UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

EXTRAÇÃO E ANÁLISE DE MEMÓRIA VOLÁTIL EM AMBIENTES ANDROID: UMA ABORDAGEM VOLTADA À RECONSTRUÇÃO DE TRAJETÓRIAS

JOÃO PAULO CLAUDINO DE SOUSA

ORIENTADOR: ROBSON DE OLIVEIRA ALBUQUERQUE
CO-ORIENTADOR: JOÃO JOSÉ COSTA GONDIM

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA
ÁREA DE CONCENTRAÇÃO INFORMÁTICA FORENSE E SEGURANÇA DA INFORMAÇÃO

PUBLICAÇÃO: PPGENE.DM - 647 A/16

BRASÍLIA / DF: 09/2016

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

EXTRAÇÃO E ANÁLISE DE MEMÓRIA VOLÁTIL EM AMBIENTES ANDROID: UMA ABORDAGEM VOLTADA À RECONSTRUÇÃO DE TRAJETÓRIAS

JOÃO PAULO CLAUDINO DE SOUSA

DISSERTAÇÃO DE MESTRADO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE.

APROVADA POR:

Robson Oliveira de Albuquerque, Dr., ENE/UNB
(ORIENTADOR)

Flávio Elias Gomes de Deus, Dr., ENE/UNB
(EXAMINADOR INTERNO)

Dino Macedo Amaral, Banco do Brasil
(EXAMINADOR EXTERNO)

Brasília, 29 de Setembro de 2016.

FICHA CATALOGRÁFICA

SOUSA, JOÃO
Extração e análise de memória volátil em ambientes Android: uma abordagem voltada à reconstrução de trajetórias [Distrito Federal] 2016.
xvii, 115p., 297 mm (ENE/FT/UnB, Mestre, Engenharia Elétrica, 2016).

Dissertação de Mestrado – Universidade de Brasília, Faculdade de Tecnologia. Departamento de Engenharia Elétrica.

1. Android 2. Memória RAM
3. NMEA

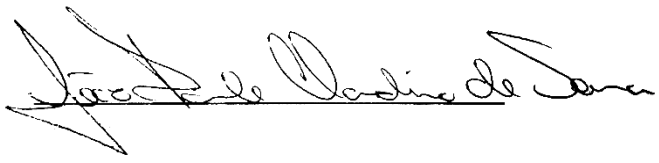
I. ENE/FT/UnB. II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

SOUSA, J. P. C. (2016). Extração e análise de memória volátil em ambientes Android: uma abordagem voltada à reconstrução de trajetórias. Dissertação de Mestrado, Publicação PPGENE.DM - 647 A/016, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 115p.

CESSÃO DE DIREITOS

NOME DO AUTOR: João Paulo Claudino de Sousa
TÍTULO DA DISSERTAÇÃO: Extração e análise de memória volátil em ambientes Android: uma abordagem voltada à reconstrução de trajetórias.
GRAU/ANO: Mestre/2016.



João Paulo Claudino de Sousa
Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910-900 – Brasília – DF - Brasil

À minha amada esposa Lillian Carine, ao meu pequeno anjo Maria Sofia e à minha querida mãe Marlene.

AGRADECIMENTOS

Aos professores João José Costa Gondim e Robson de Oliveira Albuquerque, primeiramente por terem aceitado me orientar neste projeto, mas principalmente pelo apoio, incentivo e dedicação, sem os quais os resultados apresentados neste trabalho não teriam sido possíveis.

Agradeço à Universidade de Brasília, ao Departamento de Polícia Federal e à Polícia Civil do Distrito Federal por propiciarem minha participação no Mestrado Profissional em Informática Forense.

Também sou grato à Fundação de Peritos em Criminalística Ilaraine Acácio Arce e à Associação Brasiliense de Peritos em Criminalística por todo o suporte fornecido para a participação em eventos científicos.

Agradeço aos meus colegas de trabalho e aos meus amigos de infância, em especial ao estatístico Thiago Morais de Carvalho, pela ajuda com os trabalhos acadêmicos.

Por fim, agradeço à minha amada esposa, por toda a paciência que teve neste período tão conturbado, onde nosso anjinho, Maria Sofia, passou por tantos problemas de saúde. Em todos estes momentos, ela esteve ao meu lado e meu deu apoio para não desistir do mestrado.

A todos, os meus sinceros agradecimentos.

RESUMO

EXTRAÇÃO E ANÁLISE DE MEMÓRIA VOLÁTIL EM AMBIENTES ANDROID: UMA ABORDAGEM VOLTADA À RECONSTRUÇÃO DE TRAJETÓRIAS

Autor: João Paulo Claudino de Sousa
Orientador: Robson de Oliveira Albuquerque
Programa de Pós-graduação em Engenharia Elétrica
Brasília, 29 de setembro de 2016

Dispositivos Android são amplamente utilizados e podem funcionar como receptores GPS. Informações de tempo e posicionamento possuem grande relevância no campo investigativo, todavia, os dados armazenados em mídia não-volátil podem ser limitados no que diz respeito à reconstrução de trajetórias. Este trabalho propõe um método de recuperação de mensagens com coordenadas GPS armazenadas na memória RAM de dispositivos móveis Android, a fim de reconstruir o trajeto trilhado pelo dispositivo. Estudos relacionados encontrados na literatura se mostraram limitados, uma vez que a maioria dos trabalhos produzidos são voltados à recuperação de informações de posicionamento de artefatos tipicamente encontrados em memória não-volátil. No desenvolvimento deste trabalho foi possível detalhar a arquitetura GPS em ambientes Android, viabilizando o entendimento dos principais mecanismos de armazenamento de coordenadas de posicionamento. Nesta linha, constatou-se que o protocolo NMEA 0183 tem importância fundamental na comunicação dos receptores GPS com os diversos tipos de aparelhos, uma vez que provê uma forma padronizada de transmissão dos dados de posicionamento. Ademais, foram realizados experimentos em diferentes ambientes, com aparelhos de distintas arquiteturas, para analisar a viabilidade da reconstrução de trajetórias com base nas mensagens do protocolo NMEA 0183 recuperadas da memória RAM, bem como nas estruturas textuais com características de coordenadas geodésicas. No desenvolvimento da técnica, foi possível verificar as dificuldades que podem atrapalhar o processo de extração e análise dos dados, bem como foram desenvolvidas ferramentas para auxiliar o processo.

ABSTRACT

EXTRACTION AND ANALYSIS OF VOLATILE MEMORY IN ANDROID SYSTEMS: AN APPROACH FOCUSED ON TRAJECTORY RECONSTRUCTION

Author: João Paulo Claudino de Sousa
Supervisor: Robson de Oliveira Albuquerque
Programa de Pós-graduação em Engenharia Elétrica
Brasília, 29th September of 2016

Android devices are widely used and can function as GPS receivers. Time and position information have great relevance in the investigative field, however, data stored in non-volatile media may be limited with regard to the reconstruction of trajectories. This study proposes a method for recovering messages with GPS coordinates stored in RAM memory of Android mobile devices, in order to rebuild the trajectory of the device. Related literature proved limited since the majority of the studies produced were directed to recovery of positioning information from artifacts typically found in non-volatile memory. In the development of this work, it was possible to detail GPS architecture on Android environments, providing a better understanding of the positioning coordinate storage mechanisms. Along this line, it was discovered that the NMEA 0183 protocol is critical for the communication of GPS receivers with the various types of equipment, since it provides a standard way of broadcasting positioning data. Experiments were performed in different environments, with different device architectures, to analyze the feasibility of reconstruction of trajectories based on the NMEA 0183 protocol messages and geodetic coordinates in textual format retrieved from RAM memory. In developing this technique, it was possible to verify the problems that can hinder the process of extraction and analysis of data. In addition, tools have been developed to aid the process of trajectory reconstruction.

SUMÁRIO

LISTA DE TABELAS

LISTA DE FIGURAS

LISTA DE SÍMBOLOS, NOMENCLATURA E ABREVIAÇÕES

A-GPS	Assisted GPS
ADB	Android Debug Bridge
ART	Android Runtime
DD	Decimal Degrees
DGPS	Differential Global Positioning System
DM	Decimal Minutes
DMA	Defense Mapping Agency
DMS	Degrees, Minutes and Seconds
DVM	Dalvik Virtual Machine
FROST	Forensic Recovery Of Scrambled Telephones
GLONASS	Globalnaya Navigatsionnaya Sputnikovaya
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
HAL	Hardware Abstraction Layer
IPC	Inter-Process Communication
LiME	Linux Memory Extractor
LKM	Loadable Kernel Module
LMAP	Linux Physical Memory Acquisition Tool
MCS	Master Control Station
MDA	Memory Dump Analyser
MSL	Mean Sea Level
NDK	Native Development Kit
NGA	National Geospatial-Intelligence Agency
NMEA	National Marine Eletronics Association
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
SDK	Software Development Kit
SUPL	Secure User Plane Location
TTFF	Time-To-First-Fix
SAD 69	South American Datum 1969
SIRGAS2000	Sistema de Referência Geocêntrico para as Américas
SOPS	Space Operations Squadron
USAF	United States Air Force
UTC	Universal Time Coordinated
WSG84	World Geodetic System 1984

1. INTRODUÇÃO

A utilização de dispositivos eletrônicos com capacidade computacional para o cometimento de atos ilícitos é cada vez mais crescente nos diversos cenários sociais. Neste sentido, os exames periciais de informática precisam de técnicas e procedimentos com embasamento científico para coleta, análise e apresentação das evidências encontradas. A perícia digital tem como objetivo buscar informações relativas a eventos passados em uma investigação, que pode ou não ter caráter criminal. A partir da análise dos eventos ocorridos é possível reconstruir as ações executadas nos diversos equipamentos e mídias questionados (Vecchia, 2014).

Android é o sistema operacional de mais de um bilhão de *smartphones* e *tablets* pelo mundo (Android, 2016), possuindo grande relevância para comunidade forense. Com efeito, é natural um aumento da demanda por perícias em dispositivos com o referido sistema operacional, visto que muitos crimes são cometidos com o auxílio destes aparelhos.

Neste sentido, diversos trabalhos foram realizados visando a aquisição e análise de dados provenientes tanto de memórias voláteis como não voláteis. A aquisição de memória RAM (*Random Access Memory*) possui papel importante nas investigações, uma vez que evidências coletadas neste tipo de memória podem conter informações únicas, como dados de processos em diferentes estados, arquivos, conexões de rede, informações de geoposicionamento, dentre outros dados, que geralmente não estão armazenados em memória não-volátil.

No campo investigativo, informações de posicionamento são importantes pois, podem fornecer elementos para elucidação da autoria e do *modos operandi* de algum fato delituoso. Atualmente, a maioria dos *smartphones* e *tablets* possui chips receptores GPS (*Global Positioning System*), o que os tornam fontes valiosas para coletas de dados de posicionamento. Além disso, dados de tempo e velocidade também são providos por receptores GPS, tendo relevância na elucidação de crimes e acidentes de trânsito.

As equipes periciais responsáveis pela coleta de dados de cenas de crime frequentemente encontram telefones celulares dispostos juntamente com outras evidências. Entretanto, muitas vezes os dispositivos não são tratados da forma adequada, fazendo com que dados valiosos sejam perdidos. Equipes policiais também têm contato frequente com *smartphones* apreendidos com criminosos, sendo que muitas vezes os aparelhos são recuperados logo após o cometimento do ato ilícito. A Figura 1.1 mostra um aparelho encontrado em um acidente de trânsito, onde o veículo estava em alta velocidade. O aparelho, caso fosse submetido a um exame pericial de

informática, poderia trazer informações sobre o trajeto percorrido pelo veículo, o horário em que o acidente aconteceu, bem como a velocidade nos momentos anteriores à colisão.

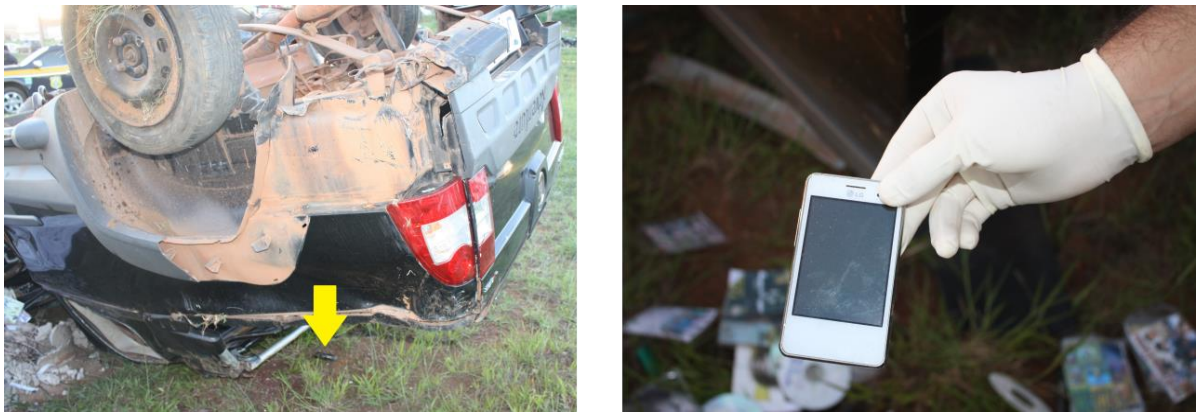


Figura 1.1: *Smartphone* coletado em cena de crime

Este trabalho teve como motivação a ausência de procedimentos e técnicas capazes de extrair e analisar dados de posicionamento associados aos últimos caminhos trilhados por um dispositivo móvel. Nesta linha, o trabalho discute como as informações de posicionamento podem ser recuperadas da memória RAM de dispositivos móveis Android, visando a reconstrução da trajetória do dispositivo nos momentos anteriores ao exame pericial. Também é feita uma análise da persistência dos dados em aparelhos com distintas arquiteturas e em diferentes condições de operação.

1.1. OBJETIVOS

Este trabalho tem por objetivo geral a determinação de procedimentos e técnicas a serem adotados em exames periciais de informática em dispositivos móveis Android, que permitam reconstruir, ainda que parcialmente, a trajetória do dispositivo com base em informações de posicionamento recuperadas da memória volátil do aparelho.

A seguir, são listados alguns objetivos específicos desta pesquisa.

1.1.1. Objetivos específicos

- Analisar a arquitetura GPS nos dispositivos Android, desde as camadas mais baixas do sistema operacional até o nível de aplicação.

- Detalhar o funcionamento dos serviços de localização, principalmente aqueles voltados para o sistema GPS.
- Detalhar o fluxo de mensagens do protocolo NMEA (*National Marine Eletronics Association*) 0183, dentro do escopo dos sistemas Android.
- Analisar os métodos e técnicas de recuperação de memória volátil para ambientes Linux, analisando a viabilidade da aplicação em sistemas Android.
- Realizar experimentos a fim de se reconstruir a trajetória de dispositivos móveis, em diversos ambientes, com diferentes condições de tráfego e clima, fazendo uso de aparelhos com múltiplas arquiteturas e versões do sistema operacional Android.
- Determinar a viabilidade da reconstrução de trajetória dos dispositivos com base em mensagens do protocolo NMEA 0183 recuperadas da memória RAM dos aparelhos.
- Determinar a viabilidade da reconstrução de trajetória dos dispositivos com base em coordenadas geodésicas em formato textual recuperadas da memória RAM dos aparelhos.
- Apresentar um procedimento para reconstrução de trajetórias de dispositivos móveis Android, com base nos dados recuperados da memória volátil.

1.2. HIPÓTESE E JUSTIFICATIVA

Acredita-se que seja possível recuperar coordenadas de posicionamento da memória volátil de dispositivos Android, de forma que, utilizando-se dos dados de geoposicionamento, dos dados temporais e dos dados de velocidade, seja viável reconstruir, ainda que parcialmente, os últimos movimentos realizados pelo aparelho.

Dispositivos GPS, em geral, fazem uso do protocolo NMEA 0183, que especifica o processo de comunicação entre um receptor GPS e os diversos sistemas capazes de prover serviços de localização. Estas mensagens transitam pelas diversas camadas do sistema Android, indicando que podem estar presentes em várias regiões da memória RAM.

Além disso, existe toda uma estrutura para tratar os serviços de localização, que fazem uso de APIs na linguagem JAVA (*Location Strategies*, 2016), que também deixam rastros de coordenadas de posicionamento na memória volátil.

1.3. MÉTODO

Na primeira parte deste trabalho foi realizada uma pesquisa descritiva buscando entender a teoria associada ao objeto de estudo. Primeiramente, foi necessário compreender como

funcionam os sistemas de navegação por satélite e sua relação com os dispositivos receptores, em especial, *smartphones* e *tablets* Android. A pesquisa englobou o estudo da especificação NMEA, uma vez que a transmissão de dados de posicionamento estava intrinsecamente relacionada com o protocolo NMEA 0183. Desta forma, foi possível entender a arquitetura GPS nos sistemas Android, desde a comunicação *driver*/receptor GPS, até os *frameworks* de aplicação, resultando em uma compreensão de como estes dados podem estar dispostos na memória RAM dos aparelhos.

Por outro lado, também foi necessário pesquisar sobre os procedimentos para recuperação de memória volátil em ambientes Android, visto que os dados necessários para reconstrução da trajetória dos dispositivos geralmente não são encontrados em mídias não voláteis.

Com a agregação de todos os fundamentos pesquisados, foi possível formular um método para obtenção de dados de posicionamento da memória volátil de sistemas Android, com a finalidade de reconstruir o trajeto percorrido por um dispositivo. O método estabelece as diretrizes para recuperação de mensagens NMEA do tipo GPGGA e GPRMC, bem como coordenadas geodésicas no formato Grau Decimais.

A segunda parte do trabalho consistiu em uma pesquisa experimental. Informações da memória RAM não persistem por muito tempo, o que pode ter grande implicação no meio forense, onde o tempo para obtenção dos dados pode ser crucial. Desta forma, foram montados cenários de testes para analisar a viabilidade da recuperação dos dados de posicionamento em diferentes ambientes, com aparelhos de arquiteturas diversas.

Por fim, todos os *dumps* de memória obtidos nos testes foram submetidos a uma ferramenta desenvolvida para extração e análise das coordenadas de posicionamento. A partir da análise dos dados, foi possível observar o comportamento da memória volátil, bem como o decaimento do número de mensagens recuperadas ao longo do tempo. Foi possível constatar que, na maioria dos experimentos, a trajetória dos dispositivos pôde ser parcialmente reconstruída, o que torna o método desenvolvido relevante para a comunidade forense.

1.4. ESTRUTURA DO TRABALHO

No capítulo 2 são apresentados conceitos relacionados aos sistemas de navegação por satélite, englobando toda a parte de geodesia, bem como a especificação NMEA. Neste capítulo também são apresentados trabalhos relacionados à forense em dispositivos GPS e em sistemas Android, com foco nos artefatos com coordenadas de posicionamento encontrados em memória não volátil.

No capítulo 3, os mecanismos de escalada de privilégios (*rooting*) e de obtenção de memória RAM em ambientes Linux são explorados. As técnicas descritas neste capítulo são analisadas tendo como referência o sistema Android, uma vez que muitos trabalhos já realizados não foram explorados no referido sistema.

No capítulo 4 descreve-se a arquitetura GPS em ambientes Android, detalhando-se um estudo sobre como as mensagens NMEA e as coordenadas geodésicas se encontram dispostas na memória RAM. Por fim, o método para obtenção de informações de posicionamento é descrito, sendo dividido em duas partes: recuperação de mensagens NMEA e recuperação de coordenadas geodésicas no formato Graus Decimais.

No capítulo 5 são apresentados os experimentos realizados, juntamente com os resultados obtidos. Nos testes realizados, mostra-se a viabilidade de reconstrução de trajetórias com base nos dados obtidos da memória RAM. Todavia, muitos fatores podem influenciar na qualidade dos dados recuperados, sendo que neste capítulo, os resultados obtidos são analisados cautelosamente.

Por fim, a conclusão é apresentada no capítulo 6, juntamente com indicações de trabalhos futuros na área de forense em dispositivos móveis.

2. SISTEMAS GLOBAIS DE NAVEGAÇÃO POR SATÉLITE

Sistemas de Navegação Global por Satélites (*Global Navigation Satellite Systems* - GNSS) são os sistemas de satélites que proveem posicionamento geoespacial com cobertura terrestre global. Entre os principais sistemas GNSS operacionais estão inclusos o sistema norte americano NAVSTAR-GPS, ou simplesmente GPS, e o sistema russo GLONASS (*Globalnaya navigatsionnaya sputnikovaya*). Existem também outros sistemas em desenvolvimento, como o sistema europeu Galileo, o qual foi projetado para uso civil, bem como o sistema chinês Compass (Vaz, Souza & Junior, 2013).

2.1. O SISTEMA GPS

O sistema GPS é de propriedade do governo dos Estados Unidos sendo operado e mantido pela Força Aérea (*United States Air Force* - USAF). O sistema provê serviços de posicionamento, navegação e tempo para civis e militares, sendo utilizado desde 1978. Cabe ressaltar que o sistema era limitado para uso militar e foi sendo gradativamente liberado para uso civil.

A USAF é responsável pela concepção, desenvolvimento, operação, manutenção e modernização do sistema. O segundo esquadrão de operações espaciais (2º *Space Operations Squadron* - 2º SOPS), o qual possui instalações no estado do Colorado nos Estados Unidos, mantém a integridade e a operacionalidade da constelação de satélites por meio de uma rede de antenas terrestres e estações de monitoramento que ficam espalhadas pelo mundo, garantindo o desempenho e a confiabilidade do sistema (Grimes, 2008).

O sistema é composto pelos segmentos espacial, de controle e do usuário, que se integram para fornecer os serviços de posicionamento.

2.1.1. Segmento espacial

O segmento espacial é formado por uma constelação de satélites que transmite sinais de rádios para os usuários. O governo americano se compromete em manter pelo menos 24 satélites operacionais, 95% do tempo. Os satélites orbitam a terra a uma altitude de aproximadamente 20.200 km, dando duas voltas no planeta por dia. Os satélites estão dispostos em seis planos orbitais igualmente espaçados, com quatro satélites em cada plano, garantindo visibilidade de pelo menos quatro satélites em qualquer ponto da terra, como pode ser visto na Figura 2.1 (*Space Segment*, 2016).

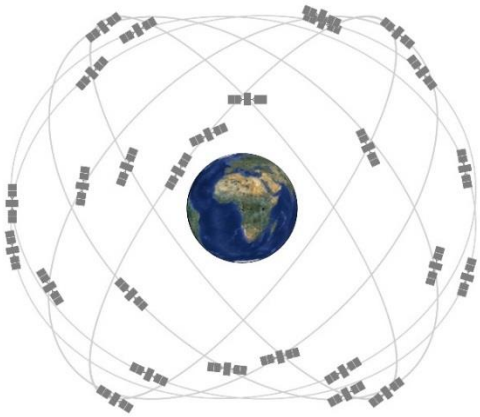


Figura 2.1: Planos orbitais da constelação GPS (*Space Segment*, 2016)

Os satélites transmitem sua posição orbital juntamente com um carimbo de tempo (*timestamp*) em duas frequências distintas, uma para uso militar (1227.6 MHz) e outra para uso civil (1575.42 MHz).

2.1.2. Segmento de controle

O segmento de controle é constituído por estações terrestres que têm por obtivo monitorar, corrigir e garantir o funcionamento do sistema. Existem diversos tipos de estações que ficam localizadas em diferentes regiões do planeta, conforme ilustrado na Figura 2.2. A estação principal (*Master Control Station* - MCS) é a única que pode se comunicar com os satélites e operá-los diretamente. Se por alguma razão a MCS ficar indisponível, uma estação alternativa pode assumir o comando. Quando da realização deste trabalho, o segmento de controle era composto pela MCS, pela MCS alternativa, por 11 antenas de comando e controle, e por 15 estações de monitoramento (*Control Segment*, 2016).

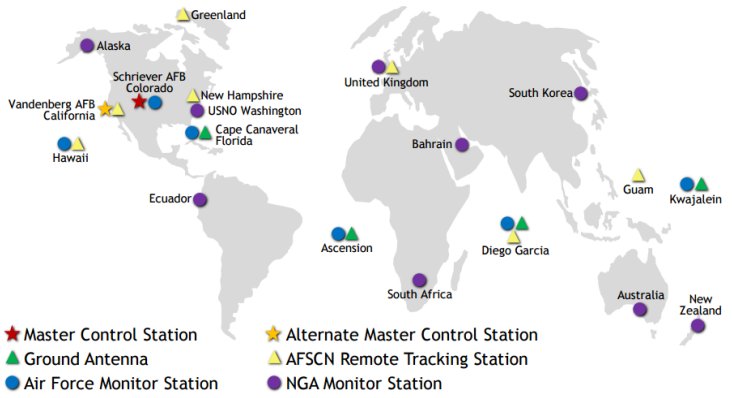


Figura 2.2: Estações do segmento de controle (*Control Segment*, 2016)

2.1.3. Segmento de usuário

Este segmento é composto pelos receptores GPS, que podem ser dispositivos dedicados ao uso em aplicações de geoposicioamento, ou podem estar embarcados em outros dispositivos, como *smartphones*, *tablets*, *smartwatches*, dentre outros. Esses receptores coletam dados enviados pelos satélites, transformando-os em coordenadas, distâncias, tempo, deslocamento e velocidade, através de processamento em tempo real ou a posteriori (Carvalho & Araújo, 2009).

Os receptores possuem, dentre outros, os seguintes componentes principais: uma antena com pré-amplificador; uma seção de radiofrequência para identificação e processamento do sinal e um microprocessador para controle do receptor, amostragem e processamento dos dados. A Figura 2.3 mostra a placa mãe de um aparelho da marca Samsung, modelo GT-P7310 (Galaxy Tab 8.9), e seu respectivo micro controlador GPS, do fabricante Broadcom, modelo BCM4751.

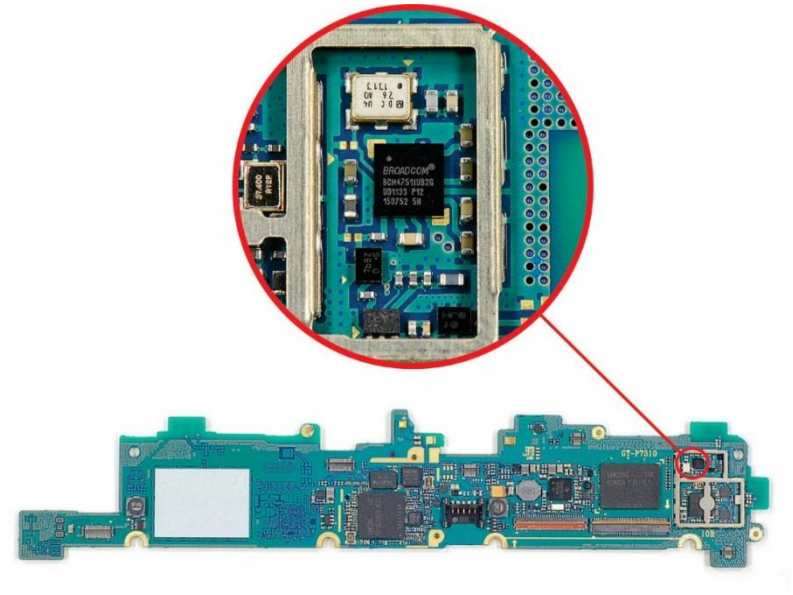


Figura 2.3: Micro controlador GPS do aparelho Samsung GT-P7310

2.1.4. Cálculo de posicionamento

Um receptor GPS calcula a distância até os satélites GPS com base no tempo de viagem da onda de rádio. Como já foi mencionado, quatro satélites são suficientes para atualizar a hora do receptor GPS, bem como proporcionar os meios necessários para o cálculo da longitude, da latitude e da altitude.

De maneira simplificada, o sistema funciona da seguinte forma: em determinado momento, o satélite começa a transmitir um padrão digital longo, chamado código pseudoaleatório. O receptor produz o mesmo padrão digital, exatamente no mesmo horário. O sinal do satélite chega ao receptor com um atraso em relação ao padrão por ele produzido. A extensão do atraso é igual ao tempo de viagem do sinal. O receptor multiplica esse tempo pela velocidade da luz para determinar qual distância o sinal viajou. Supondo que o sinal tenha viajado em linha reta, essa é a distância do receptor até o satélite (Carvalho & Araújo, 2009).

O receptor GPS utiliza o princípio matemático da trilateração para calcular o posicionamento. A trilateração funciona de maneira análoga à triangulação, todavia os parâmetros de cálculo utilizados são as distâncias ao invés de ângulos. Em (Arbelet, 2014), a trilateração é ilustrada de maneira simplificada. O receptor GPS recebe os *timestamps* e calcula o tempo de propagação (T_p), conforme a Equação 2.1:

$$Tp = Tr - Te \qquad \text{Equação 2.1}$$

Onde T_r e T_e representam, respectivamente, o *timestamp* do momento de recebimento dos dados pelo dispositivo GPS e *timestamp* de envio dos dados pelo satélite. A distância até ao satélite (D) é calculada conforme a Equação 2.2:

$$D = Tp * C \qquad \text{Equação 2.2}$$

Onde C denota a velocidade da luz. Para que a informação da distância seja útil, o receptor também tem que saber onde os satélites estão. Cada receptor GPS armazena um almanaque, que lhe diz onde cada satélite deveria estar em qualquer momento determinado (Carvalho & Araújo, 2009). Essa posição do satélite pode ser expressa em termos de coordenadas cartesianas. Dada uma distância D_i para um satélite S_i e um conjunto de coordenadas (X_i, Y_i, Z_i), pode-se obter às coordenadas do receptor (X_r, Y_r, Z_r) pela equação 2.3:

$$\begin{cases} D1 = \sqrt{(X1 - Xr)^2 + (Y1 - Yr)^2 + (Z1 - Zr)^2} \\ D2 = \sqrt{(X2 - Xr)^2 + (Y2 - Yr)^2 + (Z2 - Zr)^2} \\ D3 = \sqrt{(X3 - Xr)^2 + (Y3 - Yr)^2 + (Z3 - Zr)^2} \end{cases} \qquad \text{Equação 2.3}$$

Certamente existem fenômenos que adicionam complexidade ao cálculo de posicionamento. Os relógios dos receptores não são atômicos e precisam fazer ajustes de sua imprecisão. Também existem erros de propagação do sinal provocados por fenômenos de refração e de reflexão. Forças gravitacionais podem mudar a órbita dos satélites, forçando um constante monitoramento por parte dos órgãos responsáveis. Além disso, correções relativísticas precisam ser realizadas dada a velocidade de deslocamento dos satélites e sua altitude.

2.2. A FORMA DA TERRA

Para melhor compreender como os sistemas de navegação por satélite estabelecem o posicionamento de um determinado dispositivo são necessários alguns conceitos provenientes da Geodésia. A Geodésia é a ciência de medida e do mapeamento da superfície da Terra, ou, em outras palavras, o estudo da forma e das dimensões da terra (Zanetti, 2007).

Neste sentido, durante o uso de dispositivos GPS, é comum surgir expressões como *datum*, latitude, longitude, graus decimais, WGS84, dentre outras, que podem ter interpretações equivocadas se tratadas de forma descontextualizada.

Para se calcular uma posição na superfície da terra, em termos de latitude e longitude, alguns conceitos matemáticos e físicos precisam ser compreendidos. A forma da terra pode ter diversas interpretações em Geodésia, podendo variar desde a superfície topográfica, que tem seus limites no Monte Everest (8.850 metros acima do nível do mar) e na Fossa das Marianas (11.000 metros abaixo do nível do mar), até definições que se baseiam no campo gravitacional da terra.
De fato, a forma da terra, ao contrário do que possa parecer, não é definida por sua topografia, mas sim pelo seu campo gravitacional. A verdadeira forma da terra é conhecida como geóide, que é uma superfície equipotencial do campo da gravidade, que melhor se aproxima do nível médio dos mares (*Mean Sea Level* - MSL) em âmbito global (Janssen, 2009). O geóide é, portanto, um modelo físico da forma da terra que, em média, coincide com o valor médio do nível médio das águas do mar. O geóide é bastante irregular devido a variações de densidade no centro da terra, o que torna sua definição matemática complexa e seu uso em sistemas de posicionamento limitado. A Figura 2.4 ilustra as irregularidades do geóide.

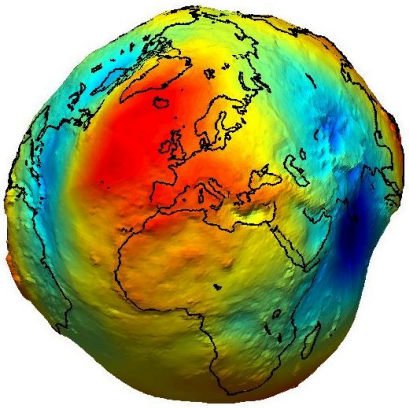


Figura 2.4: Ilustração do geóide (Physorg, 2012)

Neste sentido, o geóide precisa ser aproximado por uma superfície de referência geométrica, que diminua a complexidade matemática dos cálculos de posicionamento, como uma esfera ou um elipsoide de revolução, também conhecido como esferoide.

O planeta terra não é esférico, sendo ligeiramente achatado nos polos devido a sua rotação, o que torna o elipsoide de revolução uma boa superfície geométrica para ser utilizada como um formato aproximado da terra. Tal elipsoide pode ser obtido pela rotação de uma elipse em torno de seu semieixo menor e pode ser definido matematicamente por meio de dois parâmetros: o semieixo maior (*a*) e o semieixo menor (*b*), como ilustrado na Figura 2.5.

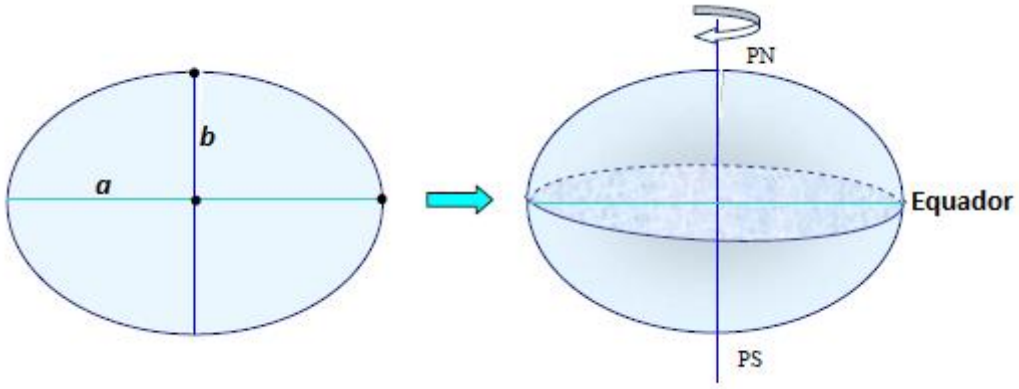


Figura 2.5: Construção do elipsoide de revolução (adaptado de Zanetti, 2007)

Em Geodésia, o elipsoide de revolução é tradicionalmente definido através dos parâmetros semieixo maior e pelo achatamento, sendo que o achatamento (*f*) é dado pela Equação 2.4:

$$f = \frac{a-b}{a} \qquad \text{Equação 2.4}$$

Existem diversos elipsoides de revolução, de várias formas e tamanhos, que são utilizados em diferentes países com o intuído de se aproximar da forma do geóide, seja em uma escala global ou para uma determinada região.

Não existe um único elipsoide que possa melhor se ajustar a todas as partes do planeta, todavia, alguns elipsoides proveem uma aproximação razoável de todo o planeta, como que é o caso do elipsoide WSG84 (*World Geodetic System* 1984), utilizado pelo sistema GPS. A Figura 2.6 ilustra como diferentes elipsoides podem melhor se ajustar a determinados pontos do globo terrestre:

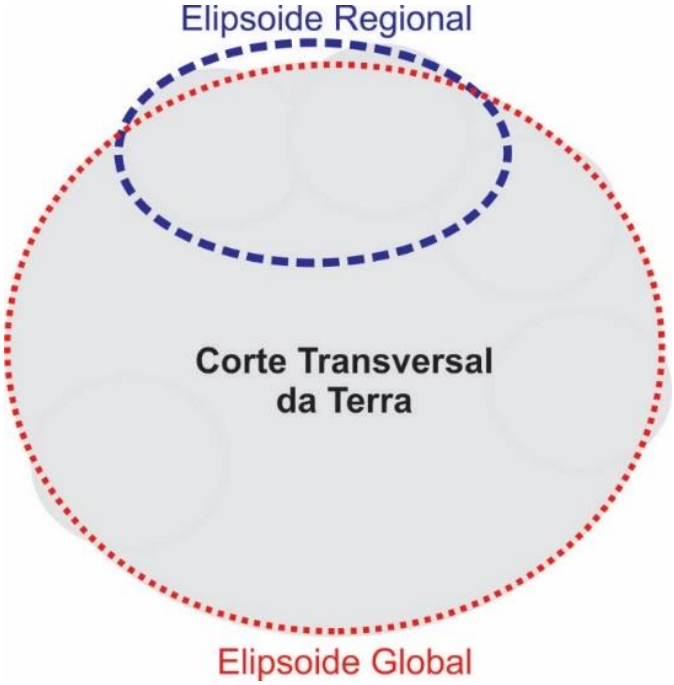


Figura 2.6: Diferenças entre elipsoides globais e regionais

2.3. SISTEMAS DE COORDENADAS

Na utilização de dispositivos GPS é muito comum o uso dos termos latitude e longitude para determinar o posicionamento de um objeto. Entretanto, o sistema de coordenadas é uma metodologia utilizada para referenciar a posição de um ponto, podendo ser definido de distintas formas, a depender da superfície geométrica utilizada. Além disso, o sistema pode ser global, tendo a sua origem geocêntrica, ou apenas local.

