Teemu Hokkanen

Paikannuslaitteet rikostutkinnassa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

8.5.2013

Metropolia

Tekijä Otsikko Sivumäärä Aika	Teemu Hokkanen Paikannuslaitteet rikostutkinnassa 38 sivua + 4 liitettä 8.5.2013
Tutkinto	insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	
Ohjaajat	yliopettaja Hannu Laine rikosylikonstaapeli Jari Javanainen

Insinöörityön aiheena on paikannuslaitteet rikostutkinnassa. Työssä selvitettiin miten ja miksi satelliittipaikannusjärjestelmää käyttävistä laitteista voidaan saada poliisin rikostutkinnassa hyödyllistä tietoa. Satelliittipaikannusjärjestelmien historiaa käytiin läpi ennen kaikkea Yhdysvaltojen kehittämän Navstar-GPS -järjestelmän (tutummin vain GPS) kautta, sillä siitä oli entisen Neuvostoliiton vastaavaa järjestelmää, Glonassia, helpompi saada tietoa ja se oli valmis hiukan ennen entisen Neuvostoliiton vastaavaa. Toimintaperiaate molemmissa on täysin sama.

Poliisin suorittamassa rikostutkinnassa on hyvin usein tarkoitus etsiä sitä tietoa, joka on tahallisesti tai tahattomasti hävitetty tai yritetty hävittää. Siksipä tässä työssä selvitettiin myös syitä, sekä ohjelmien, että käytettävien teknisten ratkaisujen osalta, jotka mahdollistavat tai haittaavat hävitettyjen tiedostojen tietosisällön etsimistä ja palauttamista. Erityisen tarkastelun alla oli edelleen ylivoimaisesti suosituin ulkoisten muistimedioiden tiedostojärjestelmä FAT32 ja flash-muistia käyttävien ulkoisten tallennusmedioiden antamat mahdollisuudet tai tekniset rajoitteet tietojen palauttamiselle.

Työn lopputuloksena kolmen eri paikannuslaitteen tallennustapa on takaisinmallinnettu ja näiden lisäksi kaksi muuta, muiden tekemien selvitysten tuloksena takaisinmallinnettua tallennustapaa on esitelty. Käytössä ollut testidata antoi mahdollisuuden tehdä ohjelmat neljällä eri tavalla tallennettujen paikkatietojen etsimiseen. Tehdyt ohjelmat toimivat odotetusti ja tuloksena saatiin useita satoja, usein jopa tuhansia, osoitemerkintöjä, joista ylivoimaisesti suurin osa oli käyttäjän tai laitteen oman ohjelmiston ulottumattomissa.

Nykyään kaupallisilla toimijoilla on tarjolla hyviä ja helppokäyttöisiä, mutta kalliita, ohjelmia paikannuslaitteiden tutkintaan. Tämän insinöörityön lopputuloksena tehdyt ohjelmat ovat ilmaisia ja tarjoavat mahdollisuuden reagoida hyvin nopeasti uuden paikannuslaitteen tullessa tutkintaan siinä tapauksessa, että se ei ole kaupallisilla ohjelmilla tuettu malli. Ohjelmien perusrakenne on yksinkertainen ja osittain modulaarinen ja siksi helposti muokattavissa, jos paikannuslaitteen tapa tallentaa paikkatietoa on kohtuudella selvitettävissä.

Avainsanat	satelliittipaikannus, GPS, Glonass, FAT, flash-muisti, rikostutkinta

Author Title Number of Pages Date	Teemu Hokkanen Navigation devices in criminal investigation 38 pages + 4 appendices 8 May 2013
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	
Instructors	Hannu Laine, Principal Lecturer Jari Javanainen, detective sergeant

The topic of this engineering thesis is navigation devices in criminal investigation. In this engineering thesis it was researched how and why useful information could be gathered from devices using the satellite navigation system to be used in the criminal investigation conducted by the police. The history of the satellite navigation systems is presented first and foremost through the United State's Navstar GPS (commonly just GPS) since it is easier to get information and was finished a little bit sooner than the former Soviet Union's counterpart, Glonass. They both work with exactly the same principle.

In the criminal investigation conducted by the police the goal is very often to find the information that has been intentionally or non-intentionally removed. That is why in this thesis the reasons, both software and used technical solutions, were researched for the causes that help or hinder the searching and restoration of the information content of deleted files. Under a special scrutiny were the chances and technical limitations imposed to the restoration of the information from the still overwhelmingly most popular file system for external memory medium, FAT32, and the flash memory used in external memory mediums.

As the result of this thesis three different data storage methods of navigation devices have been reverse engineered and on top of this two different devices researched by others have been presented. The used test data gave chance to make programs to find location data from four different storage methods. The programs worked as expected and as a result got several hundred, often even thousands, address locations, of which overwhelming majority was beyond the reach of the user or the devices own software.

Nowadays commercial organisations are offering good and easy to use, but expensive, programs for investigating locations devices. The programs made as a result of this engineering thesis are free and offer an opportunity to react very fast to a new location device brought under investigation in the event where it is a model not supported by the commercial programs. The basic structure of the programs is simple and partially modular and thus easily modified if the location devices way of storing location information can be found out with reasonable amount of work.

Keywords	satellite location, GPS, Glonass, FAT, flash-memory, criminal investigation

Sisällys

Liitteet

Liite 1. Nokiaa ja Route66:ta varten tehty ohjelmakoodi

Liite 2. Mioa ja Navmania varten tehty ohjelmakoodi

Liite 3. Navigonia varten tehty ohjelmakoodi

Liite 4. iGOta varten tehty ohjelmakoodi

1 Johdanto

Tämän työn tarkoituksena on olla avuksi rikosten esitutkinnassa. Nykyään erityisesti omaisuus- ja huumausainerikollisuudessa käytetään enenevässä määrin paikannuslaitteita (ns. GPS-laitteita). Paikannuslaitteita hyödyntäen omaisuusrikolliset etsivät murtokohteita tiedusteltuaan paikan ensin esimerkiksi Internetistä. Huumausainerikolliset taas tallentavat kätkemiensä huumausaine-erien sijainnit paikannuslaitteiden muistiin antaakseen ne sovittua korvausta vastaan taas eteenpäin huumausaineiden vastaanottajalle. Myös muiden kuin edellä mainittujen rikosten tekemisessä hyödynnetään toisinaan paikannuslaitteita. Oli rikos mikä tahansa, saattaa sen tekijällä rikoksen tekemistä varten olla tarvetta löytää itselleen entuudestaan tuntemattomaan paikkaan. Karttakirja on hidas ja vanhanaikainen suunnistuksen apuväline, ja tarvittavan paperikartan löytäminen voi olla hankalaakin. Paikannuslaite päivittyvine karttapohjineen on modernin rikollisen apuväline siinä missä tavallisenkin liikkuvan ihmisen.

Poliisin tehtävänä on selvittää rikokset ja saada niistä epäillyt ihmiset kiinni. Kun epäilty tai epäillyt on saatu riittävällä varmuudella yksilöityä, on yleensä kotietsintöjen ja takavarikkojen aika. Kun paikannuslaite on takavarikoitu, voi olla tarpeen saada selville mitä osoitteita laitteeseen on syötetty tai missä laitetta on käytetty. Pelkästään laitteen oman käyttöliittymän läpi tapahtuva tarkastelu ei välttämättä paljasta mitään, sillä käyttäjä on voinut tarkoituksella tyhjentää laitteen tallentamat suosikit tai muut talteen jäävät osoitteet. Tieto ei kuitenkaan häviä paikannuslaitteestakaan pelkästään sillä, että käyttäjä poistaa tiedon laitteen omaa käyttöliittymää käyttäen. Tällöin häviää vain tiedoston sijaintiin liittyvä ”sisällysluettelomerkintä”.

Insinöörityön tavoitteena on tehdä tietokoneohjelma, jolla päästään käsiksi siihen paikkatietoon, mitä paikannuslaitteen muistista on saatavissa, oli se sitten käyttäjän toimenpiteiden tai laitteen toiminnan seurauksena poistettua tietoa. Tämä onnistuu helpoiten siten, että laitteen muisti kopioidaan kokonaisuudessaan, eli otetaan laitteesta ns. muistivedos, josta sen jälkeen etsitään paikkapistettä kuvaavia tallenteita, olivat ne sitten vielä tallella olevia, tai jo poistettuja.

2 Maailmanlaajuiset satelliittipaikannusjärjestelmät

2.1 Satelliittipaikannusjärjestelmien tausta

GNSS-järjestelmien (Global Navigational Satellite System eli Maailmanlaajuinen satelliittipaikannusjärjestelmä) tarkkaa syntyhetkeä on hankala yksilöidä, sillä se on, kuten useimmat nykypäivän tekniikat, kerta kerralta parannellun aiemman järjestelmän viimeisin versio. Ja kuten monissa teknisissä läpimurroissa, tässäkin sotilaallinen tarve oli kehittämisen motivaationa kaikkein merkitsevin.

Kun puhutaan GPS:stä (Global Positioning System), puhutaan yleensä Yhdysvaltojen asevoimien kehittämästä Navstar GPS:stä (NAVigational Satellite Timing and Ranging) [1, s. 68]. Lähes samaan aikaan Yhdysvaltojen Navstar-projektin kanssa Neuvostoliitolla oli meneillään oma GLONASS-projektinsa (GLObalnaya NAvigatsionnaya Sputnikovaya Sistema, Maailmanlaajuinen navigointi satelliittijärjestelmä), jossa oli täsmälleen samat tavoitteet [1, s. 101]. GLONASS oli pitkään tarkoitettu vain sotilaskäyttöön ja Neuvostoliiton hajoamisen jälkeen järjestelmän ylläpidosta lipsuttiin rahoituksen vähennyttyä dramaattisesti. Tämän seurauksena satelliitteja ei ollut läheskään tarpeeksi kattamaan koko maapalloa ja järjestelmä kykeni huonoimmillaan kattamaan vain osan Neuvostoliiton raunioille syntyneen Venäjän omasta alueesta. Lokakuusta 2011 alkaen GLONASSissa on jälleen ollut järjestelmän vaatimat vähintään 24 satelliittia [2].

GLONASS ja Navstar ovat tällä hetkellä ainoat maailmanlaajuiseen paikannukseen pystyvät GNSS/GPS-järjestelmät. Järjestelmät täydentävät toisiaan erityisesti siviilikäytössä, sillä uudemmat paikannuslaitteet pystyvät hyödyntämään molempien järjestelmien satelliitteja paikannukseen. Kiinalaisten Compass ja Eurooppalaisten Galileo-järjestelmät eivät tarjoa vielä moneen vuoteen tarpeeksi kattavaa satelliittiverkostoa ollakseen maailmanlaajuisia. [3; 4.]

2.2 Navstar GPS:n historia

Navstar GPS -järjestelmä on oikeastaan jatke sitä edeltävälle LORAN (LOng RAnge Navigation) järjestelmälle, joka kehitettiin 1940 -luvulla [5, s. 173]. Tätä järjestelmää käytettiin toisessa maailmansodassa laivojen sijainnin määrittelemiseen. LORAN käytti sijainnin määrittämiseen kolmiomittausta kuten kaikki nykyiset satelliittipaikannusjärjestelmätkin. Mittaukseen käytetty tekniikka ja varsinkin sen soveltaminen tosin vaihtelivat järjestelmästä toiseen.

Vuonna 1956 Friedwardt Winterberg ehdotti suhteellisuusteorian testaamista kiertoradalle sijoitetuilla tarkoilla atomikelloilla [6]. Neuvostoliiton laukaistua Sputnikinsa kiertoradalle 1957 ajatus sai lisää tulta alleen. Ensimmäinen toimiva satelliittipaikannusjärjestelmä oli Yhdysvaltain laivaston NAVSAT (Navy Navigation Satellite System), joka tunnettiin myös nimellä TRANSIT [5, s. 163]. NAVSATin päätarkoitus oli tarjota paikannustietoa ballistisia ohjuksia kuljettaville Yhdysvaltain sukellusveneille, mutta monet muutkin alukset, jopa Neuvostoliiton puolella, käyttivät järjestelmää hyväkseen. Navsatin heikkona puolena oli kuitenkin sijainnin määrityksen hitaus. Sijainnin pystyi määrittelemään vain noin tunnin välein, joskus vasta pitemmänkin aikajakson jälkeen. [7.] Tämä olisi sinällään riittänyt laivastolle, mutta esimerkiksi Yhdysvaltojen ilmavoimille tämä ei ollut lähellekään riittävä, sillä lentokone ja ballistinen ohjus ehtii kulkea tunnissa parhaimmillaan tuhansien kilometrien matkan, tässä tapauksessa ilman tarkkaa paikkatietoa.

Vuonna 1967 Yhdysvaltain laivasto laukaisi ensimmäisen Timation -satelliittinsa Maan kiertoradalle todistaen samalla, että tarkkojen kellojen vienti avaruuteen oli mahdollista [8]. 1960-luvulla rakennettu Omega Navigation System oli ensimmäinen maailmanlaajuinen, maanpäälliseen radiotekniikkaan perustuva paikannusjärjestelmä [9]. NAVSAT- ja Omega-järjestelmät eivät kuitenkaan tarjonneet riittävää tarkkuutta paikannukseen, jolle oli jo suuria tarpeita sotilas- ja siviilipuolella. Uuden, paremman, järjestelmän tekemiseen oli siis jo tilausta, mutta järjestelmän oletettua hintaa, useita miljardeja dollareita, pidettiin kynnyskysymyksenä.

