(af2aux,2),8)
    tgd = twos_comp(int(tgdr2,2),8)

    svp.tgd = (tgd*(2**(-31)))
    svp.af0 = (af0*(2**(-31)))
    svp.af1 = (af1*(2**(-43)))
    svp.af2 = (af2*(2**(-55)))

def handle_efsf2(sf2,svp):
    ioder = sf2[2:3]
    iode = struct.unpack('=b',ioder)[0]
    crsr = sf2[:2]
    crs = struct.unpack('=h',crsr)[0]
    svp.Crs = (crs * (2**(-5)))
    delta_nr = sf2[5:7]
    delta_n = struct.unpack('=h',delta_nr)[0]
    svp.delta_n = (delta_n * (2**(-43)) * math.pi)
    M0r = sf2[8:11] + sf2[4:5]
    M0 = struct.unpack('=L',M0r)[0]
    svp.M0 = (M0 * (2**(-31)) * math.pi)
    Cucr = sf2[13:15]
    Cuc = struct.unpack('=h',Cucr)[0]
    svp.Cuc = Cuc*(2**(-29))
    er = sf2[16:19] + sf2[12:13]
    e = struct.unpack('=l',er)[0]
    svp.e = e*(2**(-33))
    Cusr = sf2[21:23]
    Cus = struct.unpack('=h',Cusr)[0]
    svp.Cus = Cus *(2**(-29))
    sqrt_Ar = sf2[24:27] + sf2[20:21]
    sqrt_A = struct.unpack('=L',sqrt_Ar)[0]
    svp.sqrt_A = sqrt_A * (2**(-19))
    toer = sf2[29:31]
    toe = struct.unpack('=H',toer)[0]
    svp.toe = toe *(2**(4))
```

```

def twos_comp(val, bits):
  if(val & (1 << (bits-1))) !=0:
    val = val-(1<<bits)
  return val

def handle_efsf3(sf3,svr):
  cicr = sf3[1:3]
  cic = struct.unpack('=h',cicr)[0]
  svr.Cic = cic*(2**(-29))
  omega0r = sf3[4:7] + sf3[:1]
  omega0 = struct.unpack('=l',omega0r)[0]
  svr.omega0 = omega0*(2**(-31))*math.pi
  cisr = sf3[9:11]
  cis = struct.unpack('=h',cisr)[0]
  svr.Cis = cis*(2**(-29))
  i0r = sf3[12:15] + sf3[8:9]
  i0 = struct.unpack('=l',i0r)[0]
  svr.i0 = i0*(2**(-31))*math.pi
  crcr = sf3[17:19]
  crc = struct.unpack('=h',crcr)[0]
  svr.Crc = crc*(2**(-5))
  omegar = sf3[20:23] + sf3[16:17]
  omega = struct.unpack('=l',omegar)[0]
  svr.omega = omega*(2**(-31))*math.pi
  omega_dotr = sf3[26:27] + sf3[25:26] + sf3[24:25]
  omega_dot = twos_comp(int(omega_dotr.encode('hex'),16),24)
  svr.omega_dot = omega_dot*(2**(-43))*math.pi

  idotaux1r = sf3[28:29]
  idotaux1 = format(int(idotaux1r.encode('hex'),16),'008b')[:6]

  idotaux2r = sf3[29:30]
  idotaux2 = format(int(idotaux2r.encode('hex'),16),'008b')

  idotr = idotaux2 + idotaux1

  idot = twos_comp(int(idotr,2),14)
```

```
  svr.idot = idot*(2**(-43))*math.pi

getcontext().prec=15
while True:
  port.reset_input_buffer()
  currentTime = time.time()
  if(currentTime-start > 60):
    print("Clock bias read: %s" % clock_biasread)
    print("Clock drift read: %s" % drift_clock)
    lastRcvTow = MostTowCount()
    print("Tow used: %s" % lastRcvTow)
    for svcp in svList:
      if(svcp.pos==1 and svcp.rcvTow==lastRcvTow):
        print("--Sat Pos %s: New Eph--" % svcp.id)
        sat_pos(svcp)
        print("---------")
        if hasattr(svcp,'Oldsf1'):
          print("--Sat Pos %s: Old Eph--" % svcp.id)
          svAux = sv(svcp.id,svcp.pr,svcp.rcvTow)
          handle_efsf1(svcp.Oldsf1,svAux)
          handle_efsf2(svcp.Oldsf2,svAux)
          handle_efsf3(svcp.Oldsf3,svAux)
          sat_pos(svAux)
          variance_ef = math.sqrt((svcp.X-svAux.X)**2 + (svcp.Y-svAux.Y)**2 + (svcp.Z-svAux.Z)**2)
          svcp.variance_ef = variance_ef
          print("--Distance: %s" % variance_ef)
          if(math.fabs(variance_ef)>3):
            svcp.health=1
          print("---------")
      if(svcp.rcvTow!=lastRcvTow):
        svcp.pos=0