O elipsoide é uma figura tridimensional e as localizações podem ser expressas utilizando-se o sistema de coordenadas geodésicas ou elipsoidicas. As posições neste sistema podem ser expressas em coordenadas cartesianas ou curvilíneas. O referido sistema possui as seguintes características (Zanetti, 2007):

- a origem situa-se no centro do elipsoide;
- o eixo $\bar{Z}$ coincide com o eixo de rotação do elipsoide;
- o eixo $\bar{X}$ passa pela interseção do plano médio equatorial com o plano do meridiano primário; e
- o eixo $\bar{Y}$ é escolhido de forma que o sistema seja dextrogiro.

No que diz respeito às coordenadas curvilíneas, a latitude geodésica ϕ de um ponto P é definida como ângulo entre a normal ao elipsoide que passa por P e o plano equatorial elipsoidico. Já a longitude geodésica λ de um ponto P é definida como o ângulo formado entre o eixo X e a projeção sobre o plano equatorial, da normal ao elipsoide nesse ponto. Já a altura *h* é obtida pela distância da superfície do elipsoide ao ponto P, ao longo da normal. A Figura 2.7 ilustra o sistema de coordenadas geodésicas.

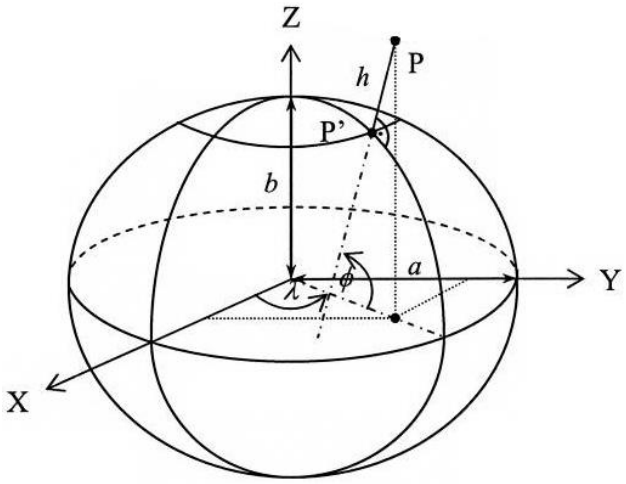


Figura 2.7: Coordenadas Geodésicas (adaptado de Janssen, 2009)

Importante observar que um determinado ponto no planeta pode ter diferentes valores de latitude e longitude, a depender do elipsoide utilizado. Como resultado, diferentes sistemas geodésicos podem apresentar discrepâncias de até centenas de metros.

Atualmente, no Brasil, o SIRGAS2000 (Sistema de Referência Geocêntrico para as Américas) é o único sistema geodésico de referência oficialmente adotado. Todavia, entre 25 de fevereiro de 2005 e 25 de fevereiro de 2015, admitia-se o uso, além do SIRGAS2000, dos referenciais SAD 69 (*South American Datum* 1969) e Córrego Alegre (IBGE, 2016).

Quando comparados os sistemas SAD 69 e SIRGAS2000, as coordenadas que representam a posição de um objeto sofrem alterações da ordem de 65 metros. Já o SIRGAS2000 e WGS84 são praticamente iguais, com os parâmetros do elipsoide muito parecidos (IBGE, 2016). A Figura 2.8 ilustra a diferença entre os sistemas WGS84 e SAD 69, para um ponto de mesma latitude e longitude dentro do campus da Universidade de Brasília.

Figura 2.8: Diferença entre os sistemas WGS84 e SAD 69

2.4. COORDENADAS GEOGRÁFICAS

Como visto anteriormente, a posição de um determinado ponto pode ser expressa em termos de sua latitude, longitude e altitude, calculadas com base em uma superfície geométrica.

Um sistema de coordenadas geográficas faz uso de um conjunto de números, letras e símbolos para expressar uma posição. Linhas no sentido norte-sul, com a mesma longitude, são conhecidas como meridianos, enquanto linhas no sentido leste-oeste, com a mesma latitude, são conhecidas como paralelos. Um meridiano é escolhido como principal, sendo atribuída a longitude 0 para ele. A escala de longitude é então dividida entre os hemi-elipsoides (no caso de sistemas geodésicos) oeste e leste, variando entre 0 e 180 graus nos dois sentidos. A latitude 0 é definida como o círculo referente ao equador do elipsoide, sendo a escala também dividida em dois hemi-elipsoides, norte e sul, que por sua vez variam entre 0 e 90 graus nos dois sentidos. A Figura 2.9 ilustra a divisão de um elipsoide em meridianos e paralelos, tendo como referência o meridiano primário (*Greenwich*) e o equador do elipsoide.

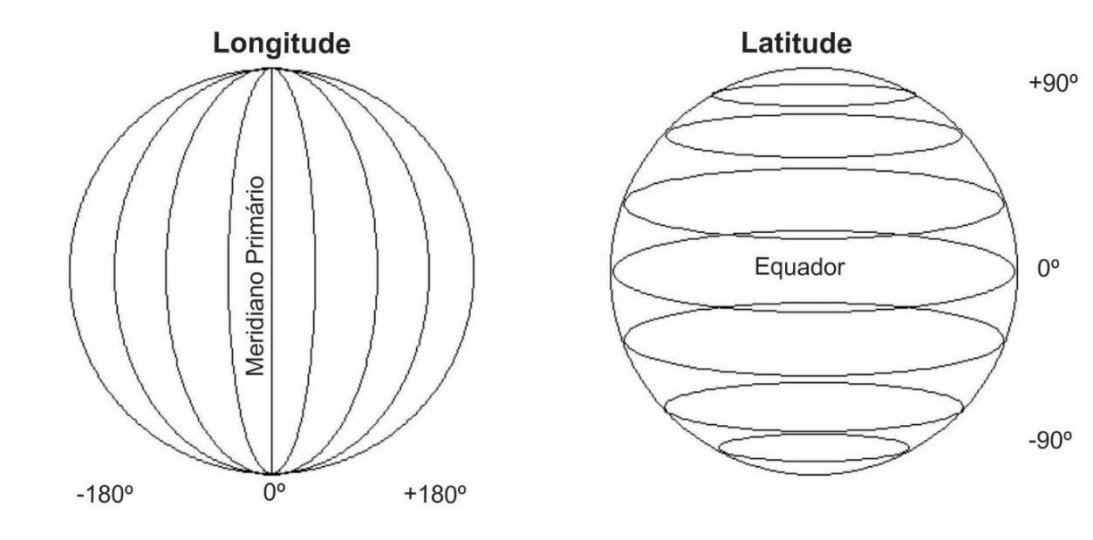


Figura 2.9: Paralelos de latitude e meridianos de longitude

Os valores numéricos da latitude e da longitude podem ser expressos em diversas notações. A unidade primária é o grau (º), que é uma unidade de medida angular. O grau pode ser dividido em 60 minutos (’) e cada minuto, por sua vez, pode ser dividido 60 segundos (”). Para melhor precisão, os segundos podem ser divididos decimalmente em frações cada vez menores. O sistema exposto é o sistema sexagesimal, de base 60, muito utilizado em medida de ângulos e tempo.

Uma das representações mais utilizadas em sistemas GPS é conhecida como Graus, Minutos e Segundos (*Degrees, Minutes, and Seconds - DMS*). Para representar a latitude em termos de graus, minutos e segundos utiliza-se, tipicamente, a notação descrita na Figura 2.10:

dd°mm’ss.ssss"[N,S], dd°mm’ss.ssss"[W,E]

Figura 2.10: Notação Graus, Minutos e Segundos

Onde:

- *dd* representa os graus;
- *mm* representa os minutos;
- *ss.ssss* representa os segundos e as frações de segundos;

- *[N,S]* e *[W,E]* representam a orientação em relação ao equador e ao meridiano principal, com base nos pontos cardeais norte (*north*), sul (*south*), leste (*east*) e oeste (*weast*).

Outro tipo de notação muito utilizada é a Graus Decimais (*Decimal Degrees* - DD). Nessa representação a latitude e a longitude são representadas como frações decimais. Valores ao norte do equador e ao leste do meridiano primário recebem valores positivos, enquanto valores ao sul do equador e ao oeste do meridiano primário recebem valores negativos. Quando expresso em graus decimais, um par de coordenadas é tipicamente representado com a notação descrita na Figura 2.11:

[-]dd.dddd [-]dd.dddd

Figura 2.11: Notação Graus Decimais

Onde:

- [-] representa a orientação em relação ao equador e ao meridiano principal;
- *dd* representa os graus;
- *dddd* representa frações decimais

Por fim, outra representação muito utilizada principalmente no protocolo NMEA 0183, o qual será explicado posteriormente, é a notação Minutos Decimais (*Decimal Minutes* - DM). Nesta notação, cada grau é dividido em 60 minutos, que por sua vez são divididos decimalmente. A Figura 2.12 ilustra a referida notação:

ddmm.mmmm[N,S]ddmm.mmmm[E,W]

Figura 2.12: Notação Minutos Decimais

Onde:

- *dd* representa os graus
- *mm.mmmm* representa os minutos e as frações de minutos

- *[N,S]* e *[W,E]* representa a orientação em relação ao equador e ao meridiano principal, com base nos pontos cardeais.

Não obstante as notações apresentadas serem muito comuns em dispositivos GPS, existem diversas outras notações utilizadas para representar a mesma informação, com pequenas variações na disposição dos símbolos. A Tabela A.1 do Apêndice mostra diversas notações para o par de coordenadas "-15.763605 , -47.86976" (Campus da Universidade de Brasília), inicialmente expresso em Graus Decimais.

Os formatos também podem ser facilmente convertidos. A Equação 2.5 é utilizada para converter um par de coordenadas no formato DMS para o formato DD:

$$Graus_{Decimais} = Graus + \frac{Minutos}{60} + \frac{Segundos}{3600} \qquad \text{Equação 2.5}$$

Já o processo inverso pode ser obtido pelas Equação 2.6:

$$\begin{cases} Graus = \lfloor Graus\_Decimais \rfloor \\ Minutos = \lfloor 60 * (Graus_{Decimais} - Graus) \rfloor \\ Segundos = 3600 * [(Graus_{Decimais} - Graus) - Minutos/60] \end{cases} \qquad \text{Equação 2.6}$$

2.5. DATUM

Os conceitos previamente apresentados são relativos a modelos matemáticos teóricos da representação da superfície da Terra e dos sistemas de coordenadas. A materialização destes conceitos é denominada *datum*. Um *datum* efetivamente define a origem e a orientação de um sistema de coordenadas e um determinado período do tempo.

Em outras palavras, em um *datum* são definidos valores como o raio equatorial, o achatamento do elipsoide, os componentes de um vetor de translação entre o centro da Terra real e do elipsoide, dentre outros elementos (d'Alge, 1995).

Neste trabalho, que tem como foco os sistemas GPS, o *datum* de referência é o WGS84. Todas as coordenadas geodésicas aqui apresentadas possuem como referência este *datum*, pois, como

visto anteriormente, diferentes sistemas podem apresentar discrepâncias para um ponto de mesma latitude e longitude.

2.5.1. WGS84

O WGS84 foi desenvolvido pelos Estados Unidos, mais especificamente pela DMA (*Defense Mapping Agency*), que atualmente se chama NGA (*National Geospatial-Intelligence Agency*). É o *datum* utilizado pelos sistemas GPS e utiliza como referência o elipsoide WGS84. Ele foi primeiramente utilizado em 1987 e vem sendo refinado de tempos em tempos para corrigir distorções causadas pelos movimentos tectônicos (Janssen, 2009).
Além disso, o referido *datum* é global, geocêntrico, fixo a Terra (gira juntamente com o planeta em seu movimento de rotação). O sistema possui uma série de parâmetros que definem o formato do elipsoide, o modelo de gravidade, a orientação utilizada, dentre outras características. Tais parâmetros podem ser visualizados na Tabela A.2 do Apêndice.

2.6. NMEA

O receptor GPS, além de realizar o processamento dos dados recebidos dos satélites, deve ser capaz de reportar esses dados para as diversas camadas do dispositivo, seja esse dispositivo um *smartphone*, um *tablet*, uma central multimídia, ou qualquer outro aparelho.

As especificações da *National Marine Eletronics Association*, em especial a NMEA 0183, têm importância fundamental na comunicação dos receptores GPS com os diversos tipos de aparelhos. NMEA 0183 é uma especificação que engloba a parte elétrica e o protocolo de dados para comunicação entre componentes eletrônicos marinhos, tais como sonares, giroscópios, pilotos automáticos, receptores GPS e muitos outros tipos de instrumentos (*National Marine Electronics Association*, 2002).

O protocolo NMEA é o mecanismo de comunicação padrão entre os receptores GPS e os *drivers* dos dispositivos receptores na arquitetura Android. Desta forma, os programas que fornecem soluções de posicionamento precisam decodificar as mensagens NMEA. A ideia do protocolo, no que diz respeito a coordenadas de posicionamento, consiste em enviar uma linha de dados, codificada em ASCII, que pode conter informações de posição, velocidade, tempo, dentre outros elementos processados pelo receptor GPS.

Cada mensagem é iniciada com o caractere “$” e terminada com os caracteres retorno de carro e a alimentação de linha (CR/LF). Além disso, as mensagens têm um prefixo de duas letras que

define o tipo de dispositivo (chamado de *TALKER Identifier*), seguido de três letras que definem o conteúdo da mensagem. Para receptores de GPS o prefixo utilizado é o "GP", para o sistema GLONASS usa-se "GL" e para outros sistemas globais de navegação por satélite utiliza-se "GN" (*National Marine Electronics Association*, 2002). Fabricantes específicos também podem definir mensagens NMEA. Todas as mensagens proprietárias se iniciam com a letra "P" seguida de três letras que identificam o fabricante. Uma mensagem de um chip Garmin, por exemplo, possui as letras "PGRM" (*NMEA Data*, 2016).

O formato da mensagem é simples, de forma que os dados contidos em uma linha são separados unicamente por vírgula (*comma-separated format*). Também há um campo, no fim de cada mensagem, para verificação de integridade dos dados (*checksum*) que é separado do restante da mensagem pelo caractere "*". Na Figura 2.13 são mostrados alguns exemplos de mensagens NMEA:

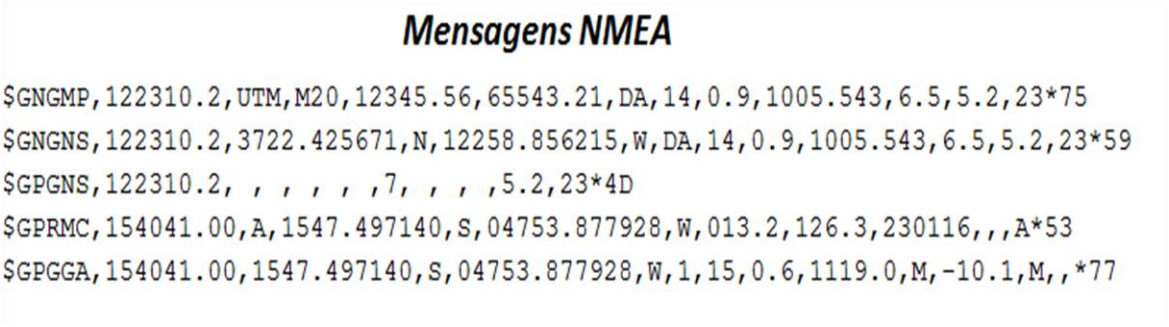


Figura 2.13: Exemplos de mensagens NMEA

Existem dezenas de mensagens NMEA, entretanto nem todas possuem aplicabilidade para os sistemas de GPS, sendo aplicáveis para outros tipos de aparelhos, como sonares e giroscópios. Os tipos mostrados na Tabela 2.1 estão diretamente relacionados com os sistemas GNSS.

Algumas mensagens, como a GSA, GST e GSV não possuem dados de posicionamento, em especial coordenadas geodésicas. Estas mensagens provêm informações sobre o número de satélites visíveis, dados estatísticos sobre a qualidade do sinal recebido, dentre outras informações que auxiliam a manutenção de uma boa precisão.

Outras mensagens, por sua vez, como a RMB e a GLL possuem coordenadas geodésicas, porém têm aplicação limitada, geralmente voltadas para ambientes marinhos. Nos experimentos realizados neste trabalho, não foram encontradas mensagens do tipo RMB e GLL.

No escopo desta pesquisa, cujo foco é a recuperação de coordenadas de posicionamento, de dados temporais e de velocidades, dois tipos de mensagens têm maior relevância: GPGGA e GPRMC.

Tabela 2.1 – Mensagens NMEA importantes para sistemas GNSS

Tipo	Descrição
GGA	*Global Positioning System Fix Data*
RMC	*Recommended Minimum Specific GNSS Data*
GSV	*GNSS Satellites in View*
RMB	*Recommended Minimum Navigation Information*
GLL	*Geographic Position - Latitude/Longitude*
GSA	*GNSS DOP and Active Satellites*
GST	*GNSS Pseudorange Error Statistics*

2.6.1. GPGGA

Mensagens deste tipo fornecem dados que permitem localizar o dispositivo por meio de coordenadas geodésicas. Mais especificamente, são providos dados de tempo, latitude, longitude, altitude e o número de satélites em uso. A Figura 2.14 mostra os campos de uma mensagem do tipo GGA.

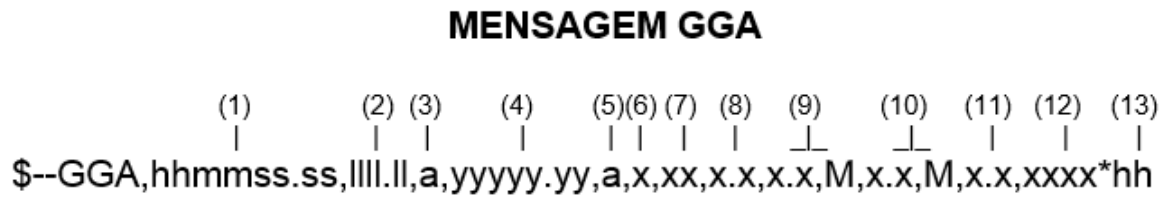


Figura 2.14: Campos da mensagem GGA

A seguir são descritos os campos:

1. Tempo no formato UTC. UTC (*Universal Time Coordinated*), é o fuso horário de referência a partir do qual se calculam todas as outras zonas horárias do mundo. Cabe ressaltar que este campo só fornece os dados de horas, minutos e segundos. A data não é fornecida.
2. Latitude.
3. Orientação da latitude (N – *North*, S - *South*).

4. Longitude.
5. Orientação da longitude (W – W*est*, E - *East*).
6. Indicador de Qualidade do GPS.
7. Número de satélites em uso (00-12), podendo ser diferente do número de satélites visíveis.
8. Diluição da precisão horizontal.
9. Altitude em metros, a partir do nível do mar.
10. Diferença entre a superfície do elipsoide e o nível médio dos mares do geoide.
11. Tempo em segundos desde a última atualização do DGPS (*Differential Global Positioning System*). Campo geralmente nulo.
12. ID da estação DGPS. Campo geralmente nulo.
13. Checksum.

Cabe ressaltar que a latitude e a longitude são descritas no formato *degreesminutes.decimal*, com dois dígitos fixos para o grau no caso de latitude (ou três para longitude), dois dígitos fixos para o minuto e um tamanho variável para a fração de decimal de minutos.

2.6.2. GPRMC

Mensagens deste tipo agregam dados de hora, data, posição, rumo e velocidade fornecidos por um receptor GPS. As mensagens são transmitidas em intervalos não superiores a 2 segundos. A Figura 2.15 mostra os campos de uma mensagem do tipo RMC.

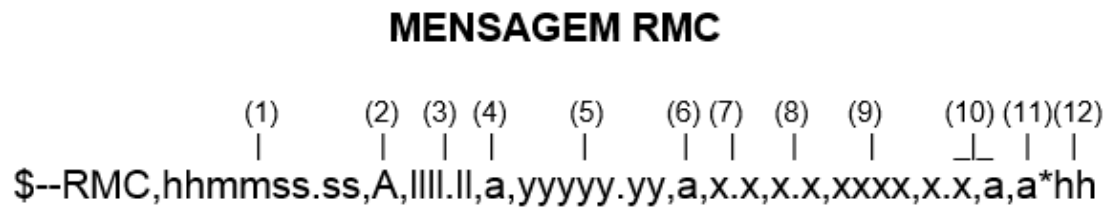


Figura 2.15: Campos da mensagem RMC

A seguir são descritos os campos:

1. Tempo no formato UTC.
2. Status. “A” indica dados validados, com coordenadas geodésicas, e “N” denota dados inválidos.

3. Latitude.
4. Orientação da latitude (N – *North*, S - *South*).
5. Longitude.
6. Orientação da longitude (W – W*est*, E - *East*).
7. Velocidade (em nós).
8. Curso.
9. Data, no formato *ddmmyy*.
10. Variação magnética.
11. Indicador de modo.
12. Checksum.

Assim como nas mensagens GPGGA, as coordenadas geodésicas são descritas no formato *degreesminutes.decimal*.

A velocidade, nas mensagens GPRMC, é fornecida em nós, onde um nó equivale a uma milha náutica por hora, ou seja, 1,852 km/h.

2.7. FORENSE DE DISPOSITIVOS GPS

No que diz respeito a recuperação de informações de posicionamento em sistemas de navegação por satélite, há estudos principalmente associados a dispositivos das marcas Garmim (Garmin, 2016) e TomTom (TomTom, 2016), conforme descrito a seguir.

Em (Nutter, 2008) foi realizado um estudo com dispositivos TomTom visando decifrar o formato dos registros com informações de posicionamento em arquivos armazenados em memória não volátil. Em (Van & Roeloffs, 2010) foi utilizado JTAG e uma distribuição Linux para extrair dados da memória RAM de aparelhos TomTom.

Em (Arbelet, 2014) foram explorados métodos forenses para extração de informações de GPS em diversos modelos de dispositivos Garmin. Em (Lim, Lee, Park & Lee, 2014) foram realizadas analises do tipo *test-driven* para decodificar formatos de arquivos de dispositivos da marca Mappy.

Também foram desenvolvidos estudos relativos a recuperação de mensagens do protocolo NMEA (Si & Aung, 2011), bem como estudos relativos à precisão da velocidade de *smartphones* obtida pelos mecanismos de GPS (Guido et al, 2014). Também foram realizados experimentos com *smartphones* Android, mostrando que dados de localização recuperados dos

aparelhos podem ser muito mais precisos do que os dados coletados pelas operadoras de telefonia (Spreitzenbarth, Schmitt & Freiling, 2012).

No que tange a análise de dados de GPS em dispositivos em Andoid, foi desenvolvido um trabalho onde coordenadas de GPS eram recuperadas de arquivos de Log e bancos de dados de aplicações, todavia não foi explorada a análise de memória volátil (Maus, Höfken & Schuba, 2011).

Por fim, em 2013, na *Black Hat Europe*, foi mostrado o funcionamento dos protocolos dos aplicativos *Waze* (*Waze*, 2016) e *Google Maps* (*Google Maps*, 2016), sendo feita uma análise das fragilidades encontradas nos mecanismos de autenticação e privacidade que permitiam ataques aos sistemas de navegação (Jeske, 2013).

Grande parte dos estudos foram realizados com base em artefatos recuperados de memória não volátil. Todavia, os conhecimentos adquiridos nestes estudos podem também ser expandidos e utilizados na recuperação de dados de memória volátil. Nas próximas seções será mostrado como coordenadas geodésicas estão comumente armazenadas em bancos de dados, arquivos de log, arquivos xml, arquivos de mídia, dentre outros artefatos.

2.7.1. Dados de GPS armazenados na memória não volátil de sistemas Android

A maioria dos dispositivos Android atuais já vêm de fábrica com uma série de aplicativos de mapa e navegação. Além disso, uma boa parcela dos aplicativos disponíveis na *Play Store* (Playstore, 2016) fazem requisição para uso dos serviços de localização.

Em sistemas Android, a biblioteca de banco de dados *SQLite* é amplamente utilizada para armazenamento estruturado dos dados, sendo uma fonte valiosa de dados de interesse pericial. O banco de dados *SQLite* possui uma estrutura simples, de forma que as tabelas, índices, *views* e *triggers* estão todas armazenadas em um único arquivo. Ao contrário da maioria dos bancos de dados SQL, o *SQLite* não possui um processo separado para o servidor. Os dados são lidos e escritos diretamente em um arquivo armazenado no disco, que é o banco de dados propriamente dito. Além disso o banco é independente de plataforma, podendo ser utilizado em máquinas de 32 ou 64 bits, bem como em sistemas *little-endian* ou *big-endian* (*About SQLite*, 2016).

Coordenadas de posicionamento são frequentemente encontradas em bancos deste tipo. Todavia, os dados são armazenados em diferentes formatos e codificações, o que torna complexa a criação de um método de extração de dados de posicionamento que independa de

aplicativo. Como exemplo, os aplicativos *WhatsApp* (*WhatsApp*, 2016) e *Facebook Messenger* (*Facebook Messenger*, 2016) armazenam as conversas dos usuários em bancos de dados *SQLite*, que ficam armazenados nos diretórios `/data/data/com.whatsapp/databases/msgstore.db` e `/data/data/com.facebook_1.orca/databases/threads_db2`, respectivamente. Ambos os bancos contêm uma tabela denominada *messages*, que armazena informações como data, destinatário, teor da conversa, coordenadas de posicionamento, dentre outras informações. As Figuras 2.16 e 2.17 mostram alguns registros que possuem informações de latitude e longitude, tanto para o aplicativo *WhatsApp*, quanto para o *Facebook Messenger*.

latitude	longitude	thumb_image	remote_resource	received_timestamp
Filter	Filter	Filter	Filter	Filter
-20.652347	-40.4840378	BLOB	5527 @s.whatsapp.net	1438303193995
-20.6522621956977	-40.4843263847961	BLOB	5527 @s.whatsapp.net	1438303210922
-20.6513809	-40.4823186	BLOB	5527 @s.whatsapp.net	1438537684340
-16.395998859911	-39.041947272681	BLOB	55(@s.whatsapp.net	1442167340013
-15.9935759	-47.9990015	BLOB	55(@s.whatsapp.net	1441420888893
-15.9209709	-48.052882	BLOB	55(@s.whatsapp.net	1441374699180

Figura 2.16: Registros presentes no banco de dados do aplicativo *WhatsApp*

coordinates	offline_threading_id	source	channel_source
Filter	Filter	Filter	Filter
{"latitude":-3.1771783,"longitude":-41.8680369,"accuracy":42.0}	5956393060	messenger	API
{"latitude":-3.1771301,"longitude":-41.8681076,"accuracy":35.08700180053711}	5956394191	messenger	API
{"latitude":-3.1771301,"longitude":-41.8681076,"accuracy":35.08700180053711}	5956393950	messenger	API
{"latitude":-3.177067,"longitude":-41.8681109,"accuracy":29.236000061035156}	5954982275	messenger	API
{"latitude":-3.1770666,"longitude":-41.868058,"accuracy":30.483999252319336}	5955009509	messenger	API
{"latitude":-3.177021,"longitude":-41.8680947,"accuracy":45.0}	5956394852	messenger	API
{"latitude":-3.1769769,"longitude":-41.8680216,"accuracy":33.0}	5955010398	messenger	API

Figura 2.17: Registros presentes no banco de dados do aplicativo *Facebook Messenger*

Como é possível observar, os dados estão armazenados em formatos distintos. O *SQLite* possui quatro tipos de dados básicos: INTEGER, REAL, TEXT, BLOB. No caso do aplicativo *WhatsApp*, os dados de latitude e longitude estão armazenados em duas colunas que utilizam o

tipo REAL (representado de acordo com o padrão da IEEE para pontos flutuantes: sinal-expoente-mantissa) (*Datatypes In SQLite Version 3*, 2016).

Já no caso do aplicativo *Facebook Messenger*, as coordenadas geodésicas estão armazenadas em uma única coluna que utiliza o tipo TEXT. Neste formato, os dados são armazenados como uma sequência de caracteres codificada em UTF-8, UTF-16BE ou UTF-16LE. As Figuras 2.18 e 2.19 mostram as diferenças no formato utilizado pelos dois aplicativos.

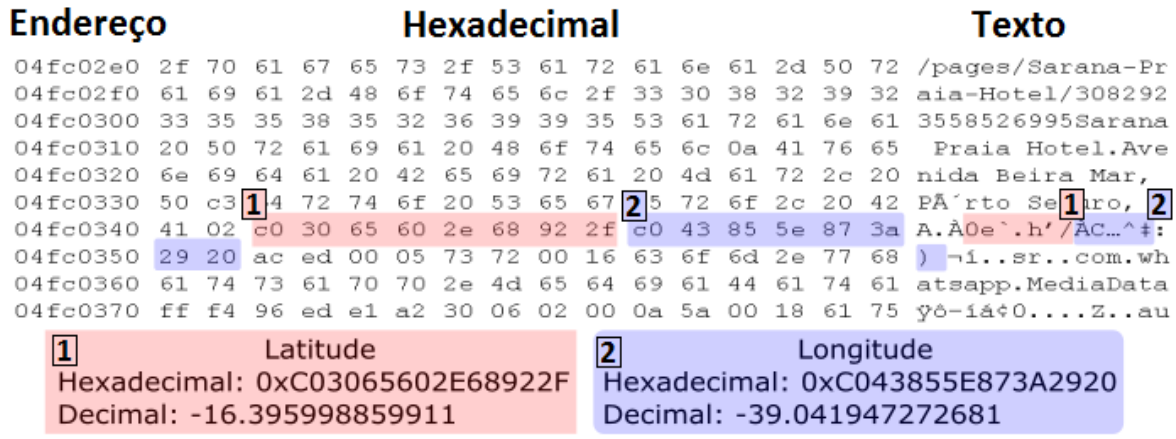


Figura 2.18: Latitude e longitude armazenadas como o tipo REAL

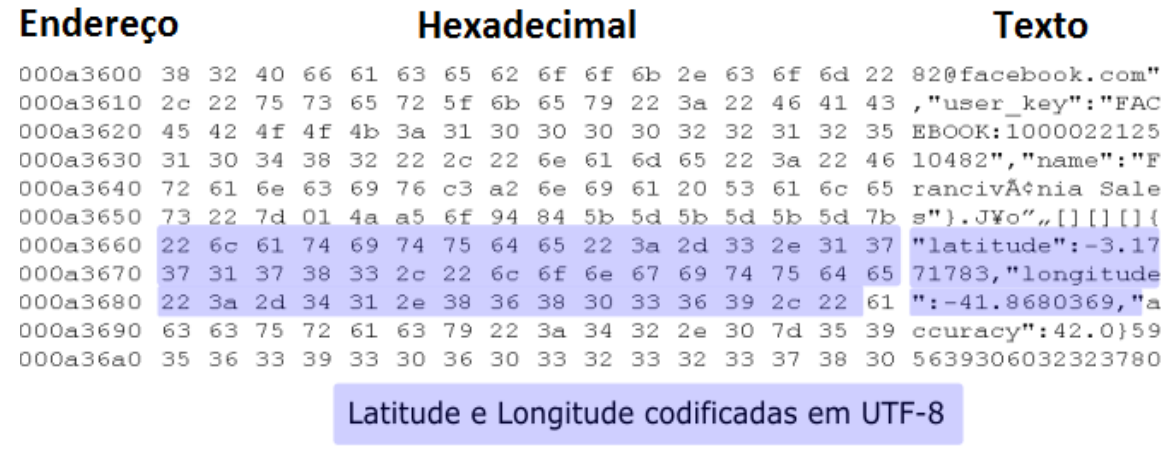


Figura 2.19: Latitude e longitude representadas em formato textual (UTF-8)

Como exposto nas figuras, os dados são representados em codificações distintas. Além disso, os dados devem ser analisados levando-se em consideração o contexto do aplicativo, uma vez que as informações de posicionamento podem não estar associadas diretamente ao dispositivo alvo.

Em (Bell,2015) foram realizados estudos de casos visando localizar e interpretar dados de posicionamento em diversos aplicativos. Na Tabela A.3 do Apêndice podem ser visualizados diversos bancos de dados *SQLite* que contêm informações de localização.

Além dos bancos *SQLite*, os aplicativos também armazenam informações de posicionamento em outros tipos de arquivos, especialmente arquivos de log, de *journaling* e do tipo XML. O aplicativo *Google Maps*, por exemplo, possui um arquivo denominado `camera.xml`, que fica armazenado no diretório `/data/data/com.google.android.apps_1.maps/shared_prefs`. De acordo com testes laboratoriais, o referido arquivo guarda dados de posicionamento da última tela visitada pelo aplicativo. Toda vez que o aplicativo é fechado e colocado em segundo plano, o arquivo "camera.xml" é sobrescrito com informações da última tela acessada. Desta forma, o aplicativo pode reiniciar a partir da última tela visitada. As informações contidas neste arquivo não permitem afirmar que o dispositivo móvel esteve fisicamente na região das coordenadas encontradas, mas sim que o aplicativo *Google Maps* teve visualizações naquela região. A estrutura deste arquivo pode ser visualizada na Figura 2.20.

```
<?mml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <float name="bearing" value="0.0" />
    <float name="lng" value="-47.97869" />
    <float name="tilt" value="0.0" />
    <float name="lat" value="-15.908299" />
    <float name="zoom" value="9.267833" />
    <long name="cameraTimestampMs" value="320277" />
</map>
```

Figura 2.20: Estrutura do arquivo camera.xml

Ainda em relação ao aplicativo *Google Maps*, toda vez que o aplicativo é utilizado, são criados diversos arquivos no diretório `data/data/com.google.android.apps_1.maps/files` seguindo o padrão de nomenclatura *DATA_LAYER_NUMBER* (ex. *DATA_st_epoch_resources_323*), que também podem armazenar dados de posicionamento (Doonan, 2013). Cabe ressaltar que a maioria dos arquivos são binários, precisando de ferramentas adequadas para visualização dos dados. A Figura 2.21 mostra coordenadas de posicionamento armazenadas em um arquivo deste tipo, ao final de uma URL.

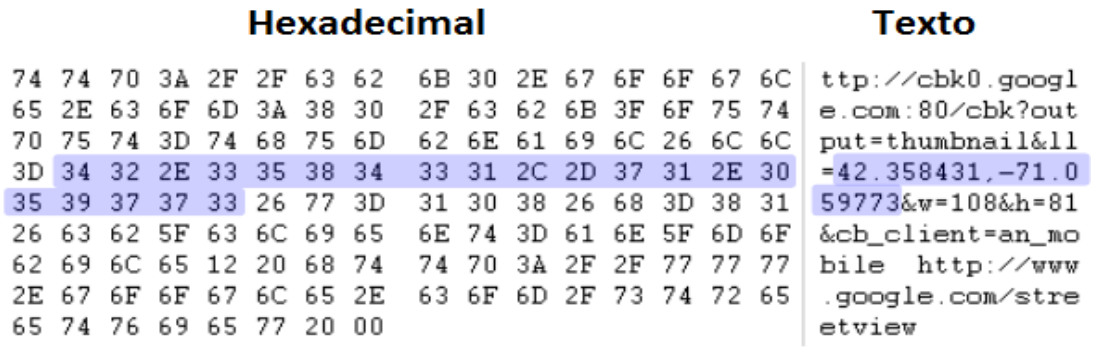


Figura 2.21: Latitude e longitude ao final de uma URL (adaptado de Doonan, 2013)

Em suma, os dados de posicionamento podem estar armazenados, na memória não volátil, em diferentes tipos de arquivos, formatos e codificações, o que implica na necessidade de uma análise detalhada dos aplicativos em cada caso concreto.

3. SISTEMAS ANDROID

A análise forense de dispositivos móveis possui muitas peculiaridades quando comparadas às abordagens pericias em computadores, uma vez que as ferramentas utilizadas no processo de extração e análise de dados dependem muito da arquitetura do dispositivo móvel. No escopo deste trabalho, cujo foco é o sistema Android, deve-se levar em consideração as melhores práticas para aquisição dos dados. Neste sentido, as próximas seções abordam três aspectos fundamentais para perícias em dispositivos móveis Android: ferramentas periciais, mecanismos de *root* e técnicas de extração de memória volátil.