Asian ratkaisivat lopulta sotilaalliset tarpeet. Kylmän sodan varustelukilvassa erityisesti sukellusveneistä laukaistavien ballististen ohjusten tarkan laukaisupaikan määrittely

paransi ohjusten iskukykyä huomattavasti. Tästä johtuen suurtakaan hintaa ei pidetty kynnyskysymyksenä järjestelmän kehittämiselle ja käyttöönotolle. Tämän päivän GPS-järjestelmän kehitys alkoi vuonna 1973 aiemmin kehitettyjen eri järjestelmien etuja pohdittaessa. Nämä järjestelmät päätettiin yhdistää uudeksi projektiksi, jonka nimeksi myöhemmin samana vuonna tuli edeltäjiensä Transitin ja Timationin satelliittien nimeämiskäytäntöjen mukaan Navstar-GPS [8].

GPS:ää kehitettiin aluksi puhtaasti sotilaskäyttöön. Neuvostoliiton hävittäjien ammuttua 1983 alas ilmatilaansa eksyneen Korean Air Linesin matkustajakoneen tappaen samalla 269 ihmistä, silloinen Yhdysvaltain presidentti Ronald Reagan lupasi GPS:n kaikkien käyttöön yleishyödylliseksi palveluksi, kunhan järjestelmä olisi tarpeeksi kehittynyt [10]. Alkuun siviilikäyttöön tarkoitettu signaali oli huomattavasti sotilaskäyttöön tarkoitettua epätarkempaa, mutta tämä rajoitus poistettiin Yhdysvaltain presidentti Bill Clintonin päätöksellä 1.5.2000 alkaen, kun oli tullut mahdolliseksi estää GPS:n käyttö mahdollisilta vihollisilta alueelliseen rajaukseen perustuen [11].

GLONASS-järjestelmän historia seuraa melko tarkkaan Navstarin historiaa, maan ja toimijoiden ollessa toki toisia. GLONASS saavutti ensimmäisen kerran täyden peiton vuonna 1995, vain vuoden Yhdysvaltojen jälkeen [12, s. 4].

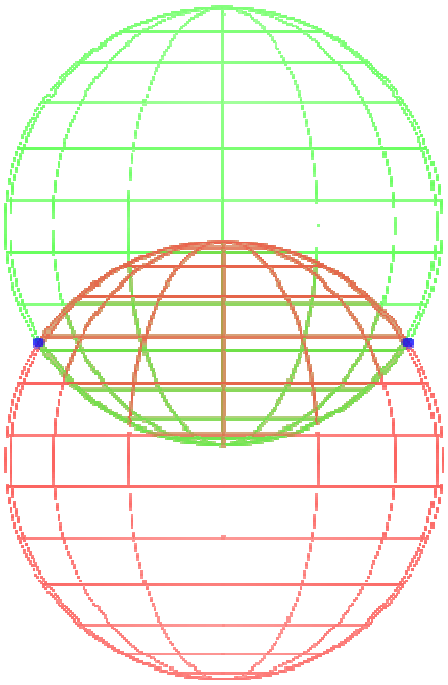
2.3 Paikannusjärjestelmien toimintaperiaate

GPS-järjestelmässä paikka määritellään laskemalla signaalia lähettävien satelliittien etäisyyttä signaalin vastaanottajaan. GPS-järjestelmä mittaa sijainnin kolmiulotteisessa avaruudessa. Satelliitit kiertävät noin 20000 km:n korkeudella Maan pinnasta, ja niiden sijainti on hyvin tarkasti tiedossa. Satelliittien signaalien vastaanottajan sijainti voidaan laskea satelliittien sijainnin ja niiden lähettämien sijainti- ja aikasignaalien signaaliviiveiden avulla.

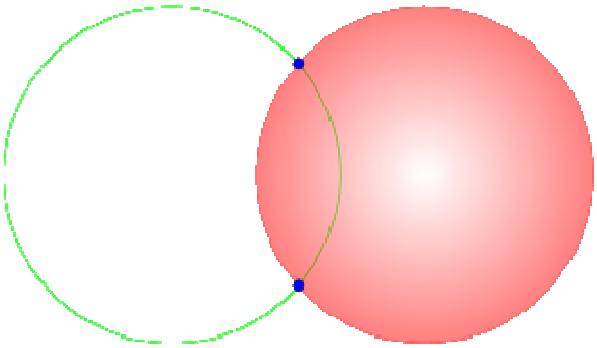
Satelliitin lähettämän signaalin eteneminen vapaasti kolmiulotteisessa avaruudessa muodostaa alati laajenevan pallokuvion. Kahden satelliitin signaaleilla, eli kahdella pallonmuotoisella säteilykuviolla, jotka menevät limittäin, saadaan leikkauspisteeksi ympyrä, jossa on ääretön määrä leikkauspisteitä. Kun tähän lisätään kolmas satelliitti, jonka

signaalin muodostama pallokuvio leikkaa ympyrän reunat, saadaan kaksi leikkauspistettä. Voidaan olettaa, että signaalin vastaanottaja on maanpinnalla tai korkeintaan joitakin kilometrejä maanpinnan yläpuolelta. Tällöin toinen leikkauspiste, joka sijaitsisi avaruudessa, voidaan jättää huomiotta. Tätä on havainnollistettu kuvioissa 1 ja 2.

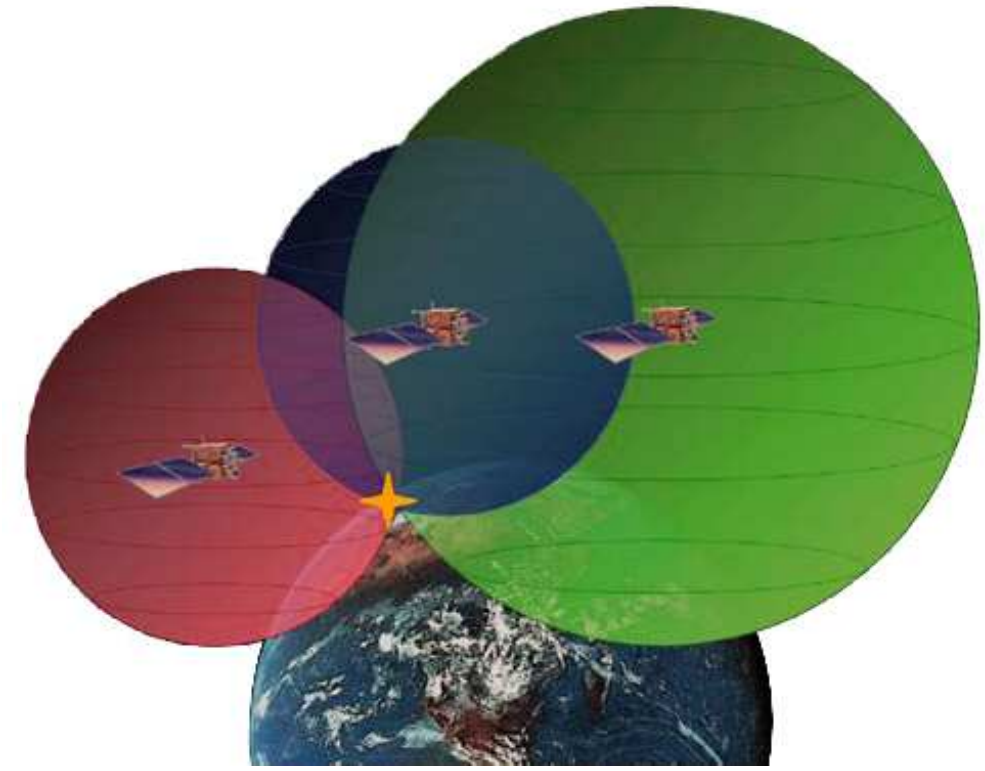
Ideaalitilanteessa paikannus onnistuisi siis kolmella satelliitilla, jotka on esitetty kuviossa 3. Tyypillisen vastaanottimen kello on huomattavasti GPS-satelliitissa olevia atomikelloja epätarkempi, joten neljättä satelliittia käytetään usein aikareferenssinä. Jos tarkka aika on tiedossa ja mittauspiste on paikallaan tai kulkee tiedossa olevalla nopeudella, periaatteessa yksikin satelliitti riittää sijainnin määrittämiseen. Tällöin joudutaan kuitenkin odottamaan satelliitin ylilennon ajan eli joitakin tunteja [13]. Myös ilmakehän eri kerrokset voivat vaikuttaa radiosignaalien kulkusuuntaan ja nopeuteen ja täten vaikuttaa lopulliseen mittaustulokseen [8]. Kellon epätarkkuuden ei tarvitse olla kummoinenkaan merkittävänkin mittausvirheen syntymiseksi. On muistettava, että radiosignaali kulkee valon nopeudella eli noin 300 000 km/s. Tällöin sadastuhannesosankin heitto ajassa vaikuttaisi paikannukseen kolme kilometriä!



Kuvio 1. Kuvassa sinisillä pisteillä merkitty kahden limittäisen pallokuvion leikkausviiva, josta muodostuu tasossa oleva ympyräkuvio [14].



Kuvio 2. Edellisen kuvan kahden pallon muodostama ympyräkuvion (vihreällä) reuna leikkaa kolmannen pallon (punaisella) kahdessa kohtaa sinisellä merkittyjen pisteiden kohdalla [14].



Kuvio 3. Kolmen paikannussatelliitin avulla suoritettu paikannus [15].

Espoossa vähintään 20° horisontin yläpuolella olevia Navstar -satelliitteja on näkyvissä viidestä yhdeksään ja GLONASS -satelliitteja kuudesta yhteentoista [16]. Näin ollen käytössä on aina vähintään 11 satelliittia sijainnin määritystä varten. Uudemmat GPS-vastaanottimet osaavat hyödyntää molempia järjestelmiä samanaikaisesti parantaen

paikannustarkkuutta etenkin syvissä laaksoissa ja taajamien ”kaupunkikanjoneissa”, eli korkeiden rakennusten väliin jäävällä katuosuudella [17].

2.4 Lyhyesti erilaisista paikannuslaitteista

Ensimmäiset GPS-vastaanottimet kykenivät ainoastaan kertomaan sijainnin tai etäisyyden tavoitellusta sijainnista. Tästä ei välttämättä ollut vielä suurta apua esimerkiksi autolla ajettaessa, sillä laitteet eivät vielä osanneet kertoa reittiä kohteeseen, ainoastaan sen sijainnin suhteessa vastaanottimeen. Jos kartta olikin saatavilla, se oli yleensä tallennettu erilliselle CD-levylle. [18 s.10; 19.]

Paikannuslaitteita on monenlaisia, eivätkä kaikki niistä ole edes tehty kulkemisen helpottamiseksi. Varsinaisia navigointilaitteita näkee usein autojen tuulilaseihin kiinnitettynä. Uudempiin autoihin se kuuluu monesti jo vakiovarusteena integroituna auton mediakeskukseen. Matkapuhelimissa, lähinnä niin kutsutuissa älypuhelimissa, se on myös jo vakiovaruste. Nykyään älypuhelimet ja autonavigaattorit osaavat kertoa, mitä reittiä haluttuun paikkaan pääsee parhaiten, ja ne priorisoivat sen tarvittaessa nopeuden, reitin pituuden tai tien päällysteen mukaan. Edellisten lisäksi moniin kameroihin on jo liitetty GPS-paikannin, jonka tehtävänä on merkitä kuviin sijaintitieto kuvien myöhempää lajittelua ja kuvien etsimistä silmällä pitäen.

Edellä kuvatuista on poliisille yleensä eniten käyttöä varsinaisten paikannuslaitteiden paikkatiedoilla. Rikosten tekijät eivät yleensä valokuvaa rikospaikkoja, eli kuvia, joissa olisi paikkadataa, ei ole kovin usein saatavilla. Toisaalta entistä useammin juuri edellä mainittu älypuhelin on ihmisten yleisesti käyttämä ”varsinainen paikannuslaite” ja sen arvo rikostutkinnassa on jo pelkästään tästä syystä kasvamassa.

3 Tiedostojärjestelmät ja tiedon tallennus

3.1 Tiedostojärjestelmistä

Paikannuslaitteissa tieto on tallennettava jonnekin eli tallennusalustalle. Jokaisella digitaalisella tallennusalustalla taas on oltava tiedostojärjestelmä. Tiedostojärjestelmän tärkein tehtävä on määritellä ne säännöt, joiden puitteissa tieto on tallennettava, jotta se olisi myöhemmin luettavissa. Tämän mahdollistamiseksi tiedostojärjestelmät pitävät kirjaa tiedostojen sijainneista, luomis- ja muokkausajoista (aikaleima), tiedostojen käyttöoikeuksista ja nimistä. Edellä kuvattu lista ei ole tyhjentävä ja erilaista toissijaista tietoa, ns. metadataa, tiedostojärjestelmät tallentavat vaihtelevia määriä eri tarkoituksiin. Yksi myöhemmin esiin tuleva ja oleellinen asia tiedostojärjestelmissä on myös se, miten ne hoitavat tallennusalustoilla olevien pienimpien tallennuslohkojen, 512 tavun kokoisten sektoreiden, ryhmittelyn. Jokaiselle sektorille ei tyypillisesti anneta omaa osoitetta, vaan näitä yhdistellään suurempiin kokonaisuuksiin, varausyksiköihin tai ryppäisiin (cluster).

Esimerkiksi yleisimmillä perinteisten tietokoneiden käyttöjärjestelmillä Linuxilla, Applen OS X:llä ja MS Windowsilla on kaikilla omat ja omanlaisensa tiedostojärjestelmänsä, joiden päälle käyttöjärjestelmät on pääsääntöisesti asennettava. Osa mainituista käyttöjärjestelmistä tukee myös toisten käyttöjärjestelmien ”nimikkotiedostojärjestelmiä”, ja osaan on mahdollista saada tuki lisäohjelmilla tai ajureilla. Varsinaisen käyttöjärjestelmän asennukseen tarjotut vaihtoehdot ovat kuitenkin tuettuja tiedostojärjestelmävaihtoehtoja suppeammat. [20; 21; 22.]