    tupSv = svPosCount()
    #checkHealth(tupSv[1])
    #tupSv = svPosCount()

    if(tupSv[0]>=4):
```

```
      lataux,lonaux,ecefx, ecefy, ecefz = getFix(tupSv[1],0)

      checkElev(tupSv[1],lataux,lonaux)
      fixSagnac(tupSv[1],ecefx, ecefy, ecefz)
    tupSv = svPosCount()
    delta_clock_tow_old = delta_clock_tow
    delta_clock_old = delta_clock
    if(tupSv[0]>=4):
      getFix(tupSv[1],1)
      tupSv = svPosCount()
      raim(tupSv[0],tupSv[1])
      for svreset in svList:
        svreset.pos = 0
    if(tupSv[0]>=3 and delta_clock_set==True and delta_clock_old
   !=0):
      clock = (lastRcvTow-delta_clock_tow_old)*(drift_clock)+
   delta_clock_old
      getFix3SAT(tupSv[1], clock)
      printSvData(tupSv[1])
      for svreset in svList:
        svreset.pos = 0
        svreset.variance = 0
        svreset.variancecn0 = 0
        svreset.delta_raim = 0
        svreset.variance_ef = 0
        svreset.variance_range = 0
    start = currentTime
    print("--Delta_Clock stored: %s" % delta_clock)
    print("--Tow stored: %s" % delta_clock_tow)
  rcv = port.read(2)
  hexr = binascii.hexlify(rcv);
  if len(hexr)!=4:
    hexr = '0000'
  if(int(hexr,16)==int('B562',16)):
    clid = binascii.hexlify(port.read(2))
    if(int(clid,16)==int('0B31',16)):
      port.read(2)
      sv_idr = port.read(4)
```

```
        sv_id = struct.unpack('=L',sv_idr)[0]
        howr = port.read(4)
        how = struct.unpack('=L',howr)[0]
        if(how!=0):
          sf1 = port.read(32)
          sf2 = port.read(32)
          sf3 = port.read(32)
          svm = checkSvList(sv_id)
          if(svm[0]==True):
            if hasattr(svm[1],'sf1'):
              svm[1].Oldsf1 = svm[1].sf1
              svm[1].Oldsf2 = svm[1].sf2
              svm[1].Oldsf3 = svm[1].sf3
            svm[1].sf1 = sf1
            svm[1].sf2 = sf2
            svm[1].sf3 = sf3
            handle_efsf1(sf1,svm[1])
            handle_efsf2(sf2,svm[1])
            handle_efsf3(sf3,svm[1])
            svm[1].pos = 1
            print("****Received eph %s" % svm[1].id)
      elif(int(clid,16)==int('0215',16)):
        port.read(2)
        rcvTowr = port.read(8)
        rcvTow = struct.unpack('=d',rcvTowr)[0]
        weekr = port.read(2)
        week = struct.unpack('=H',weekr)
        port.read(1)
        ir = port.read(1)
        i = struct.unpack('=B',ir)[0]
        port.read(4)
        for x in range (1,i):
          prMesr = port.read(8)
          prMes = struct.unpack('=d',prMesr)[0]
          cpMesr = port.read(8)
          cpMes = struct.unpack('=d',cpMesr)[0]
          doMesr = port.read(4)
          doMes = struct.unpack('=f',doMesr)[0]
```

```
    gIdr = port.read(1)
    gId = struct.unpack('=B',gIdr)[0]
    svIdr = port.read(1)
    svId = struct.unpack('=B',svIdr)[0]
    port.read(4)
    cn0r = port.read(1)
    cn0 = struct.unpack('=B',cn0r)[0]
    port.read(3)
    trkStatr = port.read(1)
    trkStat = format(int(trkStatr.encode('hex'),16),'008b')[4:]
    if(gId==0):
      print("$$$$$$$Received: %s" % svId)
      tupsv = checkSvList(svId)
      if(tupsv[0]==False):
        nSv = sv(svId,prMes,rcvTow)
        nSv.bpr = prMes
        nSv.pos = 0
        nSv.cn0 = cn0
        nSv.health = 0
        nSv.variance = Decimal(0)
        nSv.variancecn0 = Decimal(0)
        svList.append(nSv)
        port.write(pollEphSv(svId))
      elif(tupsv[0]==True):
        tupsv[1].oldcn0 = tupsv[1].cn0
        tupsv[1].cn0 = cn0

        tupsv[1].oldpr = tupsv[1].bpr
        tupsv[1].pr = prMes
        tupsv[1].bpr = prMes
        tupsv[1].oldrcvTow = tupsv[1].rcvTow
        tupsv[1].rcvTow = rcvTow

        delta_v = (Decimal(tupsv[1].oldpr) - Decimal(tupsv[1].pr))/(Decimal(tupsv[1].rcvTow)-Decimal(tupsv[1].oldrcvTow))
        delta_f = (delta_v*Decimal(l1freq))/Decimal(c)
        print("--Doppler Effect--")
```