3.1. FERRAMENTAS PERICIAIS

Em ambientes Android, existe um conjunto de ferramentas que propiciam aos desenvolvedores um ambiente completo para criação e depuração dos aplicativos, denominado *Software Development Kit* (SDK) (Simão, 2012). Todavia, parte das ferramentas utilizadas no processo de desenvolvimento são também usadas no contexto pericial, em especial a ferramenta ADB (*Android Debug Bridge*). ADB é um programa do tipo cliente-servidor que provê comunicação com um dispositivo Android por meio de uma rede virtual em cima da conexão USB (Hoog, 2011). A ferramenta é encontrada no diretório do SDK, dentro da pasta *platform-tools*.

Por meio do ADB é possível realizar diversas operações no dispositivo alvo, como realizar backup dos dados da memória interna, obter um *shell*, obter informações de log, instalar aplicativos, adicionar e remover arquivos, dentre outras operações. A Tabela 3.1 mostra alguns dos comandos que podem ser utilizados.

Tabela 3.1 – Comandos da ferramenta ADB

Comando	Descrição
adb push	copia arquivo ou diretório para o dispositivo.
adb pull	recupera arquivo ou diretório do dispositivo.
adb shell	executa *shell* remoto
adb logcat	visualiza log do dispositivo
adb install	copia o arquivo .apk para o dispositivo e instala o aplicativo
adb backup	realizada *backup* dos dados da memória interna

Neste trabalho, a ferramenta ADB foi utilizada basicamente para se obter um *shell*. A partir do *shell*, diversas tarefas podem ser executadas no sistema, principalmente se o sistema já estiver com usuário *root* habilitado. Existem diversas técnicas para depuração, mas no espoco deste trabalho, onde o foco é a analise de memória RAM, foram utilizados, com mais frequência, dois programas: *logcat* e *dumpsys*.

3.1.1. *Logcat*

A ferramenta *logcat* direciona para a saída do usuário uma lista continua e atualizada de todas a mensagens de debug emitidas pelos aplicativos, bem como pela interface de rádio (Hoog, 2011). Cada mensagens de log é iniciada com uma letra que indica o tipo de mensagem, conforme descrito na Tabela 3.2.

Tabela 3.2 – Tipos de mensagem do *logcat* (Hoog, 2011)

Comando	Descrição
V	*Verbose* (menor prioridade)
D	*Debug*
I	*Info*
W	*Warning*
E	*Error*
F	*Fatal*
S	*Silent* (maior prioridade)

As mensagens de depuração contêm dados relevantes tais como *timestamps* de eventos dos aplicativos, informações de mensagens SMS, informações de células de telefonia, coordenadas de posicionamento, redes wireless, dentre outros dados importantes no contexto pericial. Na Figura 3.1 podem ser observadas mensagens do tipo *Debug* com dados de liberação de memória RAM.

```
D/dalvikvm(509): GC_CONCURRENT freed <1K, 9% free 21739K/23751K, paused 7ms+24ms
D/dalvikvm(509): GC_CONCURRENT freed 4614K, 20% free 19206K/23943K, paused 6ms+18ms
D/dalvikvm(509): GC_CONCURRENT freed <1K, 12% free 21254K/23943K, paused 7ms+26ms
D/dalvikvm(509): GC_CONCURRENT freed 4614K, 22% free 18687K/23943K, paused 6ms+17ms
D/dalvikvm(509): GC_CONCURRENT freed <1K, 14% free 20735K/23943K, paused 7ms+22ms
```

Figura 3.1: Mensagens de debug do *logcat*

3.1.2. *Dumpsys*

O *dumpsys* (Hoog, 2011) é um programa que provê informações sobre o estado do sistema operacional, com informações detalhadas sobre os serviços em execução e, em especial, sobre a memória volátil.

Neste trabalho a opção *meminfo* do programa *dumpsys* foi muito utilizada, visto que esta opção provê informações detalhadas da alocação de memória RAM. Utiliza-se a seguinte sintaxe: `dumpsys meminfo <package_name|pid> [-d]`. A Figura 3.2 mostra a saída do comando, tendo como referência o aplicativo *Google Maps*.

```
Applications Memory Usage (kB):
Uptime: 3207370 Realtime: 3207370

** MEMINFO in pid 2641 [com.google.android.apps.maps] **
                   Shared  Private     Heap     Heap     Heap
                     Pss    Dirty    Dirty     Size    Alloc     Free
                  ------  ------   ------   ------   ------   ------
       Native      2049     860     1996    16064    10388      163
       Dalvik     17466    7304    17012    26435    17957     8478
       Cursor         0       0        0
       Ashmem         2       8        0
    Other dev         4      36        0
     .so mmap      2588    4492     1348
    .jar mmap         0       0        0
    .apk mmap       537       0        0
    .ttf mmap        17       0        0
    .dex mmap      7251       0       44
   Other mmap        37      12       24
      Unknown      6639     392     6616
        TOTAL     36590   13104    27040    42499    28345     8641

 Objects
               Views:      105         ViewRootImpl:        1
         AppContexts:        3           Activities:        1
              Assets:        2        AssetManagers:        2
       Local Binders:       33        Proxy Binders:       35
    Death Recipients:        4
     OpenSSL Sockets:        2

 SQL
                heap:       72          MEMORY_USED:       72
  PAGECACHE_OVERFLOW:       14          MALLOC_SIZE:       46

 DATABASES
      pgsz     dbsz   Lookaside(b)          cache  Dbname
         1       11             39         6/16/2  gmm_myplaces.db
```

Figura 3.2: Mensagens de debug do *logcat*

Como é possível observar, há uma grande quantidade de informações sobre a memória utilizada pelo aplicativo, tais como o tamanho da *heap*, a quantidade

de memória livre, a quantidade de memória mapeada para bibliotecas nativas, dentre outros dados.

3.2. MECANISMOS DE *ROOT*

Sistemas Linux, em especial sistemas Android, possuem diversas falhas de segurança que permitem acesso como usuário *root*. Em (Thomas, Beresford & Rice, 2015) foi realizado um estudo com 20.400 dispositivos, onde foi mostrado que, em média, 87,7% dos aparelhos estavam expostos a pelo menos uma de 11 vulnerabilidades críticas bem conhecidas, conforme Tabela 3.3.

Tabela 3.3 – Vulnerabilidades em sistemas Android (Thomas, Beresford & Rice, 2015)

Vulnerabilidade	Data	Categoria
KillingInTheNameOf	2010-07-13	sistema *e kernel*
exploid udev	2010-07-15	*kernel*
levitator	2011-03-10	*kernel*
Gingerbreak	2011-04-18	sistema
zergRush	2011-10-06	sistema
APK duplicate file	2013-02-18	assinatura
APK unchecked name	2013-06-30	assinatura
APK unsigned shorts	2013-07-03	assinatura
vold asec	2014-01-27	sistema
Fake ID	2014-04-17	assinatura
TowelRoot	2014-05-03	*kernel*

Atualmente, o processo de *rooting* para grande parte dos aparelhos pode ser encontrado em buscas na internet, em especial na comunidade XDA *Developers*, que é um site voltado para desenvolvedores e entusiastas na área de dispositivos móveis (XDA *Developers*, 2016). Também existem soluções do tipo *One Click Root*, como o software *Kingo Root* (*Kingo Root*, 2016) e *iRoot* (*iRoot*, 2016), que facilitam o trabalho, porém funcionam como uma caixa preta, onde não é possível, em uma análise superficial, identificar o *exploit* utilizado.

Existem diversas formar de se obter acesso como usuário *root* em ambientes Android. Algumas técnicas envolvem substituição de partições de *recovery*, implicando na reinicialização do aparelho. Outras técnicas, também conhecidas como *soft root*, consistem na obtenção de um *shell* de usuário *root* mediante exploração de falhas de segurança. O *soft root* geralmente é realizado explorando alguma vulnerabilidade no *kernel* do Android, em um processo já em execução como *root*, em algum aplicativo com o bit *set-uid* habilitado, bem como em outros

mecanismos que permitam escalada de privilégios (Drake *et al*, 2014). No contexto deste trabalho foram utilizadas técnicas de *soft root*.

A seguir serão apresentados alguns métodos para escalada de privilégios, abrangendo desde ataques mais antigos, que funcionavam em versões inferiores à 2.2 do sistema Android, até técnicas mais recentes. No site *http://androidvulnerabilities.org* há uma compilação das diversas técnicas mais conhecidas. Apesar de algumas vulnerabilidades atingirem uma grande quantidade de sistemas Android, muitas delas são específicas para determinadas versões do referido sistema ou até mesmo para dispositivos específicos. Maiores informações sobre os métodos podem ser encontradas de acordo sua descrição na lista de vulnerabilidades conhecidas (CVE, 2016).

3.2.1. *RageAgainstTheCage adb* - CVE-2010-EASY

Uma falha do ano de 2010 que afetava versões Android anteriores à 2.2 explorava o fato de o *deamon* ADB não averiguar o valor de retorno da chamada *setuid*. Sebastian Krahmer utilizou essa falha de validação para criar o *exploit RageAgainstTheCage*. A Figura 3.3 mostra o *exploit* em execução.

```
* daemon started successfully *
$ id
id
uid=2000(shell) gid=2000(shell) groups=1003(graphics),1004(in
(mount),1011(adb),1015(sdcard_rw),3001(net_bt_admin),3002(net
$ /data/local/tmp/rageagainstthecage-arm5.bin
/data/local/tmp/rageagainstthecage-arm5.bin
[*] CVE-2010-EASY Android local root exploit (C) 2010 by 743C

[*] checking NPROC limit ...
[+] RLIMIT_NPROC={3301, 3301}
[*] Searching for adb ...
[+] Found adb as PID 77
[*] Spawning children. Dont type anything and wait for reset!
[*]
[*] If you like what we are doing you can send us PayPal mone
[*] 7-4-3-C@web.de so we can compensate time, effort and HW c
[*] If you are a company and feel like you profit from our wo
[*] we also accept donations > 1000 USD!
[*]
[*] adb connection will be reset. restart adb server on deskt
$
C:\Users\Consultant>adb kill-server

C:\Users\Consultant>adb shell
* daemon not running. starting it now *
* daemon started successfully *
# id
id
uid=0(root) gid=2000(shell) groups=1003(graphics),1004(input)
```

Figura 3.3: *Exploit RageAgainstTheCage* (Benn, 2010)

O *exploit* precisa ser executado em um *shell* ADB. Basicamente, o procedimento consiste em realizar múltiplos *forks* de um processo até exceder o limite de processos por usuário. Esse limite, conhecido como RLIMIT_NPROC, especifica o número de processos que podem ser executados para um determinado UID. Quando o limite é excedido, o *exploit* mata o processo adbd, forçando-o a reiniciar. O *deamon* ADB (adbd) é iniciado com privilégios de *root*, mas logo em seguida diminui seus privilégios para o usuário *shell* (AID_SHELL). Entretanto, o adbd não consegue realizar mudança para o usuário *shell*, uma vez que o limite de processos para aquele usuário foi atingido. A chamada *setuid* falha, mas o adbd não detecta esta falha e continua executando com privilégio de *root* (Drake *et al*, 2014).

Cabe ressaltar que a versões Android mais atuais já corrigiram esta falha de segurança. Entretanto, no contexto pericial, aparelhos antigos também podem ser objeto de exame.

3.2.2. *towelroot* - CVE-2014-3153

O *exploit towelroot* utiliza uma falha no *kernel* de versões Android anteriores à 4.4.2. A função *futex_requeue* (*kernel/futex.c)* nas versões do *kernel* anteriores à 3.14.5 não assegurava que chamadas teriam endereços *futex* diferentes, permitindo que usuários pudessem ganhar privilégios.

O processo utilizado no *exploit* é complexo, envolvendo manipulação da pilha (*stack*) e dos parâmetros utilizados nas chamadas de sistema *futex*. Uma explicação mais abrangente do processo pode ser encontra em (Nugroho, 2014).

Apesar da falha de segurança ser complexa, o uso do *exploit* em ambientes Android é trivial, bastando instalar um aplicativo e executá-lo (Hotz, 2014).

3.2.3. *pipe inatomic* - CVE-2015-1805

As implementações das funções *pipe_read* e *pipe_write* em versões do *kernel* anteriores à 3.16 não levavam em consideração possíveis falhas nas chamadas *__copy_to_user_inatomic* e *__copy_from_user_inatomic*, o que possibilitava a usuários sem privilégios causarem instabilidades no sistema, bem como ascenderem a usuário *root*. Apesar da falha já ser conhecida desde 2014, somente em fevereiro de 2016 a empresa Google foi notificada quanto a existência de aplicativos explorando tal vulnerabilidade, sendo lançado um *patch* para correção do problema.

Pelo fato de a correção ser recente, muitos aparelhos ainda se encontram vulneráveis, sendo uma boa opção para escalada de privilégios, inclusive em aparelhos com versões mais recentes do Android, como *Lollipop* e *Marshmallow*.

3.3. AQUISIÇÃO E ANÁLISE DE MEMÓRIA VOLÁTIL

A recuperação de memória volátil em dispositivos Android vem sendo amplamente explorada nos últimos tempos, dada a importância, para o mundo forense, das informações contidas neste tipo de memória.

Um dos primeiros trabalhos apresentados nessa área foi publicado por (Thing, Ng & Chang, 2010). A pesquisa tinha como proposta a apresentação de um sistema automatizado para análise de memória volátil. Foram criados um ambiente de troca de mensagens, uma ferramenta de aquisição de memória (*Memory Acquisition Tool - memgrab*) e uma ferramenta de análise de memória (*Memory Dump Analyser - MDA*) para verificação dos diferentes cenários de comunicação. No entanto, a ferramenta de aquisição de memória limitava-se a localizar e recuperar regiões de memória específicas para cada processo, baseando-se nos arquivos `/proc/pid/maps` e `/proc/pi/mem` do sistema de arquivos *procfs*.

Em (Sylve, Case, Marziale & Richard, 2012) as limitações da ferramenta mencionada foram contornadas com a produção de um módulo do *kernel* para extração de memória (conhecido como *dmd* ou *LiME - Linux Memory Extractor*). O módulo realizava a extração da memória percorrendo a estrutura *iomem_resource* do *kernel* e realizando a tradução de endereços. Atualmente, o referido módulo é uma das principais ferramentas de extração de memória volátil em sistema Linux, sendo amplamente utilizado em pesquisas, como em (Macht, 2013). Entretanto, esta abordagem tem como empecilho a necessidade de compilação do módulo para cada versão específica de *kernel*.

Existem trabalhos que visam produzir um módulo de aquisição de memória que seja independente de *kernel*. Em (Stüttgen & Cohen, 2014) foi apresentada uma nova técnica para inserção de módulo de aquisição de memória pré-compilado, que funciona independentemente da versão de *kernel*. A técnica consiste em injetar um pequeno módulo, denominado *parasite*, em um outro módulo de *kernel* já validado e inserido no sistema alvo. O trabalho teve como resultado a produção da ferramenta *Linux Physical Memory Acquisition Tool* (*LMAP*). Contudo, a pesquisa não abarcou a arquitetura ARM, muito utilizada em sistemas Android.

Em (Leppert, 2012) foi mostrada uma abordagem para aquisição e análise da *heap* dos aplicativos. Todavia, tal abordagem não pode ser utilizada em casos forenses reais, visto que os aplicativos precisam estar com a *flag android:debuggable* habilitada, o que não acontece na maioria dos casos reais.

Por fim, (Müller & Spreitzenbarth, 2013) apresentaram um conjunto de ferramentas denominado *Forensic Recovery Of Scrambled Telephones* (FROST) que utiliza ataques do tipo *cold boot* para recuperar chaves criptográficas utilizadas para cifrar partições do sistema Android, no sistema conhecido como *Full Disk Encryption* (FDE). Ainda nesta linha, (Hilgers, Macht, Muller & Spreitzenbarth, 2014) apresenta um outro trabalho onde a memória *post-mortem* recuperada pelo FROST é submetida a análise com o uso do *framework Volatility* (*Volatility*, 2016).

A seguir serão mostradas algumas técnicas que podem ser utilizadas para extração de memória RAM em ambientes Linux. Em especial, serão analisadas as ferramentas *LiME* e LMAP.

3.3.1. Aquisição de memória volátil em sistemas Linux

Os primeiros métodos utilizados para aquisição de memória RAM em ambientes Linux não necessitavam de softwares específicos ou mecanismos de inserção de módulos no *kernel* em execução. Existiam interfaces construídas dentro do sistema operacional que permitiam a leitura e escrita da memória física por aplicações com as devidas permissões. Por exemplo, o dispositivo `/dev/mem` podia ser lido com as ferramentas *dd* ou *cat* e redirecionado para um arquivo ou para a rede (Ligh, Case, Levy & Walters, 2014).

Entretanto, tal abordagem tinha alguns empecilhos. Muitas arquiteturas não mapeavam a memória de forma contígua, começando do *offset* 0. Como resultado, durante a extração de memória, se não fossem tomadas as devidas cautelas, algumas regiões de memória críticas, destinada a hardware específico, eram acessadas, causando instabilidade no sistema operacional. Além disso, o dispositivo `/dev/mem` somente podia endereçar 896MB de RAM.

O dispositivo `/dev/kmem` também foi historicamente utilizado para aquisição de memória em arquiteturas de 32 bits. Enquanto o dispositivo `/dev/mem` exportava memória física, o dispositivo `/dev/kmem` exportava o espaço de endereçamento virtual do *kernel* (*kernel virtual address space*). Assim como `/dev/mem` apresentava falhas de segurança, a interface `/dev/kmem` também trazia riscos de segurança ao expor a memória do *kernel* ao ambiente de usuário (*userland address space*).

Neste sentido, devido a questões de segurança, interfaces deste tipo foram desabilitadas na maioria das distribuições Linux modernas. Em especial, sistema Android não provêm suporte a dispositivos que expõem a memória física, nem APIs que deem suporte às aplicações para extração de memória em nível do espaço de usuário.

3.3.2. fmem

Para contornar os problemas com o uso dos dispositivos `/dev/mem` e `/dev/kmem`, foi criada a ferramenta *fmem*. A ferramenta opera carregando um *driver* no *kernel* que cria um dispositivo de caractere (*character device*) denominado `/dev/fmem`. O dispositivo funciona de forma parecida com o `/dev/mem`, no sentido de exportar memória física para outras aplicações, todavia possui uma série de vantagens em relação à técnica anterior. Uma delas consiste em verificar se a página demandada está realmente na memória (utilizando-se da função *page_is_ram*), antes de acessá-la, evitando causar instabilidades no sistema operacional. Além disso, o limite de 896 MB é superado (Ligh, Case, Levy & Walters, 2014).

No entanto, o examinador ainda deve tomar cuidado com as regiões de memória a serem acessadas. É necessário verificar em `/proc/iomem` quais tipos de memória são do tipo *System RAM*, uma vez que, como já mencionado anteriormente, o mapeamento da memória pode não ser contíguo. A Figura 3.4 mostra como as regiões da memória do tipo *System RAM* podem estar dispersas.

```
root@mestrado-VirtualBox:/home/mestrado# cat /proc/iomem
00000000-00000fff : reserved
00001000-0009fbff : System RAM
0009fc00-0009ffff : reserved
000c0000-000c7fff : Video ROM
000e2000-000ef3ff : Adapter ROM
000f0000-000fffff : reserved
  000f0000-000fffff : System ROM
00100000-dffeffff : System RAM
  01000000-017bbc4b : Kernel code
  017bbc4c-01d2537f : Kernel data
  01e90000-01fd2fff : Kernel bss
dfff0000-dfffffff : ACPI Tables
e0000000-e7ffffff : 0000:00:02.0
  e0000000-e012ffff : vesafb
f0000000-f001ffff : 0000:00:03.0
  f0000000-f001ffff : e1000
f0400000-f07fffff : 0000:00:04.0
  f0400000-f07fffff : vboxguest
f0800000-f0803fff : 0000:00:04.0
f0804000-f0804fff : 0000:00:06.0
  f0804000-f0804fff : ohci_hcd
f0806000-f0807fff : 0000:00:0d.0
  f0806000-f0807fff : ahci
fec00000-fec003ff : IOAPIC 0
fee00000-fee00fff : Local APIC
fffc0000-ffffffff : reserved
100000000-1347fffff : System RAM
134800000-137ffffff : RAM buffer
```

Figura 3.4: Estrutura `/proc/iomem`

Apesar da abordagem utilizando o dispositivo `/dev/fmem` ser vastamente utilizada em arquiteturas Intel, ela não é adequada para sistemas Android. Em (Sylve, Case, Marziale & Richard, 2012) a ferramenta *fmem* foi testada em ambientes Android com a arquitetura ARM, sendo encontrados alguns problemas. O primeiro deles é o fato de não existir, na arquitetura ARM, a função *page_is_ram*, o que pode provocar acesso a regiões de memória mapeadas para hardware específico. Além disso, também foi verificado que a ferramenta *dd* comumente distribuída nas ROMs Android não manipulava endereços acima de 0x80000000 de forma correta, uma vez que eram utilizados inteiros de 32 bits para armazenar o offset. Por fim, foi detectada uma alta taxa de sobrescrita da memória devido a trocas de contexto entre *kernel space* e *user space*.

3.3.3. LMAP

A ferramenta LMAP, que foi resultado do trabalho realizado por (Stüttgen & Cohen, 2014), é uma boa opção para extração de memória RAM em ambientes Linux, dada sua versatilidade e facilidade de uso. A técnica desenvolvida simplifica o processo de aquisição de memória em ambientes de resposta a incidentes, visto que o analista pericial não precisa se preocupar com questões de compatibilidade de *kernel*, bem como prévia compilação de módulo. Cabe ressaltar que a ferramenta é parte de um projeto maior, chamado *Rekall*, que envolve tanto extração quanto análise de memória volátil (*Rekall*, 2016).

Módulos de *kernel* carregáveis (*Loadable Kernel Module* - LKM) possuem código que estendem as funcionalidades do *kernel*, sendo diretamente ligados ao núcleo do sistema em execução. Desta forma, as regiões de memória ficam acessíveis aos módulos, uma vez que eles são executados com o mais alto nível de privilégio.

A técnica consiste em executar código arbitrário em um *kernel*, contornando os mecanismos de validação de compatibilidade de módulos, bem como parasitando um LKM já disponível no sistema alvo, alterando o fluxo de execução do processo.

Parasitar um LKM é uma técnica muito usada por *malwares*, tendo sido explorada em outros contextos (Styx, 2012). Devido ao LKM ser um objeto realocável, é possível combinar códigos de dois módulos distintos realocando-os em um único arquivo.

Neste tipo de técnica, geralmente o símbolo de inicialização (*init_module*) do módulo parasitado é alterado para a rotina de inicialização do módulo parasita, fazendo com que o *kernel* execute o código desejado. A Figura 3.5 ilustra o funcionamento da técnica.

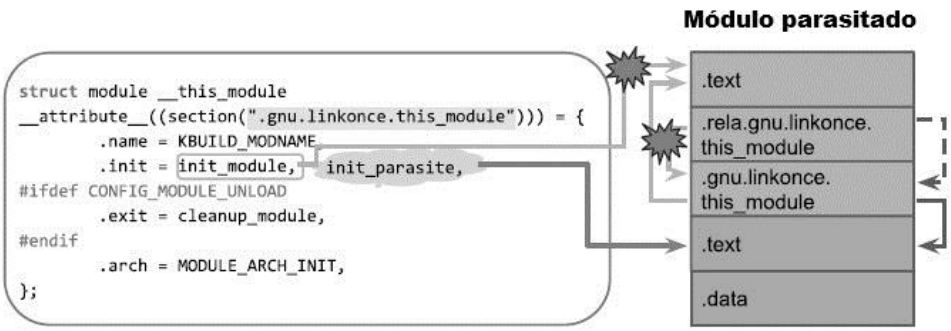


Figura 3.5: Módulo parasitado (Stüttgen & Cohen, 2014)

No contexto de aquisição de memória, nem todo LKM disponível no sistema pode ser utilizado. A ferramenta LMAP percorre o diretório `/lib/modules` em busca de um módulo compatível e altera as funções *init* e *cleanup* de forma a executar o código injetado. De acordo com os criadores da ferramenta, a técnica deve funcionar em qualquer Linux x86 64 bits, do *kernel* 2.6 ao 3.X. A Figura 3.6 mostra o uso da ferramenta em um sistema Ubuntu 14.04, com o *kernel* 3.19. Neste caso, o módulo *lp.ko* possuía os requisitos necessários para ser parasitado.

```
root@mestrado-VirtualBox:/usr/local/lmap/lmap/release# ./lmap -a dump
[+] Found suitable host /lib/modules/3.19.0-25-generic/kernel/drivers/char/lp.ko
[+] Using underscore prefixes and noop_llseek (kernel 3.x) relocation hooking method (2)
[+] Successfully injected parasite into /lib/modules/3.19.0-25-generic/kernel/drivers/char/lp.ko
[+] Injected pmem module has been loaded
[+] Created device node /dev/pmem
[+] Starting to dump memory
[+] [0000000000001000 - 000000000009efff] [WRITTEN]
[+] [0000000000100000 - 00000000558effff] [WRITTEN]
[+] Acquired 350350 pages (1435033600 bytes)
[+] Size of accessible physical address space: 1435435008 bytes (2 segments)
[+] Successfully wrote elf image of memory to dump
```

Figura 3.6: Ferramenta LMAP

Em contrapartida, a ferramenta não foi desenvolvida para as arquiteturas *ARM* e *x86* 32 bits, funcionando somente para a arquitetura *x86* 64 bits, o que inviabiliza o uso da ferramenta na maioria dos dispositivos Android.

Além disso, a maioria dos fabricantes Android faz uma sanitização do sistema antes de liberá-lo. Como o hardware é conhecido, não há necessidade de disponibilizar muitos LKMs, uma vez que os *drivers* já são previamente compilados juntamente com o *kernel*. A Tabela 3.4 mostra a diferença entre o número de módulos encontrados em uma distribuição Ubuntu 14.04 e

diferentes aparelhos Android. Enquanto em distribuições Linux comuns existem milhares de módulos, em sistemas Android tem-se apenas algumas dezenas.

Tabela 3.4 – Quantitativo de módulos no diretório `/lib/modules` ou `/system/lib/modules`

Distribuição Linux ou Aparelho	Total de módulo (.ko)
Ubuntu 14.04	*8226*
Samsung GT-P5200 (4.2.2 - Jelly Bean)	*11*
Samsung SM-G3812B (4.2.2 - Jelly Bean)	*18*
Sony LT22i (4.1.2 - Jelly Bean)	*131*
Sony Z2 (5.0.2 - Lollipop)	*0*

3.3.4. *LiME*

A extração e a análise da memória RAM são passos fundamentais para que seja remontado o caminho percorrido por um dispositivo Android, com base em coordenadas geodésicas, visto que a maioria das informações provenientes do chip receptor GPS não são registradas em arquivos de logs, bancos de dados ou qualquer outro artefato tipicamente armazenado em memória não-volátil.

Nesta pesquisa, optou-se por utilizar o módulo *LiME* para extração da memória volátil, visto que seu funcionamento, do ponto de vista forense, não causa grandes impactos na evidência e o módulo já foi amplamente testado em sistemas Android. Somente um arquivo precisa ser transferido para o dispositivo alvo. Além disso, o processo de carga do módulo altera pouco a evidência, visto que seu tamanho é reduzido (~70 KB) e necessita de poucas funções do *kernel* para aquisição da memória (Heriyanto, 2013).

O módulo *LiME*, assim como o fmem, também carrega um módulo no *kernel*, mas, ao invés de criar um dispositivo de caractere para o espaço de usuário, toda a captura é realizada em nível de *kernel*. Sendo assim, nas próximas seções serão mostrados os procedimentos necessários para compilação e uso do módulo *LiME*.

3.3.4.1. Preparação do módulo *kernel LiME*

Em sistemas Linux, durante o processo de carga de um módulo *kernel*, são feitas diversas verificações de compatibilidade com o intuito de garantir que o código tenha sido compilado para a versão específica do *kernel* em execução (Sylve, Case, Marziale & Richard, 2012). Para

o módulo *LiME* não é diferente. Por isto, o módulo deve ser compilado utilizando-se um ambiente específico, com o código fonte do *kernel* do aparelho a ser examinado e com o uso de ferramentas de compilação voltadas para a arquitetura do dispositivo. A maioria dos fabricantes de *smartphones* e *tablets* Android disponibiliza o código fonte do sistema operacional dos dispositivos, incluindo os *drivers* necessários. Nesta pesquisa, foram utilizados aparelhos das marcas Samsung e Sony, sendo possível realizar o download dos códigos fonte em seus sites[1][2].

A arquitetura do dispositivo a ser examinado também deve ser levada em consideração. A arquitetura *ARM* é utilizada em grande parte dos *smartphones* Android, visto que a concepção dos processadores, baseados no conjunto de instruções RISC (*Reduced Instruction Set Computing*), reduz o consumo de energia, contudo mantém a alto desempenho de processamento (ARM, 2005). Todavia, é cada vez mais comum o uso de outras arquiteturas, como a x86, em dispositivos Android. Consequentemente, o conjunto de ferramentas de programação (*toolchain*) utilizado para compilar o módulo deve levar em consideração a arquitetura do dispositivo. Especialmente para a arquitetura ARM, pode ser necessário o uso de um compilador cruzado (*cross compiler*) para se obter arquivos binários que se destinem a esta arquitetura.

As ferramentas para compilação e interação com os dispositivos Android podem ser obtidas do próprio site de desenvolvimento Android[3]. Em especial, o Android SDK (*Software Development Kit*) e o Android NDK (*Native Development Kit*) precisam ser baixados, instalados e configurados corretamente. O código fonte do *LiME* pode ser obtido em sua página do repositório github[4].

A documentação do *LiME* detalha como deve ser o processo de compilação do módulo *kernel* levando em consideração a arquitetura *ARM* (504ENSICS Labs, 2013). Para simplificar o processo, são definidas algumas variáveis de ambiente, conforme ilustrado na Figura 3.7, que apontam para os diretórios com as ferramentas de compilação e para o código fonte.

Definidas as variáveis, deve-se obter uma cópia do arquivo de configuração do *kernel* (config.gz). O arquivo de configuração, de acordo com a documentação do *LiME*, pode ser obtido do diretório `/proc/` do dispositivo alvo, com o uso da ferramenta ADB, conforme ilustrado na Figura 3.8.

[1] http://opensource.samsung.com - acessado em 05 jan. 2016
[2] http://developer.sonymobile.com/knowledge-base/open-source - acessado em 17 jan. 2016
[3] http://developer.android.com - acessado em 02 jan. 2016
[4] https://github.com/504ensicslabs/lime - acessado em 03 fev. 2016

```
$ export SDK_PATH=/path/to/android-sdk-linux/
$ export NDK_PATH=/path/to/android-ndk/
$ export KSRC_PATH=/path/to/kernel-source/
$ export CC_PATH=$NDK_PATH/toolchains/arm-linux-androideabi-
X.X.X/prebuilt/linux-86/bin/
$ export LIME_SRC=/path/to/lime/src
```

Figura 3.7: Compilação do módulo *LiME*

```
$ cd $SDK_PATH/platform-tools
$ ./adb pull /proc/config.gz
$ gunzip ./config.gz
$ cp config $KSRC_PATH/.config
```

Figura 3.8: Obtenção do arquivo config.gz

Todavia, muitos fabricantes não disponibilizam o arquivo de configuração. Desta forma, uma das maneiras de contornar este problema é verificar a existência do arquivo no próprio diretório do código fonte do *kernel*.

O próximo passo consiste em preparar o código fonte do *kernel* para o módulo. Junto com o código fonte do *LiME* também existe um arquivo *Makefile* de amostra, que deve ser alterado de acordo com as configurações do dispositivo alvo. Por fim, é realizada a compilação do módulo, tendo como resultado um arquivo com o nome *lime-<versão do kernel>. ko*, o qual será carregado no sistema alvo, conforme ilustrado na Figura 3.9.

```
$ cd $KSRC_PATH
$ make ARCH=arm CROSS_COMPILE=$CC_PATH/arm-eabi- modules_prepare
$ cd $LIME_SRC
$ make
```

Figura 3.9: Preparação do código fonte do *kernel*

3.3.4.2. Utilização do módulo *kernel LiME*

O processo de inserção do módulo em um *kernel* em execução para realização do *dump* da memória requer privilégios de usuário administrador (*root*). Por questões de segurança, o acesso como usuário *root*, por padrão, não é permitido pela maioria dos fabricantes. Por isto, a utilização de *exploits* que permitiam escalada de privilégios é um procedimento comum em análises do tipo *Live Forensics* e pode ser a única alternativa para recuperação de memória volátil.

Neste sentido, antes de ser a realizada a carga do módulo *LiME*, o dispositivo a ser examinado deve ser submetido a algum mecanismo de *rooting*. Com as permissões de usuário *root* habilitadas, o primeiro passo para inserção do módulo *LiME* é copiar o arquivo para a memória interna do aparelho ou para o cartão de memória externo. Esta tarefa pode ser feita utilizando o ferramenta adb.

O *LiME* pode extrair o conteúdo da memória de duas formas distintas. A primeira consiste em realizar o *dump* diretamente para o cartão de memória ou para a memória interna do aparelho. A segunda consiste na cópia dos dados por meio de um túnel TCP.

Além disso, existem três formatos de arquivos de saída, que podem ser do tipo *raw* (simples concatenação dos espaços de memória do tipo *System RAM*, na estrutura *iomem_resource*), *padded* (preenche todos os espaços de memória diferentes do tipo *System RAM* com o valor 0) ou *LiME* (inclui a inserção de cabeçalhos contendo metadados de cada espaço de endereçamento).

O caminho para o arquivo de saída também deve ser especificado pelo parâmetro *path*. Nesta pesquisa, optou-se por realizar o *dump* para o cartão de memória no formato *raw*, conforme ilustrado na Figura 3.10:

```
$ adb push lime.ko /sdcard/lime.ko
$ insmod /sdcard/lime.ko "path=/sdcard/ram.lime format=raw"
```

Figura 3.10: Utilização do modulo *LiME*

4. MÉTODO PROPOSTO

Neste capítulo será apresentada inicialmente a estrutura da arquitetura GPS em sistemas Android. Com base nos estudos e experimentos realizados, foi possível detalhar todas as camadas da arquitetura GPS, bem como examinar o comportamento das mensagens NMEA na memória RAM. Também foi possível verificar que grande parte das informações textuais de latitude e longitude estão armazenadas, na memória volátil, no formato Graus Decimais.

Em seguida, são descritos os procedimentos e técnicas desenvolvidos tanto para recuperar mensagens NMEA, quanto coordenadas geodésicas no formato Graus Decimais.

4.1. ARQUITETURA GPS EM SISTEMAS ANDROID

O sistema operacional Android possui o código fonte aberto (*open source*) o que facilita o entendimento de sua arquitetura. Do ponto de vista de desenvolvimento, o sistema pode ser dividido em camadas, que vai desde a camada de aplicação até os *drivers* dos diversos periféricos.

A Figura 4.1 ilustra a arquitetura do sistema Android. A camada superior consiste nos *frameworks* de aplicação que são utilizados por desenvolvedores de aplicativos para suprir necessidades específicas em seus códigos. Nesse contexto, a API *Google Location Services*, que é parte do *Google Play Services*, provê um *framework* que automatiza tarefas como a escolha de um provedor de localização e a detecção de movimentação (*Location and Maps*, 2016).

O mecanismo *Binder Inter-Process Communication* (IPC) permite que os *frameworks* ultrapassem as barreiras de um processo específico e façam chamadas aos serviços do sistema Android. Já os serviços do sistema realizam tarefas como gerenciamento de janelas, gerenciamento de mídias, dentre outros.

A camada de abstração de hardware (*Hardware Abstraction Layer* - HAL) define uma interface padrão a ser utilizada pelos fabricantes de hardware de tal forma que as camadas superiores do sistema Android não necessitem saber de detalhamentos em baixo nível da implementação de *drivers*. Para cada tipo de hardware há uma implementação de HAL específica, que geralmente é um módulo de biblioteca compartilhada (arquivo .so). O desenvolvimento de *drivers* para ambientes Android segue o mesmo procedimento utilizado em outros sistemas Linux.