Monet tietokoneiden kanssa tekemisissä olevat tietävät, että tiedoston poistaminen eli ”siirtäminen roskakoriin” ei tuhoa tietoa, vaikka ”roskakorin” tyhjentäisikin. Kaikissa tiedostojärjestelmissä tässä toimitaan pääsääntöisesti nopeimman tavan mukaan, eli ainoastaan tiedoston sijaintiin liittyvät tiedot muutetaan kuvaamaan vapaata tallennustilaa. Tieto itsessään säilyy alkuperäisessä paikassaan, kunnes se ylikirjoitetaan uudella tiedolla. Käyttöjärjestelmien on toki mahdollista asetuksin tai lisäohjelmin huolehtia poistettuihin tiedostoihin liittyvän datan välitön ylikirjoitus.

3.2 FAT-tiedostojärjestelmät

Microsoft kehitti 1970-luvun lopussa ja 1980-luvun alussa FAT-tiedostojärjestelmän (File Allocation Table, Tiedostonvaraustaulukko), jota käytettiin Microsoftin MS-DOS käyttöjärjestelmässä. Alkujaan se oli kehitetty yksinkertaiseksi tiedostojärjestelmäksi alle 500 kt:n levykkeille. Nykyään on olemassa FAT12-, FAT16- ja FAT32-tiedostojärjestelmät, joista viimeisin on käytännössä ainoa käytössä oleva versio FATista. Numerot nimen perässä ilmoittavat, kuinka montaa bittiä tiedostojärjestelmä käyttää tiedonvaraustaulukon merkinnöissä [23].

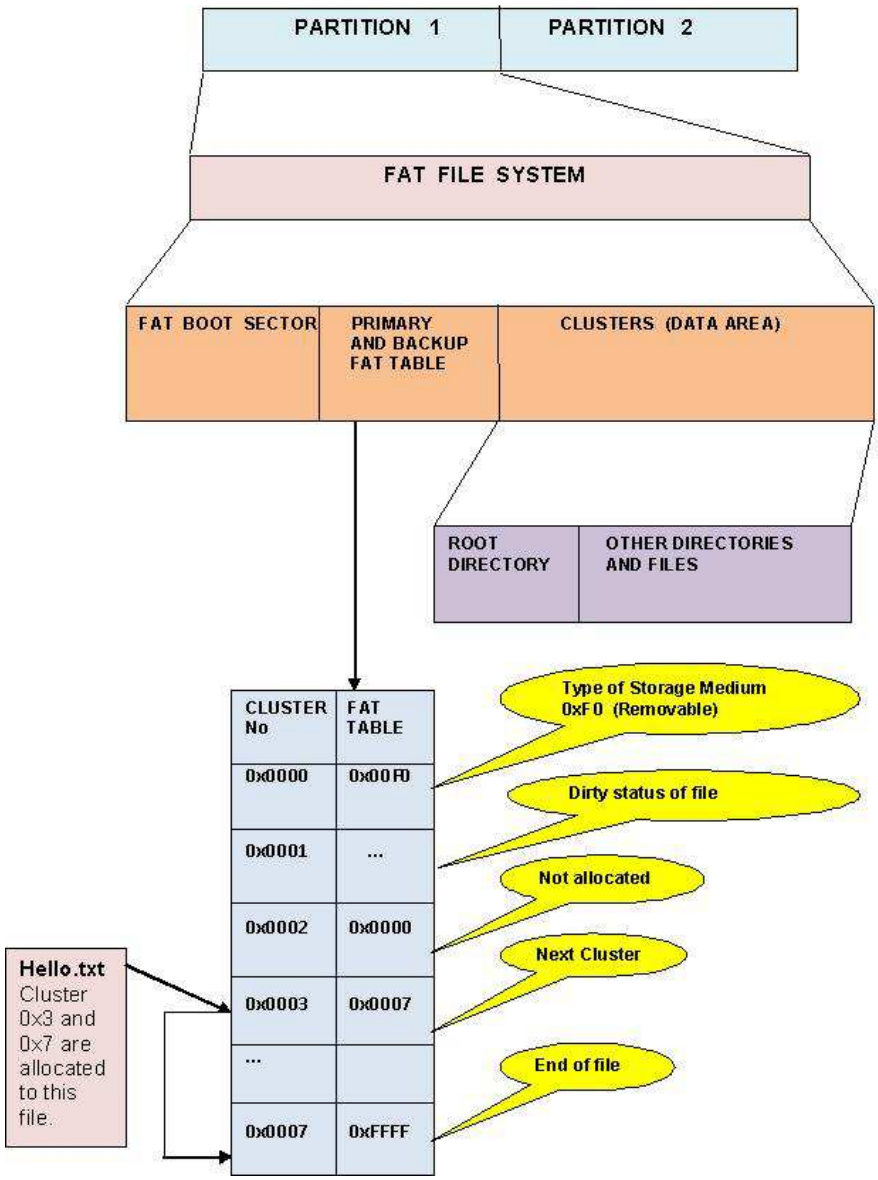
Tietojen tallennuksen kannalta FAT-tiedostojärjestelmä on kätevin, sillä se on tuettu kaikissa yleisimmissä kotikäyttöjärjestelmissä [20; 21; 22]. Näin eri käyttöjärjestelmien välillä voidaan käyttää FAT-tiedostojärjestelmällä alustettua muistimediaa tiedostojen siirtämiseen. Sen lisäksi se on pääasiallisena tiedostojärjestelmänä erilaisten sulautettujen järjestelmien ja digikameroiden käyttöjärjestelmistä esimerkiksi juuri paikannuslaitteiden käyttöjärjestelmiin.

Tämä yleiseksi muodostunut tiedostojärjestelmä tekee olemassa olevien tiedostojen etsimisestä helppoa, kun lähes millä tahansa tietokoneella voi lukea muistimedialla olevia tiedostoja. Näin esimerkiksi juuri paikannuslaitteen muistimedialta voidaan lukea tallella olevat versiot tiedostoista, joihin paikannuslaite tallentaa suosikit, äskettäin syötetyt reittipisteet ja muuten vieraillut reittipisteet riippuen toki siitä, mitä kyseisellä laitteella voi tallentaa tai mitä laite itse tallentaa.

3.3 FAT-tiedostojärjestelmän toimintaperiaate

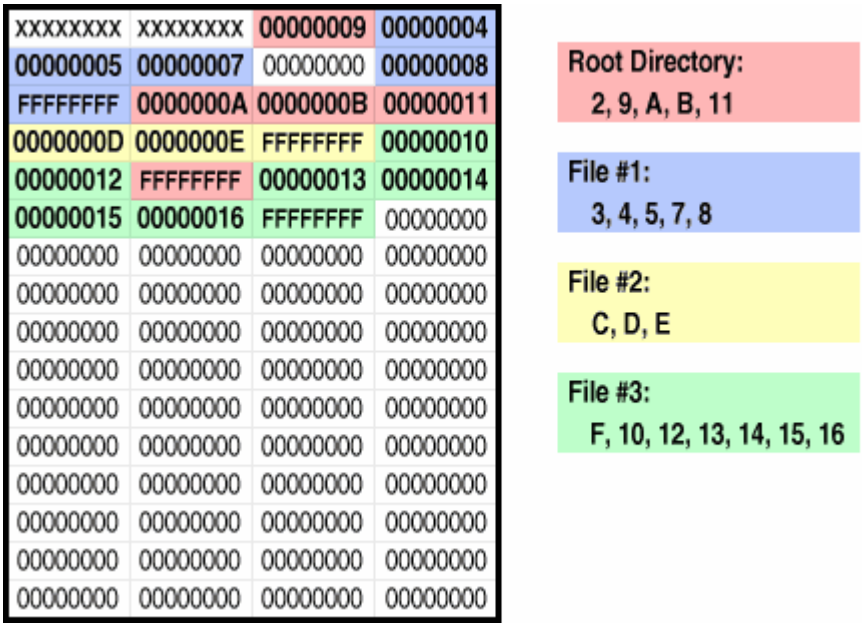
FAT-tiedostojärjestelmän ytimenä on nimensä mukaisesti tiedonvaraustaulukko, mutta se ei vielä riitä kuvaamaan koko järjestelmän toimintaa. FAT-tiedostojärjestelmässä kaiken juuressa on ns. hakemistoluettelo (directory entry). Tähän luetteloon kirjataan muun muassa tiedoston tai hakemiston nimi, tiedoston osalta koko, aikaleimat ja tiedonvaraustaulussa olevan ensimmäisen merkinnän sijainti. Edelleen tähän merkitään, jos tiedosto on poistettu.

Kaikki käytettävissä olevat varausyksiköt on määritelty tiedonvaraustaulukossa, FAT:ssa. FAT-taulukossa olevat merkinnät, FAT32:ssa 32 bittiä pitkät, kertovat, missä varausyksikössä on tiedoston seuraava osa, jos yksi ei riittänyt tiedoston tallentamiseen. Tämä tieto on samalla osoite seuraavaan kohtaan tiedonvaraustaulukkoa. Tiedoston viimeisen osan kohdalla FAT-taulukossa ei enää ole seuraavan varausyksikön osoitetta, vaan tiedoston päättymistä kuvaava bittisarja. Kuvioissa 4 on esitetty tiedonvaraustaulukon toimintaa tarkemmin. [23.]



Kuvio 4. FAT-tiedostojärjestelmän toimintaperiaate [24].

Huonona puolena FAT-tiedostojärjestelmissä on pirstaloituminen. FAT-tiedostojärjestelmissä ei ole minkäänlaista mekanismia pirstaloitumisen hallintaan tai estämiseen, vaan tiedon tallennus aloitetaan ensimmäisestä vapaasta varausyksiköstä ja sen täyttyessä jatketaan seuraavaan vapaaseen varausyksikköön, jolloin tiedoston eri osien väliin voi jäädä useita muita tiedostoja tai niiden osia [23]. Tämä hidastaa erityisesti kovalevyjen, nimenomaan perinteisten pyörivien levyjen, toimintaa. Kovalevyllä, jossa tiedostot ovat pirstaloituneet, lukupää joutuu siirtymään uuteen kohtaan kovalevyä, mahdollisesti useitakin kertoja, yhden tiedoston luvun aikana. Tämä kuluttaa aikaa ja pitää prosessoria tyhjäkäynnillä, kun se joutuu odottamaan tietoja kovalevyltä. Jälkikäteen pirstaloitumista voidaan toki korjata erilaisilla ohjelmilla, joiden tarkoitus on siirtää tiedostojen osat peräkkäisiin varausyksikköihin [25].



Kuvio 5. Tiedostojen pirstaloituminen FAT-tiedostojärjestelmässä [26].

3.4 Tiedostojen palautus FAT-tiedostojärjestelmästä

FAT-tiedostojärjestelmästä poistettujen tiedostojen palauttaminen on helppoa, jos alkuperäisten tiedostojen osat ovat vielä ylikirjoittamattomia, eli tiedonvaraustaulukon merkinnät ovat tallella. Ainoa mitä FAT-tiedostojärjestelmä tekee, kun se huolehtii tiedoston poistosta, on kirjoittaa hakemistoluetteloon tiedoston nimen ensimmäisen merkin päälle bittisarjan, joka vastaa heksamerkkejä \xE5. Seuraavien varausyksiköiden

osoitteet ovat varsinaisessa tiedonvaraustaulukossa ja pysyvät muuttumattomina. Poistettujen tiedostojen osat voidaan täten yhdistää tiedostojen mahdollisesta pirstaloitumisesta huolimatta.

Puhuttaessa FAT-tiedostojärjestelmän pirstaloitumisesta on myös syytä muistaa, että tiedoston seuraava osa on aina ”edessä”, eli järjestysnumeroltaan suuremmassa varausyksikössä. Esimerkiksi Microsoftin NTFS-tiedostojärjestelmässä on toisinaan tiedoston palautuksen kannalta hankalia ”takaisinpäin lukuja” (negative filerun), eli seuraava varausyksikkö voikin sijaita järjestyksessä aikaisemmassa varausyksikössä [23].

Jos poistettua tiedostoa kuvaava merkintä on tallella hakemistoluettelossa, mutta tiedoston käyttämistä varausyksiköistä osa on otettu uuden tiedoston käyttöön, voi tiedoston osittainen palautus onnistua vielä kohtalaisen hyvin. Pääsääntöisesti automatisoidut tiedostonpalautusohjelmat tarkastavat poistetun tiedoston koon tiedostojärjestelmän jäljellä olevista tiedoista ja alkavat rakentaa tiedostoa tiedoston tiedossa olevaa alkua kuvaavan varausyksikön kohdalta. Usein automaattiset tiedostojen palautusohjelmat jatkavat tiedoston osien kopiointia aina seuraavaan tiedonvaraustaulukossa olevaan varausyksikköön, kunnes tiedoston viimeisen varausyksikön ilmoittava merkintä tulee vastaan tai kun tiedoston tiedossa olevan koon verran tavuja on tallennettu. Toisinaan käy niin, että palautettavaan tiedostoon tulee palautettua osia, jotka eivät siihen kuulu. Usein esimerkiksi vain tiedoston alkuosa, kuvatiedostossa esimerkiksi vain yläosa tai tekstitiedoston alku, on palautettavissa, mutta toisinaan palautus voi onnistua lähes täydellisesti [23; 27].

Jos tiedoston alkukohtaan liittyvät tiedot on hakemistoluettelosta ylikirjoitettu, on tiedoston etsiminen entistä vaikeampaa, vaikka tuhotun tiedoston merkinnät tiedonvaraustaulukossa olisivat koskemattomat. Tällöin tiedoston sisältämä data on etsittävä suoraan ”levyn pinnalta” eli tallennusmedian varsinaisesta tallennustilasta. Tällöin yhtenäinen tiedosto on vielä palautettavissa, jos sillä on esimerkiksi etukäteen tunnettu ”otsikko” (header) eli tiedoston alussa oleva merkkisarja, josta tiedoston tyypin joissakin tapauksissa tunnistaa. Joskus myös pirstaloitunut tiedosto on palautettavissa, mutta se vaatii erikoistuneita ohjelmistoja tai käsityötä [27]. Pakkaamattoman tiedoston, kuten esimerkiksi tekstiä ja muita merkkejä sisältävän tiedoston kohdalla ei välttämättä

ole tarpeenkaan saada koko tiedostoa, sillä tiedoston osakin voi riittää, jos tarvittava tieto on juuri siinä osassa, jota ei ole ylikirjoitettu.