```
        print("-RcvTow: %s" % tupsv[1].rcvTow)
        print("-OldRcvTow: %s" % tupsv[1].oldrcvTow)
        print("-Expected: %s Hz" % delta_f)
        print("-Expected: %s m/s" % delta_v)
        print("-Observed: %s Hz" % doMes)
        variance = math.fabs(((Decimal(doMes)-delta_f)/Decimal(doMes)))
        print("-Error observed/expected: %s" % variance)
        if variance>tupsv[1].variance:
          tupsv[1].variance = variance
        print("------------------")
        print("--CN0 variation--")
        print("-Old: %s" % tupsv[1].oldcn0)
        print("-New: %s" % tupsv[1].cn0)
        variancecn0 = math.fabs((tupsv[1].oldcn0-tupsv[1].cn0)/tupsv[1].oldcn0)
        print("-delta: %s" % variancecn0)
        print("-------------------")
        if variancecn0>tupsv[1].variancecn0:
          tupsv[1].variancecn0 = variancecn0
        lastRcvTow = rcvTow
        if(delta_clock_set==True):
          time_drift = (rcvTow - delta_clock_tow)*drift_clock
          rangeD = Decimal(tupsv[1].pr) - Decimal(c*(delta_clock+time_drift))
          tupsv[1].rangeD = rangeD
          print("-Estimated range: %s" % rangeD)
          print("-Tow: %s" % tupsv[1].rcvTow)
          print("-Pseudo range: %s" % tupsv[1].pr)
          print("-------------------")
        if(tupsv[1].pos==0):
          print("****Sv %s ephem polled." % tupsv[1].id)
          port.write(pollEphSv(svId))
    port.read(1)
elif(int(clid,16)==int('0122',16)):
  if(delta_clock_set==False):
    port.read(2)
    port.read(4)
```

```
        c_bias = port.read(4)
        clock_biasread = struct.unpack('=l',c_bias)[0]
        clock_biasread = clock_biasread*(10**-9)
        c_drift = port.read(4)
        drift_clock = struct.unpack('=l',c_drift)[0]
        drift_clock = drift_clock*(10**-9)
```

Bibliography

[1] Christopher J. Hegarty Elliott D. Kaplan. *Understanding GPS. Principles and applications*, volume 59. Artech House, Norwood, Massachusetts, USA, 1997.

[2] LD Landau. Global Positioning System Directorate System Engineering & Integration. *Zhurnal Eksperimental'noi i Teoreticheskoi Fiziki*, 1937.

[3] [online] available at: https://www.e-education.psu.edu/geog862/node/1756 [accessed 23 jun. 2019].

[4] Kevin Brown. *Reflections on relativity.* lulu.com, Morrisville, North Carolina , United States, 2004.

[5] U-blox, (2019). [online] https://www.u-blox.com/sites/default/files/products/documents/evk-m8t-userguide-(ubx-14041540).pdf [accessed: 24 jun. 2019].

[6] "raspberry pi 1 - sparkfun electronics", wikipedia.org, 2019. [online]. available: https://pt.wikipedia.org/wiki/raspberry-pi [accessed: 14 jan. 2019].

[7] Ettus, (2019). [online] https://kb.ettus.com/n200/n210 [accessed: 24 jun. 2019].

[8] [online] available at: https://www.u-blox.com/sites/default/files/products/documents/u-blox8-m8-receiverdescrprotspec-(ubx-13003221)-public.pdf [accessed: 24 jun. 2019].

[9] [online] available at: https://medium.com/signals-of-change/gps-spoofing-of-ships-could-be-a-new-cyberweapon-5b389dcc72ae [accessed: 24 jun. 2019].

[10] Kai Borre and Dennis M Akos. *A Software-Defined GPS and Galileo Receiver*. Birkhäuser, Basel, Switzerland, 2007.

[11] Todd E Humphreys, Brent M Ledvina, Virginia Tech, Mark L Psiaki, Brady W O Hanlon, and Paul M Kintner. Assessing the Spoofing Threat : Development of a Portable GPS Civilian Spoofer. *Proceedings of the 21st International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2008) September 16 - 19, 2008 Savannah International Convention Center Savannah, GA*, pages 2314–2325, 2009.

[12] Kang Wang, Shuhua Chen, and Aimin Pan. Time and position spoofing with open source projects. *Black Hat*, 148, 2015.

[13] Mohammad Reza Mosavi, Sadaf Azarshahi, Iman Emamgholipour, and Ali Asghar Abedi. Least squares techniques for GPS receivers positioning filter using pseudo-range and carrier phase measurements. *Iranian Journal of Electrical and Electronic Engineering*, 10(1):18–26, 2014.

[14] Nielsen and Michael A. *Neural Networks and Deep Learning*. Determination Press, 2015.

[15] [online] available at: https://github.com/osqzss/gps-sdr-sim [accessed 23 jun. 2019].

[16] P B Ober and D Harriman. On the Use of Multiconstellation-RAIM for Aircraft Approaches. *Proceedings of Ion*, (September):26–29, 2006.