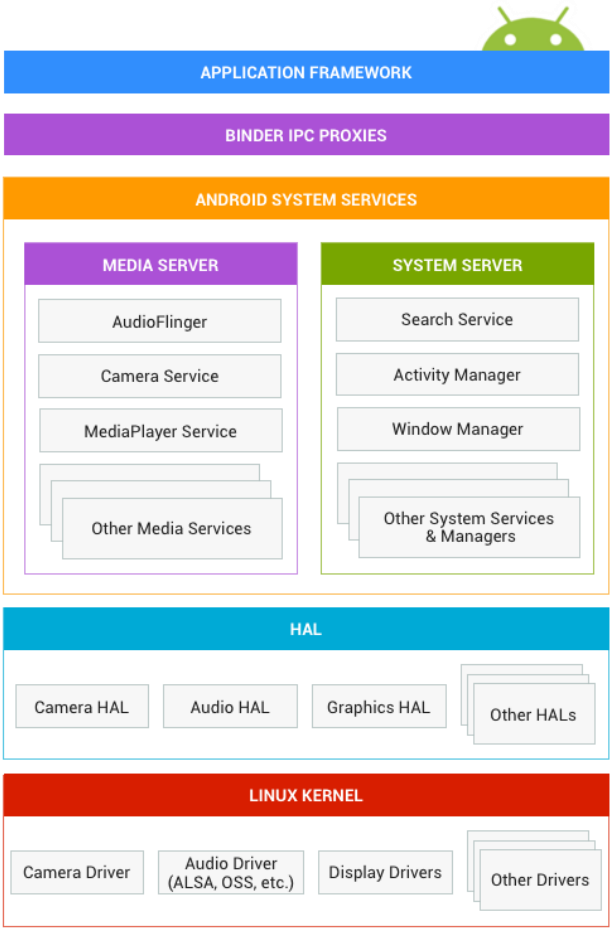


Figura 4.1: Arquitetura de desenvolvido do sistema Android (*Android Interfaces and Architecture*, 2016)

No que diz respeito a arquitetura GPS em ambientes Android, as camadas são semelhantes às anteriormente descritas. Nas camadas superiores estão os aplicativos e os *frameworks* que fazem uso de serviços de posicionamento. Em especial, neste trabalho, foram utilizados os aplicativos *Google Maps* e *Waze*.

Os aplicativos, por sua vez, fazem uso dos *frameworks* de localização, que geralmente são desenvolvidos na linguagem de programação Java. Como exemplo, a classe *LocationManager* é comumente usada para prover acesso aos serviços de localização, possuindo, dentro outros, o método *addNmeaListener*.

Em nível de sistema, funcionando como base para os serviços de localização, existe toda uma *engine* para receber os dados do receptor GPS e fornecê-los para camadas superiores. A partir da versão 4.0 dos sistemas Android, alguns fabricantes começaram a utilizar o projeto GPSD para monitorar os receptores GPS embutidos nos aparelhos (GPSD, 2016).

O principal programa do projeto é um *deamon* de serviço denominado *gpsd* que coleta os dados do receptor GPS e reporta para as camadas superiores em formato NMEA 0183 ou JSON. O *deamon* fica geralmente localizado no diretório `/system/bin`, todavia essa localização pode mudar dependendo da plataforma Android e do fabricante do dispositivo.

A *engine* também possui arquivos de configuração, geralmente em formato xml, que variam de nome e localização a depender da plataforma. Alguns exemplos de arquivos são: `glconfig.xml`, `gps.xml`, `Jupiter.xml`, `gpsconfig.xml`, `gps.conf`, `secgps.conf`, etc. Tais arquivos geralmente estão localizados nos diretórios `/system/etc`, `/system/etc/gps`, `/vendor/etc/`, `/data/gps`, dentro outros (*Understanding Android GPS Architecture*, 2012).

Em sistemas Android, a localização também pode ser obtida por outros meios além do receptor GPS. *Cell-IDs* e *Wi-Fi* também podem ser utilizadas para estabelecer a localização do dispositivo, todavia a precisão dos dados é diferente daquela obtida no modo GPS.

Além disso, o sistema de GPS pode ser auxiliado por servidores conectados à internet, funcionando em um modo conhecido como A-GPS (*Assisted* GPS) e utilizando o protocolo SUPL (*Secure User Plane Location)* para obter informações que melhorem o tempo para obter o primeiro posicionamento (*Time-To-First-Fix* - TTFF). Desta forma, existem outros arquivos que fazem parte da *engine* e que auxiliam em todos esses serviços mencionados.

Nas camadas inferiores existem os *drivers* e os *chips* receptores GPS. Os *drivers* utilizam API´s de baixo nível para se comunicar com o receptor GPS e geralmente consistem de um ou mais arquivos que se iniciam com o prefixo gps e terminam com a extensão *.so* (*`gps.default.so`*, *`gps.aries.so`*, etc), comumente localizados nos diretórios */System/Lib/hw/* e */Vendor/Lib/hw/* (*Understanding Android GPS Architecture*, 2012). A Figura 4.2 ilustra a arquitetura GPS em ambientes Android.

Como pode ser visto na Figura 4.2, as mensagens NMEA perpassam praticamente todas as camadas da arquitetura GPS em ambientes Android, desde a comunicação *driver*/receptor até os *frameworks* em nível de aplicação. Como consequência, tais informações podem ser encontradas em diferentes regiões de memória, sendo de grande interesse forense.

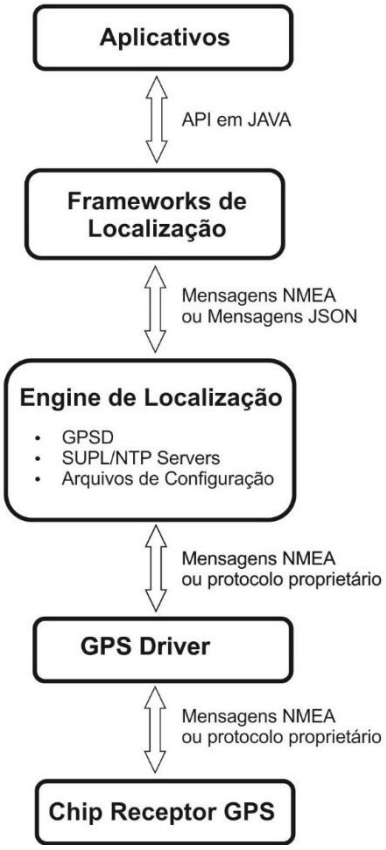


Figura 4.2: Arquitetura GPS em sistemas Android

4.2. RECUPERAÇÃO DE MENSAGENS NMEA

Para realização dos testes de extração de mensagens NMEA da memória RAM, foi necessário primeiro entender como os dados estão dispostos na memória.

As mensagens, como já dito na Seção 4.1, são transmitidas por diversas camadas do sistema operacional. Por padrão, a especificação do protocolo NMEA estabelece que as mensagens sejam transmitidas em codificação ASCII. As camadas inferiores da arquitetura GPS (*deamon gpsd e os drivers*) são escritas na linguagem de programação C, que geralmente trata *strings* como uma sequência de bytes codificados em ASCII. Desta forma, algumas mensagens serão encontradas na memória como uma simples sequência de caracteres codificada em ASCII, como pode ser visto na Figura 4.3.

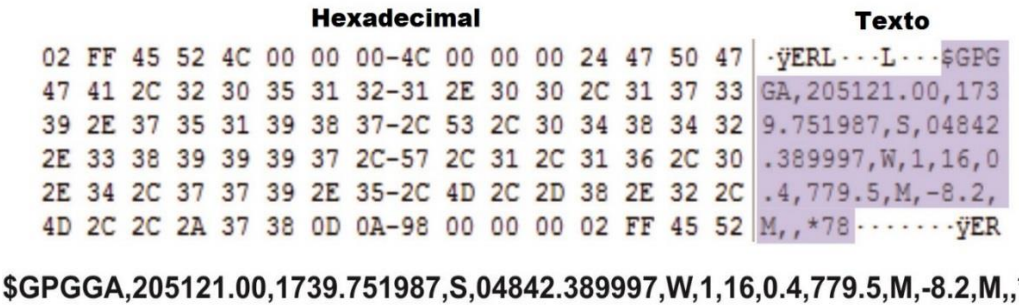


Figura 4.3: Mensagem GPGGA em codificação ASCII.

Por outro lado, os aplicativos e *frameworks*, em ambientes Android, são escritos utilizando-se a linguagem de programação Java, que é baseada no *charset* Unicode, o qual suporta a maioria dos símbolos utilizados no mundo. A especificação original do Unicode definia os caracteres como elementos de tamanho fixo, com 16 bits. Entretanto, o padrão veio sendo alterado com o passar dos anos para permitir representações que necessitem de mais de 16 bits. Logo, as mensagens do protocolo NMEA, quando utilizadas nas camadas superiores da arquitetura GPS, utilizam mais bytes, como pode ser visto na Figura 4.4.

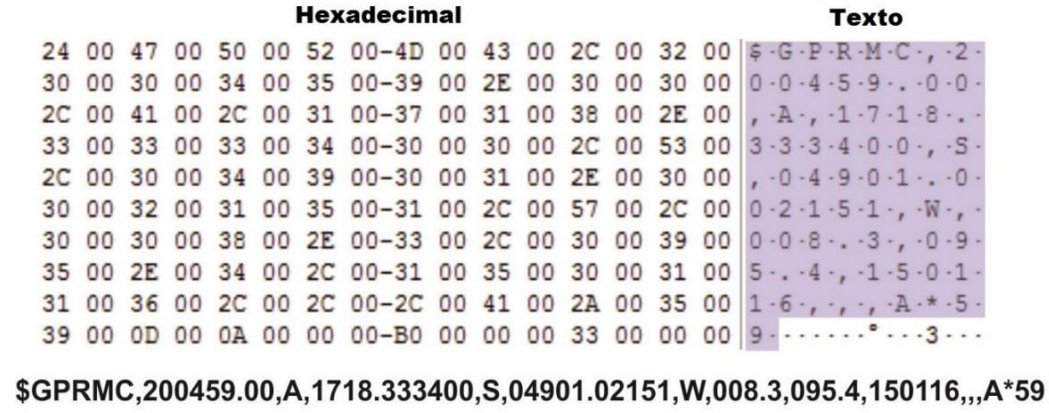


Figura 4.4: Mensagem GPRMC em codificação Unicode.

4.2.1. Apresentação do algoritmo para recuperação de mensagens NMEA

Sabendo que as mensagens do protocolo NMEA 0183 podem estar dispostas na memória em diferentes codificações, foi criado um algoritmo para recuperação das referidas informações. Cabe ressaltar que o algoritmo leva em consideração mensagens NMEA bem formadas e

válidas, ou seja, mensagens com coordenadas geodésicas que tenham relevância para reconstrução de trajetórias.

O algoritmo funciona de maneira simples. Percorre-se o *dump* de memória até que seja encontrado um caractere do tipo '$'. Quando o símbolo é encontrado, os bytes subsequentes são lidos, visando encontrar os padrões textuais "GPRMC" e "GPGGA", tanto em codificação ASCII como Unicode.

Caso os padrões não sejam encontrados, o *dump* de memória volta a ser percorrido até que o caractere '$' seja encontrado novamente. Do contrário, os valores da mensagem NMEA são lidos e validados, inclusive com a verificação de *checksum*. Caso a validação ocorra com sucesso, a mensagem NMEA é salva. Do contrário, retorna-se a busca por mais mensagens NMEA. O fluxograma da Figura 4.5 ilustra o processo de recuperação de mensagens NMEA.

4.3. RECUPERAÇÃO DE COORDENADAS GEODÉSICAS NO FORMATO GRAUS DECIMAIS

Dados de posicionamento também são encontrados na memória RAM em formatos textuais. As classes dos *frameworks* de localização interpretam as mensagens do protocolo NMEA ou de outros protocolos vindos das camadas inferiores e transformam em outros artefatos mais abstratos e acessíveis ao desenvolvedor. Desta forma, as coordenadas geográficas podem ser encontradas na memória RAM como cadeias de caracteres de formato numérico.

As informações, assim como no caso das mensagens do protocolo NMEA, podem estar codificadas em ASCII ou Unicode, a depender do processo que está manipulando a informação. As Figuras 4.6 e 4.7 mostram sequencias de caracteres com coordenadas geodésicas em diferentes codificações.

Os experimentos realizados neste trabalho, que serão apresentados no próximo capítulo, mostraram que a maioria das coordenadas geográficas em formato textual presentes na memória RAM estavam no formato Graus Decimais.

Por conseguinte, optou-se pela criação de um algoritmo para recuperação deste tipo de informação, dada sua maior importância no cenário de reconstrução de trajetórias.

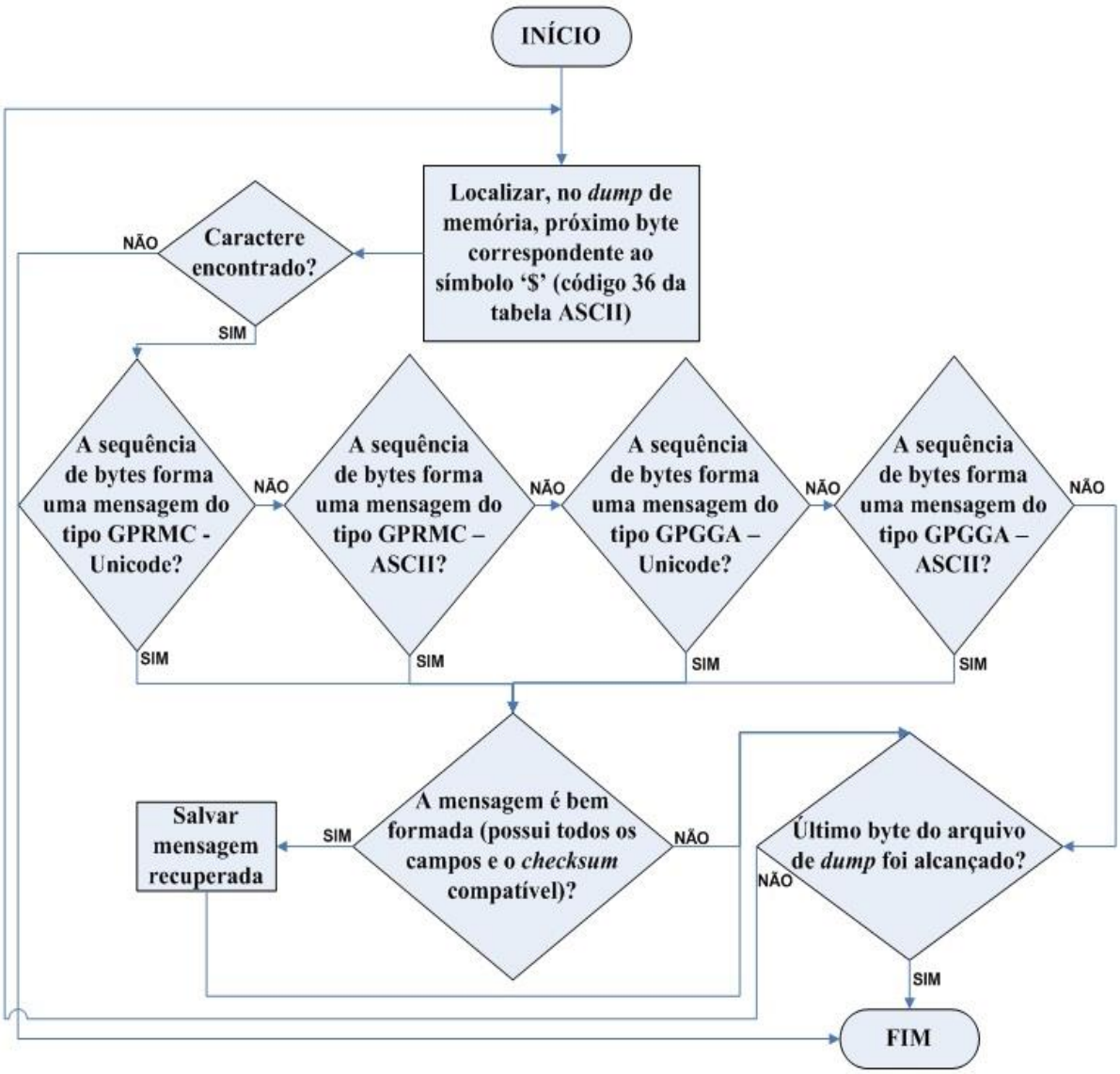


Figura 4.5: Algoritmo para buscas de mensagens NMEA.

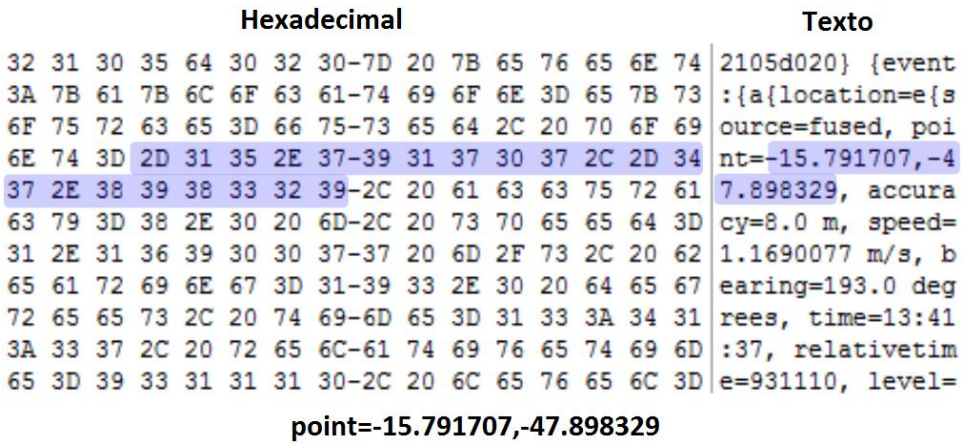


Figura 4.6: Coordenadas geográficas no formato Graus Decimais codificada em ASCII.

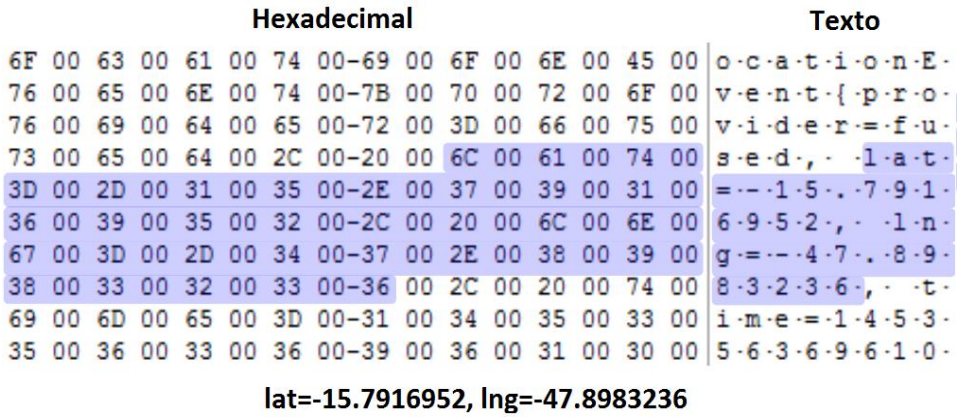


Figura 4.7: Coordenadas geográficas no formato Graus Decimais codificada em Unicode.

4.3.1. Apresentação do algoritmo para recuperação de coordenadas geodésicas no formato Graus Decimais

Encontrar dados textuais que representem números em um *dump* de memória pode se tornar uma tarefa complexa, uma vez que é necessário contextualizá-los. A Figura 4.8 mostra uma região de memória que contém dois números que poderiam ser interpretados como coordenadas geográficas. Entretanto, neste caso, a referida região não apresenta relação com dados de posicionamento, estando associada a nomes de arquivos de cache gerados por um aplicativo do tipo *e-commerce*. Um algoritmo que busque as informações de forma mais abrangente trará uma grande quantidade de falsos positivos, dificultando o trabalho pericial.

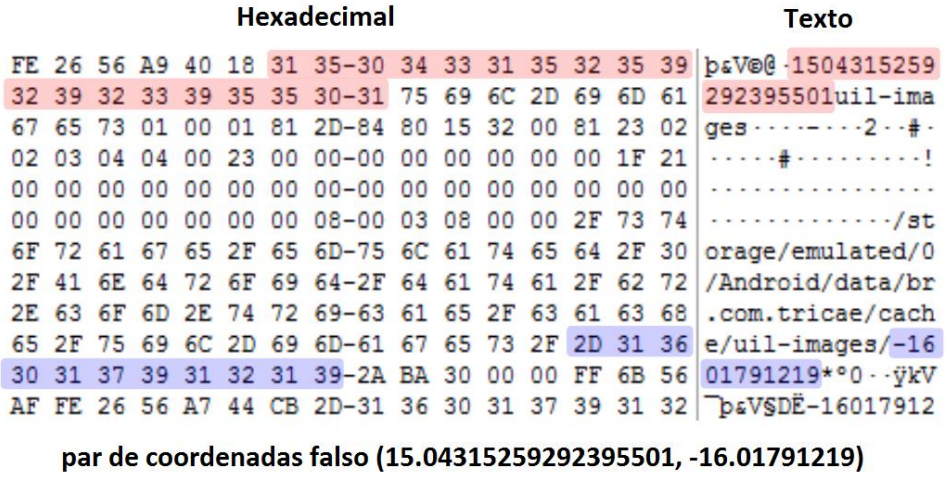


Figura 4.8: Falso positivo para coordenadas geográficas.

Os dados das coordenadas geográficas geralmente estão associados com outras sequências de caracteres, como “lat”, “latitude”, “long”, “longitude”, “coord”, “coordinate”, “la.”, “lo.”, “point”, “gps” dentre outras. Porém nem sempre estas expressões estão presentes. Um algoritmo que limite os dados recuperados com base em expressões pode ter uma grande quantidade de falsos negativos, podendo descartar informação de interesse investigativo.

Neste sentido, optou-se por criar um algoritmo que localize as coordenadas geodésicas com base em duas regras:

- limitação geográfica;
- princípio da localidade.

4.3.1.1. Limitação geográfica

As informações numéricas que sejam candidatas a coordenadas geográficas devem estar dentro de um limite geográfico, que será estabelecido pelo examinador. Ou seja, para que o algoritmo seja executado, deverá ser fornecido um par de coordenadas geodésicas de referência, bem como o limite de abrangência, em quilômetros, de forma que todos os dados recuperados estejam dentro região geográfica estabelecida. Como exemplo, a Figura 4.9 ilustra um limite de 5 km estabelecido em torno de uma região central de Brasília, de coordenadas -15.791171, -47.891168.

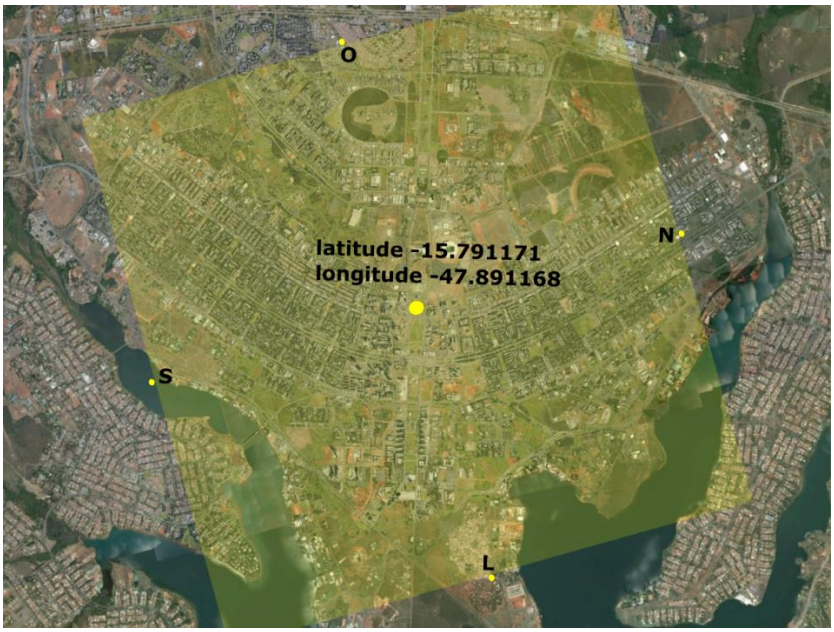


Figura 4.9: Limites geográficos em torno das coordenadas -15.791171, -47.891168.

Para limitar geograficamente a região de interesse investigativo, foi utilizada a fórmula de *Haversine*. Tal equação é muito utilizada em geodesia para calcular a distância entre dois pontos de uma esfera a partir de suas latitudes e longitudes (Chopde & Nichat, 2013). A fórmula apresenta este nome pelo fato de a expressão ser construída com base na função *Haversine* utilizada no ramo da trigonometria esférica. O semi-seno-verso (que em inglês, é denominado *haversine*) é empregado na solução do triângulo de posição em várias Tábuas para Navegação Astronômica (Miguens, 1999).

Dados dois pontos de uma esfera de raio *R*, com latitudes φ_1 e φ_2 e longitudes λ_1 e λ_2, bem como a distância *d* entre dois pontos, a fórmula de *haversine* é dada pela Equação 4.1:

$$hav\left(\frac{d}{R}\right) = hav(\varphi_2 - \varphi_1) + \cos(\varphi_1)\cos(\varphi_2)\, hav(\lambda_2 - \lambda_1) \quad \text{Equação 4.1}$$

Onde *hav* é a função *haversine*, denotada pela Equação 4.2:

$$hav(\theta) = sin^2\left(\frac{\theta}{2}\right) = \frac{1-\cos(\theta)}{2} \quad \text{Equação 4.2}$$

A distância *d* pode ser obtida pela aplicação do arco seno na fórmula de *Haversine*, conforme a Equação 4.3:

$$d = 2\mathrm{R}arcsin\left(\sqrt{sin^2\left(\frac{\varphi_2-\varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\, sin^2\left(\frac{\lambda_2-\lambda_1}{2}\right)}\right) \quad \text{Equação 4.3}$$

Durante a implementação da ferramenta de busca de coordenadas posicionamento, observou-se que o custo computacional para calcular a distância entre dois pontos toda vez que fosse encontrado um par de coordenadas era muito alto. Consequentemente, optou-se por estabelecer quatro limites (ao norte, ao sul, ao leste e ao oeste), de acordo com a distância estabelecida pelo examinador, formando uma região quadrada de interesse pericial.

Para os limites geográficos em torno das coordenadas -15.791171, -47.891168 da Figura 4.9, por exemplo, tem-se os seguintes limites: -15.746204919704063, -47.891168 ao norte, -15.836137080295936, -47.891168 ao sul, -15.791171, -47.84443833937693 ao leste e -15.791171, -47.93789766062307 ao oeste.

Por conseguinte, o custo computacional é reduzido, uma vez que, ao ser encontrado um par de coordenadas, são feitas apenas comparações com as coordenadas limítrofes, ao invés de se utilizar funções trigonométricas, bem como operações com ponto flutuante.

Para se calcular as coordenadas limítrofes foram feitas algumas manipulações na expressão para cálculo de distância entre pontos. Para se encontrar os limites latitudinais, fixa-se a longitude de forma a se encontrar as latitudes dos extremos norte e sul. Para se encontrar os limites longitudinais, fixa-se a latitude de forma a se encontrar as longitudes dos extremos leste e oeste.

Desta forma, dados dois pontos de uma esfera de raio *R*, com latitudes φ_1 e φ_2 e longitudes λ_1 e λ_2, bem como a distância *d* entre dois pontos, tem-se:

- Para os limites latitudinais, faz-se $\lambda_1 = \lambda_2$, obtendo-se a Equação 4.4 para φ_2:

$$\varphi_2 = 2\arcsin\left(sin\left(\frac{d}{2R}\right)\right) + \varphi_1 \qquad \text{Equação 4.4}$$

- Para os limites longitudinais, faz-se $\varphi_1 = \varphi_2$, obtendo-se a Equação 4.5 para λ_2:

$$\lambda_2 = 2\arcsin\left(\frac{\sin\left(\frac{d}{2R}\right)}{\cos(\varphi_1)}\right) + \lambda_1 \qquad \text{Equação 4.5}$$

4.3.1.2. Princípio da localidade

Variáveis de um programa geralmente são armazenadas próximas umas das outras na memória RAM, bem como vetores e matrizes são armazenados em sequência de acordo com seus índices. Baseado neste princípio, o sistema de memória tende a manter dados e instruções próximos aos que estão sendo executados no topo da hierarquia de memória (Santana, 2014).

Dados de latitude e longitude, de acordo com o que foi verificado nos experimentos, geralmente estão dispostos na memória RAM em regiões próximas umas das outras. Neste sentido, para que o algoritmo seja executado, também deverá ser fornecida uma distância máxima, em bytes, a ser percorrida pelo algoritmo em busca dos dados desejados.

Ou seja, o algoritmo, ao encontrar uma informação que tenha as características de uma coordenada geodésica, deverá buscar a outra informação que forma o par de coordenadas nas proximidades do primeiro dado já encontrado, limitando-se a distância máxima previamente estabelecida.

Na Figura 4.10 é mostrado um exemplo onde fora estabelecido como entrada do algoritmo um limite de 100 bytes. Cabe salientar que este limite é definido pelo examinador. No caso ilustrado, a longitude está presente nas proximidades da latitude -15.8817297, dentro do limite de 100 bytes anteriores ao primeiro byte da latitude.

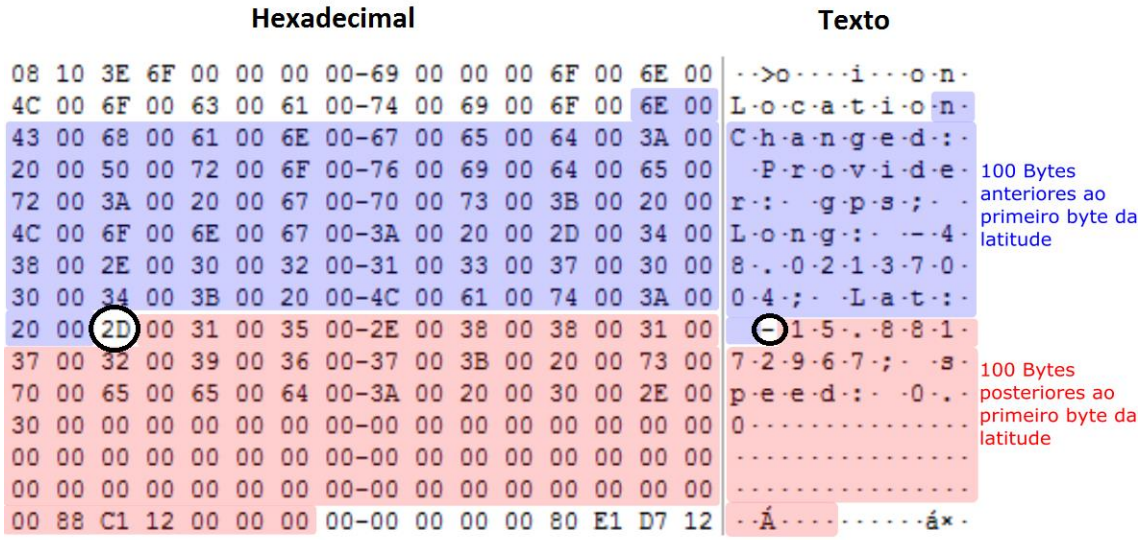


Figura 4.10: Limite máximo de 100 bytes.

4.3.1.3. Algorítmo

Por fim, na Figura 4.11 é mostrado o fluxograma com o algoritmo para recuperação de coordenadas geodésicas no formato Graus Decimais. O Algoritmo recebe como entrada três valores:

1. um par de coordenadas como referência para a busca;
2. a distância máxima, em quilômetros, que limitará geograficamente a busca;
3. e o total de bytes que limitará a busca em torno de uma coordenada encontrada.

O *dump* de memória é percorrido até que seja encontrado valor correspondente a um dígito numérico ou ao caractere "-". Quando o valor é encontrado, os bytes subsequentes são lidos, visando encontrar a latitude no formato Graus Decimais, dentro da abrangência geográfica. No momento que a latitude é encontrada, um fragmento de memória é extraído, com base no total de bytes passado como parâmetro, e inicia-se a busca por um valor de longitude também válido.

Implementações dos algoritmos na linguagem de programação *python* estão disponíveis na plataforma GitHub e podem ser acessadas em: *https://github.com/jpclaudino/nmeaSearch*.

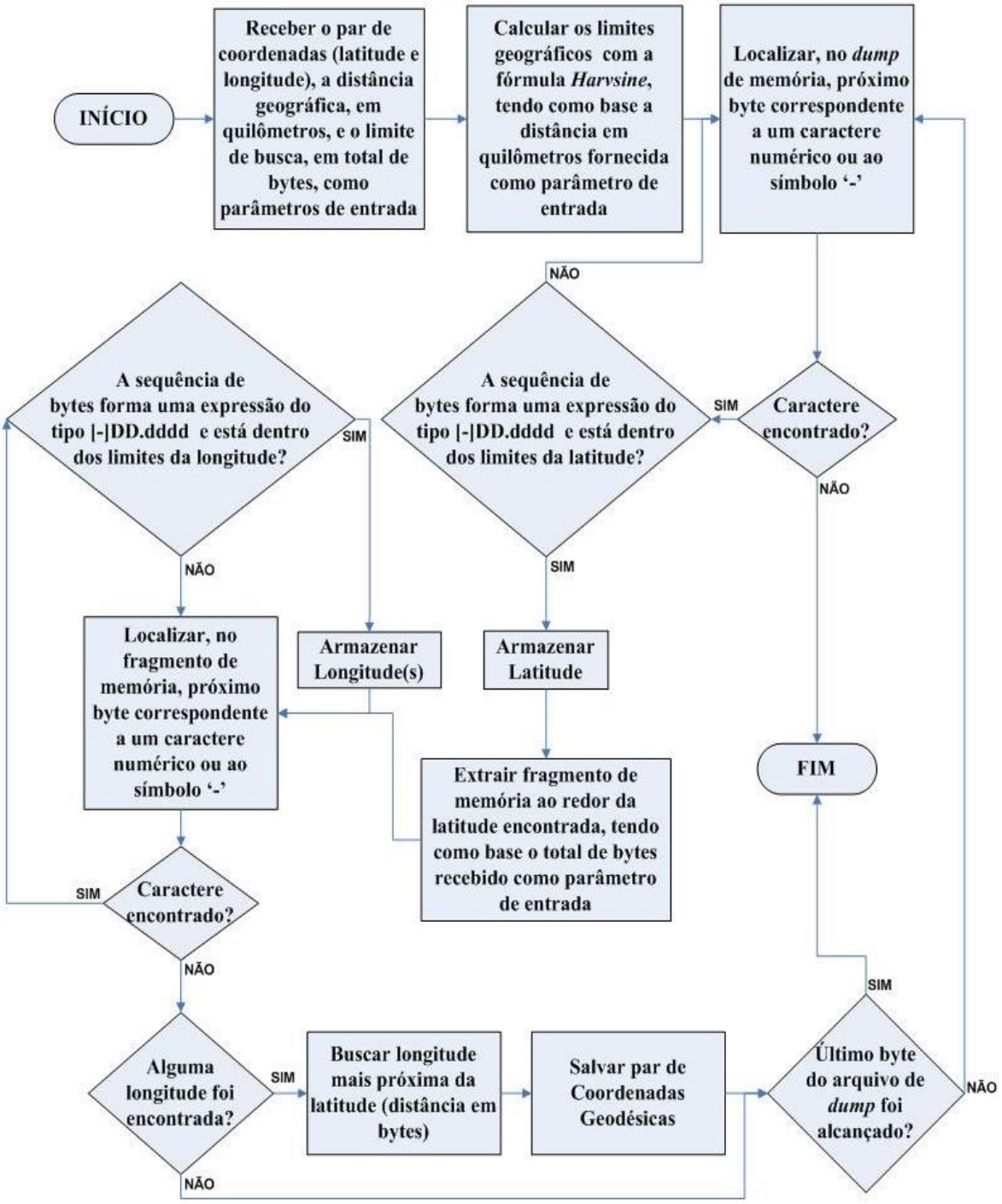


Figura 4.11: Algoritmo para buscas de coordenadas geodésicas no formato Graus Decimais.

5. EXPERIMENTOS E RESULTADOS OBTIDOS

Para analisar a viabilidade da recuperação de dados de GPS em ambientes de reposta a incidentes, foram realizados testes em diferentes cenários com diferentes dispositivos móveis. Os aparelhos escolhidos possuíam características diversas, com diferentes versões do sistema Android e diferentes arquiteturas (*ARM* e *x86*).

Além disso, foi construída uma ferramenta para extração e análise de dados (Sousa, 2016), que percorre o *dump* da memória RAM em busca de mensagens da especificação NMEA 0183, bem como coordenadas geodésicas no formato graus decimais (doravante denominadas coordenadas DD). O programa tem como entrada um arquivo contendo a imagem da memória volátil e retorna como saída arquivos de log e um arquivo no formato gpx. A ferramenta faz uso das bibliotecas pynmea2 (Flanagan, 2013) e gpxpy (Krajina, 2010) para interpretar as mensagens NMEA e escrever arquivos no formato GPX, respectivamente. Cabe ressaltar que a ferramenta foi desenvolvida na linguagem de programação Python, podendo ser facilmente adaptada aos principais frameworks de análise de memória volátil, com o *Volatility* e o *Rekall*.

5.1. PRÉ-REQUISITOS PARA TODOS OS EXPERIMENTOS

Os experimentos foram realizados com o uso dos aplicativos *Waze* e *Google Maps*. Os aplicativos foram escolhidos devido ao fato de serem os mais utilizados pelos usuários Android, no que tange a serviços de localização. O *Waze* é um dos maiores aplicativos de trânsito e navegação do mundo baseado em uma comunidade colaborativa (*Waze,* 2016), enquanto o *Google Maps* é o aplicativo desenvolvido pela empresa Google que traz os serviços de localização para o universo dos dispositivos móveis (*Google Maps,* 2016).