Viimeisin vaihtoehto, lähinnä tekstidatan kohdalla sen tarvitsevan pienemmän tallennustilan johdosta, on datan etsiminen tiedostojen jäännöstilasta (slack space, file slack). FAT-tiedostojärjestelmässä varausyksikön koko on 512 tavun kahden monikerta eli 512, 1024, 2048, 4096 jne. riippuen tallennusmedian koosta, 4096:n tavun ollessa oletus 256 Mt – 8 Gt:n tallennusmedioissa. Jos tällaisessa tallennusmediassa tiedoston tarvitsema tila on esimerkiksi 1347 tavua, jää 2749 tavua käyttämättä. Jos tällä paikalla on aiemmin ollut koko varausyksikön tilan tarvinnut tiedosto tai tiedoston osa, uuden tiedoston käyttämättä jättämä tila on luettavissa ja esimerkiksi juuri tekstipohjainen tieto palautettavissa sellaisenaan. Tämä tekstipohjainen tieto voi hyvinkin olla esimerkiksi myöhemmin tarkemmin kuvailtua paikkatietoa laitteen käyttämästä muistista.

3.5 Muut tiedostojärjestelmät

Myös monissa muissa tiedostojärjestelmissä on mahdollista etsiä poistettuja tiedostoja, jos tiedostojärjestelmän kirjanpitoa ei ole vielä pyyhitty yli [26; 28]. Tiedostojärjestelmien tapa kohdella tuhottujen tiedostojen sijaintimerkintöjä tosin vaihtelee. Joissakin tiedoston sijaintiin liittyvät oleelliset tiedot ylikirjoitetaan tiedoston poiston mukana tai ne ylikirjoittuvat nopeasti sen jälkeen [28; 29; 30].

Edellisen lisäksi esimerkiksi Applen OS X:n käyttö- ja tiedostojärjestelmissä on sisäänrakennettuja pirstaloitumisen hallintaan tarkoitettuja toimintoja ja useimmissa käyttöjärjestelmissä on mukana tai saatavissa erikseen ajettava ohjelma tai useampi, joilla tiedostojen pirstaloitumista voi korjata [25; 31; 32]. Tuhotun datan osalta pirstaloitumisen estämiseksi ajettavat ohjelmat ovat kuitenkin hankalia, sillä pirstaloitumisen korjaaminen väistämättä tarkoittaa datan siirtelyä paikasta toiseen. Tällöin vielä olemassa olevat, kuitenkin käyttäjän näkyvistä poistuneet tiedostot tai niiden osat tulevat helposti ylikirjoitetuiksi eikä niitä voida enää millään keinoin palauttaa.

Tässä kohtaa lienee syytä kuitenkin huomauttaa, että paikannuslaitteiden käyttöjärjestelmissä ei yleensä ole toimintoja, joilla pirstaloitumista ehkäistäisiin. Tällaiselle toimin-

teelle ei ole varsinaisesti tarvetta käytettyjen tallennusalustojen käyttäessä flash-muistia.

4 Flash-muistien toiminta tietoteknisen tutkinnan kannalta

Jokaisessa paikannuslaitteessa on jonkinlainen muisti, joissakin kaksi ja jopa kolmella erillisellä muistilla varustettu malli on tullut vastaan. Jos laitteessa on enemmän kuin yksi muisti, on näistä yleensä vähintään yksi korttipaikkaan kiinnitettävä muistikortti, yleensä SD-kortti. Sisäinen muisti, jollainen laitteista tyypillisesti aina löytyy, on kuitenkin tyypiltään flash-muistia, kuten ulkoiset kortitkin.

Flash-muisteissa tieto tallennetaan erikoistransistoreihin, joita kutsutaan kelluvahilatransistoreiksi. Niiden englanninkielinen nimi, floating gate transistor, on varmasti monille elektroniikan kanssa askaroiville paljon tutumpi. Kelluvahilatransistorien hyvänä puolena on ilman muuta se, että tiedon tallennus tapahtuu suhteellisen nopeasti. Erityisesti pienten tiedostojen osalta ero on selvä, sillä kovalevyistä tuttua hakuaikaa ei käytännössä ole, sillä siirreltävää lukupäätä ei tarvita. Ilman liikkuvia osia toimivat muistimediat kestävät myös tärinää ja tärähdyksiä huomattavasti pyöriviä kovalevyjä paremmin ja ovat siksi korvaamassa perinteiset kovalevyt monissa laitteissa [33, s. 1].

Kelluvahilatransistorit kuitenkin kuluvat käytössä huomattavasti nopeammin, koska tiedon tallennukseen tarvitaan suhteellisen korkea jännite [33; 34, s. 7]. Kun muistiyksikkö kuluu, siihen ei enää voi luotettavasti tallentaa dataa. Nykyiset flash-muistien valmistajat lupaavat yleensä 10 000 - 100 000, joillekin jopa 1 000 000, kirjoituskertaa, eli erillistä tallennusta, jokaiselle muistimedian muistialkiolle [34; 35; 36]. Varsinkaan edellä mainittu 10 000 kertaa ei kuitenkaan riitä pitkään, sillä flash-muistien toimintatavasta ja rakenteesta johtuen muutamankin isomman tiedoston kirjoittaminen voi kuluttaa esimerkiksi tiedostojärjestelmän käyttämiä osia flash-muistista erittäin nopeasti. [36.]

Tallennusalustan käyttöikää pidennetään kulumisen tasoittamisella (wear leveling). Erilaisia tekniikoita kulumisen tasoittamiseksi on useita. Osa toimii tiedostojärjestelmätasolla, kun taas osassa muistimedian ohjainpiiri huolehtii siitä, että muistialkiot kuluvat tasaisesti. Jos kulumisen tasoittamisesta huolehtii tiedostojärjestelmä, on koko muistin kopioiminen mahdollista ohjelmallisesti. Jos taas kulumisen tasoittamisesta huolehtii ohjainpiiri, on kulumisen tasoittamisen mahdollistamiseksi (sekä tuotantoprosessissa sattuvien virheiden varalta) muistimedioissa usein jonkin verran ylimääräistä tilaa. Tä-

mä tila ei ole normaalein toimenpitein käyttöjärjestelmälle tai sovelluksille nähtävissä, mutta suoraan fyysisen muistipiirin kopioimalla, on tälläkin tavalla mahdollista saada näkyviin aiemmin muistimedialla olleita tiedostoja [34; 37].

Muistipiirien fyysinen tutkiminen on kuitenkin tämän työn aihealueen ulkopuolella eikä sitä käsitellä tämän enempää. On silti hyvä muistaa, että oli tutkittava data saatu sitten tiedostojärjestelmä tai suoraan fyysinen muistipiiri kopioimalla, myöhemmin esiteltävät ohjelmalliset menetelmät paikkatietojen etsimiseksi toimivat täsmälleen samalla tavalla.

5 Paikannuslaitteiden muistimedioiden hyödyntäminen esitutkinnassa

5.1 Paikkadatan etsintä paikannuslaitteelta

Vanhemmilla paikannuslaitteilla ei voinut tallentaa juuri muuta kuin muutaman suosikkipisteensä. Nykyisillä paikannuslaitteilla, olivat ne sitten autonavigaattoreita tai muita laitteita, voi merkitä muistiin useita suosikkikohteitaan ja useimmat niistä tallentavat automaattisesti myös aiempia kohteita sekä neuvovat tien perille. Jotkut laitteet jopa tallentavat määräajoin laitteen sijainnin, jolloin kuljettua reittiä voi tarkastella jälkikäteen usein hyvinkin tarkkaan. Osa tiedon tallentumisesta vaatii käyttäjän toimenpiteitä, mutta osa tallentuu käyttäjästä riippumatta. Näin sana satelliittinavigointilaite tai pelkkä navigaattori on oikeampi sana kuvaamaan myöhemmin käsiteltäviä laitteita.

Paikannuslaitteita on tutkittu täysin automaattisin työkaluin, että heksaeditoreiden toiminnallisuuksia tarjoavia työkaluja käyttäen. Useimmiten paikkadataa oli saatavilla runsain määrin. Dataa on ollut sekä varsinaisissa tiedostojärjestelmässä olemassa olevissa tiedostoissa että allokoimattomalla alueella. Varsinkin allokoimattomalla alueella on usein moninkertainen määrä tallennettua paikkadataa, olkoonkin, että kyseessä on hyvin usein saman datan eli käytännössä alkuperäisen tiedoston aiempi tallennusversio. Joissakin malleissa on useita eri tiedostoja paikkadatan tallentamiseen ja näistäkin voidaan pitää myös tiedoston edellinen versio tallennettuna.

Toinen syy paikkadatan moninkertaiseen löytymiseen allokoimattomalta alueelta on huomattavasti spekulatiivisempi. On mahdollista, että laitteet tallentavat suosikkitiedostoja tavalla tai toisella eri paikkoihin tiedoston päivityksen yhteydessä. Tällä voitaisiin ainakin ehkäistä tiedoston turmeltuminen niissä tapauksissa, että paikannuslaitteesta loppuu virta yllättäen kesken. Asiaa on pohdittu (ainakin Suomen poliisin piirissä) laajemminkin, mutta selkeää syytä siihen, miksi allokoimattomalta alueelta voi löytyä selkeästi toistakymmentä eri versiota tai poistettua tallennetta täsmälleen samasta tiedostosta, ei ole löytynyt. Tämä asia tarvitsisi lisää tutkimista.

Tämän työn aihepiirissä ongelmaksi jää siis näiden paikkadataa sisältävien, mutta hävitettyjen tai hävinneiden tiedostojen etsiminen ilman tiedostojärjestelmää. Tällöin pitää

vain tietää, mitä tiedoston sisällä on ja suorittaa esimerkiksi tätä sisältöä vastaavan merkkisarjan etsintä. Näin esimerkiksi juuri poliisin tietoteknisessä tutkinnassa tehdään, kun haetaan vaikkapa poistettuja kuvatiedostoja. Esimerkiksi jokaisen jpg-tiedoston alussa (file header) on vakio bittisarja, josta tiedoston tunnistaa juuri jpg-tiedostoksi.

Tiedoston pirstaloitumisen myötä paikannuslaitteen muistimedialta voi löytyä paljon paikkadataa sisältävien tiedostojen osia, joissa ei ole minkäänlaista tiedoston sisäistä tunnistetta. Toisaalta, paikkadataa sisältävissä tiedostoissa ei sellaista välttämättä ole alun alkaenkaan. Näin ollen paras, ja joissakin tapauksissa ainoa tapa, on tutkia laitteiden tallentamaa tietoa ja takaisinmallintaa (reverse engineer) tiedon tallennustapa. Tähän takaisinmallintamiseen olen saanut myös arvokasta apua poliisiorganisaatiossa työskenteleviltä kollegoiltani.

Se, miten muistimedioilla olevaan dataan pääsee käsiksi, vaihtelee. Asiaan vaikuttaa kolme tiedon tallennuksen osa-aluetta, joita ovat edellä käsitelty tallennusalusta, tallennusalustan käyttöä ohjaava tiedostojärjestelmä sekä se, miten ohjelma itse tietoa tallentaa. Ohjelma voi tallentaa tiedon selväkielisenä, mutta usein erilaisin merkein tai vaikkapa xml-tagein eroteltuina kenttinä. Joskus tiedot on tallennettu osittain selväkielisenä, mutta esimerkiksi varsinaiset koordinaatit binäärimuotoisena kokonaislukumuuttujassa, jolloin niitä ei voi suoraan lukea esimerkiksi yksinkertaisella teksti- tai heksaeditorilla, vaan ne on palautettava desimaaliluvuksi luettavaan muotoon saamiseksi. Joskus tiedot ovat suoraan muistimediaa luettaessa erittäin vaikea löytää, koska data on voitu häivyttää (engl. obfuscate) tavalla tai toisella. Tämä häivyttäminen ei välttämättä tarkoita varsinaista salausta, vaan esimerkiksi merkkien muokkaamista jollakin maskilla tai merkistön kierrätystä jollakin tavalla niin, että ilman tietoa häivytykseen käytetystä tekniikasta tekstiä tai muita merkkejä ei voi lukea oikein, vaikka merkistökoodaus olisi sinänsä oikea. Näihin ei ole tässä työssä perehdytty, eikä niitä ole sen enempää tutkittukaan.

Navigaattorivalmistajiltakin on apua kysytty, ja yhtä poikkeusta lukuun ottamatta kaikki ovat kieltäytyneet kertomasta, miten he tallentavat tiedon, joko yrityssalaisuuteen vedoten tai yksinkertaisesti jättämällä vastaamatta. Asian suhteen on toki tehty paljonkin tutkimusta muiden maiden viranomaisten ja navigaattoreiden tutkintaan ohjelmia suunnittelevien yritysten taholla. Muiden maiden viranomaisten tutkimusten tuloksia

onkin käytetty, mutta yritysten työ taas on yleensä tämän yrityksen liikesalaisuudeksi tai heidän ja navigaattorivalmistajan väliseksi salaisuudeksi luokiteltua tietoa.