Os dispositivos foram incialmente submetidos a *exploits* para escalada de privilégios e acesso como usuário administrador (*root*). Foram utilizados os softwares *Kingo Root* e *iRoot*, que automaticamente exploram falhas de segurança de acordo com as vulnerabilidades específicas do aparelho. Importante frisar que não foram utilizadas técnicas que envolvem substituição da partição *Recovery*, nem qualquer outra técnica que pudesse reiniciar o aparelho.

Também foram previamente compilados módulos *LiME* para extração da memória RAM de acordo com as características dos aparelhos. Cabe ressaltar que o processo de compilação do *kernel* e do módulo *LiME* é diferente para cada tipo de dispositivo. Para que a compilação ocorra com sucesso, deve ser observado, nos arquivos README presentes no diretório raiz do código fonte do *kernel*, qual *toolchain* deve ser utilizado.

Para realização dos testes, todos os dispositivos foram colocados em modo avião, de forma a evitar o acesso às redes wireless e de telefonia, bem como o acionamento do modo GPS assistido (A-GPS). Também foram removidos os cartões SIM (*simcards*), por precaução, ainda que o modo avião já estivesse habilitado. Desta forma, garantiu-se o uso dos aplicativos no modo GPS *only*.

Para se obter os dados, foram realizados testes sob diferentes condições de tráfego e clima. Os dispositivos foram montados em um veículo de teste, que percorreu três trajetos distintos. O primeiro trajeto consistiu em um pequeno trecho de 4 km, o segundo em um trecho médio de aproximadamente 15 km e o terceiro trajeto em um trecho longo de 150 km.

5.2. EXPERIMENTO EM TRAJETOS CURTOS (4 KM)

Aplicativos que fazem uso de GPS são cada vez mais utilizados por usuários de dispositivos móveis, até mesmo para percorrer trajetos curtos, como nos casos de aplicativos do tipo *fitness-tracking*, que são geralmente utilizados por praticantes de esportes. Os experimentos em trajetos curtos foram realizados com objetivo de reproduzir cenários onde o tempo de resposta de equipes periciais é menor, como nos casos de flagrante delito.

Neste experimento, foram utilizados 4 dispositivos, cujas características podem ser visualizadas na Tabela 5.1. Versões mais recentes do Android - *Marshmallow* (6.X) e *Nougat* (7.X) – não foram analisadas, visto que no período de testes poucos aparelhos já possuíam a versão *Marshmallow* e a versão *Nougat* sequer tinha sido lançada (Dashboards, 2016).

Tabela 5.1 – Especificações dos dispositivos móveis

Fabricante/ Modelo	Versão do Android	Chipset	Memória
Sony LT22i (Smartphone)	*4.1.2 - Jelly Bean*	*NovaThor U8500 ARMv7 Processor rev 1 (v7l)*	*820588 kB*
Sony Z2 (Smartphone)	*5.0.2 - Lollipop*	*Qualcomm MSM8974PRO-AB ARMv7 Processor rev 1 (v7l)*	*2846028 kB*
Samsung GT-P5200 (Tablet)	*4.2.2 - Jelly Bean*	*Intel(R) Atom(TM) CPU Z2560 @ 1.60GHz*	*944372 kB*
Samsung SM-G3812B (Smartphone)	*4.2.2 - Jelly Bean*	*PXA1088 ARMv7 Processor rev 3 (v7l)*	*804440 kB*

O experimento foi realizado em uma via no Distrito Federal, com velocidade máxima de tráfego de 60 km/h. O trajeto percorrido pelo veículo pode ser visualizado na Figura 5.1.

Figura 5.1: Trajeto de curta distância

Em casos de resposta a incidentes, é uma boa pratica forense isolar o aparelho das ondas eletromagnéticas, colocando o aparelho em modo avião e em gaiolas de Faraday. Entretanto, nem sempre as melhores práticas são seguidas por quem tem o primeiro contato com uma determinada evidência. Neste sentido, foram realizados dois tipos de testes, a seguir descritos:

1. Serviços de localização habilitados.
2. Serviços de localização desabilitados.

No primeiro, os serviços de localização do Android não foram interrompidos. Desta forma, os dados do receptor GPS continuaram sendo recebidos pelos aplicativos *Waze* e *Google Maps*, fazendo mais uso da memória RAM, simulando casos onde o dispositivo não é isolado.

No segundo caso, os serviços de localização foram desabilitados, no instante que o veículo parou, de forma que os aplicativos não mais conseguiam determinar trajetórias. Desabilitar os serviços de localização, assim como colocar o aparelho em modo avião, são tarefas simples, que podem facilmente ser realizadas pelas equipes de primeiro contato com cenas de crime, que podem não ter o suporte necessário para isolamento eletromagnético dos aparelhos.

Os testes em trajetos curtos consistiram em analisar o comportamento da memória RAM ao longo do tempo, verificando a viabilidade de reconstrução do caminho percorrido pelo dispositivo, mesmo já tendo sido passado um certo período de tempo. Foram realizados *dumps* da memória dos aparelhos durante um período de 30 minutos. Para cada dispositivo, após ser percorrido o trajeto de 4 km com um dos aplicativos ligados, o veículo foi parado e foi iniciada a recuperação dos dados da memória. A cada 5 minutos um novo *dump* da memória era

produzido. Para o dispositivo Sony Z2, os *dumps* foram realizados a cada 10 minutos, visto que o processo de cópia integral da memória RAM, neste aparelho, demandava mais de 5 minutos, dadas as limitações na velocidade de escrita do cartão *microSD*.

As imagens das memórias dos dispositivos foram então submetidas ao processamento pela ferramenta de extração e análise.

Nos testes em trajetos curtos, para o método de extração de coordenadas DD, foi utilizado o par de coordenadas "-15.801526, -47.911293" como referência, com uma abrangência de 3 km e um limite de 100 bytes para busca.

Os resultados serão apresentados em duas etapas, quais sejam:

1. Resultados para os experimentos com os serviços de localização habilitados.
2. Resultados para os experimentos com os serviços de localização desabilitados.

Em ambas as abordagens, foi feita uma análise tanto das mensagens NMEA recuperadas, quanto das coordenadas DD. Primeiro são mostrados os resultados gerais e logo em seguida cada dispositivo é analisado separadamente.

5.2.1. Resultados com os serviços de localização habilitados

As Tabelas 5.2 e 5.3 mostram a quantidade de coordenadas recuperadas de acordo com o tempo em que o *dump* foi realizado, nos experimentos com os serviços de localização habilitados. Na Tabela 5.2 é mostrado o quantitativo de mensagens NMEA recuperadas, ao passo que na Tabela 5.3 os dados são relativos às coordenadas DD.

Cabe ressaltar que o quantitativo leva em consideração somente mensagens GPGGA e GPRMC válidas, ou seja, que continham coordenadas GPS.

Tabela 5.2 – Quantitativo de mensagens NMEA recuperadas por *dump*, com os serviços de localização habilitados.

Dispositivo	**Tempo transcorrido em minutos**						
	0	**5**	**10**	**15**	**20**	**25**	**30**
Sony LT22i	*275*	*364*	*355*	*382*	*396*	*431*	*316*
Sony Z2	*1175*	*N/A*	*195*	*N/A*	*159*	*N/A*	*125*
Samsung GT-P5200	*294*	*274*	*284*	*302*	*342*	*311*	*345*
Samsung SM-G3812B	*279*	*273*	*272*	*255*	*346*	*324*	*238*

Tabela 5.3 – Quantitativo de coordenadas DD recuperadas por dump, com os serviços de localização habilitados

Dispositivo	Tempo transcorrido em minutos						
	0	5	10	15	20	25	30
Sony LT22i	*186*	*200*	*167*	*138*	*156*	*158*	*147*
Sony Z2	*338*	*N/A*	*744*	*N/A*	*1056*	*N/A*	*871*
Samsung GT-P5200	*278*	*698*	*280*	*331*	*635*	*655*	*383*
Samsung SM-G3812B	*739*	*791*	*755*	*828*	*754*	*759*	*740*

Como é possível observar nos gráficos da Figura 5.2, a quantidade de mensagens recuperadas quando os serviços de localização estavam ativos varia com o passar do tempo, como era de se esperar, visto que o receptor GPS continua operando normalmente. Entretanto, as coordenadas de posicionamento tendem a se agrupar no ponto de repouso do veículo.

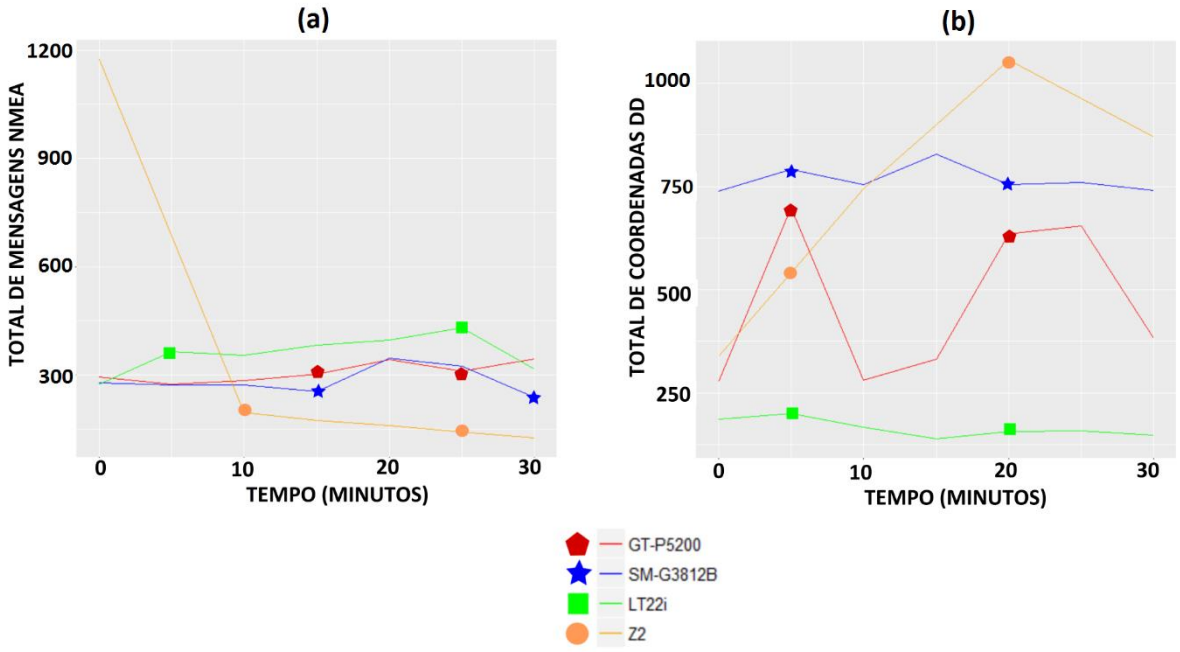


Figura 5.2: Total de dados recuperados com os serviços de localização habilitados: a) mensagens NMEA; b) coordenadas DD.

A seguir será mostrado como se deu a recuperação das mensagens NMEA e das coordenadas DD em cada um dos aparelhos e como foi possível, em alguns cenários, reconstruir parcialmente a trajetória dos dispositivos móveis. Os resultados foram analisados com o auxílio do software *Google Earth* (*Google Earth*, 2016). Os arquivos GPX gerados pela ferramenta de

extração e análise foram importados no *Google Earth*, viabilizando a plotagem das coordenadas GPS.

5.2.1.1. Sony LT22i

Neste aparelho, a trajetória percorrida pelo dispositivo pôde ser parcialmente recuperada. No atual cenário de testes (com os serviços de localização habilitados), foi possível observar que a quantidade de mensagens NMEA recuperadas não diminuiu com o passar do tempo, entretanto, as coordenadas geodésicas tendiam a se agrupar na posição de repouso final do veículo. A Figura 5.3 mostra a evolução dos dados recuperados com o passar do tempo.

No primeiro *dump*, que foi realizado logo após o veículo parar, foram recuperadas coordenadas ao longo de todo o percurso. Após 5 minutos, a quantidade de informação sobre o trajeto começou a diminuir, entretanto ainda foram encontradas mensagens NMEA com informações de velocidade e tempo ao longo do trajeto.

Entre 15 e 25 minutos - item (d) da Figura 5.3 -, a quantidade de coordenadas recuperadas caiu drasticamente, todavia ainda foram encontradas mensagens NMEA com informações de posicionamento a aproximadamente 470 metros da posição de repouso final do veículo. Por fim, aos 30 minutos, todas a mensagens se agrupavam na posição de repouso final do veículo.

Na abordagem de recuperação de coordenadas DD, somente o final do trajeto percorrido pelo dispositivo pôde ser parcialmente recuperado, conforme ilustrado na Figura 5.4.

No experimento, a quantidade de coordenadas DD recuperadas ao longo do trajeto diminuiu com o passar do tempo. Entretanto, os dados recuperados permitiram reconstruir parcialmente a trajetória do dispositivo no final do percurso.

No item (d) da Figura 5.4, é possível observar uma nova coordenada (-15.791, -47.898186) fora do traçado da pista. Na verdade, a discrepância se deu devido à região de memória associada a latitude estar corrompida, implicando na recuperação parcial dos dados de posicionamento.

Outro processo estava fazendo uso da área de memória que outrora fora utilizada pela API de localização. Desta forma, os dados foram plotados em uma região inesperada do mapa. Na Figura 5.5 é possível observar uma comparação entre a região de memória corrompida com outra região onde os dados foram corretamente recuperados.

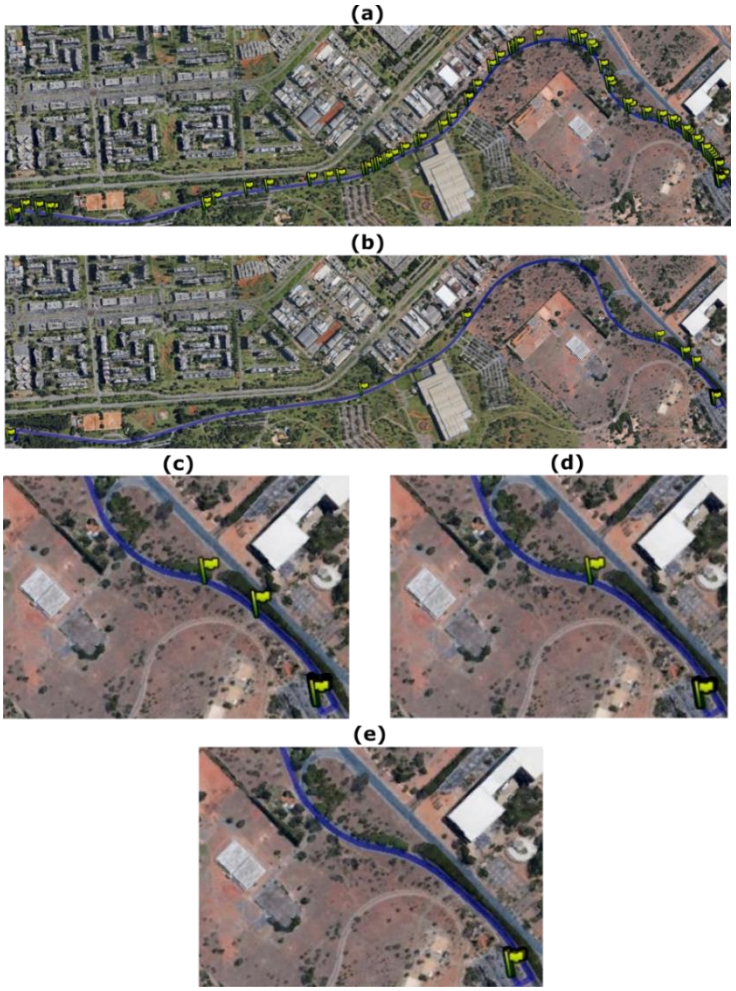


Figura 5.3: Coordenadas recuperadas: (a) instante inicial; (b) 5 minutos; (c) 10 minutos; (d) 15, 20 e 25 minutos (e) 30 minutos.

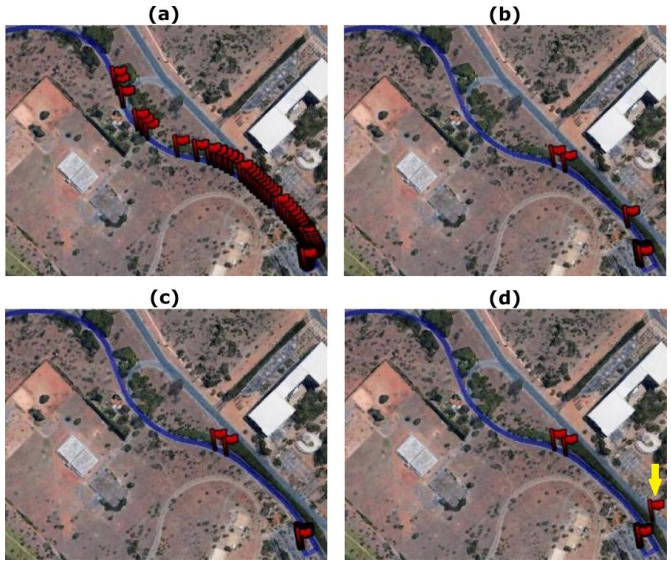


Figura 5.4: Coordenadas recuperadas: (a) instante inicial; (b) 5 minutos; (c) 10 minutos e (d) 15, 20, 25 e 30 minutos.

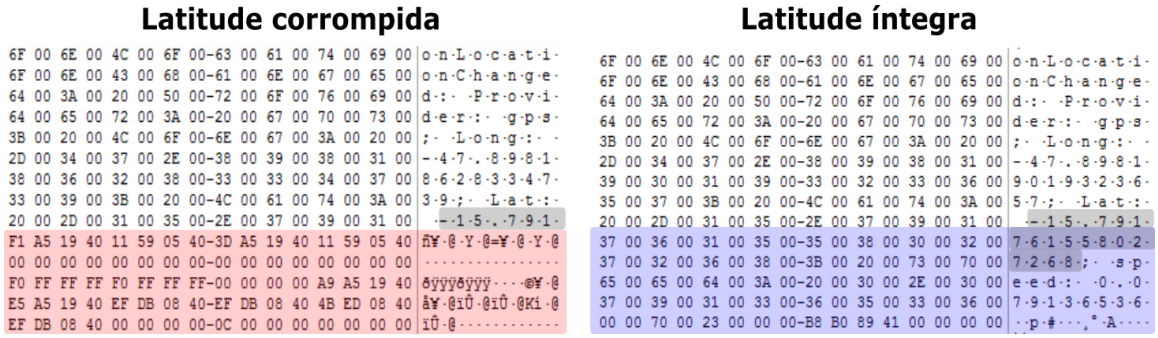


Figura 5.5: Comparação entre uma região de memória “suja” e uma região íntegra.

5.2.1.2. Sony Z2

Como visto no gráfico da Figura 5.2, este aparelho apresentou um comportamento diferente dos outros aparelhos examinados, com uma alta taxa de mensagens recuperadas nos primeiros *dumps*, seguido uma queda abrupta no total de mensagens recuperadas nos *dumps* posteriores. Isto ocorreu devido ao fato de o aparelho possuir a versão Lollipop do Android, que utiliza o ambiente de execução *Android Runtime* (ART) ao invés do seu predecessor, *Dalvik Virtual Machine* (DVM). O sistema de gerenciamento de memória do ART foi aprimorado, especialmente no que diz respeito a liberação de memória e *garbage collection* (*Debugging ART Garbage Collection*, 2014). Além disso, o aparelho também apresentava como diferencial a capacidade de memória RAM (aproximadamente 3 Gbytes) e processamento.

No cenário de testes com os serviços de localização habilitados, o aparelho apresentou comportamento parecido com o do aparelho Sony LT22i, exceto pelo fato de os dados estarem concentrados no final do trajeto. No primeiro *dump*, foram recuperadas diversas coordenadas, sendo possível reconstruir a trajetória do dispositivo ao final do percurso. Aos 10 minutos, a quantidade de informação recuperada diminui, entretanto, os dados ainda permitiram reconstruir parcialmente a trajetória. Aos 20 e 30 minutos os dados se concentraram na posição de repouso. A Figura 5.6 ilustra a perda de informação sobre o trajeto, como passar do tempo.

Em relação às coordenadas DD, todos os dados recuperados estavam associados à posição de repouso final do veículo, conforme ilustrado na Figura 5.7. Diferentemente do que aconteceu na recuperação de mensagens NMEA, sequer o primeiro *dump* apresentou informações sobre o trajeto. Portanto, não foi possível reconstruir a trajetória do veículo.

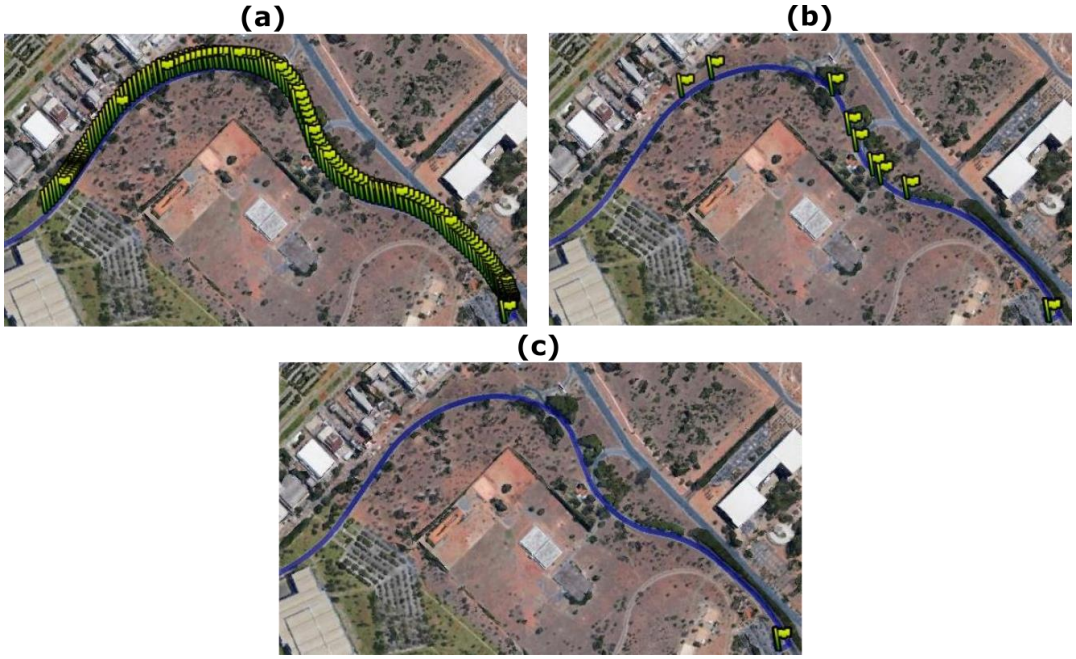


Figura 5.6: Coordenadas recuperadas: (a) instante inicial; (b) 10 minutos; (c) 20 e 30 minutos.

Figura 5.7: Coordenadas agrupadas na posição de repouso do veículo.

5.2.1.3. Samsung GT-P5200

Nos testes com o *tablet* Samsung GT-P5200, o qual possui arquitetura distinta dos outros dispositivos, foi possível reconstruir parcialmente a trajetória final do veículo, com as mensagens NMEA recuperadas.

Assim como no caso do *smartphone* Sony LT22i, a quantidade de mensagens recuperadas não diminuiu com o passar do tempo, porém as coordenadas geográficas tendiam a se concentrar na posição de repouso final do veículo, como ilustrado na Figura 5.8. As mensagens NMEA

foram gradativamente sendo perdidas, entretanto, mesmo depois de transcorridos 30 minutos, algumas mensagens com informações relevantes sobre o trajeto foram recuperadas.

Na análise com coordenadas DD, todas as informações de posicionamento recuperadas estavam associadas a posição de repouso final do veículo, inviabilizando a reconstrução da trajetória do dispositivo.

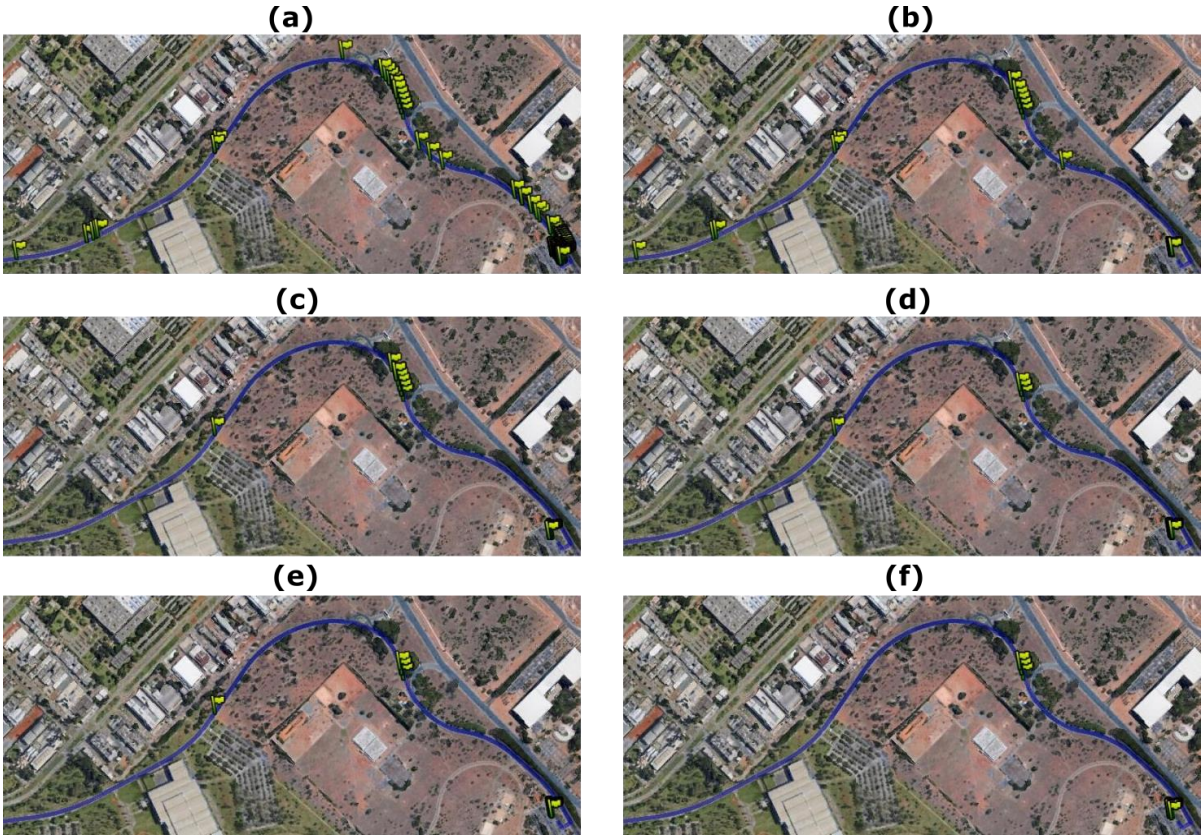


Figura 5.8: Coordenadas recuperadas: (a) instante inicial; (b) 5 minutos; (c) 10 minutos; (d) 15 e 20 minutos; (e) 25 minutos e (f) 30 minutos.

5.2.1.4. Samsung SM-G3812B

No experimento realizado, quase a totalidade das mensagens NMEA recuperadas se concentraram na posição de repouso final do veículo, com exceção de algumas poucas mensagens NMEA, que apontavam para uma região a aproximadamente 400 metros do destino final, conforme ilustrado na Figura 5.9. A mensagem, que era do tipo GPRMC, pode ser visualizada no trecho a seguir:

$GPRMC,134101.000,A,1547.435081,S,04754.045440,W,30.3,95.3,270216,,,A*67 - Speed in Km/h: 56.11

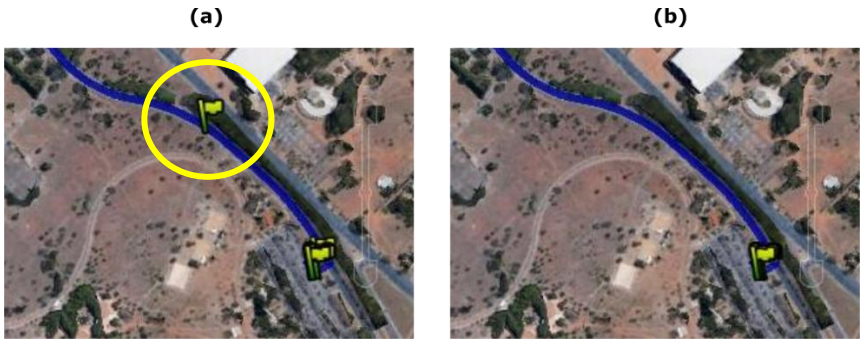


Figura 5.9: Coordenadas recuperadas: (a) instante inicial e (b) 5 a 30 minutos.

Na abordagem utilizando coordenadas DD, também não foi possível reconstruir a trajetória do dispositivo. As coordenadas geodésicas recuperadas se concentraram na posição de repouso final do veículo, com exceção de um ponto, conforme indicado no item (a) da Figura 5.10.

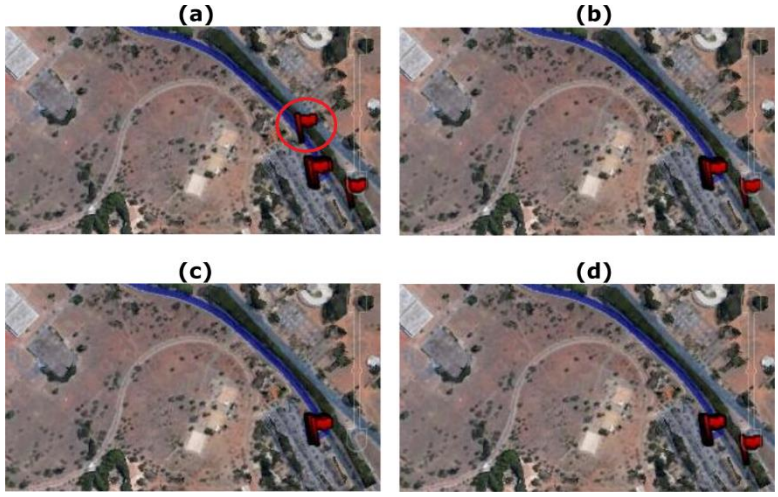


Figura 5.10: Coordenadas recuperadas: (a) instante inicial e 5 minutos; (b) 10 minutos; (c) 15 e 20 minutos e (d) 25 e 30 minutos.

Entretanto, as coordenadas DD encontradas fora da posição de repouso final do veículo não apresentavam informações de velocidade, tempo, número de satélites, nem qualquer outro dado que pudesse ajudar a associá-lo ao último trajeto do veículo. As informações encontradas, ao contrário da maioria dos dados associados a API de localização, estavam codificadas em ASCII e associadas aos mecanismos de comunicação entre processos (IPC - *Inter-Process Communication*), conforme ilustrado na Figura 5.11.

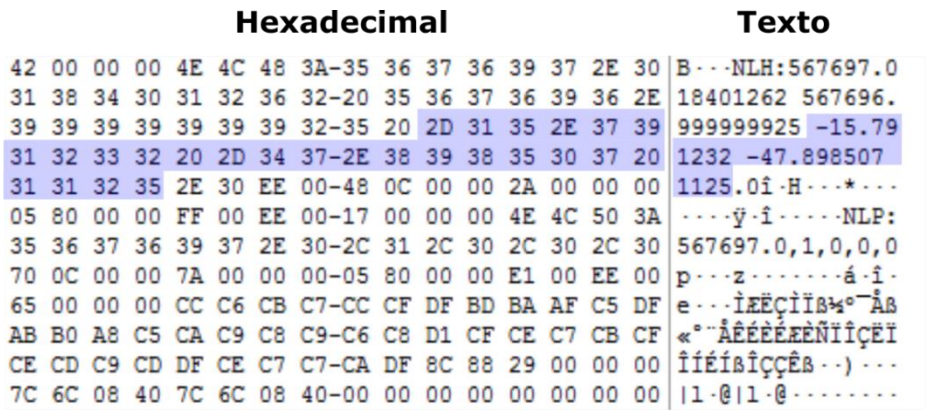


Figura 5.11: Coordenadas codificadas em ASCII - IPC

5.2.2. Resultados com os serviços de localização desabilitados

As Tabelas 5.4 e 5.5 mostram a quantidade de coordenadas recuperadas de acordo com o tempo em que o *dump* foi realizado, nos experimentos com os serviços de localização desabilitados. Na Tabela 5.4 é mostrado o quantitativo de mensagens NMEA recuperadas, ao passo que na Tabela 5.5 os dados são relativos às coordenadas DD.

Tabela 5.4 – Quantitativo de mensagens NMEA recuperadas por *dump*, com os serviços de localização desabilitados.

Dispositivo	**Tempo transcorrido em minutos**						
	0	**5**	**10**	**15**	**20**	**25**	**30**
Sony LT22i	*214*	*144*	*114*	*93*	*75*	*69*	*71*
Sony Z2	*743*	*N/A*	*299*	*N/A*	*131*	*N/A*	*113*
Samsung GT-P5200	*307*	*87*	*65*	*3*	*3*	*2*	*2*
Samsung SM-G3812B	*86*	*46*	*26*	*24*	*22*	*12*	*8*

Tabela 5.5– Quantitativo de coordenadas DD recuperadas por *dump*, com os serviços de localização desabilitados.

Dispositivo	**Tempo transcorrido em minutos**						
	0	**5**	**10**	**15**	**20**	**25**	**30**
Sony LT22i	*170*	*106*	*29*	*5*	*23*	*9*	*6*
Sony Z2	*882*	*N/A*	*536*	*N/A*	*214*	*N/A*	*231*
Samsung GT-P5200	*340*	*1*	*27*	*17*	*17*	*17*	*18*
Samsung SM-G3812B	*184*	*193*	*177*	*173*	*161*	*139*	*147*

Neste cenário, a quantidade de mensagens recuperadas reduz com o passar do tempo, como é possível visualizar nos gráficos da Figura 5.12. Entretanto, em alguns dispositivos, as coordenadas de posicionamento recuperadas forneceram mais informações sobre o percurso completo, mesmo depois de transcorrido 30 minutos, se comparadas com aquelas recuperadas nos testes com os serviços de localização habilitados.

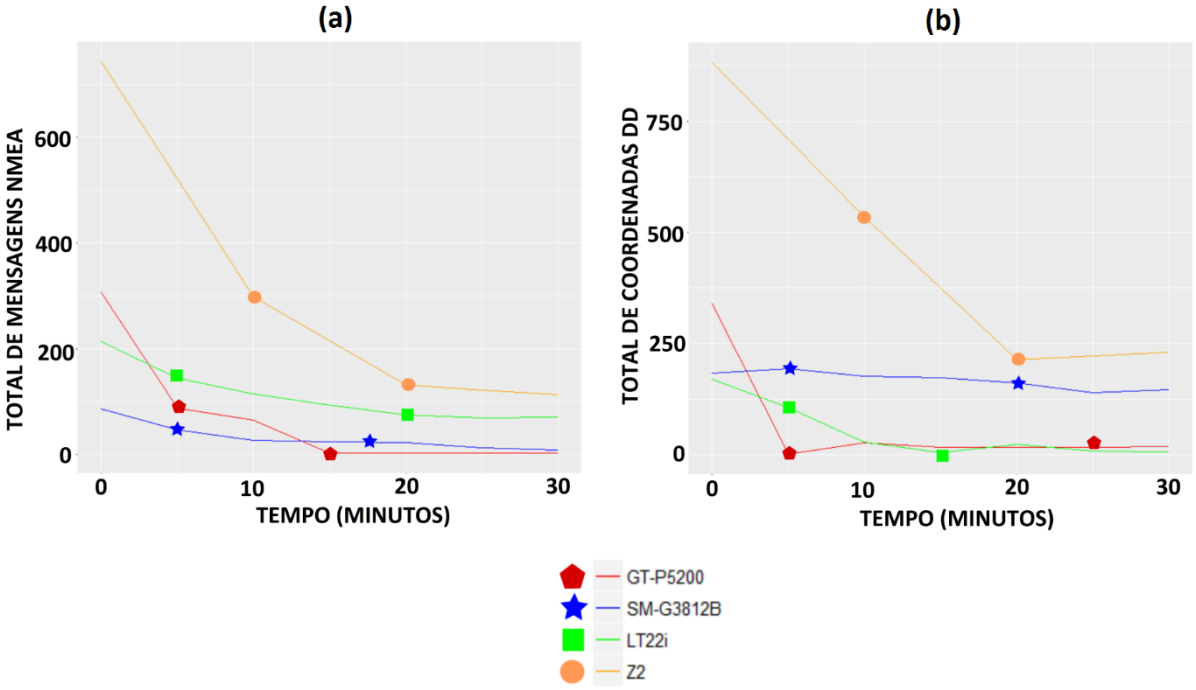


Figura 5.12: Total de dados recuperados com os serviços de localização desabilitados: a) mensagens NMEA; b) coordenadas DD.

A seguir será feita uma análise dos dados recuperados para cada dispositivo.

5.2.2.1. Sony LT22i

Neste cenário de testes, a quantidade de mensagens NMEA recuperadas diminuiu com o passar do tempo, entretanto, os dados recuperados permitiram reconstruir a trajetória com mais precisão, quando comparadas com os experimentos com os serviços de localização habilitados, mesmo depois de transcorridos 30 minutos, como pode ser visualizado no Figura 5.13.

Poucos dados foram perdidos, visto que o *garbage collector* não liberou totalmente as páginas de memória usadas pelo aplicativo, reduzindo consideravelmente a quantidade de memória sobrescrita.

Na abordagem de recuperação de coordenadas DD, a quantidade de informação ao longo do trajeto diminuiu com o passar do tempo, conforme ilustrado na Figura 5.14. Entretanto, os dados recuperados permitiram reconstruir parcialmente o final da trajetória do dispositivo, com exceção dos dados recuperados do último *dump* (30 minutos), que somente forneciam informações sobre a posição de repouso do veículo.

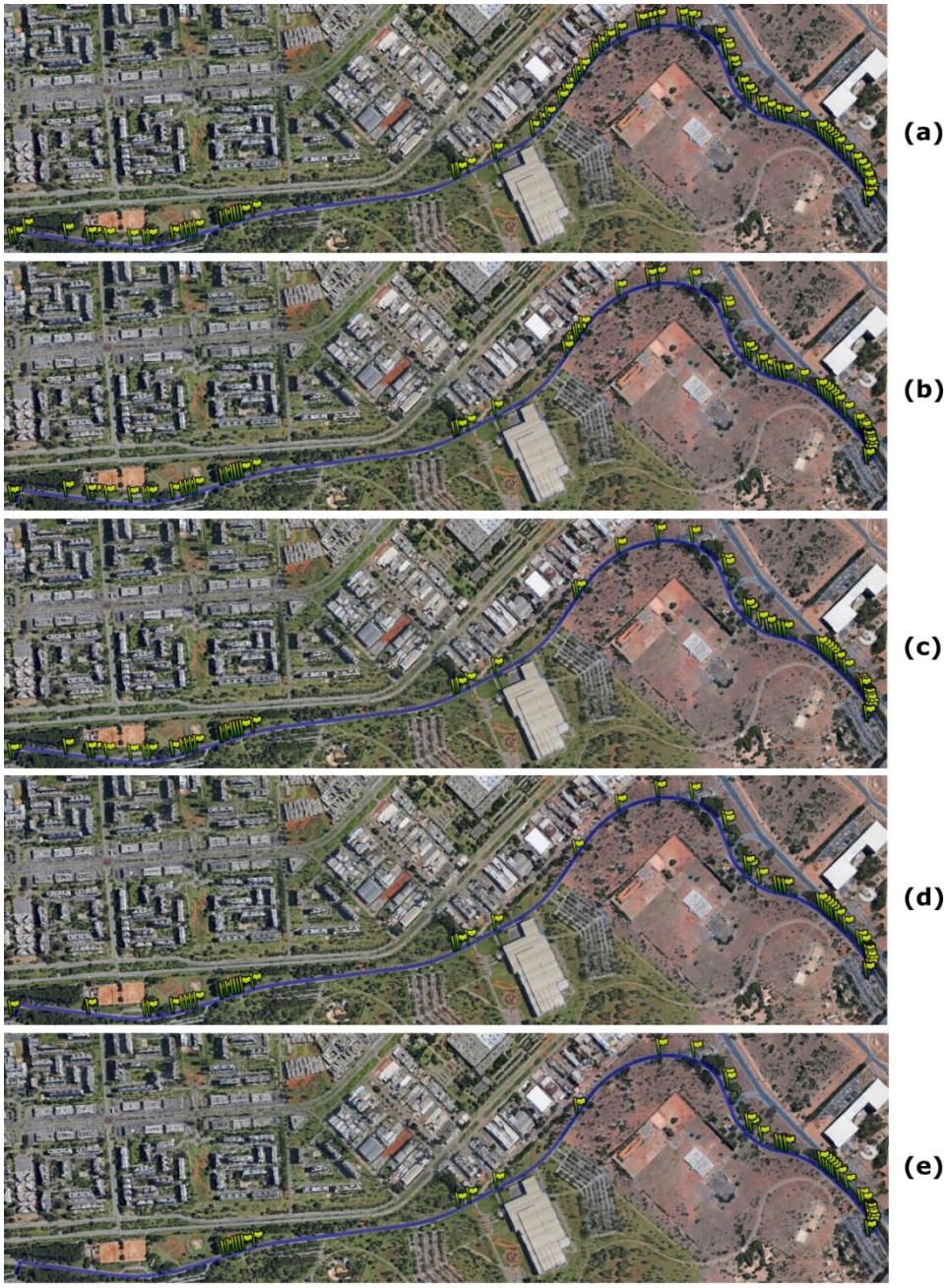


Figura 5.13: Coordenadas recuperadas: (a) instante inicial; (b) 5 minutos; (c) 10 minutos; (d) 15 minutos e (e) 20 a 30 minutos.

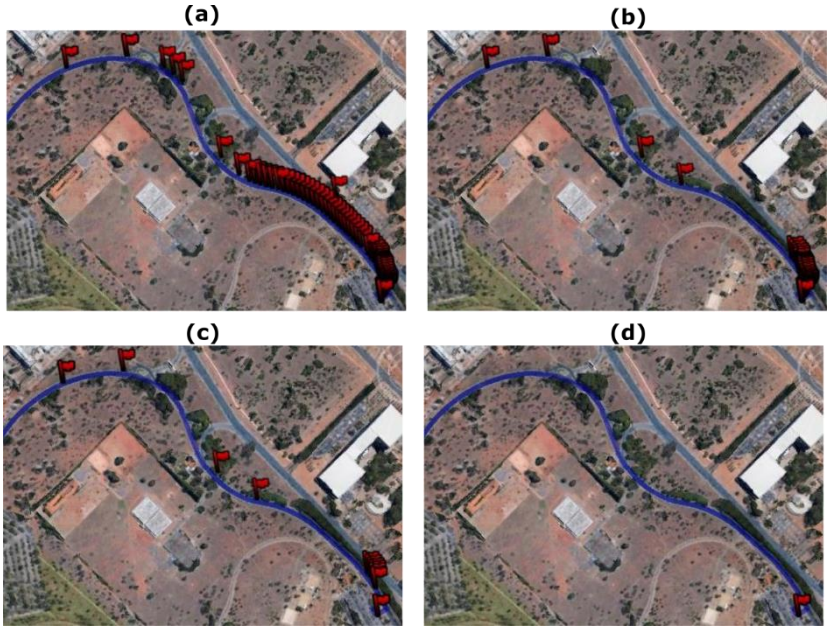


Figura 5.14: Coordenadas recuperadas: (a) instante inicial; (b) 5 minutos; (c) 10 minutos e (d) 15, 20, 25 e 30 minutos.

5.2.2.2. Sony Z2

As mensagens NMEA recuperadas permitiram reconstruir parcialmente a trajetória do dispositivo mesmo depois de transcorridos 30 minutos, como pode ser visto na Figura 5.15. Pouca informação sobre o trajeto se perdeu. Mesmo com uma queda de 743 para 113 mensagens recuperadas, diversos pontos do trajeto ainda puderam ser plotados.

Na abordagem de recuperação das coordenadas DD, os dados recuperados permitiram reconstruir parcialmente a trajetória do dispositivo mesmo depois de transcorridos 30 minutos, como pode ser visto na Figura 5.16.

É possível observar a existência de coordenadas que foram plotadas fora do traçado da pista, no item (a) da Figura 5.16. Estas coordenadas estavam associadas a dados que são enviados periodicamente para servidores do aplicativo *Waze*, com informações do mapa que está sendo visualizado.

As primeiras versões do aplicativo *Waze* eram distribuídas sob a Licença Pública Geral GNU v2 e utilizavam códigos *open source* do projeto *RoadMap*. O código fonte de versões antigas do *Waze* pode ser encontrado em sua *wiki*[5].

[5] https://wiki.waze.com/wiki/Source_code - acessado em 15 jan. 2016

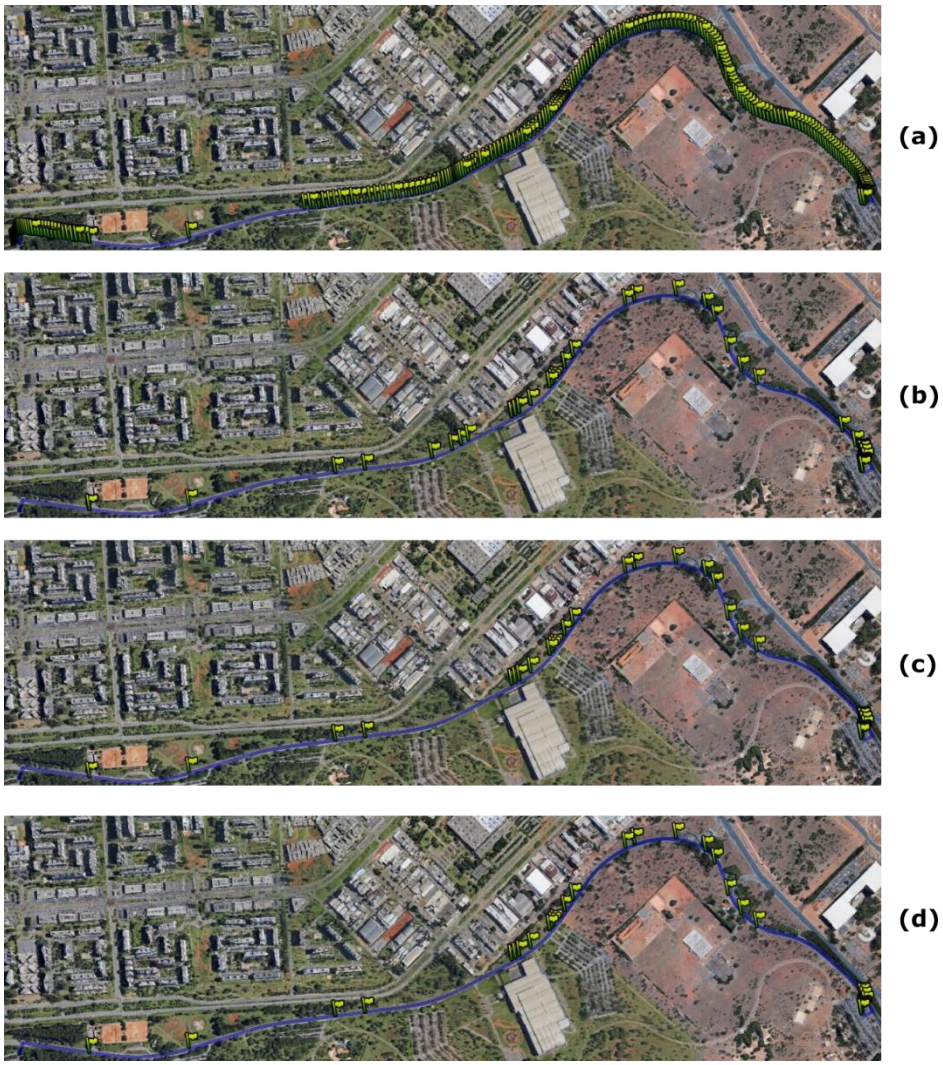


Figura 5.15: Coordenadas recuperadas: (a) instante inicial; (b) 10 minutos; (c) 20 minutos e (d) 30 minutos.

Nos arquivos *RealtimeNet.c*, *RealtimeNet.h* e *RealTimeNetDefs.h* são encontradas referências às funções e definições dos dados que são enviados para os servidores. Em especial, a macro *RTNET_FORMAT_NETPACKET_2MapDisplayed* define o formato da mensagem com as informações do mapa visualizado, as quais são compatíveis com os dados recuperados da memória, conforme ilustrado na Figura 5.17.

Também é possível observar o envio das mensagens utilizando-se o modo *Debug*, conforme visualizado no trecho a seguir:

[14:41:24 Debug] Realtime_SendCurrentViewDimentions() - Sending 'MapDisplayed'... [Realtime.c:2313 (Realtime_SendCurrentViewDimentions)]

...

[14:41:25 Debug] OnAsyncOperationCompleted_MapDisplayed__only() - 'MapDisplayed' was sent successfully [Realtime.c:2268 (OnAsyncOperationCompleted_MapDisplayed__only)]

Em suma, nem todas as coordenadas recuperadas são relativas ao trajeto percorrido pelo dispositivo. Muitas informações fazem parte do contexto do aplicativo que está sendo utilizado, e podem não ser úteis para a reconstrução da trajetória do dispositivo. O examinador deverá estar atento a estas questões para que não incorra em erros na análise pericial.

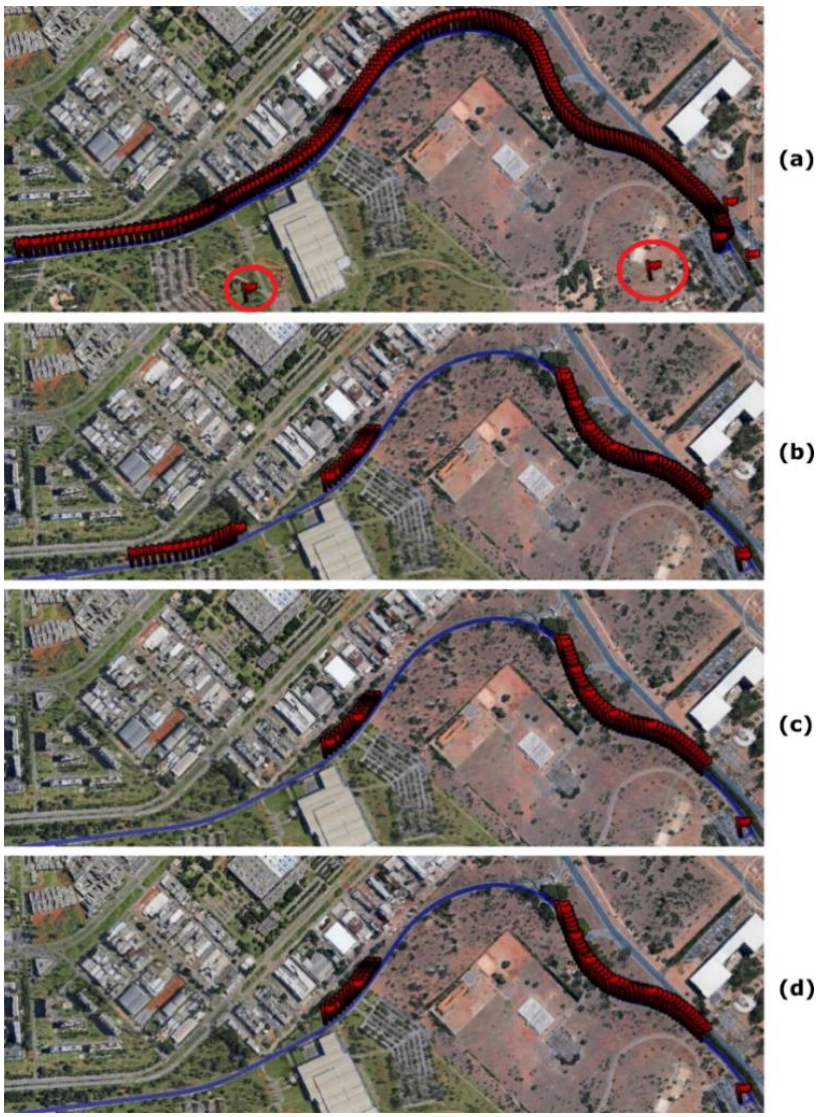


Figura 5.16: Coordenadas recuperadas: (a) instante inicial; (b) 10 minutos; (c) 20 minutos e (d) 30 minutos.

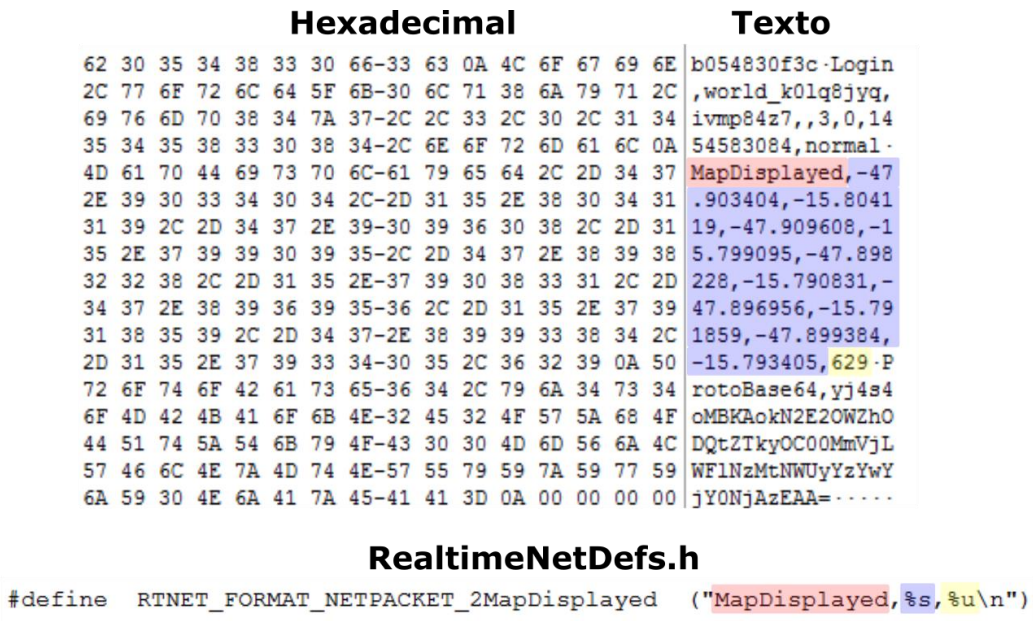


Figura 5.17: Assimilação entre o conteúdo recuperado no *dump* de memória e as definições no código fonte do projeto *Roadmap*.

5.2.2.3. Samsung GT-P5200

No cenário de testes com os serviços de localização desabilitados, durante os 10 primeiros minutos, uma grande quantidade de coordenadas foi recuperada, permitindo reconstruir parcialmente a trajetória do veículo.

Contudo, a partir dos 15 minutos, as informações se perderam. Como mostrado na Tabela 5.4, foram recuperadas 63 mensagens NMEA do *dump* ocorrido aos 10 minutos, enquanto que do *dump* realizado aos 15 minutos, somente foram recuperadas 3 mensagens. A Figura 5.18 mostra a queda abrupta na quantidade de informação recuperada depois dos 15 minutos.

Neste cenário, também não foi possível reconstruir a trajetória do veículo com base somente nas coordenadas DD. Todas as informações de posicionamento recuperadas estavam associadas a posição de repouso final do veículo.

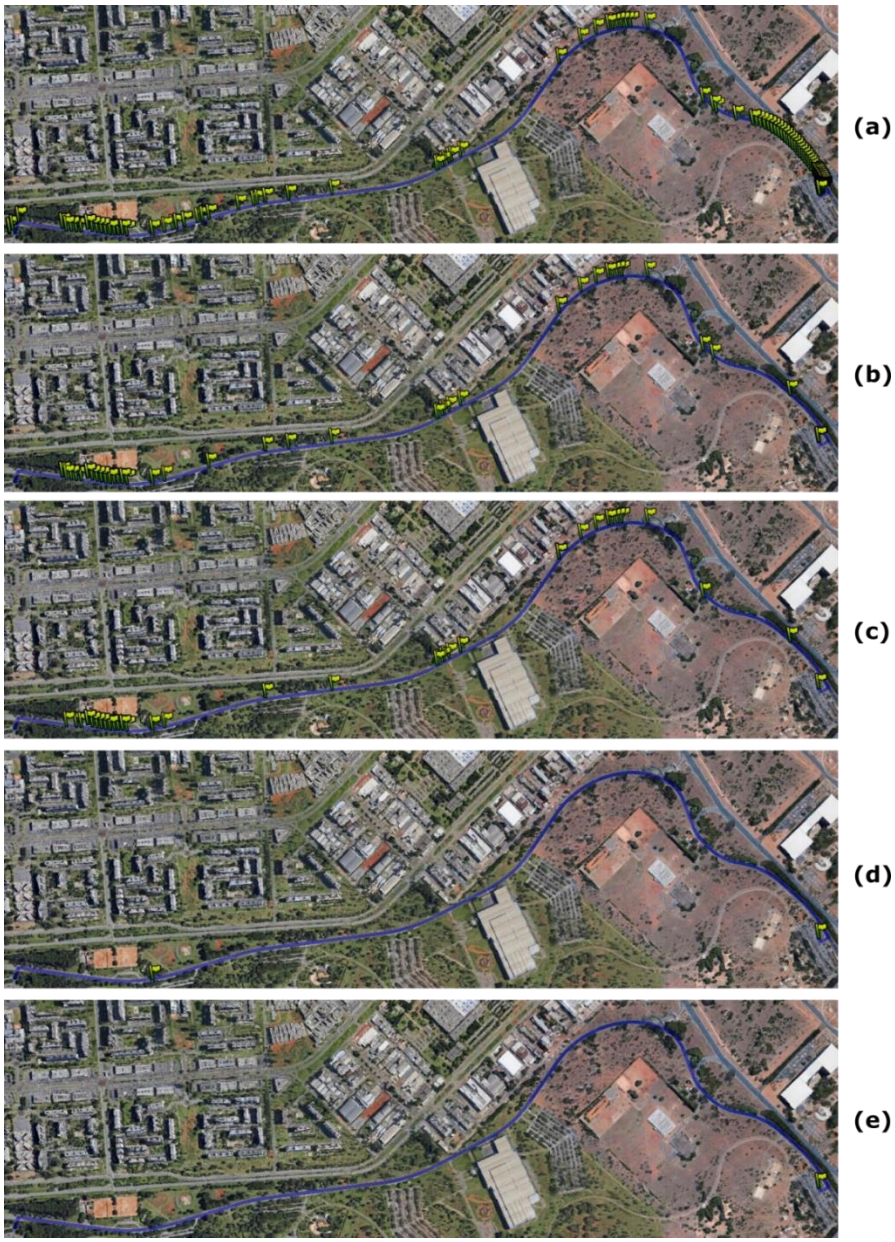


Figura 5.18: Coordenadas recuperadas: (a) instante inicial; (b) 5 minutos; (c) 10 minutos; (d) 15 e 20 minutos; (e) 25 e 30 minutos.

5.2.2.4. Samsung SM-G3812B

No cenário de testes com os serviços de localização desabilitados, foram recuperados dados alusivos à região final do percurso, conforme ilustrado na Figura 5.19. Os dados recuperados mostraram um comportamento diferenciado do aparelho aos 20, 25 e 30 minutos. Esperava-se que a mensagens ao longo do percurso, com o passar do tempo, fossem removidas permanentemente da memória.

Aos 20 minutos, é possível visualizar na Figura 5.19 a presença das coordenadas "-15.789462933333333, -47.90742388333334". Entretanto, aos 25 minutos a coordenada não foi plotada, sendo novamente encontrada aos 30 minutos. Tal comportamento se deu devido aos mecanismos de gerenciamento de memória e troca de contexto dos processos, no sistema Android.

A mesma página da memória RAM, com o tamanho de 4096 bytes, foi encontrada nos *dumps* realizados aos 20 e 30 minutos, todavia em regiões de memória diferentes, conforme ilustrado na Figura 5.20.

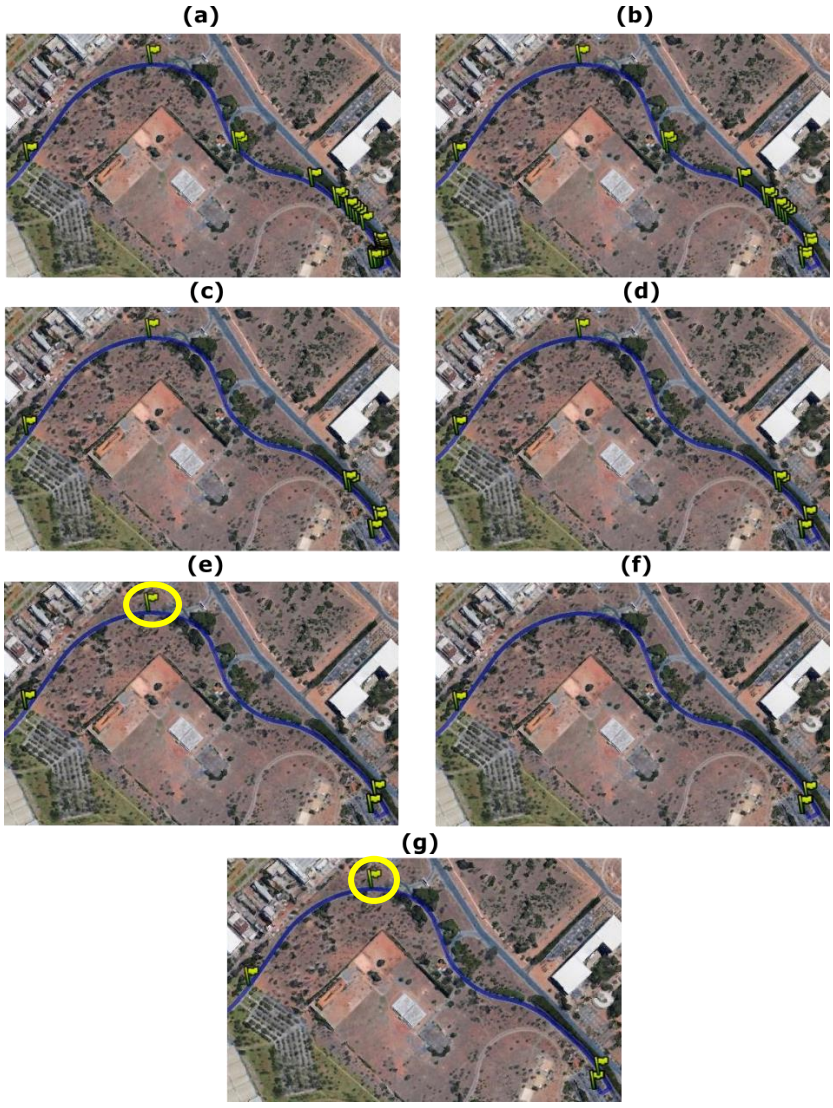


Figura 5.19: Coordenadas recuperadas: (a) instante inicial; (b) 5 minutos; (c) 10 minutos; (d) 15 minutos; (e) 20 minutos; (f) 25 minutos e (g) 30 minutos.

Figura 5.20: Mesma página de memória encontrada em *dumps* distintos, porém em regiões de memória diferentes.

Na abordagem de recuperação de coordenadas DD, as coordenadas geodésicas recuperadas se concentraram na posição de repouso final do veículo, com exceção de um ponto encontrado algumas dezenas de metros antes da posição de repouso final, conforme ilustrado na Figura 5.21.

Figura 5.21: Coordenadas recuperadas: instante inicial a 30 minutos.

5.3. EXPERIMENTO EM TRAJETOS MÉDIOS (15 KM)

Em ambientes urbanos, ferramentas colaborativas, como o *Waze*, são muito utilizadas por condutores de veículos para pesquisar trajetos e condições de tráfego. Não obstante o uso destes tipos de aplicativos ser muito útil para os usuários, é muito comum a ocorrência de acidentes de trânsito em virtude do manuseio de *smartphones* concomitantemente com a direção do veículo.

Para simular casos deste tipo, foi percorrido um trajeto de 15 km. Os dispositivos Samsung SM-G3812B e Sony Z2 foram testados em condições de clima desfavoráveis para o uso de GPS, com chuva intensa e nuvens carregadas. Assim como no caso anterior, o veículo foi parado e foi iniciada a recuperação dos dados da memória.

Foram utilizados os mesmos dispositivos descritos na Tabela 5.1. O trajeto percorrido abarcou três vias do Distrito Federal (DF-075, DF-003 e DF-085) e pode ser visualizado na Figura 5.22.

Figura 5.22: Trajeto de média distância

Neste cenário os serviços de localização não foram desligados, simulando casos de acidentes de trânsito, onde o tempo de reposta das equipes periciais pode ser superior a uma hora.

Foram realizados *dumps* da memória dos aparelhos no período de uma hora. Nos primeiros 30 minutos, uma imagem era produzida a cada dez minutos. Depois deste período, foi feito um último *dump* depois de passados 60 minutos do veículo em repouso.

As imagens das memórias foram então submetidas ao processamento pela ferramenta desenvolvida. Nos testes em trajetos médios, para a abordagem de recuperação de coordenadas

DD, foi utilizado o par de coordenadas -15.845186, -47.974762 como referência, com uma abrangência de 10 km e um limite de 100 bytes para busca.

A seguir serão descritos os resultados, primeiramente de maneira geral, e logo em seguida s as peculiaridades de cada dispositivo.

5.3.1. Resultados

As Tabelas 5.6 e 5.7 mostram, respectivamente, a quantidade de mensagens NMEA e a quantidades de coordenadas DD recuperadas de acordo com o tempo em que o *dump* foi realizado.

Tabela 5.6 – Quantitativo de mensagens NMEA recuperadas por dump (15 km).

Dispositivo	**Tempo transcorrido em minutos**				
	0	**10**	**20**	**30**	**60**
Sony LT22i	*200*	*363*	*177*	*333*	*272*
Sony Z2	*294*	*985*	*234*	*180*	*228*
Samsung GT-P5200	*177*	*243*	*337*	*315*	*330*
Samsung SM-G3812B	*11*	*293*	*336*	*303*	*377*

Tabela 5.7 – Quantitativo de coordenadas DD recuperadas por *dump* (15 km).

Dispositivo	**Tempo transcorrido em minutos**				
	0	**10**	**20**	**30**	**60**
Sony LT22i	*234*	*246*	*230*	*211*	*228*
Sony Z2	*836*	*935*	*3348*	*2560*	*850*
Samsung GT-P5200	*365*	*2215*	*2387*	*1941*	*910*
Samsung SM-G3812B	*45*	*545*	*628*	*604*	*464*

Em relação a abordagem visando a recuperação de mensagens NMEA, é possível observar no gráfico do item (a) da Figura 5.23 que a quantidade de mensagens recuperadas é variável, assim como no caso dos experimentos em trajeto curto, uma vez que os serviços de localização permaneceram habilitados.

Na abordagem de recuperação das coordenadas DD, com exceção do *smartphone* Sony LT22i, que não teve grande variabilidade na quantidade de dados recuperados, todos os aparelhos tiveram um crescimento no número de coordenadas recuperadas nos primeiros 20 minutos, seguido de uma redução nos *dumps* realizados aos 30 e 60 minutos, como pode ser observado no gráfico do item (b) da Figura 5.23. Apesar da grande quantidade de coordenadas recuperadas de alguns *dumps*, boa parte dos dados se concentravam na posição de repouso final do veículo.

Além disso, o decréscimo na quantidade de dados recuperados após os 20 minutos é normal, uma vez que os aparelhos utilizam mecanismos de economia de energia.

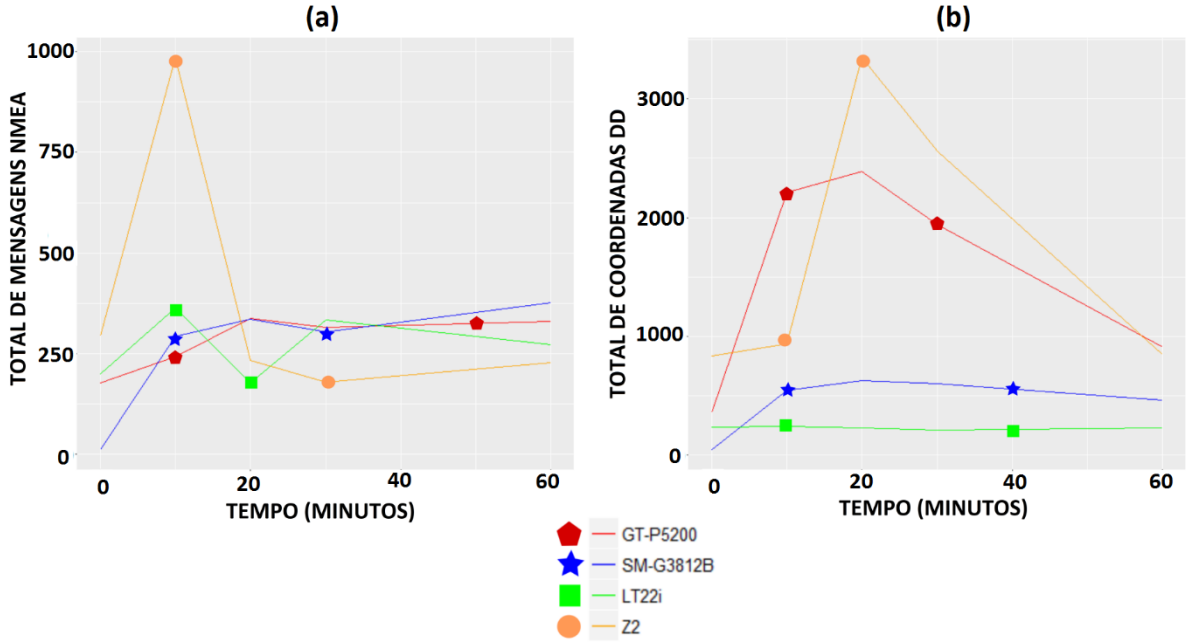


Figura 5.23: Total de dados recuperados: a) mensagens NMEA; b) coordenadas DD.

Importante observar que, mesmo depois de transcorrido o intervalo de 60 minutos após o veículo entrar em repouso, foi possível, em alguns dispositivos, recuperar informações do trajeto percorrido, bem como dados de velocidade, que podem ser muito importantes em investigações de acidentes de trânsito.

A seguir serão mostrados os resultados, como se deu a recuperação das mensagens em cada um dos dispositivos e como os fatores climáticos podem influenciar na precisão dos dados.

5.3.1.1. Sony LT22i

Neste aparelho, na abordagem de recuperação de mensagens NMEA, a trajetória percorrida pelo dispositivo pôde ser parcialmente recuperada, conforme ilustrado na Figura 5.24. Nos primeiros 10 minutos, foi possível recuperar boa parte do final do trajeto percorrido. Com o passar do tempo, as informações foram sendo perdidas, entretanto, mesmo depois de transcorridos 60 minutos, algumas mensagens NMEA ao longo do trajeto ainda foram recuperadas, conforme mostrado no trecho de log a seguir, reportado pela ferramenta de extração e análise:

*$GPRMC,214029.000,A,1552.630832,S,04800.033300,W,34.325,265.1,180216,,,A*52 - Speed in Km/h: 63.56*

*$GPRMC,214033.000,A,1552.633845,S,04800.071033,W,32.546,264.4,180216,,,A*5E - Speed in Km/h: 60.27*

Na abordagem de recuperação de coordenadas DD, os dados recuperados permitiram reconstruir parcialmente a trajetória do veículo, em especial a região central do percurso, conforme ilustrado na Figura 5.25.

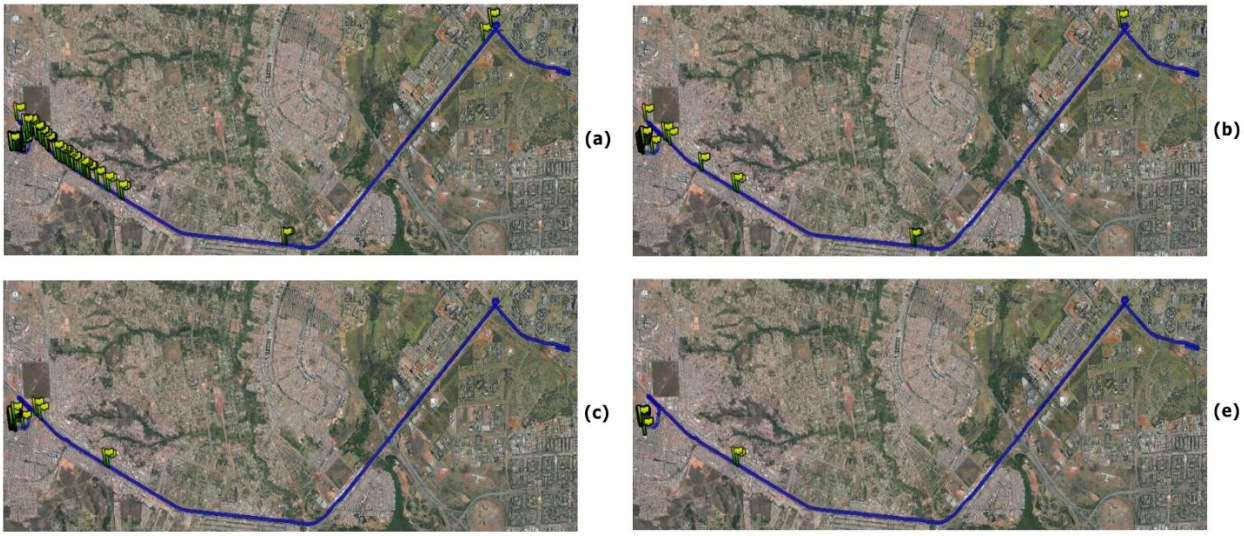


Figura 5.24: Coordenadas recuperadas: (a) instante inicial; (b) 10 minutos; (c) 20 e 30 minutos e (d) 60 minutos.