5.2 Erilaiset tallennustavat ja tallentuva tieto

Työssä on jo aiemmin viitannut erilaisiin tallennustapoihin sille tiedolle, jota tässä työssä tutkitaan ja nyt, valotetaan tarkemmin, mitä sillä tarkoitetaan. Ensinnäkin on huomioitava se, mitä tallennetaan. Kuten aiemmin jo mainittiin, tallennettava data vaihtelee laitteesta laitteeseen. Osa tallentaa vain osoitteen ja sen koordinaatit. Osa taas tallentaa edellisten lisäksi aikaleiman josta näkee, milloin osoite on ollut tarpeen. Osassa taas saatetaan tallentaa osoitteeseen liittyvä puhelinnumero tai muuta tietoa, jopa kuva.

Monet navigaattorit tallentavat tietoa, jonka merkitystä ei ole saatu selville, vaikka tieto itsessään on selvästi luettavissa. Tätäkään tietoa valmistajat eivät hevillä anna. Paikkatietoa sisältävä tietue voi olla aivan selväkielinen, ja kaikki data on tallennettu suoraan luettavassa muodossa, mutta tämä ei auta, jos ei tiedä mikä merkitys tietueeseen kirjatulla tiedolla on.

Tieto on voitu tallentaa myös eri merkistökoodauksilla. Osa voi olla koodattu laitetta tutkivan kannalta helposti ymmärrettävästi ja eri käyttöjärjestelmissä yleisesti tuetulla Iso-Latin 1:llä tai jollakin versiolla UTF:stä. Tämäkin on selvitettävä ja otettava huomioon mikäli, haluaa saada erilaiset hakumenetelmät toimimaan yhtäläisesti ja tulokset näkymään siististi ja käytännöllisesti.

Usein esimerkiksi koordinaatit on tallennettu binäärimuotoisina 32-bittisinä kokonaislukuina, vaikka kaikki muut tekstit ja numerot ovat tallennettu merkkimuotoisina. Kokonaisluvut käsittelevät nimensä mukaisesti vain kokonaislukuja ja paikkapistettä kuvaava lopputulos on saatavissa vastan jonkin laskutoimituksen tai laskutoimituksien jälkeen.

Aivan viimeisimpänä ja tiedon etsinnän kannalta hankalimpia ovat ne valmistajat, jotka tallentavat joko suurimman osan tai kaikki sijaintiin liittyvät tietonsa jollakin omalla koodaustavallaan. Vaikka olisi itse luonut paikkamerkinnät ja näkisi laitteen käyttöliit-

tymän läpi kaiken sen tiedon, mitä laite käyttäjälle näyttää, voi paikkatiedon selvittäminen suoraan paikkatietoa sisältävää tiedostoa tarkastelemallakin olla äärimmäisen haastavaa.

5.3 Paikkatietueita etsivien ohjelmien tekeminen

Tätä työtä varten ei ole hankittu paikannuslaitteita tai tehty niillä itse kenttäkokeita, vaan turvauduttu ainoastaan niihin laitteisiin, joita minä tai kollegani ovat saaneet työtehtävissään tehtäväksi tutkia. Laitteiden hankintaan ja toiminnan takaisinmallintamiseen omin kenttäkokein ei ole ollut tätä työtä tehtäessä mahdollisuutta, vaan olen tutkinut ainoastaan jonkun muun tallentamia tai jonkun muun liikkumisen seurauksena laitteen tallentamia tietoja. Omat kokeet olisivat saattaneet antaa arvokasta lisätietoa tallennettujen paikkapisteiden yhteydessä tallennetusta datasta. On myös syytä muistaa, että navigaattorivalmistajilla voi olla myös mallikohtaisia eroja siinä, mitä ja miten tietoa tallennetaan. Uudemmissa laitteissa voi myös olla uusien teknisten ratkaisujen lisäksi aivan uudenlainen tapa tallentaa paikkatietoja.

Ohjelmien tarkoituksena on siis etsiä laitteilla käyttäjän tai laitteen oman ohjelmiston toiminnan yhteydessä tallennettuja paikkatietoja, jotka muodostuvat yleensä osoitteesta ja koordinaateista. Nämä paikkatiedot tallennetaan paikannuslaitteissa usein eri tiedostoihin riippuen siitä, onko kyseessä niin sanottu suosikkipaikka vai kenties viimeksi tallennettu paikka. Yhden paikan kuvausta kutsutaan tässä insinöörityössä paikkatietueeksi ja tietokoneohjelmien tarkoituksena on etsiä näitä paikkatietueita riippumatta siitä, ovatko ne vielä käyttäjän luettavissa vai onko tiedosto poistettu tai jopa osittain ylikirjoittunut. Kaikki paikkatietueet haetaan kuitenkin kokonaisina, sillä pelkkien koordinaattien haku ei useimmissa tapauksissa ole käytännössä mahdollista. Esimerkiksi pelkät binäärimuodossa olevat neljän tavun kokonaisluvut on usein mahdotonta erottaa muista neljän tavun merkkijonoista pelkän sisällön perusteella.

Kolme neljästä tehdystä ohjelmasta käyttää niin sanottuja säännöllisiä lausekkeita datan etsimiseen. Säännölliset lausekkeet ovat etukäteen määriteltyjä määrämuotoisia merkkijonoja. Säännöllisiä lausekkeita käytetään siis etsimään määrämuotoista dataa, johon nämä merkkijonot sopivat. Määrämuotoinen data on siis tietoa, jossa on joitakin

vakio-osia. Nämä vakio-osat voivat olla esimerkiksi sanoja, merkkejä tai vaikkapa toistuvia bittisarjoja. Vakio-osa ei kuitenkaan automaattisesti tarkoita tietyn pituista merkkijonoa, vaan se voi myös olla esimerkiksi jollekin eri kappalemäärien välille osuva joukko kirjaimia tai numeroita. Koska osoitetietueet ovat yleensä hyvin määrämuotoisia, on säännöllisten lausekkeiden käyttö näiden tietueiden etsintään perusteltua. Valitsin säännöllisten lausekkeiden käsittelyyn PCRE-kirjaston (Perl Compatible Regular Expression), johon Google on tehnyt C++-liittymäpinnan. PCRE on tiuhaan päivitetty ja hyvin dokumentoitu [38].

Kun osoitetietueita haetaan, pitää myös määritellä se, että mitkä osat kutakin osoitetietuetta ovat vakioita ja mitkä eivät. Näitä vakio-osia voi käyttää säännöllisen lausekkeen ”ankkureina” eli kohtina joiden on osuttava ohjelmiin ohjelmoituihin säännöllisiin lausekkeisiin, jotta osoitetietue tunnistuu juuri haetunkaltaiseksi osoitetietueeksi. Tämän lisäksi on tiedettävä osoitetietueen ehdolliset eli ei-vakiot osat ja niiden tallennustapa. Osoitetietueen eri osista on myös tiedettävä niiden minimi ja maksimi pituudet. Esimerkiksi vaihtelevan pituisesta vakio-osasta käy tekstikenttä jossa on osoite tai muu paikkaan liittyvä yksilöivä selite. Se voi Suomessa olla esimerkiksi kaksikirjaiminen paikkakunta (Ii) tai pitkä kadunnimi talon numeron, rappukirjaimen ja asunnon numeron kera. Itella Oyj:n ohjeen mukaan 50 merkkiä pitää riittää katuosoitteeseen myös silloin, kun kyseessä on yritys [39, s. 5].

Merkkijonon maksimipituudella on merkitystä, sillä ohjelmien testauksessa havaittiin, että esimerkiksi säännöllisen lausekkeen yhdelle osalle annettu mahdollisuus äärettömään määrään merkkejä kuormittaa ohjelmalle varattua muistitilaa, eikä ohjelma toimi toivotusti. Ero oli vieläpä varsin merkittävä. Esimerkiksi testauksessa käytetystä Route66:n muistivedoksesta saatiin vain vajaa 300 tietuetta kun merkkijonojen pituutta ei oltu rajoitettu ja ohjelman ajaminen kesti noin 15 minuuttia. Kun merkkijonot rajoitettiin Itellan ohjeen mukaan 50 merkkiin, löysi ohjelma yli 7000 osoitetietuetta ja vieläpä puolet lyhyemmässä ajassa.

Koordinaattipisteet tallennetaan ainakin Navstar GPS -järjestelmää hyväksi käyttävissä laitteissa WGS-84 -koordinaattijärjestelmällä (World Geodetic System 84), joka on voitu todentaa laitteita tutkittaessa [43]. Esittelen ohjelmissa tehtyjä ratkaisuja jonkin verran navigaattoreiden ja niiden tallennustapaa esittelevässä kohdassa. Ohjelmien lähdekoo-

deissa, jotka ovat tämän työn liitteinä, on myös kommentoitu ohjelmien eri osia. On myös syytä huomauttaa, että esimerkeissä esitetyt paikkapisteet ovat täysin mielikuvituksellisia, mutta todellisia tallenteita vastaavia.

Tehtyihin ohjelmiin voi myöhemmin ilmetä muutostarpeita, jos laitteita tutkittaessa huomataan, että ohjelma ei toimi tarpeeksi täydellisesti. Rajallinen määrä testidataa ei välttämättä tarjonnut kaikkia niitä variaatioita, jotka ovat mahdollisia. Varsinaista testidataa tämä työ ei esittele, sillä se ei ole julkista tietoa. Tekemäni takaisinmallintamisen oikeellisuuden olen voinut tarkistaa juuri paikkaa kuvaavia osoitetietoja ja koordinaattipisteitä keskenään vertailemalla eli syöttämällä koordinaatit johonkin karttapalveluun ja katsomalla mihin osoitteeseen nämä vievät [40; 41].

5.4 Tutkimuksen kohteena olleet navigointilaitteet

5.4.1 Nokia ja Route 66

Route 66 on ensimmäinen tutkinnan kohteena ollut navigaattori, jota tutkittiin heksaeditoria toiminnallisuuden tarjoavalla työkalulla. Tietueet ovat selväkielisiä, 32-bittisinä kokonaislukuina tallennettuja koordinaatiopisteitä lukuun ottamatta. Tämän havainnon tekemistä vaikeutti suuresti ensimmäisenä käsittelyyn tullut Route 66 -navigaattori, jonka toinen koordinaattia ilmoittava kokonaisluku oli aina 2147483647. Silloin ei ollut vielä tiedossa, että luku pitäisi jakaa nimenomaan 100 000:lla vaan oletettiin, että jälkimmäisessä luvussa oli vain pari desimaalia enemmän. Näin ollen koordinaattipari sattui kuitenkin Suomen alueelle ja alkuun epäiltiin, että käytössä oli vain väärä koordinaattijärjestelmä. Kun viimein huomattiin, että heksana luku oli \x7FFFFFFF, ymmärrettiin kyseessä olevan laitteen toimintavirheen tai jonkinlaisen oletusarvon. Laite on siis tallentanut suurimman positiivisen luvun, jonka etumerkillisellä 32-bittisellä kokonaisluvulla voi ilmaista.

Route 66 ja aikanaan Nokian julkaisemien navigaattoreiden paikkapisteiden tallennusperiaate on esitetty taulukossa 1. Taulukossa kohta TIETO kertoo, miten tieto on tallennettu eli miltä se näyttää kun sitä lukee esimerkiksi heksaeditorilla, AINA? kertoo, onko tieto tallennettu jokaiseen paikkapistettä kuvaavaan tietueeseen, ja SELITE kuvaa

lyhyesti, mitä kukin tietueen osa merkitsee. Näissä navigaattoreissa käytetään utf-8 -merkistökoodausta.

Taulukko 1. Nokian ja Route 66 navigaattoreiden tallentama data [42].

TIETO	AINA?	SELITE
<#>	E	Talonumero tai kysymysmerkki, pienempi kuin ja suurempi kuin merkkien välissä.
katu/paikka	K	Kadun tai paikan nimi
[kaupunki]	E	Kaupungin nimi kulmasulkeissa, mahdollisesti useammalla kielellä. Eri versiot nimestä eroteltu puolipisteellä ja välilyönnillä.
(#####)	E	Postinumero kaarisulkeissa. Esimerkiksi autokorjaamoa kuvaavassa merkinnässä postinumeron tilalla voi olla lisätieto.
lisätieto	E	Esimerkiksi yrityksiä kuvaavassa paikkatiedoissa osoite voi näkyä selväkielisenä lisätiedossa. Tässä voi olla tallessa myös esimerkiksi puhelinnumero tai muuta tietoa.
[aikaleima]	K	Aikaleima muodossa pp.kk, hh:mm kulmasulkeiden sisällä
longitudi	K	Longitudi tallennettuna 32-bittisenä kokonaislukuna, joka pitää jakaa 100000:lla varsinaisen koordinaattipisteen saamiseksi.
latitudi	K	Latitudi tallennettuna 32-bittisenä kokonaislukuna, joka pitää jakaa 100000:lla varsinaisen koordinaattipisteen saamiseksi.

Tavanomainen katuosoite kiinteistön numeron kera tallentaa kaikkiin tietueen eri osiin, paitsi lisätietoon, siihen sopivaa dataa. Muuta paikkaa tai paikkakuntaa kuvaava tietue taas voi pitää sisällään ainoastaan vakio-osat. Lisätieto taas voi olla esimerkiksi jotakin yritystä kuvaavassa tietueessa yrityksen osoite ja puhelinnumero, jos paikan kohdalle on tallennettu vain yrityksen nimi. Nokiaa ja Route66:ta varten tehty ohjelma osaa lukea kaikki variaatiot.

Esimerkki osoitteesta heksa- tai tekstieditorilla luettaessa:

<12> Esimerkkitie [Esbo; Espoo] (02123) [01.01, 12:00] ... 7.[[42]

Esimerkissä loppupään merkit kuvastavat 32-bittisen kokonaisluvun tavukohtaista tulkkausta utf-8:lla. Jokaisen tietueen jälkeen on nolla-arvo, eli \x00. Kaikki ”tyhjät” merkit eivät myöskään ole välilyöntejä eli \x20, vaan saattavat olla myös jompikumpi rivinvaihto merkki \x0a (uusi rivi, new line) tai \x0d (rivinvaihto, carriage return). Ohjelmassa teksti vakioidaan näiden merkkien osalta ja rivinvaihtomerkit vaihdetaan välilyönneiksi ennen tietueen tallentamista tiedostoon. Ilman tätä vakiointia tietue tallentuisi useammalle riville ja vieläpä epäsäännöllisesti, sillä rivinvaihtomerkkejä on erityisesti lisätietoa sisältävässä kentässä.

5.4.2 Mio ja Navman

Mio ja Navman molemmat tallentavat tiedot samanlaisessa xml-muodossa. Kaikkien eri tietueen osien merkitystä ei ole saatu selville, mutta oleellisimmat ovat onneksi itsestään selviä. Taulukossa 2 on esitelty eri tagit ja niiden rajaama tieto, jos tagien rajaama tieto on tunnistettu. Tiedot on esitetty aina tagien nimillä ja epäselväksi jääneiden tagien merkitys on selitteessä esitetty vain kysymysmerkillä ja mahdollisesti muita merkityksiä sisältävien kohdalla kysymysmerkki on selitteen lopussa sulkeissa. Monien epätietoisuutta aiheuttavien merkintöjen kohdalla tallennettu tieto on vain numero, joka toisinaan voi olla myös negatiivinen. Usein jotakin tietoa ei ole edes tallennettu, mutta kaikille on varattu paikkansa jokaisessa paikkapistettä kuvaavassa tietueessa. Kaikkien eri tagien rajaamaa tietoa ei ohjelmalla haeta, sillä tieto jonka tarkoitusta tai syytä ei tiedä ei hyödytä mitään. Kaikista oleellisin tieto, eli varsinainen sijainti saadaan kuitenkin luettua. Sulkuihin on merkitty tagit, joiden rajaamaa tietoa ei ohjelmalla haeta.

Taulukko 2. Mio ja Navman navigaattoreiden tallentaman paikkadatatietueen eri osat [42]

TAGI	SELITE
(<Item class=”xxxx”>)	Minkälainen paikkatietue on kyseessä: suosikki, monen paikkapisteen reitti vai jokin muu. Tämä on kirjattu lainausmerkkien väliin (?)
(<DisplayName>)	Näytöllä näkyvä nimi (?)
(<Behaviour>)	?
<LastUsed>	Nollasta alkava järjestysnumero, joka ilmoittaa monennesko paikkapiste on tallentuneiden paikkapisteiden listalla tiedoston sisällä.
(<StickyOrder>)	?
(<Object class=”xxxx”>)	Minkätyyppinen tietue on kyseessä (?)
<name>	Paikkapisteen nimi
<name2>	Toinen nimikenttä, mahdollisesti lisätiedoille (?)
<lat>	latitudi (selväkielisenä, mutta täytyy jakaa 100000:lla)
<long>	longitudi (selväkielisenä, mutta täytyy jakaa 100000:lla)
<entryName>	Kadun tai paikan nimi
<entryLat>	Latitudi, kenties joskus itse syötetty (Testidatassa tässä oli sama arvo kuin aiemmassa latitudi-kentässä tai pelkkä nolla) (?)
<entryLong>	Longitudi, kenties joskus itse syötetty. (Testidatassa tässä oli sama arvo kuin aiemmassa longitudi-kentässä tai pelkkä nolla) (?)
(<entryHeading>)	Kuljettava suunta. Onko kyseessä linnuntietä kuljettu reitti lähtöpisteestä määränpäähän vai jokin muu, ei ole selvinnyt. Siitäkään ei ole tietoa, että mitä suuntaa ilmoittavaa järjestelmää käytetään. (?)
(<entrySideOfLine>)	?
<entryHouseNo>	Katuosoitteeseen liittyvä talon numero
<entryPostcode>	Postinumero
(<entryResolved>)	Onko kuljettava/käytettävä reitti löytynyt (testidatassa tallentuneet arvot olivat joko 0 tai 1) (?)
(<entryValid>)	Onko reitti kelvollinen (testidatassa tallentuneet arvot olivat joko 0 tai 1)(?)
(<entryMaxRoad>)	?
(<entryHalfSize>)	?
(<source>)	?
(<resultType>)	?
(<photo>)	Kuvatiedoston sijainti tai pelkkä nimi (?)
(<typeName>)	?
<phoneNum>	Puhelinnumero
<description>	Kuvaus. Esimerkkejä sisällöstä ei testidatassa ollut. (?)
<type>	Paikkapisteen tyyppi ?

Näiden navigaattoreiden kohdalla tehdyn ohjelman toimintaperiaatteena oli vain ottaa vastaan annetun tagin sisältö merkkijonona ja etsiä tagien rajaama informaatio suoraan tiedoston kopiosta ja sen jälkeen täydentää sitä seuraavien tagien rajaamalla informaatiolla. Jos kyseessä oli koordinaattipisteen haku, tämä luettiin luvuksi ja tehtiin tarvittavat laskutoimitukset. Seuraava esimerkki yhden paikkapisteen sisältävästä tietueesta valottaa asiaa. Tietueeseen tallennetut paikkaa kuvaavat tiedot on lihavoitu.

```
<Item class="Favourite">
<DisplayName></DisplayName>
<Behaviour>1</Behaviour>
<LastUsed>2</LastUsed>
<StickyOrder>-1</StickyOrder>
<Object class="Location">
<name>Nimi</name>
<name2></name2>
<lat>1234567</lat>
<long>7654321</long>
<entryName>Testikatu</entryName>
<entryLat>1234567</entryLat>
<entryLong>7654321</entryLong>
<entryHeading>225</entryHeading>
<entrySideOfLine>1</entrySideOfLine>
<entryHouseNo>12</entryHouseNo>
<entryPostcode></entryPostcode>
<entryResolved>1</entryResolved>
<entryValid>1</entryValid>
<entryMaxRoad>0</entryMaxRoad>
<entryHalfSize>0</entryHalfSize>
<source>5</source>
<resultType>3</resultType>
<photo></photo>
<typeName></typeName>
<phoneNum></phoneNum>
<description></description>
<type>road</type>
</Object>
</Item>                                    [42]
```

5.4.3 Navigon

Navigon tallentaa datan selväkielisenä ja suoraan luettavana, mutta senkään kaikkia tiedostoon tallentamien eri tietueiden osien merkitystä en ole saanut selville. Tekstin merkkikoodaus on todennäköisimmin ISO Latin 1, mutta se voi olla myös ISO Latin 15. Oleellisen tiedon tästäkin laitteesta saa hyvin esille ja esitettävään muotoon, mutta koko tietueen purkaminen osiin pidentää ohjelman ajoaikaa, eikä sitä ole siksi tehty. Koko tietueen purkamiseen liittyvä ongelma havainnollistuu parhaiten, kun esitellään kaksi samassa tiedostossa sijaitsevaa, mutta aivan eri määrän informaatiota sisältävää tietuetta.

[||][0][6]|TUUSULA|04123|20.12345|60.12345[3]|UUSIMAA|[2]|Etelä-Suomi|Etelä-Suomi|3842[0]|Suomi||15

[||][0][17]|ESIMERKKITIE|00312|20.12345|60.12345[16]|42|[8]|ESIMERKKITIE|00300|20.11223|60.11223[6]|HELSINKI|00100|20.11111|60.22222[3]|UUSIMAA|[2]|Etelä-Suomi|Etelä-Suomi|3842[0]|Suomi||15 [42]

Nämä näyttäisivät vielä hiukan erilaisilta, jos kyseessä olisi esimerkiksi suoraan koordinaattipisteinä ja käyttäjän nimeämällä nimellä tallennetut paikkapisteet. Ilmeisesti hakasuluissa olevilla numeroilla olisi jokin merkitys, kun tarkastellaan, kuinka paljon dataa kustakin paikkapisteestä tallennetaan. Ne voivat myös ottaa kantaa siihen, minkälainen paikka on kyseessä. Onko kyseessä esimerkiksi jonkin kaupungin keskusta, kuten ensimmäisessä ilmeisesti on, vai kenties kaupungissa sijaitseva katu, kuten jälkimmäisessä.

Ensimmäisessä esimerkissä on kyseessä ollut ilmeisesti Tuusulan keskusta ja loppuosa tietueesta kertoo, missä osassa ja missä maassa tämä Tuusulan keskusta sijaitsee. Toisessa tietueessa on ollut katu Helsingissä. Tässä tietueessa on esimerkiksi koordinaatteja kolmet kappaleet, joista ensimmäinen on varsinainen paikkapiste, toinen ilmeisesti se piste, mihin navigaattori neuvoo, jos ei syötä talon numeroa vaan pelkän kadun, ja kolmas navigaattorin valmistajan tai karttapohjan toimittajan näkemys kyseisen kaupungin keskipisteestä tai paikasta, johon navigaattori neuvoo menemään, jos kohteena on pelkkä kaupunki ilman tarkempaa osoitetta. Tietueen alun eri osat on vielä esitelty jäljempänä olevalla esimerkillä, jossa on kauttaviivalla (/) erotettu ne eri vaihtoehdot, joita jokin tietueen osa voi pitää sisällään. Pohjana on käytetty edellä olevista esimerkeistä jälkimmäistä.

[Käyttäjän tietueelle antama nimi||][0][17]|Tie/Paikka/Koordinaattipisteet käyttäjälle näkyvänä|Postinumero|longitudi|latitudi[16]|
Talon(kiinteistön)numero/Kaupunki|[8]|ESIMERKKITIE|00300|
20.11223|60.11223[6]|HELSINKI|00100|20.11111|60.22222[3]|UUSIMAA|
[2]|Etelä-Suomi|Etelä-Suomi|3842[0]|Suomi||15 [42]

Kuten edellä kerrottiin, tehty ohjelma purkaa vain alkuosan tietueesta omiin sarakkeisiinsa, näiden purettavien tietojen ollessa niitä, jotka on yllä avattu. Tietueen loppuosa tallennetaan ylimääräisenä tietona, kun siitä on ensin korvattu hakasulut ja erotinmerkit välilyönneillä. Nämä erikoismerkit ovat kuitenkin hyödyllisiä tietueiden etsinnässä, sillä ne toimivat hyvin ankkureina, eli säännöllisen lausekkeen vakio-osina. Tietueen

perässä on myös rivinvaihtoon käytetyt kaksi merkkiä, jotka heksana esitettynä ovat \x0d0a.

5.4.4 iGO 8 ja iGO Primo eli Becker ja Blaupunkt

Blaupunkt ja Becker molemmat käyttävät iGO 8 tai iGO Primo navigaattoriohjelmistoja laitteissaan. Nämä voivat tallentaa käytetyn reitin, kaupungin tai osoitteen. Paikkapisteen sisältävää tietuetta en ole purkanut itse, vaan käytetyt tiedot perustuvat täysin Keskusrikospoliisilta saatuun asiakirjaan [42]. Tämäkään ei ole täysin tyhjentävä, mutta kattaa jälleen tärkeimmäin. Näille laitteille tehty ohjelma pureutuu ainoastaan syötettyihin osoitteisiin. Myöhemmin taulukossa 3 esitettävä paikkatietueen rakenne sisältää myös ehdollisia kohtia, eli tietueen osia, joita ei joka paikkapistettä tallennettaessa tallenneta. Tietueen alussa on myös hyvin paljon tietoa, joka pitää laskea auki, ja siksi tietueen eri osat on esitelty osien tarvitsemaa tavumäärää (eli kahdeksan bitin muodostamaa merkkiä) esittävässä taulukossa.

Tietueen alkuosa on saadun testidatan perusteella yhtä poikkeusta lukuun ottamatta kiinteämittainen ja tätä tietoa käytetään hyväksi kun tietue puretaan osiin ja luettavaan muotoon. Navigaattorin paikkatietueen tekstiosuudet on myös pääsääntöisesti tallennettu siten, että joka merkin jälkeen tulee nollamerkki eli \x00. Tämä viittaisi ensisijaisesti unicode-koodaukseen, mutta tätä ei ole saatu varmistettua. Merkkien lukeminen PCRE:tä käyttäen pitäisi onnistua myös silloin kun merkit ovat unicode-muotoisia, mutta tätä ei saatu toimimaan ja on huomattava, että kaikki teksti ei kaikissa osoitteissa ole unicode-muotoista. Ongelma unicode-merkkien lukemisessa on kierretty siten, että nollat poistetaan tekstin seasta ennen tiedon tallentamista tiedostoon. Nollamerkkiä ei useinkaan voi tallentaa sellaisenaan, se kun tulkitaan ohjelmoinnissa usein merkkijonon päättymismerkiksi. Siksi valtaosa niin sanotusta tekstipohjaisesta datasta on alla esitetty vievän tuplamäärän tavuja etsittävään tekstiin nähden, sillä nämä nollamerkit on otettu mukaan kun datan tallennustapaa käsitellään. Tämä näkyy säännöllisessä lausekkeessa esimerkiksi osoitteen pituutena, joka on Itellan ohjeen mukainen 50 merkkiä kaksinkertaisena, eli 100 merkkiä. Taulukossa 3 on esitelty iGO-laitteiden paikkatietueiden rakenne. TAVUT ilmaisevat kuinka monta tavua tietueen osa vie, AINA? onko tietueen osa aina tallennettu ja SELITE kertoo, mitä siihen kohtaa tietuetta on tallennettu.

Taulukko 3. iGO 8:n ja iGO Primon paikkatietueiden tallennustapa [42, 43].