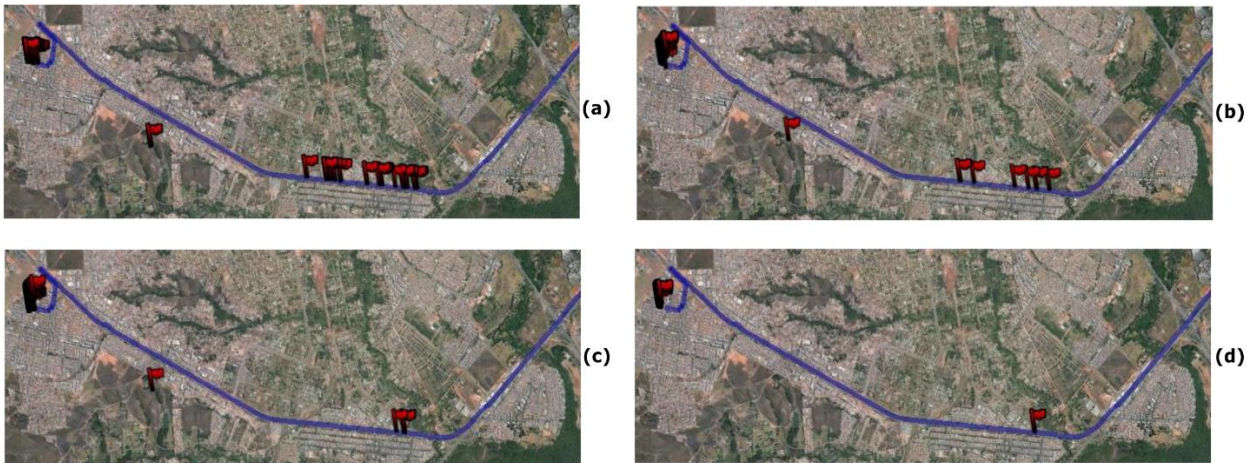


Figura 5.25: Coordenadas recuperadas: (a) instante inicial; (b) 10 minutos; (c) 20 e 30 minutos e (d) 60 minutos.

A quantidade de coordenadas geodésicas recuperadas ao longo do trajeto diminuiu com o passar do tempo, entretanto, mesmo depois de transcorridos 60 minutos, foi possível encontrar um par de coordenadas a aproximadamente 6,4 km da posição de repouso do veículo.

Também é possível observar que algumas coordenadas recuperadas estavam fora do traçado da pista. Assim em como em alguns experimentos em trajeto curto, parte da memória estava corrompida, implicando em latitudes e longitudes incorretas.

5.3.1.2. Sony Z2

Ao contrário do que aconteceu nos experimentos de trajeto curto, neste cenário todas as mensagens NMEA recuperadas estavam associadas com a posição de repouso final do veículo.

Na abordagem de recuperação de coordenadas DD, os dados recuperados de todos os *dumps* de memória permitiram reconstruir quase a totalidade do trajeto percorrido pelo veículo. Todavia, em um determinado trecho, as coordenadas estavam associadas a uma via que não foi utilizada no cenário de testes, com pode ser observado na Figura 5.26. De fato, nos experimentos para recuperação de mensagens NMEA, todos os dados recuperados se agrupavam na posição de repouso final do veículo, contrapondo as coordenadas DD plotadas.

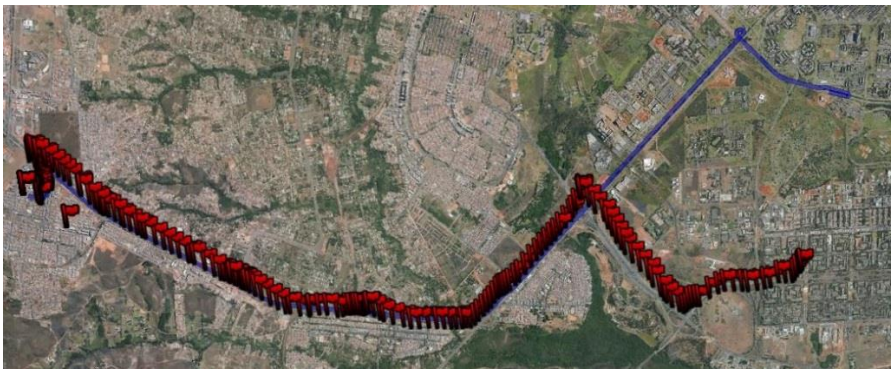


Figura 5.26: Coordenadas DD recuperadas do comando GPSPath.

Analisando os *dump*s de memória, foi possível constatar uma grande quantidade de coordenadas, codificadas em ASCII, que não apresentavam as características dos dados tipicamente encontrados nas regiões de memórias associadas às APIs de localização e aos aplicativos. Tais mensagens, conforme exposto no trabalho de (Jeske, 2013), fazem parte do protocolo do aplicativo *Waze*. Mais especificamente, trata-se de uma mensagem do tipo *GPSPath*, que contém informações sobre um caminho já percorrido pelo aparelho. Conforme ilustrado na Figura 5.27, a mensagem GPSPath estava associada à data de 06/02/2016, sendo que os testes foram realizados no dia 27/02/2016.

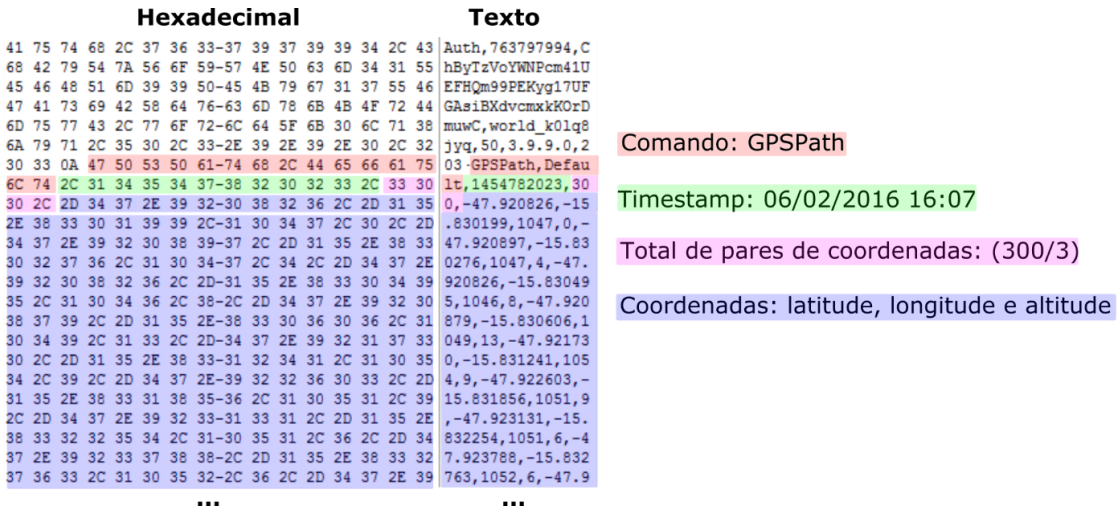


Figura 5.27: Mensagem *GPSPath*.

Removendo-se as coordenadas descritas, foi possível analisar os dados verdadeiramente associados ao último trajeto de veículo. Assim como aconteceu com a recuperação das mensagens NMEA, todos os dados se concentraram na posição do repouso final do veículo, com exceção de três pares de coordenadas, que estavam fora do traçado da pista, conforme ilustrado na Figura 5.28.

Estas coordenadas que se destacaram também não estavam associadas ao verdadeiro trajeto do veículo. Na verdade, elas faziam parte de dados da *engine* de localização, com dados do último posicionamento do dispositivo (*lastLocation*), obtido pelo provedor de localização do tipo *Network* na data de 26/02/2016, como mostra a Figura 5.29.

Em resumo, todos os dados recuperados relativos ao último trajeto do veículo apontavam para a posição de repouso final do veículo, assim como no caso das mensagens NMEA.

5.3.1.3. Samsung GT-P5200

Com a recuperação das mensagens NMEA, assim como no caso do Sony LT22i, a trajetória pôde ser parcialmente recuperada e a quantidade de mensagens diminuiu com o passar do tempo, conforme mostrado na Figura 5.30. Entretanto, neste dispositivo, foi possível recuperar uma quantidade maior de informação sobre o percurso, mesmo depois de transcorrida uma hora após o veículo entrar em repouso.

Figura 5.28: Pares de coordenadas fora do traçado da pista.

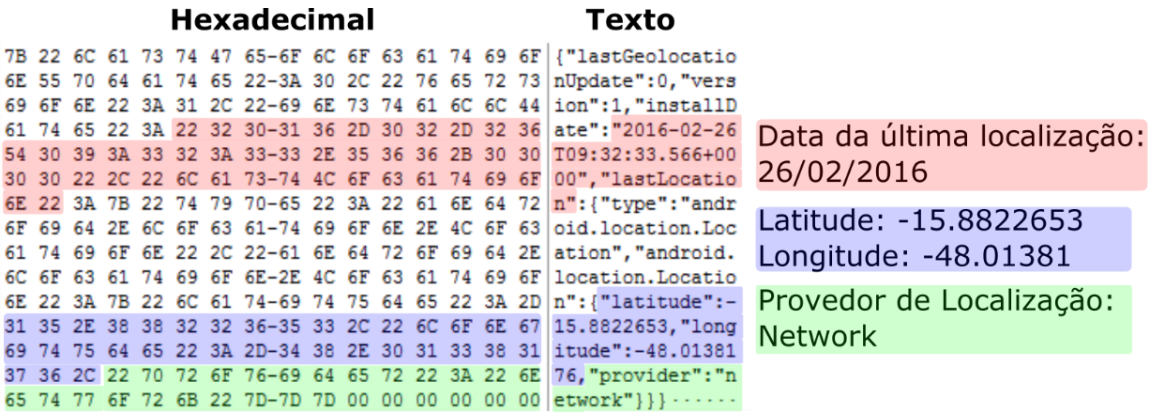


Figura 5.29: Dados da última localização obtida pelo *Network Provider*.

Importante observar que é possível construir uma *timeline* com as informações de tempo coletadas das mensagens GPRMC.

Segue um trecho do log com as mensagens NMEA associadas aos pares de coordenadas "-15.870947983333334, -47.97440975" e "-15.8220278, -47.95059725", que estão separados por aproximadamente 7,3 km:

*$GPRMC,211546.00,A,1549.321668,S,04757.035835,W,031.7,189.8,180216,,,A*5D - Speed in Km/h: 58.70*

*$GPRMC,213423.00,A,1552.256879,S,04758.464585,W,000.0,244.0,180216,,,A*5E - Speed in Km/h: 0.0*

As informações de tempo das mensagens NMEA descritas estão em UTC, no formato hora, minutos, segundos (*hhmmss*). Com estas informações, é possível observar que o trajeto de 7,3 km foi percorrido em 18 minutos e 36 segundos, indicando uma velocidade média de 23,5 km/h.

De fato, os experimentos foram realizados em um período de tráfego intenso naquela região (entre 19h15 e 19h34), implicando na redução da velocidade do veículo.

Em suma, ainda que o trajeto não possa ser completamente reconstruído com as informações coletadas da memória, é possível inferir dados sobre o comportamento do dispositivo com apenas poucas mensagens recuperadas.

Na abordagem de recuperação das coordenadas DD, foi possível recuperar dois grupos de coordenadas a aproximadamente 1,5 km e 3,5 km da posição de repouso final do veículo, mesmo depois de transcorridos 60 minutos, conforme ilustrado na Figura 5.31. No item (b) da referida figura, é possível observar um plotagem fora do traçado da pista. Assim como já mostrado em outros experimentos, trata-se de dados corrompidos, que possuem pequenos fragmentos de regiões tipicamente associadas aos *frameworks* dos serviços de localização. Os dados corrompidos podem ser visualizados na Figura 5.32.

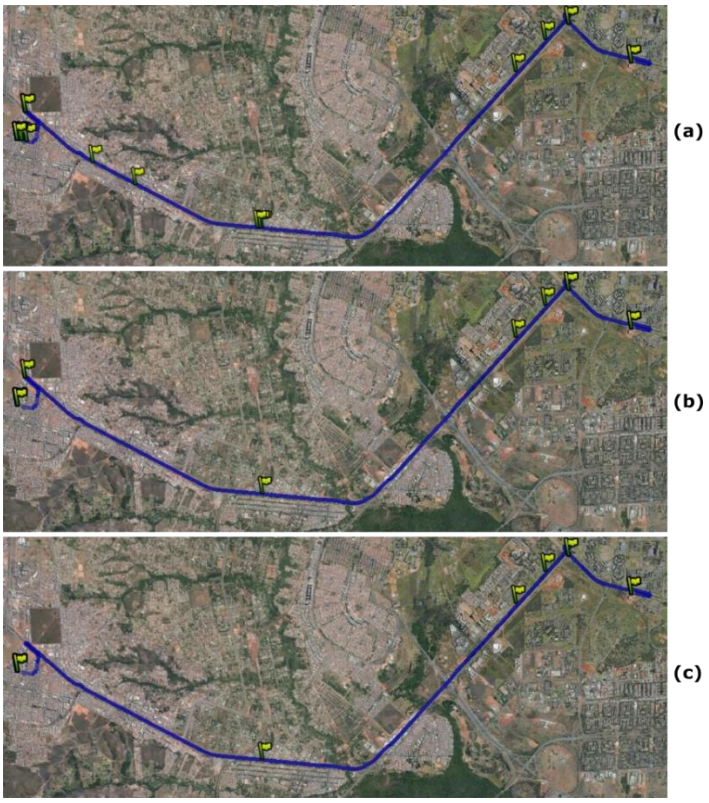


Figura 5.30: Coordenadas recuperadas: (a) instante inicial; (b) 10 minutos e (c) 20 a 60 minutos.

Figura 5.31: Coordenadas recuperadas: (a) instante inicial, 10 e 20 minutos; (b) 30 minutos.

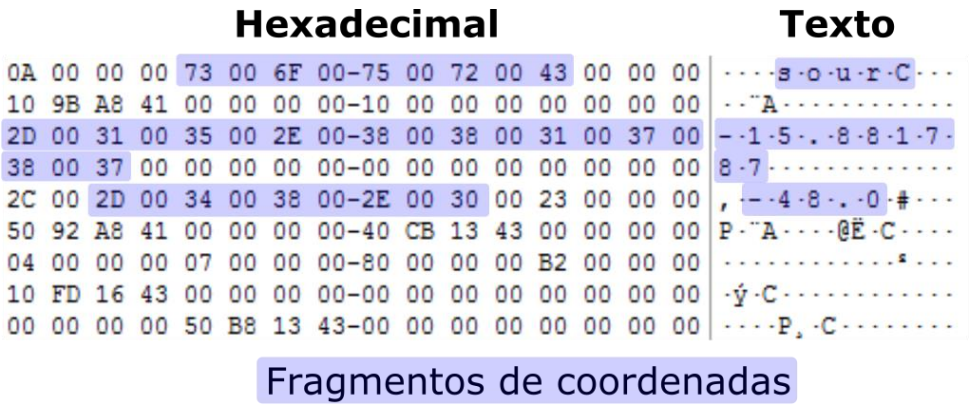


Figura 5.32: Fragmentos corrompidos.

5.3.1.4. Samsung SM-G3812B

Os experimentos realizados com o aparelho Samsung SM-G3812B foram feitos sob condições climáticas adversas (chuva forte, com baixa visibilidade do espaço). Os testes mostraram que a precisão dos receptores GPS nestas condições pode ser altamente afetada. Os dados coletados do aparelho Samsung SM-G3812B mostraram erros no cálculo de posicionamento que variaram entre 1200 m a 1500 m do percurso original, como pode ser visualizado na Figura 5.33.

Neste sentido, o especialista que realizar a coleta e análise dos dados deve levar em consideração outros aspectos das mensagens NMEA, as quais possuem informações sobre o número de satélites visíveis, bem como a precisão dos dados recebidos.

O trecho de log a seguir mostra as mensagens NMEA recuperadas com erro no cálculo de posicionamento. Importante observar que os campos relativos à variação magnética estavam

vazios nas mensagens GPRMC. Já nas mensagens GPGGA, o campo de diluição horizontal de precisão continha o valor 7, um valor elevado, que indica a baixa precisão dos dados obtidos.

*$GPGGA,164152.423,1549.838105,S,04756.322022,W,1,04,7.0,139.7,M,-9.8,M,,*4D*

*$GPRMC,164201.423,A,1549.908438,S,04756.334291,W,28.9,190.6,270216,,,A*50 - Speed in Km/h: 53.5228*

*$GPGGA,164201.423,1549.908438,S,04756.334291,W,1,04,7.0,139.8,M,-9.8,M,,*43*

*$GPRMC,164307.427,A,1550.427097,S,04756.368864,W,33.3,189.2,270216,,,A*5F - Speed in Km/h: 61.6716*

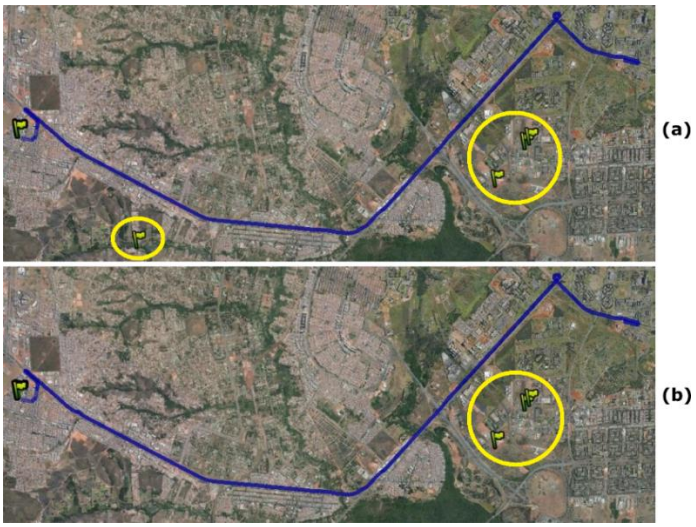


Figura 5.33: Coordenadas recuperadas: (a) instante inicial e 10 minutos e (b) 20 a 60 minutos.

Nos resultados com as coordenadas DD, os dados coletados do aparelho também mostraram erros no cálculo de posicionamento, como mostrado na Figura 5.34. Além disso, as coordenadas recuperadas se concentraram no final do trajeto. Também foi possível analisar a precisão dos dados recebidos, uma vez que estas informações estavam disponíveis nos dados recuperados, conforme ilustrado na Figura 5.35.

Figura 5.34: Coordenadas recuperadas: (a) instante inicial; (b) 10, 20 e 30 minutos e (c) 60 minutos.

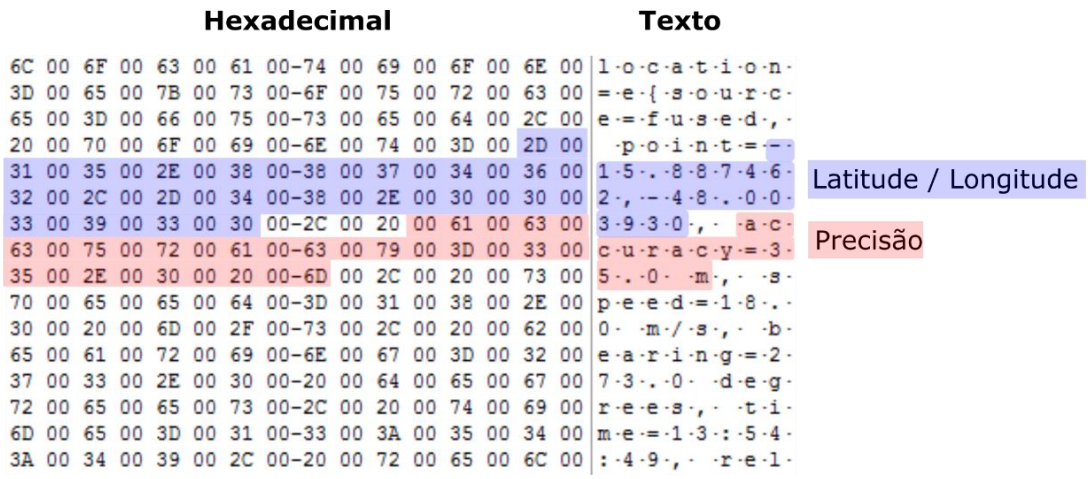


Figura 5.35: Dados de posicionamento e precisão dos serviços de localização.

5.4. EXPERIMENTO EM TRAJETOS LONGOS (150 KM)

Aplicativos com serviços de localização também são frequentemente utilizados em trechos longos para auxiliar viajantes. Desta forma, é comum a presença de dispositivos ainda em funcionamento com o receptor GPS habilitado em acidentes de trânsito ocorridos em viagens.

Neste experimento, foi utilizado somente o *tablet* Samsung GT-P5200, visto que o experimento tinha como objetivo exclusivamente analisar a viabilidade da recuperação de coordenadas de geoposicionamento, mesmo depois de transcorrido um período de tempo superior a duas horas. A análise do comportamento dos dispositivos de diferentes arquiteturas já havia sido feita nos experimentos de trajeto curto e médio, sendo estudadas as discrepâncias entre os diferentes aparelhos, principalmente no que tange a diferentes versões do Android.

Foi percorrido um trajeto de 150 km em aproximadamente duas horas. Ao final do trajeto, após 10 minutos com o veículo em repouso, um único *dump* de memória foi produzido. Cabe ressaltar que não foram produzidos múltiplos *dumps* de memória, pois o objetivo do experimento não era analisar o decaimento da quantidade de informações recuperadas com o passar do tempo, mas sim analisar a quantidade de dados que persistiram na memória neste período de duas horas.

Na fase de análise, para recuperação das coordenadas DD, foi utilizado o par de coordenadas -17.295098, -49.028472 como referência, com uma abrangência de 100 km e um limite de 100 bytes para busca.

O trajeto percorrido abarcou diversas rodovias no estado do Goiás e pode ser visualizado na Figura 5.36.

Figura 5.36: Trajeto de longa distância.

5.4.1. Resultados

Os resultados da extração de dados utilizando a técnica de recuperação de mensagens NMEA podem ser vistos na Tabela 5.8 e na Figura 5.37. A maioria das mensagens NMEA recuperadas, tanto no teste atual como nos testes em trajetos médios e curtos, estava em formato *Unicode*, indicando que a maior parte dos dados coletados estava em regiões de memória destinada aos aplicativos e *frameworks*.

Tabela 5.8 – Quantitativo de mensagens NMEA recuperadas (150 km).

Tipo de mensagem	**Quantidade**
GPRMC - Unicode	*200*
GPRMC - ASCII	*3*
GPGGA - Unicode	*188*
GPGGA - ASCII	*1*

Figura 5.37: Mensagens NMEA recuperadas – 150 km.

Neste cenário, os dados foram coletados após uso contínuo e por um longo período do aplicativo *Google Maps*. Mesmo após o decurso de 2 horas, foi possível recuperar informações de grande parte do trajeto.

O aplicativo fez uso de mais memória RAM do que nos testes anteriores, e mais dados persistiram por mais tempo. Algumas informações podem ter grande relevância forense, podendo indicar o comportamento do usuário do dispositivo ao longo do trajeto. A seguir são mostradas algumas mensagens que foram recuperadas, bem como velocidade desenvolvida em determinados trechos:

*$GPRMC,202910.00,A,1730.081949,S,04844.532432,W,073.5,138.7,150116,,,A*52 (Speed in Km/h: 136.12)*

*$GPRMC,202915.00,A,1730.161759,S,04844.461142,W,077.1,138.5,150116,,,A*50 (Speed in Km/h: 142.78)*

*$GPRMC,202343.00,A,1725.883507,S,04848.124062,W,057.0,133.0,150116,,,A*57 (Speed in Km/h: 105.56)*

Os dados recuperados, ainda que não forneçam precisamente todo o trajeto percorrido, contêm informações que permitem inferir aspectos como clima, tempo e velocidade média em determinados trechos.

Na abordagem de recuperação de coordenadas DD, no único *dump* realizado foram encontradas 110 coordenadas geodésicas, as quais foram plotadas no *Google Earth* e podem ser visualizadas na Figura 5.38.

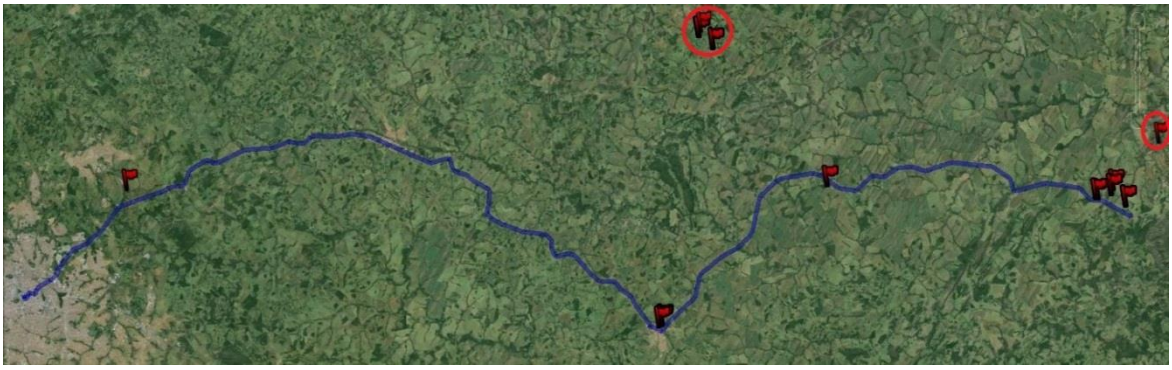


Figura 5.38: Coordenadas DD recuperadas – 150 km.

É possível observar a existência de alguns pares de coordenadas plotados a quilômetros de distância do trajeto original. Como no experimento realizado a abrangência de busca foi maior (100 km), surgiram alguns falso positivos que não fazem parte da reconstrução da trajetória. Neste sentido, o examinador deve estar atento ao fato de que quanto maior a abrangência

geográfica estabelecida como parâmetro de entrada, maior a chance de aparecerem dados não associados ao percurso do dispositivo móvel (falso positivos).

5.5. ANÁLISE COMBINADA

Os métodos de recuperação de mensagens NMEA e de recuperação de coordenadas geodésicas textuais no formato Graus Decimais podem ser combinados para aprimorar a reconstrução da trajetória dos dispositivos. A seguir serão mostrados três exemplos, um para cada tipo de experimento realizado (trajeto curto, médio ou longo), onde os dados puderam ser combinados para fornecer mais informações sobre o trajeto trilhado pelo dispositivo.

5.5.1. Trajetos curtos

Analisando conjuntamente os dados recuperados do aparelho Sony Z2 (*dump* realizado aos 30 minutos), nos experimentos com os serviços de localização desabilitados, foi possível verificar uma maior quantidade de pontos plotados a região final do caminho percorrido, como pode ser visualizado na Figura 5.39. É possível observar na imagem que os dados se complementaram, viabilizando uma melhor compreensão do trajeto percorrido.

Figura 5.39: Análise combinada do aparelho Sony Z2, no experimento de trajeto curto.

Com a análise combinada, tendo como base as mensagens NMEA plotadas, ao se inserir os dados das coordenadas DD recuperadas, foi possível aumentar em aproximadamente 900m a extensão do trajeto recuperado.

5.5.2. Trajetos médios

No caso em análise, os dados recuperados do primeiro *dump* de memória do aparelho Sony LT22i foram plotados conjuntamente no software *Google Earth*. É possível observar, na Figura 5.40, dois grandes grupos de coordenadas recuperadas. O primeiro se concentrou na região final do percurso e está associado às mensagens NMEA. O segundo se concentrou na região central do percurso e diz respeito às coordenadas geodésicas. Novamente, com a análise conjunta foi possível obter mais informação sobre o percurso completo.

Figura 5.40: Análise combinada do aparelho Sony LT22i, no experimento de trajeto médio.

Também neste exemplo, tendo como base as mensagens NMEA plotadas, ao se inserir os dados das coordenadas DD recuperadas, foi possível aumentar em aproximadamente 1800m a extensão do trajeto recuperado.

5.5.3. Trajeto longo

No caso do experimento em trajeto longo, somente o aparelho Samsung GT-P5200 foi analisado. Os dados recuperados do *dump* de memória foram plotados conjuntamente e podem ser visualizados na Figura 5.41.

Não obstante o baixo número de coordenadas no formato Graus Decimais recuperadas, os dados plotados se complementaram, podendo ter relevância a depender do contexto em evidência.

Figura 5.41: Análise combinada do aparelho Samsung GT-P5200, no experimento de trajeto longo.

6. CONCLUSÃO

Neste trabalho foi demonstrada a viabilidade da reconstrução de trajetórias de dispositivos móveis Android com base em coordenadas de posicionamento recuperadas da memória volátil. Além disso, foi feita uma análise da arquitetura GPS em sistemas Android, o que resultou na produção de uma ferramenta para recuperação de dados do protocolo NMEA 0183 e de coordenadas geodésicas no formato Graus Decimais, relevantes no contexto forense.

Os experimentos foram realizados com quatro dispositivos móveis, das marcas Sony e Samsung, possuindo diferentes versões do sistema operacional Android. Foram percorridos três trajetos distintos (4 km, 15 km e 150 km), em condições de tráfego e clima diversas.

Como foi demonstrado, as mensagens do protocolo NMEA puderam ser recuperadas da memória RAM, mesmo depois de transcorrido um certo período de tempo, permitindo a reconstrução parcial dos trajetos trilhados. O mesmo ocorreu com as coordenadas geodésicas no formato Graus Decimais, as quais também permaneceram na memória RAM, possibilitando a remontagem do último caminho percorrido.

Fazendo uso conjunto dos dados recuperados, foi possível recuperar, em certos aparelhos, quase que a totalidade do trajeto percorrido. Além disso, dados como velocidade, tempo, quantidade de satélites visíveis, dentre outras informações pertinentes ao campo investigativo, puderam ser recuperadas.

Também foi possível observar que diversos fatores influenciam no processo de recuperação dos dados da memória RAM, tais como:

- o estado do sistema operacional, em especial o número de processos em execução e os mecanismos de gerenciamento de memória;
- a ferramenta de extração de dados, visto que as abordagens utilizando compilação de *kernel* demandam tempo, não sendo convenientes para cenários de resposta a incidentes;
- a versão do sistema operacional do dispositivo e;
- a arquitetura do dispositivo.

Desta forma, cada exame de dispositivo deve ser tratado de forma individualizada, uma vez que não é possível determinar qual o tempo mínimo para realização da extração da memória RAM antes que os dados se percam, tampouco como se dará o comportamento dos mecanismos de gerenciamento de memória.

Constatou-se que a quantidade de informação relevante no contexto pericial decai com o passar do tempo, visto que a memória RAM é constantemente utilizada e os mecanismos de *gargabe collection* liberam, de tempos em tempos, as páginas menos acessadas. Desta forma, em alguns experimentos os dados recuperados restringiram-se a posição de repouso final do veículo, possuindo pouca relevância no contexto pericial.

Entretanto, ainda que algumas mensagens NMEA possam ser perdidas devido aos mecanismos de gerencialmente de memória, nos casos analisados, foram recuperadas muitas coordenadas GPS dos trajetos percorridos, o que pode ter grande relevância no contexto de uma investigação.

6.1. PUBLICAÇÃO

Algumas dessas contribuições foram aceitas para publicação em congressos científicos, como o caso do artigo *Extraction and analysis of volatile memory in Android systems: an approach focused on trajectory reconstruction based on NMEA 0183 standard*, aceito para publicação no evento *The 9th International Workshop on Digital Forensics* realizado em conjunto com a conferência *11th International Conference on Availability, Reliability and Security (Salzburg, Austria).*

6.2. TRABALHOS FUTUROS

Um dos grandes empecilhos para extração de dados da memória volátil é o procedimento utilizado atualmente, com o uso do módulo *LiME*, que depende da prévia compilação do módulo com base no *kernel* do dispositivo em evidência. A compilação do *kernel* e do módulo demanda tempo, entretanto, em ambientes de reposta a incidentes, onde o tempo para extração da memória RAM pode ser crucial, tarefas como a compilação do módulo podem prejudicar a recuperação dos dados.

Como já mencionado, em (Stüttgen e Cohen, 2014) foi apresentada uma técnica para extração de memória RAM que independe de prévia compilação de *kernel*. Contudo, a arquitetura *ARM* não foi abarcada. Como trabalho futuro, o referido estudo pode ser adaptado para a arquitetura *ARM*.

Esforços também podem ser empreendidos para se manter uma grande base de dados com módulos *LiME* previamente compilados.

Neste trabalho, os dados de localização providos pelos mecanismos de *Cell-ID* e *Wi-fi* não foram explorados. Futuramente pode ser realizado um estudo para analisar como estes dados

estão dispostos na memória RAM e como eles podem ser utilizados para reconstrução de trajetórias, fazendo um paralelo com o trabalho já realizado em (Jeske, 2013).

Os aplicativos *Google Maps* e *Waze* também mantêm informações em mídias não voláteis. Outro trabalho futuro poderia abordar dados recuperados de mídias não voláteis que podem auxiliar na reconstrução de trajetórias, complementando o trabalho aqui realizado.

Além disso, os dispositivos estão em constante evolução e periodicamente são lançadas correções de segurança que inibem a escalada de privilégios, o que demanda estudos constantes dos mecanismos de *rooting*.

REFERÊNCIAS BIBLIOGRÁFICAS

504ENSICS Labs. (2013). LiME – Linux Memory Extractor: Instructions v1.2. Disponível em: <https://lime-forensics.googlecode.com/files/LiME_Documentation_1.1. pdf>. Acesso em: 7 jan. 2016.

About SQLite. (2016). Disponível em <http://www.sqlite.org/about.html>. Acesso em: 17 mai. 2016.

Android. (2016). Disponível em: <https://www.android.com/intl/pt-BR_br/history/>. Acesso em: 28 set. 2016.

Android Interfaces and Architecture. (2016). Disponível em: <https://source.android.com/devices>. Acesso em: 13 jan. 2016.

Arbelet, A. (2014). Garmin satnav forensic methods and artefacts: an exploratory study. Tese de Doutorado. Edinburgh Napier University, Reino Unido.

ARM. (2005). ARM Architecture Reference Manual. Disponível em: <https://www.scss.tcd.ie/~waldroj/3d1/arm_arm.pdf >. Acesso em: 05 mai. 2016.

Bell, C. (2015). Providing Context to the Clues: Recovery and Reliability of Location Data from Android Devices. Dissertação de Mestrado. University of Central Florida. Estados Unidos.

Benn. (2010). Android Root Source Code: Looking at the C-Skills. Disponível em <https://www.nccgroup.trust/us/about-us/newsroom-and-events/blog/2010/september/android-root-source-code-looking-at-the-c-skills>. Acesso em: 19 mai. 2016.

Carvalho E. A. & Araújo P. C. (2009). Noções básicas de sistema de posicionamento global GPS. Universidade Federal do Rio Grande do Norte, Natal.

Chopde, N. R., & Nichat, M. (2013). Landmark based shortest path detection by using A* and Haversine formula. International Journal of Innovative Research in Computer and Communication Engineering, 1(2), 298-302.

Control Segment. (2016). Disponível em <http://www.gps.gov/systems/gps/control>. Acesso em: 05 mai. 2016.

CVE. (2016). Disponível em: <https://cve.mitre.org>. Aceso em: 14 jan. 2016.

d'Alge, J. C. L. (1995). Coordenadas geodésicas e sistemas de informação geográfica. In *Congresso e Feira para Usuarios de Geoprocessamento da America Latina* (GISBRASIL'99), v. 5.

Dashboards. (2016). Disponível em: <https://developer.android.com/about/dashboards/index.html>. Acesso em: 28 set. 2016.

Datatypes In SQLite Version 3. (2016). Disponível em <https://www.sqlite.org/datatype3.html>. Acesso em: 17 mai. 2016.

Debugging ART Garbage Collection. (2014). Disponível em <https://source.android.com/devices/tech/dalvik/gc-debug.html#art_gc_overview>. Acesso em: 28 jan. 2016.

Doonan, D. (2013). Android GPS Forensics. Disponível em: <http://dandoonan.blogspot.com.br>. Acesso em: 25 fev. 2016.

Drake, J. J., Lanier, Z., Mulliner, C., Fora, P. O., Ridley, S. A., & Wicherski, G. (2014). Android hacker's handbook. John Wiley & Sons.

Facebook Messenger. (2016). Disponível em: <https://www.messenger.com>. Acesso em: 28 set. 2016.

Flanagan, Tom. (2013). pynmea2. Disponível em: <https://github.com/Knio/pynmea2>. Acesso em: 26 jan. 2016.