TAVUT	AINA?	SELITE
1 - 3	K	Ilmeisesti jotakin tietueeseen, kokoon tai rakenteeseen, liittyviä merkkejä. Merkitystä ei ole selvitetty eikä tavumääräkään ole aivan varma. Näitä merkkejä ei ohjelmalla etsitä.
2	K	Vuosi kahden tavun (16-bitin) little-endian kokonaislukuna. (Merkitsevin tavu on \x07 vuoteen 2047 asti ja siksi sitä käytetään ensimmäisenä säännöllisen lausekkeen ”ankkurina”!)
1	K	Päivämäärän kuukausi yhden tavun (8-bitin) kokonaislukuna.
1	K	Päivämäärän päivä yhden tavun (8-bitin) kokonaislukuna.
4	K	Kellonaika neljän tavun (32-bitin) little-endian kokonaislukuna. Kellonaika tallennetaan millisekunteina vuorokauden alusta lähtien GMT/UTC-ajassa.
4	E	Jos navigaattoriin asetettu osoitetieto on asetettu määränpääksi (koordinaattipisteiden jälkeen tallennettava tieto), tässä on neljä tavua, jotka testidatassa olivat kaikki \xffffffff.
4	K	Longitudi neljän tavun (32-bitin) little-endian kokonaislukuna. Tämä luku pitää vielä kertoa luvulla 1.1920929e-07:lla, jotta saadaan varsinainen koordinaattipiste. Kertojana oleva luku ilmaisee moneenko osaan yksi aste on jaettu.
4	K	Latitudi neljän tavun (32-bitin) little-endian kokonaislukuna. Tämä luku pitää vielä kertoa 1.1920929e-07:lla, jotta saadaan varsinainen koordinaattipiste. Kertojana oleva luku ilmaisee moneenko osaan yksi aste on jaettu.
16 - 26	K	Osoitteen tallennustapa alla olevan listan mukaisesti: – hist_address: navigaattoriin on annettu kadun nimi – hist_geo: navigaattoriin on annettu paikan koordinaatit – hist_cos: navigaattoriin on annettu pelkkä kaupunki – hist_flag: navigaattoriin annettu paikka on asetettu määränpääksi – hist_junction: navigaattoriin on annettu osoitteeksi risteys
2	K	\x0000
6	K	sys (eli ”sana” sys nollamerkit välissä)
2	K	\x0000
4 - ?	K	Osoite ilman postinumeroa tai paikkaan liittyvä selite
4	E	Maa edeltävällä alaviivalla, esim. _FIN tai _SWE. Välissä EI ole nollamerkkejä.
4 - ?	E	Postinumero, paikkakunta ja kadun nimi
6 (- 8)	K	\x000000000000 nollamerkkien jono tietueen lopussa. Ainakin yhdessä tapauksessa yhden keskellä olevan nollamerkin tilalla oli \x01

5.4.5 TomTom

TomTom on maailman johtava navigaatiolaitteiden valmistaja ja siksi sitä on tutkittukin maailmanlaajuisesti eniten. TomTomiin on ollut jo vuosia kohtuullisen hyvä tuki erilaisin ohjelmin tai lisäosin jo käytössä oleviin tietoteknisen tutkinnan ohjelmiin. Laitteen yleisyydestä huolimatta käytössä ei ollut sopivaa testidataa, mutta Iso-Britannian Metropo-

litan Policen tutkijan aiheeseen liittyvästä selvityksestä on seuraavassa oleelliset tiedot[44].

TomTomeissa jokainen osoitetietue alkaa (lähes) samalla bittisarjalla. Heksamerkein esitettynä tämä 14 merkkiä pitkä alkuosa tai ”otsikko” on seuraavanlainen:

04 00 XX 00 00 00 04 00 YY 00 00 00 08 00

jossa XX:n ollessa:
01 kaupungin keskusta
02 risteys
03 talon tai kiinteistön numero
04 mikä tahansa kohta katua

ja YY:n ollessa:
01 syötetty laitteen karttaa katsomalla tai postinumerolla
03 suosikki
04 koti
05 syötetty osoitteen avulla
06 syötetty kiinnostavan paikan (Point Of Interest, POI) kautta
07 viimeisen lasketun osoitteen alku [44]

Taulukossa 4 on esitelty TomTomin tallentamien paikkatietueiden eri osat. Taulukossa TAVUT kertovat kuinka monta tavua tietueen osa vie ja SELITE mitä näihin tavuihin tallennettu tieto merkitsee.

Taulukko 4. TomTom navigaattoreiden paikkapisteen [44]

TAVUT	SELITE
14	Otsikko
4	Longitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla.
4	Latitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla.
2	\x0800
4	Toinen longitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla. Ilmoittaa lähimmän tien.
4	Toinen latitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla. Ilmoittaa lähimmän tien.
1 tai 3	Nimen ensimmäisen osan pituus, johon on lisätty \x1b. Pelkkä \x1b on tyhjä. Tavu on \x08 jos nimessä on yli 100 merkkiä. Tällöin heksamerkin \x08 perässä on 16-bittinen little-endian kokonaisluku, josta saadaan nimen ensimmäisen osan pituus
0-?	Nimen ensimmäinen osa, pituus edeltävässä tavussa TAI kahdessa tavussa
1 tai 3	Nimen toisen osan pituus, johon on lisätty \x1b. Pelkkä \x1b on tyhjä. Tavu on \x08 jos nimessä on yli 100 merkkiä. Tällöin heksamerkin \x08 perässä on 16-bittinen little-endian kokonaisluku, josta saadaan nimen toisen osan pituus
0-?	Nimen toinen osa, pituus edeltävässä tavussa TAI kahdessa tavussa
1 tai 3	Nimen kolmannen osan pituus, johon on lisätty \x1b. Pelkkä \x1b on tyhjä. Tavu on \x08 jos nimessä on yli 100 merkkiä. Tällöin heksamerkin \x08 perässä on 16-bittinen little-endian kokonaisluku, josta saadaan nimen kolmannen osan pituus
0-?	Nimen kolmas osa, pituus edeltävässä tavussa TAI kahdessa tavussa
?	Sisällöstä ei tietoa, loppuu kuitenkin heksamerkkeihin \x08 XX YY 14 00, jossa XX vaihtelee ja YY on alle \x03
4	Sisällöstä ei tietoa, neljäs tavu aina \x00
4	Kolmas longitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla. Kuvaa lähintä risteystä
4	Kolmas latitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla. Kuvaa lähintä risteystä
14	Sisällöstä ei tietoa.
4	Neljäs longitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla. Kuvaa toiseksi lähintä risteystä
4	Neljäs latitudi 32-bittisenä little-endian kokonaislukuna. Kokonaisluku pitää jakaa 100000:lla. Kuvaa toiseksi lähintä risteystä

Suoraan tekstidatana nimille varattu tila näyttäisi seuraavanlaiselta, kun merkinnän pituutta ilmaisevat arvot (näkyvät huutomerkkinä, pisteenä ja numerona 9) on lihavoitu:

!London**.**Downing Street SW1A**9**Downing Street SW1A 10, London [44]

Edellä kuvatusta, varsin perusteellisesta selvityksestä huolimatta TomTom paikannuslaitteiden tutkiminen on nykyään aiempaa haastavampaa, sillä laitteet näkyvät tietoko-

neiden suuntaan verkkolaitteina ja niiden kopiointi on tämän johdosta huomattavasti vaikeampaa. Käytännössä tämä tapahtuu kytkeytymällä suoraan laitteen muistipiiriin, josta johtimet viedään tietokoneen USB-porttiin kytkettävään sovittimeen [45].

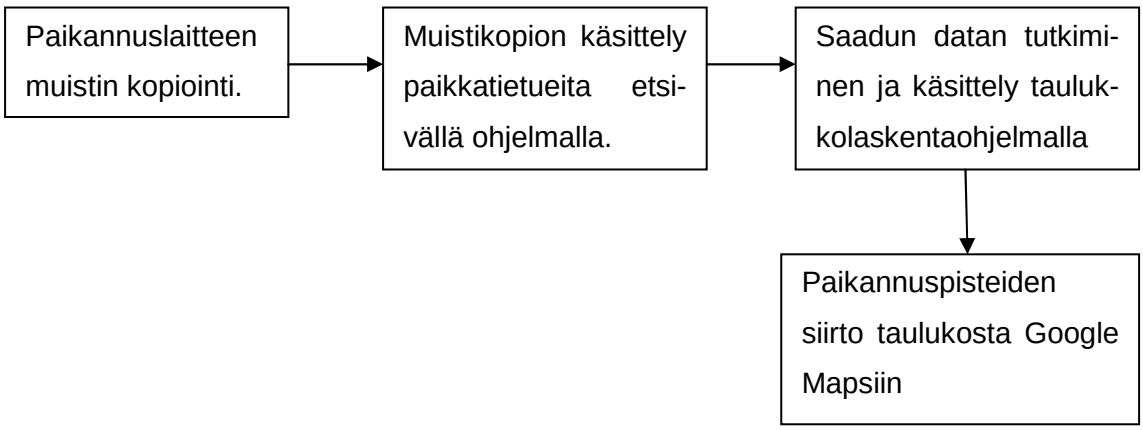
5.5 Tehtyjen ohjelmien käyttäminen

Ensimmäinen vaihe tehtyjen tietokoneohjelmien hyödyntämiseksi on kopioida laitteen muisti tai muistit jollekin ulkopuoliselle muistimedialle tai suoraan sille tietokoneelle, jolla tekemiäni ohjelmia on tarkoitus käyttää. Kopiointi tulisi aina suorittaa oikeusvarmasti. Oikeusvarma kopiointi tarkoittaa sitä, että kopioitava data ei missään tutkinnan vaiheessa pääse muuttumaan. Tämä voidaan varmistaa esimerkiksi siten, että kopio otetaan kirjoitussuojainta käyttäen. Kirjoitussuojaimia on kahdenlaisia: ohjelmallisia ja fyysisiä laitteita. Ohjelmallinen kirjoitussuojain on ohjelma tai esimerkiksi rekisteriarvo, joka kytketään päälle (tai pois päältä) ja tällöin muistimedia on kirjoitussuojattu eikä sillä olevaa dataa voi muuttaa, poistaa tai lisätä. Fyysinen kirjoitussuojaimen tarkoitus on aivan sama kuin ohjelmallisen kirjoitussuojaimen, mutta se on toteutettu erillisenä laitteena, jonka tuloporttiin liitetään kopioitava muistimedia ja lähtöporttiin tietokone tai muuta laite, jolla kopio otetaan. Kopion muuttumattomuus voidaan varmistaa myöhemmin laskemalla siitä kopion ottamisen yhteydessä tarkistussumma, esimerkiksi MD5, jota sitten verrataan myöhemmin laskettuun tarkistussummaan.

Kopiota ei saa esimerkiksi pakata, vaan sen on oltava alkuperäistä vastaava muistivedos (dump). Kopion ottamiseen voi käyttää esimerkiksi erilaisissa unix-pohjaisissa järjestelmissä olevaa dd-ohjelmaa tai vaikkapa Access Datan FTK Imager -ohjelmaa, jonka voi ladata ilmaiseksi yrityksen nettisivulta [46; 47]. Saatu kopio syötetään tehtyjen ohjelmien parametriksi.

Tietueen löytämisen jälkeen hakuosumaa on usein vielä käsiteltävä, jotta se voidaan lopuksi tulostaa taulukkolaskentaohjelmien ymmärtämään taulukkoon, josta rikostutkintaa suorittavan poliisin on se helppo lukea ja käsitellä. Taulukkolaskentaohjelmilla voi myös yleensä varsin helposti poistaa osoitetietueiden kopiot, joita voi tulla toista kymmentäkin per tietue riippuen siitä, kuinka monta eri tallennetta samasta tietueesta löytyy. Yhtenä tärkeimpänä asiana tiedon käsittelyssä on ollut koordinaattipisteiden saaminen luettavaan muotoon. Tämän jälkeen on mahdollista muuntaa nämä taulu-

koidut koordinaattipisteet esimerkiksi Google Mapsin ymmärtämään kml-tiedostomuotoon (Keyhole Markup Language) karttanäkymään sijoittamista varten [48]. Prosessi on esitetty kuviossa 6.



Kuvio 6. Insinöörityön lopputuloksena tehdyn ohjelman osuus paikannuslaitteen tutkinnassa.

6 Yhteenveto

Saatuani tämän aiheen mieleeni yli neljä vuotta sitten, ei kaupallisilla toimijoilla ollut juuri tarjota ohjelmistoja varsinkaan käyttäjän tai paikannuslaitteen käyttöjärjestelmän poistaman datan etsintään. Nykyään tilanne on huomattavasti parempi, mutta vielä aivan hiljattain esimerkiksi iGO-laitteet eivät olleet tuettuna Suomen poliisin yleisesti käytössä olevilla ohjelmilla, vaan paikkapisteet piti käytännössä purkaa käsin. Työ on opettanut itselleni erityisesti säännöllisten lausekkeiden ja merkkijonojen käsittelyä. Tekemäni ohjelmat olisi voinut niputtaa yhdeksikin, mutta pidin ne erillään. Mitään erityistä perustetta tähän ei ollut. Jos tulevaisuudessa tulen tehneeksi lisää eri merkeille ja malleille sopivia ohjelmia, yhdistän ohjelmat varmasti yhteen ja käyttäjä joutuu jollakin valitsimella kertomaan, minkä merkkistä laitetta ollaan tutkimassa.