Garmin. (2016). Disponível em: < https://www.garmin.com>. Acesso em: 28 set. 2016.

Google Earth. (2016). Disponível em: <https://www.google.com/earth>. Acesso em: 02 jan. 2016.

Google Maps. (2016). Disponível em: < https://play.google.com/store/apps/details?id=com.google.android.apps.maps&hl=pt_BR>. Acesso em: 17 jan. 2016.

GPSD. (2016). Disponível em: <http://www.catb.org/gpsd/index.html>. Aceso em: 14 jan. 2016.

Grimes, J. G. (2008). Global positioning system standard positioning service performance standard. GPS Navster, Department of Defense.

Guido, G., Gallelli, V., Saccomanno, F., Vitale, A., Rogano, D., & Festa, D. (2014). Treating uncertainty in the estimation of speed from smartphone traffic probes. In *Transportation Research Part C: Emerging Technologies*, v. 47, p. 100-112.

Heriyanto, A. P. (2013). Procedures and tools for acquisition and analysis of volatile memory on android smartphones. In *Proceedings of the 11th Australian Digital Forensics Conference.* Edith Cowan University, Perth, Australia.

Hilgers, C., Macht, H., Muller, T., & Spreitzenbarth, M. (2014). Post-mortem memory analysis of cold-booted android devices. In *IT Security Incident Management & IT Forensics* (IMF), *Eighth International Conference*. IEEE. p. 62-75.

Hoog, A. (2011). Android forensics: investigation, analysis and mobile security for Google Android. Elsevier.

Hotz, G. (2014). towelroot. Disponível em: < https://towelroot.com>. Acesso em: 17 mai. 2016.

IBGE. (2016). Disponível em: <http://www.ibge .gov.br/home/geociencias/geodesia/pmrg/faq.shtm#11>; Acesso em: 25 fev. 2016.

IRoot. (2016). Disponível em: <http://www.iroot.com>; Acesso em: 23 jan. 2016.

Janssen, V. (2009). Understanding coordinate systems, datums and transformations in Australia. Survey Infrastructure and Geodesy, Land and Property Information, NSW Department of Lands, 346 Panorama Avenue, Bathurst NSW 2795, Australia.

Jeske, T. (2013). Floating car data from smartphones: What google and waze know about you and how hackers can control traffic. BlackHat Europe.

Kingo Root. (2016). Disponível em: <https://www.kingoapp.com>; Acesso em: 23 jan. 2016.

Krajina, Tomo. (2010). gpxpy -- GPX file parser. Disponível em: <https://github.com/tkrajina/gpxpy>. Acesso em: 26 jan. 2016.

Leppert, S. (2012). Android memory dump analysis. *Student Research Paper*, *Chair of Computer Science*, v. 1.

Ligh, M. H., Case, A., Levy, J., & Walters, A. (2014). The art of memory forensics: detecting malware and threats in windows, linux, and Mac memory. John Wiley & Sons.

Lim, K. S., Lee, C., Park, J. H., & Lee, S. J. (2014). Test-driven forensic analysis of satellite automotive navigation systems. Journal of Intelligent Manufacturing, v. 25(2), p. 329-338.

Location and Maps. (2016). Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/location/index.html>. Acesso em: 13 jan. 2016.

Location Strategies. (2016). Disponível em: <http://developer.android.com/intl/pt-br/guide/topics/location/strategies.html>. Acesso em: 27 jan. 2016.

Macht, H. (2013). Live memory forensics on android with volatility. *Friedrich-Alexander University Erlangen-Nuremberg*.

Maus, S., Höfken, H., & Schuba, M. (2011). Forensic analysis of geodata in android smartphones. In *International Conference on Cybercrime, Security and Digital Forensics*.

Miguens, A. P. (1999). Navegação: a ciência e a arte–volume II–navegação astronômica e derrotas.

Mularie, W. M. (2000). Department of defense world geodetic system 1984, its definition and relationships with local geodetic systems. National Geospatial-Intelligence Agency, Tech. Rep, 152.

Müller, T., & Spreitzenbarth, M. (2013). Frost. *In Applied Cryptography and Network Security* p. 373-388. Springer Berlin Heidelberg.

National Marine Electronics Association. (2002). NMEA 0183--Standard for interfacing marine electronic devices. Version 3.01.

NMEA Data. (2016). Disponível em: <http://www.gpsinformation.org/dale/nmea.htm>. Acesso em: 12 jan. 2016.

Nugroho, Y. (2014). Exploiting the Futex Bug and uncovering Towelroot. Disponível em: < http://tinyhack.com/2014/07/07/exploiting-the-futex-bug-and-uncovering-towelroot>. Acesso em: 15 mai. 2016.

Nutter, B. (2008). Pinpointing TomTom location records: A forensic analysis. In *Digital Investigation*, v. 5(1), p. 10-18.

PhysOrg. (2012). Disponível em <http://phys.org/news/2012-10-atomic-clocks-good-earth-geoid.html>. Acesso em: 07 fev. 2016.

Playstore. (2016). Disponível em: <https://play.google.com>. Acesso em: 28 set. 2016.

Rekall. (2016). Disponível em: <http://www.rekall-forensic.com>. Acesso em: 15 mai. 2016.

Santana. E. (2014). Disponível em: < http://producao.virtual.ufpb.br/books/edusantana/introducao-a-arquitetura-de-computadores-livro/livro/livro.pdf>. Acesso em: 15 abr. 2016.

Si, H., & Aung, Z. M. (2011). Position data acquisition from NMEA protocol of global positioning system. In *International Journal of Computer and Electrical Engineering*, v. 3(3), p. 353.

Simão, A. M. D. L. (2012). Proposta de método para análise pericial em smartphone com sistema operacional android, Dissertação de Mestrado, UnB, Brasília.

Sousa, João. nmeaSearch. (2016). Disponível em: <https://github.com/jpclaudino/nmeaSearch>. Acesso em: 01 mar. 2016.

Space Segment. (2016). Disponível em <http://www.gps.gov/systems/gps/space>. Acesso em: 05 mai. 2016.

Spreitzenbarth, M., Schmitt, S., & Freiling, F. (2012). Comparing Sources of Location data From Android Smartphones. In *Advances in Digital Forensics VIII*. p. 143-157. Springer Berlin Heidelberg.

Stüttgen, J., & Cohen, M. (2014). Robust Linux memory acquisition with minimal target impact. *Digital Investigation*, v. 11, p. S112-S119.

Styx. (2012). Infecting loadable kernel modules, kernel versions 2.6.x/3.0.x. Phrack. Disponível em <http://phrack.org/issues/68/11.html#article>. Acesso em: 15 mai. 2016.

Sunearthtools. (2016). conversão coordenadas. Disponível em: < http://www.sunearthtools.com/dp/tools/conversion.php?lang=pt>. Acesso em 23 jan. 2016.

Sylve, J., Case, A., Marziale, L., & Richard, G. G. (2012). Acquisition and analysis of volatile memory from android devices. *Digital Investigation*, v. 8(3), p. 175-184.

Thing, V. L., Ng, K. Y., & Chang, E. C. (2010). Live memory forensics of mobile phones. *Digital Investigation*, v. 7, p. S74-S82.

Thomas, D. R., Beresford, A. R., & Rice, A. (2015). Security metrics for the android ecosystem. In: *Proceedings of the 5th Annual ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices*. ACM. p. 87-98.

TomTom. (2016). Disponível em: <https://www.tomtom.com>. Acesso em: 28 set. 2016.

Understanding Android GPS Architecture. (2012). Disponível em: <http://forum.xda-developers.com/showthread.php?t=2063295>. Acesso em 14 jan. 2016.

Van E. O., & Roeloffs, M. (2010). Forensic acquisition and analysis of the Random Access Memory of TomTom GPS navigation systems. In *Digital Investigation*. v. 6(3), p. 179-188.

Vaz, J. A.; Souza, R. P.; Junior, E. S. (2013). Comparação da cobertura e acurácia entre os sistemas GLONASS e GPS obtidas dos dados de observação de uma estação da rede brasileira de monitoramento contínuo. Revista Brasileira de Cartografia, n. 65/3.

Vecchia, E. D. (2014). Perícia digital: da investigação à análise forense. Millennium Editora. Campinas, São Paulo.

Volatility. (2016). Disponível em: <http://www.volatilityfoundation.org/>. Acesso em: 28 set. 2016.

Waze. (2016). Disponível em: <https://play.google.com/store/apps/details?id=com.waze&hl=pt_BR>. Acesso em: 17 jan. 2016.

WhatsApp. (2016). Disponível em: <https://www.whatsapp.com>. Acesso em: 28 set. 2016.

XDA Developers. (2016). XDA-Developers Android Forums. Disponível em: <http://forum.xda-developers.com>. Acesso em 23 jan. 2016.

Zanetti, M. A. Z. (2007). GEODÉSIA. Universidade Federal do Paraná, Curitiba.

APÊNDICES

A – TABELAS

Tabela A.1 – Múltiplas notações de coordenadas geográficas (Sunearthtools, 2016)

Notação	Valor
[N,S]dd°mm'ss.ssss", [W,E]dd°mm'ss.ssss"	S15°45'48.978, W47°52'11.136"
dd°mm'ss.ssss"[N,S], dd°mm'ss.ssss"[W,E]	15°45'48.978"S, 47°52'11.136"W
[-]dd mm ss.ssss, [-]dd mm ss.ssss	-15 45 48.978, -47 52 11.136
[-]dd.dddd [-]dd.dddd	-15.763605 -47.86976
[-]dd.ddddd°,[-]dd.ddddd°	-15.763605°,-47.86976°
[-]milliseconds, [-]milliseconds	-229080114
dd.ddddd[N,S]dd.ddddd[E,W]	15.763605S47.86976W
ddmm.mmmm[N,S]ddmm.mmmm[E,W]	1545.8163S4752.1856W
dd°mm'ss.sss"[N,S], dd°mm'ss.sss"[W,E]	15°45'48.978"S, 47°52'11.136"W
ddmmss.sss[N,S]ddmmss.sss[W,E]	154548.978S475211.136W
[N,S] dd mm.mmm [W,E] dd mm.mmm	S 15 45.8163 W 47 52.1856
dd:mm:ss[N,S],dd:mm:ss[W,E]	15:45:49S,47:52:11W
dd:mm:ss.ss[N,S] dd:mm:ss.ss[W,E]	15:45:48.978S 47:52:11.136W
dd°mm'ss"[N,S] dd°mm'ss"[W,E]	15°45'49"S 47°52'11"W
[-]dd°mm'ss" [-]dd°mm'ss"	-15°45'49" -47°52'11"
dd mm' ss" [N,S] dd mm' ss" [W,E]	15d 45' 49" S 47d 52' 11" W
dd.dddd[N,S] dd.dddd[W,E]	15.763605S 47.86976W
[-]dd° mm.mmmm [-]dd° mm.mmmm	-15° 45.8163, -47° 52.1856

Tabela A.2 – Parâmetros do WGS84 (Mularie, 2000)

Parâmetro	Valor
Semi-eixo maior do elipsoide	6378137.0 m
Fator de achatamento (1/f)	298.257223563
Velocidade angular média nominal da terra (ω)	7292115×10^{-11}
Constante gravitacional da terra (GM)	$(3986004.418 \pm 0.008) \times 10^{8}$ m^3/s^2

Tabela A.3 – Bancos de dados com informações de posicionamento

Banco de dados	Aplicativo
gmm_storage.db	*Google Maps*
suggestions.db	*Google Earth*
tts.db	*Waze*
user.db	*Waze*
mytracks.db	*MyTracks*
RunKeeper.sqlite	*RunKeeper*
workout.db	*Map My Walk*
360LocationDB	*Life360, FriendLocator*
messaging.db	*Life360, FriendLocator*
nc.db	*Life360*

Banco de dados	Aplicativo
dumpLogsDatabase	*FriendLocator*
fsq.db	*Swarm*
threads_db2	*Facebook Messenger*
viber_messages	*Viber*
msgstore.db	*WhatsApp*
naver_line	*LINE*
scout.db	*FieldTrip*
oneweather.db	*OneWeather*
weather.db	*GO Weather*
forecast_accu.db	*Accuweather*
ContextLog_0.db	*Pre-installed Samsung app/feature*
event	*Pre-installed Amazon shopping app*
herrevad	*Google Mobile Services*
https_www.google.com_0.localstorage	*web browser*
NetworkUsage.db	*Google Mobile Services*
locdatabase	*Android Location Tracker*

B – CÓDIGO FONTE DA FERRAMENTA

```
#   Android GPS Forensics (AGF).
#   Extraction and analysis of volatile memory in Android systems:
#   an approach focused on trajectory reconstruction.
#
#   Copyright (C) 2016 João Paulo Claudino (jpclaudino@gmail.com)
#   Copyright (C) 2016 João José Costa Gondim (joao.gondim@gmail.com)
#
#   Available at: https://github.com/jpclaudino/nmeaSearch
#
#   This program is free software: you can redistribute it and/or
#   modify it under the terms of the GNU General Public License as     #
published by the Free Software Foundation, either version 3 of the #
License, or (at your option) any later version.
#
#   This program is distributed in the hope that it will be useful,
#   but WITHOUT ANY WARRANTY; without even the implied warranty of
#   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#   GNU General Public License for more details.
#
#   NmeaSearch: this program receives as parameter a memory dump and  #
returns a GPX file with all coordinates retrieved from NMEA       #
sentences and log files
#

__author__ = 'Claudino'

from argparse import ArgumentParser
import sys
import os
import pynmea2
import datetime
from fileUtil import *

nmeaGPRMC_UTF16 = b'$\x00G\x00P\x00R\x00M\x00C\x00'
nmeaGPGGA_UTF16 = b'$\x00G\x00P\x00G\x00G\x00A\x00'
nmeaGPRMC_ASCII = b'$GPRMC'
nmeaGPGGA_ASCII = b'$GPGGA'

class BadMessage(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

def findCheckSum(offset,pieceOfFile,isUTF16):
    # Find marker (*)
    end = offset + 200
    while(offset < end):
        checksum = pieceOfFile[offset]
        if checksum == 42:
            if(isUTF16):
                return offset + 6
            else:
                return offset + 3
        offset += 1
    return 0
```

```
def parseNmea(pieceOfFile):
    offset = 0
    messageListGPRMC_UTF16 = []
    messageListGPGGA_UTF16 = []
    messageListGPRMC_ASCII = []
    messageListGPGGA_ASCII = []
    while offset < len(pieceOfFile):
        dollarSign = pieceOfFile[offset]
        # Dollar sign is the starting character of the message
        if(dollarSign == 36):
            try:
                type = findMessageType(offset,pieceOfFile)
                if type == 1:
                    messageListGPRMC_UTF16.append(getMessageUTF16(offset,
pieceOfFile))
                elif type == 2:
                    messageListGPGGA_UTF16.append(getMessageUTF16(offset,
pieceOfFile))
                elif type == 3:
                    messageListGPRMC_ASCII.append(getMessageASCII(offset,
pieceOfFile))
                elif type == 4:
                    messageListGPGGA_ASCII.append(getMessageASCII(offset,
pieceOfFile))
            except:
                pass
        offset += 1
    return
messageListGPRMC_UTF16,messageListGPGGA_UTF16,messageListGPRMC_ASCII,messag
eListGPGGA_ASCII

def getMessageUTF16(offset, pieceOfFile):
    try:
        messageSize = findCheckSum(offset, pieceOfFile,True)
        if messageSize != 0:
            nmeaMessage = pieceOfFile[offset:messageSize]
            nmeaUTF16Message = nmeaMessage.decode('utf-16-le')
            nmeaASCIIMessage = nmeaUTF16Message.encode('ASCII', 'ignore')
            msg = pynmea2.parse(nmeaASCIIMessage.decode())
            return msg
        else:
            raise BadMessage("UTF-16 message not found!")
    except:
        raise BadMessage("UTF-16 message not found!")


def getMessageASCII(offset, pieceOfFile):
    try:
        messageSize = findCheckSum(offset, pieceOfFile, False)
        if messageSize != 0:
            nmeaMessage = pieceOfFile[offset:messageSize]
            msg = pynmea2.parse(nmeaMessage.decode())
            return msg
        else:
            raise BadMessage("ASCII message not found!")
    except:
        raise BadMessage("ASCII message not found!")


def findMessageType(offset,pieceOfFile):
```

```
        try:
            pieceUTF16 = pieceOfFile[offset:offset + 12]
            if pieceUTF16 == nmeaGPRMC_UTF16:
                return 1 # Type GPRMC encoded in UTF16
            if pieceUTF16 == nmeaGPGGA_UTF16:
                return 2 # Type GPGGA encoded in UTF16
            pieceASCII = pieceOfFile[offset:offset + 6]
            if pieceASCII == nmeaGPRMC_ASCII:
                return 3 # Type GPRMC encoded in ASCII
            if pieceASCII == nmeaGPGGA_ASCII:
                return 4 # Type GPGGA encoded in ASCII
        except:
                return 0
        return 0

def gpsFinder(basepath):
    try:
        fin = open(basepath, "rb")
        foutGPX = open(basepath + "_" + GPXFile,'wb')
        foutLOG = open(basepath + "_" + LOGFile,'wb')
        foutCoordinates = open(basepath + "_" + CoordinatesFile,'wb')
    except:
        print("File not Found")
        exit(0)
    listGPRMC, listGPGGA,
listGPRMC_UTF16,listGPGGA_UTF16,listGPRMC_ASCII,listGPGGA_ASCII =
[],[],[],[],[],[]
    for piece in read_in_chunks(fin):
        lGPRMC_UTF16,lGPGGA_UTF16,lGPRMC_ASCII,lGPGGA_ASCII =
parseNmea(piece)
        listGPRMC_UTF16 = listGPRMC_UTF16 + lGPRMC_UTF16
        listGPGGA_UTF16 = listGPGGA_UTF16 + lGPGGA_UTF16
        listGPRMC_ASCII = listGPRMC_ASCII + lGPRMC_ASCII
        listGPGGA_ASCII = listGPGGA_ASCII + lGPGGA_ASCII
    listGPRMC = listGPRMC_UTF16 + listGPRMC_ASCII
    listGPGGA = listGPGGA_UTF16 + listGPGGA_ASCII
    buildLog(foutLOG,listGPGGA, listGPGGA_ASCII, listGPGGA_UTF16,
listGPRMC, listGPRMC_ASCII, listGPRMC_UTF16)
    try:
        sortedListGPRMC = sorted(listGPRMC+listGPGGA,key=lambda msg:
msg.timestamp)
        buildGPX(foutGPX,sortedListGPRMC)
        buildCoordinatesLog(foutCoordinates,sortedListGPRMC)
    except:
        buildGPX(foutGPX,listGPRMC+listGPGGA)
        buildCoordinatesLog(foutCoordinates,listGPRMC+listGPGGA)
    fin.close()
    foutGPX.close()
    foutLOG.close()
    foutCoordinates.close()

def main(argv):
    print("Starting NMEA sentences recovery!")
    print(datetime.datetime.now())
    # parser options
    parser = ArgumentParser(description='GPS Data Recovery')
    parser.add_argument(dest='infile', help="The argument needs to be a
dump file.")
    options = parser.parse_args()
    # checks for the input file
    if options.infile is None:
```

```
        parser.print_help()
        sys.exit(1)
    if not os.path.exists(options.infile):
        print('Error: "{0}" file is not found!'.format(options.infile))
        sys.exit(1)
    gpsFinder(options.infile)
    print(datetime.datetime.now())
    print("Nmea sentences write to gpx and log files ")

if __name__ == '__main__':
    main(sys.argv[1:])



#
#   StringSearch: this program receives as parameter a memory dump,
#   the refencial latitude and longitude, a geographical distance in
kilometers and
#   a search distance in bytes, and returns a GPX file
#   with all textual coordinates retrieved from memory, and log files.
#

__author__ = 'Claudino'

import os
from argparse import ArgumentParser
from fileUtil import *
from math import radians, cos, sin, asin, degrees
import sys
import datetime

AVG_EARTH_RADIUS = 6371
DECIMALSIZE = 50
BYTERANGE = 100

def haversine_getLatitude(point1, d):
    """ Calculates the latitude of a point distant "d" kilometes from the
reference point, passed as a parameter.

    :input: one tuple, containing the latitude and longitude of a fixed
point
    in decimal degrees, and the distance to the next point (which has the
same longitude of the input tuple), in kilometers.

    :output: Returns latitude of a the point in degrees.

    """
    # unpack latitude/longitude
    lat1, lng1 = point1

    # convert all latitudes/longitudes from decimal degrees to radians
    lat1, lng1 = map(radians, (lat1, lng1))

    # calculate latitude
    lat2 = (2 * asin( (sin(d/ (2*AVG_EARTH_RADIUS) )) ) ) + lat1

    return degrees(lat2)

def haversine_getLongitude(point1, d):
    """ Calculates the longitude of a point distant "d" kilometes from the
reference point, passed as a parameter.
```

```
    :input: one tuple, containing the latitude and longitude of a fixed
point
    in decimal degrees, and the distance to the next point (which has the
same latitude of the input tuple), in kilometers.

    :output: Returns latitude of a the point in degrees.

    """
    # unpack latitude/longitude
    lat1, lng1 = point1

    # convert all latitudes/longitudes from decimal degrees to radians
    lat1, lng1 = map(radians, (lat1, lng1))

    # calculate latitude
    lng2 = (2 * asin( (sin(d/ (2*AVG_EARTH_RADIUS) )) / (cos(lat1)) ) )  +
lng1

    return degrees(lng2)

class CoordinateNotFound(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

class NotFloatCoordinate(Exception):
    def __init__(self, value):
        self.value = value
    def __str__(self):
        return repr(self.value)

def getLimits(point,distance):
    northLimit = haversine_getLatitude(point,distance)
    southLimit = haversine_getLatitude(point,-distance)
    eastLimit = haversine_getLongitude(point,distance)
    westLimit = haversine_getLongitude(point,-distance)
    return northLimit,southLimit,eastLimit,westLimit

def getDegreesFromLimits(limits):
    degreesList = []
    for limit in limits:
        degreesList.append(int(limit))
    return degreesList

def getStringDegrees(degrees):
    stringDegreesList = []
    for degree in degrees:
        stringDegreesList.append(str(degree))
    return stringDegreesList

def getASCIIDegrees(degrees):
    asciiDegreesList = []
    for degree in degrees:
        asciiDegreesList.append(degree.encode(ASCII))
    return asciiDegreesList

def getUTFDegrees(degrees):
    utfDegreesList = []
    for degree in degrees:
```

```
        utfDegreesList.append(degree.encode(UTF16))
    return utfDegreesList

def checkDegrees(degreeDownLimit,degreeUpLimit, piece, offset,encoding):
    try:
        stringDegreeDownLimit = degreeDownLimit.decode(encoding)
        intDegreeDownLimit = int(stringDegreeDownLimit)
        stringDegreeUpLimit = degreeUpLimit.decode(encoding)
        intDegreeUpLimit = int(stringDegreeUpLimit)
        for intDegree in range(intDegreeDownLimit,intDegreeUpLimit+1):
            stringDegree = intDegree.__str__()
            bytesDegree = stringDegree.encode(encoding)
            tempOffset = offset
            isValidDegree = True
            for byteDegree in bytesDegree:
                if byteDegree != piece[tempOffset]:
                    isValidDegree = False
                    break
                tempOffset += 1
            if isValidDegree:
                return piece[offset:tempOffset]
        raise CoordinateNotFound("Degree not found.")
    except:
        raise CoordinateNotFound("Degree not found.")

def checkLimits(stringLatitude,stringLongitude,limits):
    try:
        floatLatitude = float(stringLatitude)
        floatLongitude = float(stringLongitude)
        northLimit,southLimit,eastLimit,westLimit = limits
        if( (floatLatitude < northLimit) and (floatLatitude > southLimit)
and (floatLongitude < eastLimit) and (floatLongitude > westLimit) ):
            return True
        else:
            return False
    except:
        return False

def getMessage(offset,piece,degreeList,limits,encoding):
    northDegree = degreeList[0]
    southDegree = degreeList[1]
    eastDegree = degreeList[2]
    westDegree = degreeList[3]
    latitudeDegree = checkDegrees(southDegree,northDegree, piece,
offset,encoding)
    latitude =
getDecimalDegreeCoordinate(latitudeDegree,offset,piece,encoding)
    offsetNewPiece = 0
    newPiece = getNewPiece(offset, piece)
    longitude = None
    longitudesList = []
    while (offsetNewPiece < len(newPiece)):
        try:
            longitudeDegree = checkDegrees(westDegree,eastDegree, newPiece,
offsetNewPiece,encoding)
            longitude =
getDecimalDegreeCoordinate(longitudeDegree,offsetNewPiece,newPiece,encoding
)
            longitudesList.append((longitude,abs(BYTERANGE-
offsetNewPiece)))
        except:
```

```
            pass
        offsetNewPiece += 1
    if not longitudesList:
        raise CoordinateNotFound("Longitude not found.")
    else:
        longitude = getBestLongitude(longitudesList)
        stringLatitude = intListToString(latitude,encoding)
        stringLongitude = intListToString(longitude,encoding)
        if(checkLimits(stringLatitude,stringLongitude,limits)):
            print(stringLatitude + "  " + stringLongitude)
            return stringLatitude,stringLongitude,newPiece
        else:
            raise NotFloatCoordinate("Latitude and longitude must be in
decimal degrees")


def getBestLongitude(longitudesList):
    distance = BYTERANGE
    minLong = None
    for longitude in longitudesList:
        if(longitude[1] < distance):
            minLong =  longitude[0]
            distance = longitude[1]
    return minLong

def intListToString(bytelist,encoding):
    frame = bytearray()
    for intByte in bytelist:
        frame.append(intByte)
    return frame.decode(encoding)

def getNewPiece(offset, piece):
    if(offset < BYTERANGE):
        return piece[0:offset + BYTERANGE]
    return piece[offset - BYTERANGE:offset + BYTERANGE]

def getDecimalDegreeCoordinate(degree,offset,piece,encoding):
    coordinateByteArray = []
    tempOffset = offset
    for byteDegree in degree:
        coordinateByteArray.append(piece[tempOffset])
        tempOffset += 1
    if( encoding == ASCII):
        getDecimalDegreeCoordinateASCII(coordinateByteArray, piece,
tempOffset)
    else:
        getDecimalDegreeCoordinateUTF(coordinateByteArray, piece,
tempOffset)
    return coordinateByteArray

def getDecimalDegreeCoordinateASCII(coordinateByteArray, piece,
tempOffset):
    if (piece[tempOffset] == pointSign[0]):
        coordinateByteArray.append(piece[tempOffset])
        count = 0
        while (count < DECIMALSIZE):
            tempOffset += 1
            if (isNumber(piece[tempOffset])):
                coordinateByteArray.append(piece[tempOffset])
                count += 1
            else:
```

```
                if (count == 0):
                    raise CoordinateNotFound("Message without decimal
part")
                else:
                    break
    else:
        raise CoordinateNotFound("Message without . sign")

def getDecimalDegreeCoordinateUTF(coordinateByteArray, piece, tempOffset):
    if (piece[tempOffset] == pointSign[0] and piece[tempOffset + 1] ==
zeroSign[0]):
        coordinateByteArray.append(piece[tempOffset])
        coordinateByteArray.append(piece[tempOffset + 1])
        count = 0
        while (count < DECIMALSIZE):
            tempOffset += 2
            if (isNumber(piece[tempOffset]) and piece[tempOffset + 1] ==
zeroSign[0]):
                coordinateByteArray.append(piece[tempOffset])
                coordinateByteArray.append(piece[tempOffset + 1])
                count += 1
            else:
                if (count == 0):
                    raise CoordinateNotFound("Message without decimal
part")
                else:
                    break
    else:
        raise CoordinateNotFound("Message without . sign")

def isNumber(byte):
    for number in numbers:
        if (byte == number):
            return True
    return False

def searchCoordinates(piece,limits,asciiDegreesList,utfDegreesList):
    offset = 0
    messageListASCII = []
    messageListUTF = []
    while offset < len(piece):
        if(isNumber(piece[offset]) or piece[offset] == negativeSign[0] ):
            try:
                try:

messageListASCII.append(getMessage(offset,piece,asciiDegreesList,limits,ASC
II))
                except:

messageListUTF.append(getMessage(offset,piece,utfDegreesList,limits,UTF16))
            except:
                pass
        offset += 1
    return messageListASCII,messageListUTF

def stringSearch(coordinate,distance,basepath):
    print("Starting GPS strings search!")
    print(datetime.datetime.now())
    try:
        fin = open(basepath, "rb")
        foutGPX = open(basepath + "_" + StringGPXFile,'wb')
```

```
        foutLOG = open(basepath + "_" + StringLOGFile,'wb')
        foutMemoryLOG = open(basepath + "_" + StringMemoryLOGFile,'wb')
    except:
        print("File not Found")
        exit(0)
    listUTF, listASCII = [],[]
    limits = getLimits(coordinate,distance)
    degreesFromLimits = getDegreesFromLimits(limits)
    stringDegreesFromLimits = getStringDegrees(degreesFromLimits)
    asciiDegreesList = getASCIIDegrees(stringDegreesFromLimits)
    utfDegreesList = getUTFDegrees(stringDegreesFromLimits)
    for piece in read_in_chunks(fin):
        lASCII,lUTF =
searchCoordinates(piece,limits,asciiDegreesList,utfDegreesList)
        listUTF = listUTF + lUTF
        listASCII = listASCII + lASCII
    messages = listASCII+listUTF
    for message in messages:
        print(message.__str__())
    buildGPXfromCoordinatesList(foutGPX,messages)
    buildLogFromCoordinatesList(foutLOG,messages)
    buildMemoryLogFromCoordinatesList(foutMemoryLOG,messages)
    fin.close()
    foutGPX.close()
    foutLOG.close()
    foutMemoryLOG.close()
    print(datetime.datetime.now())
    print("GPS sentences write to gpx and log files ")

def main(argv):
    print("Starting GPS Data (coordinates in decimal degrees format)
recovery!")
    print(datetime.datetime.now())
    # parser options
    parser = ArgumentParser(description='GPS Data Recovery')
    parser.add_argument(dest='infile', help="The argument needs to be a
dump file.")
    parser.add_argument('-d', '--distance', dest='distance', help="Distance
in km")
    parser.add_argument('-b', '--bytes', dest='bytes', help="bytes range")
    parser.add_argument('-la', '--latitude',  dest='latitude',
help="Latitude in decimal degrees")
    parser.add_argument('-lo', '--longitude',  dest='longitude',
help="Longitude in decimal degrees")
    options = parser.parse_args()
    # checks for the input file
    if options.infile is None:
        parser.print_help()
        sys.exit(1)
    if not os.path.exists(options.infile):
        print('Error: "{0}" file is not found!'.format(options.infile))
        sys.exit(1)
    if not options.distance.isdigit():
        print('Error: distance must be in km!'.format(options.infile))
        sys.exit(1)
    if not options.bytes.isdigit():
        print('Error: bytes must be an integer!'.format(options.infile))
        sys.exit(1)
    else:
        BYTERANGE = int(options.bytes)
    try:
```

```
        coordinates = (float(options.latitude),float(options.longitude))
    except:
        print('Error: latitude and longitude must be in decimal
degrees!'.format(options.infile))
        sys.exit(1)
    stringSearch(coordinates,int(options.distance),options.infile)
    print(datetime.datetime.now())
    print("Nmea sentences write to gpx and log files ")

if __name__ == '__main__':
    main(sys.argv[1:])


#
#   fileUtil: auxiliary procedures and settings
#

__author__ = 'Claudino'

import gpxpy
import gpxpy.gpx

GPXFile = "NMEA.gpx"
LOGFile = "NMEA.log"
StringGPXFile = "StringCoordinates.gpx"
StringLOGFile = "StringCoordinates.log"
StringMemoryLOGFile = "StringCoordinatesMemory.log"
CoordinatesFile = "Coordinates .log"

pointSign = b'.'
negativeSign = b'-'
numbers = b'0123456789'
zeroSign = b'\x00'
ASCII =  'ASCII'
UTF16 = 'utf-16-le'

def read_in_chunks(file_object, chunk_size=1024000):
    """Lazy function (generator) to read a file piece by piece.
    Default chunk size: 1k."""
    while True:
        data = file_object.read(chunk_size)
        if not data:
            break
        yield data

def buildGPXfromCoordinatesList(wfile,listMessages):
    gpx = gpxpy.gpx.GPX()
    name = 1
    for msg in listMessages:
        if msg != None:
            try:
                gpx_waypoint=gpxpy.gpx.GPXWaypoint(name=name,
latitude=msg[0], longitude=msg[1])
                gpx.waypoints.append(gpx_waypoint)
                name += 1
            except:
                pass
    wfile.write(gpx.to_xml().encode('utf-8'))

def buildLogFromCoordinatesList(wfile,listMessages):
    sentence = "latitude,longitude"
```

```
    wfile.write(sentence.encode('utf-8'))
    for msg in listMessages:
        if msg != None:
            try:
                sentence = "\n" + str(msg[0]) + "," + str(msg[1])
                wfile.write(sentence.encode('utf-8'))
            except:
                pass

def buildMemoryLogFromCoordinatesList(wfile,listMessages):
    for msg in listMessages:
        try:
            wfile.write(msg.__str__().encode('utf-8'))
        except:
            pass

def buildGPX(wfile,listNMEAMessages):
    gpx = gpxpy.gpx.GPX()
    name = 1
    for msg in listNMEAMessages:
        if msg != None:
            try:
                if msg.latitude != None and msg.longitude != None and
msg.latitude != 0.0 and msg.longitude != 0.0:
                    gpx_waypoint=gpxpy.gpx.GPXWaypoint(name=name,
latitude=msg.latitude, longitude=msg.longitude)
                    gpx.waypoints.append(gpx_waypoint)
                    name += 1
            except:
                pass
    wfile.write(gpx.to_xml().encode('utf-8'))

def buildCoordinatesLog(wfile,listNMEAMessages):
    sentence = "latitude,longitude"
    wfile.write(sentence.encode('utf-8'))
    for msg in listNMEAMessages:
        if msg != None:
            try:
                if msg.latitude != None and msg.longitude != None and
msg.latitude != 0.0 and msg.longitude != 0.0:
                    if msg.timestamp != None:
                        sentence = "\n" + str(msg.latitude) + "," +
str(msg.longitude) + "," + str(msg.timestamp)
                    else:
                        sentence = "\n" + str(msg.latitude) + "," +
str(msg.longitude) + ",0"
                    wfile.write(sentence.encode('utf-8'))
            except:
                pass

def buildLog(wfile,listGPGGA, listGPGGA_ASCII, listGPGGA_UTF16, listGPRMC,
listGPRMC_ASCII, listGPRMC_UTF16):
    countMessages(listGPGGA, listGPGGA_ASCII, listGPGGA_UTF16, listGPRMC,
listGPRMC_ASCII, listGPRMC_UTF16, wfile)
    for msg in listGPRMC:
        printMsg(wfile,msg, True)
    for msg in listGPGGA:
        printMsg(wfile,msg, False)

def printMsg(wfile,msg,printSpeed):
    try:
```

```
        if printSpeed:
            sentence = "\n"+ msg.__str__() + "     -    Speed in Km/h: "  +
str(msg.spd_over_grnd * 1.852)
            wfile.write(sentence.encode('utf-8'))
        else:
            sentence = "\n"+ msg.__str__()
            wfile.write(sentence.encode('utf-8'))
    except:
        pass

def countMessages(listGPGGA, listGPGGA_ASCII, listGPGGA_UTF16, listGPRMC,
listGPRMC_ASCII, listGPRMC_UTF16, wfile):
    validGPGGAMessages = list(filterList(listGPGGA))
    validGPRMCMessages = list(filterList(listGPRMC))
    wfile.write(("Number of Unicode GPRMC messages: " +
str(len(listGPRMC_UTF16))).encode('utf-8'))
    wfile.write(("\nNumber of ASCII GPRMC messages: " +
str(len(listGPRMC_ASCII))).encode('utf-8'))
    wfile.write(("\nTotal Number of GPRMC messages: " +
str(len(listGPRMC))).encode('utf-8'))
    wfile.write(("\nTotal Number of Valid GPRMC messages: " +
str(len(validGPRMCMessages))).encode('utf-8'))
    wfile.write("\n***********************************".encode('utf-8'))
    wfile.write(("\nNumber of Unicode GPGGA messages: " +
str(len(listGPGGA_UTF16))).encode('utf-8'))
    wfile.write(("\nNumber of ASCII GPGGA messages: " +
str(len(listGPGGA_ASCII))).encode('utf-8'))
    wfile.write(("\nTotal Number of GPGGA messages: " +
str(len(listGPGGA))).encode('utf-8'))
    wfile.write(("\nTotal Number of Valid GPGGA messages: " +
str(len(validGPGGAMessages))).encode('utf-8'))
    wfile.write("\n***********************************".encode('utf-8'))

def filterList(list):
    for msg in list:
        if msg.is_valid: yield msg
```