Toinen asia on ohjelmien ajoon käytetty aika. Ylivoimaisesti suurin osa ohjelman ajoajasta kuluu juuri merkkijonon etsintään ja käsittelyyn. Osa käytössäni olevasta testidatasta oli vain joitakin kilotavuja, eli vain osavedos paikannuslaitteen muistista, mutta joistakin laitteista käytössä oli myös täydelliset muistivedokset. Esimerkiksi osana testidataa ollut Route66:n muistivedos oli 512Mt ja näitä laitteita varten tehdyltä ohjelmalta kului 7043 löydetyn osoitetietueen etsimiseen, käsittelemiseen ja tallentamiseen noin 7,5 minuuttia. IGO-laitteita varten tehty ohjelma taas käytti 2Gt:n muistivedoksesta 3544 osoitetietueen etsimiseen hiukan reilu 4,5 minuuttia. Olin myös aiemmin testannut Miolle ja Navmanille tehtyä ohjelmaa, kokonaiseen muistivedokseen. Tätä muistivedosta minulla ei enää ole, mutta muistelin ohjelman ajoajan olleen noin 15 minuuttia. Viimeksi mainittu ohjelma oli myös ainoa, joka ei käyttänyt säännöllisiä lausekkeita vaan ohjelmassa etsittiin tietueen alkupää ja alettiin sen jälkeen kerätä tietueen osia, kunnes tietue käsiteltiin loppuun tai tietue katkesi kesken. Ohjelmat käyttävät laskentaan vain yhtä säiettä, vaikka nykyään olisi hyvin usein mahdollisuus käyttää vähintään kahta, usein useampaakin säiettä samaan aikaan. Omassa kotikoneessani on esimerkiksi neljä fyysistä ja neljä virtuaalista ydintä, jotka näkyvät käyttöjärjestelmälle kahdeksana laskentaytimenä. Tällöin muistikopio voitaisiin lukea osissa, hiukan limittäin, jotta osoitetietue ei katkea juuri siitä kohtaa josta tiedosto on leikattu, ja antaa eri säikeiden käsiteltäväksi. Alun perin säikeiden käyttö oli tarkoituksenikin, mutta ajan puutteen takia jätin ratkaisun toteuttamatta.

Vaikka tässä työssä on esitelty monta paikannuslaitevalmistajaa, on markkinoilla vielä useampia, joiden tallennustapaa en ole ehtinyt ollenkaan tutkia. Tässä työssäkin esitellyissä olisi monessa vielä lisäselvityksiin aihetta. Kokonaisuutena arvioiden voin kuitenkin todeta, että työ on onnistunut, sillä tavoite tiedon saamisesta laitteista ulos täyttyi.

Lähteet

1. Hasik, James M ym. 2002. The Precision Revolution: GPS and the Future of Aerial Warfare. USA: Naval Institute Press.

2. 3 GLONASS satellites in final orbit. 2011. Verkkodokumentti. ITAR-TASS. <http://www.itar-tass.com/en/c32/264486.html>. Luettu 29.6.2012.

3. Beidou satellite navigation system to cover whole world in 2020. 2010. Verkkodokumentti. Chinese People's Liberation Army. <http://eng.chinamil.com.cn/news-channels/china-military-news/2010-05/20/content_4222569.htm>. Luettu 29.6.2012.

4. Launch of first 2 operational Galileo satellites. 2011. Verkkodokumentti. Euroopan komissio. <http://ec.europa.eu/enterprise/policies/satnav/galileo/satellite-launches/index_en.htm>. Luettu 29.6.2012.

5. The American Practical Navigator, julk. no 9. 2002. National Imagery and Mapping Agency.

6. Dr. Friedwardt Winterberg. 2012. Verkkodokumentti. University of Nevada. <http://www.physics.unr.edu/FacWinterberg.html>. Luettu 30.6.2012.

7. The Navy Navigation Satellite System. 1967. Video. US Navy.

8. Beard R.L., Murray J. ja White J.D. 1986. GPS Clock Technology and the Navy PTTI Programs at the U.S. Naval Research Laboratory. Washington: Naval Research Laboratory.

9. The Omega Navigation System. 1969. Video. US Navy.

10. Yogendran, S. 2002. Verkkodokumentti. Geographic Information System. <http://www.gisdevelopment.net/technology/gps/techgp0017a.htm>. Luettu 1.7.2012.

11. President Clinton: Improving the Civilian Global Positioning System (GPS). 2000. Verkkodokumentti. Valkoinen talo, Office of the Press Secretary. <http://clinton4.nara.gov/WH/New/html/20000501_2.html>. Luettu 1.7.2012.

12. Roßbach, Udo. 2000. Positioning and Navigation Using the Russian Satellite System GLONASS. Universität der Bundeswehr München.

13. USNO NAVSTAR Gloval Positioning System. Verkkodokumentti. US Naval Observatory. <http://tycho.usno.navy.mil/gpsinfo.html>. Luettu 1.7.2012.

14. Global Positioning System. 2012. Verkkodokumentti. Wikipedia. <http://en.wikipedia.org/wiki/Gps>. Luettu 1.7.2012.

15. How GPS Determine Position by Triangulation Method. 2013. Verkkodokumentti. Technology Hub – Technoworld

<http://straight2masti.com/how-gps-determine-position-by-triangulation-method>. Luettu 18.3.2013. Kuvan taustavärin muokkaus Joachim Ollas.

16. Global Navigation Satellite System GNSS (GPS/GLONASS). 2012. Verkkodokumentti. CalSky. <http://www.calsky.com/?GPS=>. Luettu 1.7.2012.

17. Consumer GPS/GLONASS. 2011. GPS World. <http://www.gpsworld.com/gnss-system/receiver-design/consumer-gpsglonass-12359>. Luettu 1.7.2012.

18. Mitsubishi Electric Advance, julk. no 91. 2000. Mitsubishi Electric Corporation.

19. Corporate Profile, History. 2012. Verkkodokumentti. Pioneer. <http://pioneer.jp/corp/profile-e/history/>. Luettu 25.9.2012.

20. LinuxFilesystemsExplained. 2011. Ubuntu Community Help Wiki. <https://help.ubuntu.com/community/LinuxFilesystemsExplained>. Luettu 1.7.2012.

21. Overview of Mac OS X File Systems. 2005. Verkkodokumentti. Apple. <https://developer.apple.com/library/mac/#documentation/Performance/Conceptual/FileSystem/Articles/MacOSXAndFiles.html>. Luettu 1.7.2012.

22. Overview of FAT, HPFS, and NTFS File Systems. 2007. Verkkodokumentti. Microsoft. <http://support.microsoft.com/kb/100108>. Luettu 1.7.2012.

23. Microsoft Extensible Firmware Initiative FAT32 File System Specification. 2000. Redmond: Microsoft Corporation.

24. The Fat-filesystem. 2013. Verkkodokumentti. Filetraffic.eu <http://filetraffic.eu/s/thefatfilesystem.jpg>. Luettu 1.1.2013

25. Murphy, David. 2008. The Disk Defrag Difference. Verkkodokumentti. Maximum PC. <http://www.maximumpc.com/article/the_disk_defrag_difference>. Luettu 4.7.2012.

26. Stoffregen, Paul. 2005. Understanding FAT32 Filesystems. Verkkodokumentti. <http://www.pjrc.com/tech/8051/ide/fat32.html>. Luettu 4.7.2012.

27. Adroit Photo Forensics 2013. 2013. Digital Assembly. Verkkodokumentti. <http://digital-assembly.com>. Luettu 28.4.2013.

28. Bekolay, Trevor. 2010. Recover Deleted Files on an NTFS Hard Drive from a Ubuntu Live CD. Verkkodokumentti. <http://www.howtogeek.com/howto/13706/recover-deleted-files-on-an-ntfs-hard-drive-from-a-ubuntu-live-cd/>. Luettu 4.7.2012.

29. Raise Data Recovery. 2012. Verkkodokumentti. UFS Explorer software. <http://www.ufsexplorer.com/rdr_hfsp.php>. Luettu 4.7.2012.

30. extundelete: An ext3 and ext4 file undeletion utility. Verkkodokumentti. Sourceforge.net. <http://extundelete.sourceforge.net/>. Luettu 4.7.2012.

31. Carrier, Brian. 2005. Why Recovering a Deleted Ext3 File Is Difficult… Verkkodokumentti. Sys-Con. <http://linux.sys-con.com/node/117909>. Luettu 4.7.2012.

32. Disk Utility 12.x: About disk defragmentation. 2012. Verkkodokumentti. Apple. <http://support.apple.com/kb/PH5862>. Luettu 4.7.2012.

33. Han, Sang-Wook. 1998. Flash memory wear leveling system and method. USA:n patentti numero: 6016275.

34. Lampinen, Juha. 2012. Tohtori, rikosinsinööri. Experimental study on extracting overwritten data from flash memory-based devices. University College Dublin

35. USB Flash Wear-Leveling and Life Span. 2007. Corsair (FAQ). <http://docs.aboutnetapp.ru/FAQ_flash_drive_wear_leveling.pdf>. Luettu 5.7.2012.

36. Perdue, Ken. 2010. Wear Leveling. Spansion.

37. Breeuwsma, Marcel ym. 2007. Forensic Data Recovery from Flash Memory. Small Scale Digital Device Forensics Journal, vol 1, no. 1.

38. PCRE – Perl Compatible Regular Expressions. 2012. Verkkodokumentti <http://www.pcre.org>. Luettu 18.4.2013

39. Ohjeistus aikakauslehtien osoiterekistereille ja osoitteille. 2007. Itella Oyj.

40. Karttapaikka. 2013. Verkkodokumentti. Maanmittauslaitos. <http://www.maanmittauslaitos.fi/kartat/karttapaikka>. Luettu 18.4.2013

41. Google Maps. 2013. Verkkodokumentti. Google. <http://maps.google.fi>. Luettu 18.4.2013.

42. Poliisin rikostutkinnan yhteydessä saatu testidata

43. Hevosoja, Jani. iGO8 ja iGO Primo –navigaattoriohjelmien datan tulkinta (Becker/Blaupunkt). 2011. Keskusrikospoliisi, rikostekninen laboratorio.

44. Nutter, Beverley. Pinpointing TomTom location records: a forensic analysis. 2008. Digital Investigation, vol 1, issues 1-2.

45. Sampo Ojala. 2012. Rikosinsinööri.12.12.2012 pidetty esitelmä KRP:n rikosteknisen laboratorion digitaaliseen todistusaineistoon liittyvistä teknisistä ratkaisuista

46. dd. 2011. Linux.fi-wiki. Verkkodokumentti <http://linux.fi/wiki/Dd>. Luettu 18.4.2013

47. Access Data. 2013. Verkkodokumentti. Access Data. <http://www.accessdata.com>. Luettu 18.4.2013

48. “Choon-Chern Lim (Mike)” (nimimerkki). 2009. KMLCSV Converter. Verkkodokumentti. <http://sourceforge.net/projects/kmlcsv/>. Luettu 18.4.2013.

Nokiaa ja Route66:ta varten tehty ohjelmakoodi

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <pcrecpp.h>

using namespace std;

/*****************************************************************************
 inter funktio muuttaa string-muuttujaan luetut tavut int-muuttujaksi.
 Funktio voi ottaa vastaan 1:stä n:ään tavua ja käsittelee ne tavu kerrallaan.
 Käytännön kokorajoitus tulee vastaan siinä, että int-muuttuja on neljän tavua. Samaa
 funktiota voitaisiin kuitenkin käyttää varsin sujuvasti myös long int-muuttujalle,
 joka on tyypillisesti kahdeksan tavua.
 Tavut luetaan väliaikaiseen int-muuttujaan ja ensin varmistetaan, että muuttujaan
 ei jää muuta kuin tämä yksi tavu. Tämän jälkeen tavu siirretään oikealle paikalleen
 ja kopioidaan oikealle kohdalle palautettavaa int-muuttujaa.
 *****************************************************************************/

float inter (string hit, int bytes, int rp) {
    int rots = bytes - 1, temp, sum = 0;
    for (int i = 0; i < bytes; i++) {
        temp = hit [rp--];
        temp |= 0xffffff00;
        temp ^= 0xffffff00;
        if (rots > -1) {
            temp = temp << (8 * rots);
            sum |= temp;
            rots--;
        }
    }
    return (sum);
}

/*****************************************************************************
 tabber-funktio paloittelee string-muodossa olevan hakuosuman alkuosan sarkaimella
 erotelluiksi eri sarakkeisiin tallennetun osoitetietueen eri osien sisällön tai
 ylipäätään olemassaolon mukaan. Vaikka tallennettava tiedosto onkin .csv-päätteinen
käytetään tabulaattoria siksi, että osoitetiedoissa voi olla pilkkuja
*****************************************************************************/

string tabber (string hit) {
    int pos = 0, cols = 0;
    pos = hit.find(">");
    if (pos > -1)
        hit.replace(pos+1, 1, "\t");
    else {
        string tab = "\t";
        tab.append(s);
        hit.assign(tab);
    }
```

```
    pos = hit.find(" ["); //kaupunki
    if (pos > -1)
        hit.replace(pos, 1, "\t");
    else
        cols++;

    pos = hit.find("] "); //postinumero
    if (pos > -1)
        hit.replace(pos + 1, 1, "\t");
    else
        cols++;

    pos = hit.find("]\x0d"); //ei postinumeroa, vaan tekstiä
    if (pos > -1){
        hit.replace(pos + 1, 1, "\t\t");
        cols--;
    }
    else
        cols++;

    pos = hit.find("\x0a[");
    hit.replace(pos, 1, cols + 1, '\t');

std::replace(hit.begin(), hit.end(), '\x0d', ' ');
    return hit;
}

main(int argc, char *argv[])
{
    int len, hitNum = 0;
    float lat, lon;
    string hit;
    stringstream ss;
    char outname[64];

    ifstream file(argv[1], ios::in | ios::binary);
    string sample ( (istreambuf_iterator<char>(file.rdbuf())), (istreambuf_iterator<char>()));

    strcpy(outname, argv[1]);
    strcat(outname, ".csv");

    ofstream output;
    output.open (outname, ios::out);
    if (!output)
    {
        cerr << "File could not be opened." << endl;
        exit(1);
    }
    output << "NUMERO\tKATU\tKAUPUNKI\tPOSTINUMERO\tSELITE\tAIKA\tLAT\tLONG\t" <<
endl;

        // määritellään pcre:n asetukset
    pcrecpp::RE_Options options;
    options.set_caseless(true).set_multiline(true).set_ungreedy(true);
    pcrecpp::StringPiece input(sample);
```

```
/*****************************************************************************
 Luodaan haettava säännöllinen lauseke (grep), aiemmin luotujen asetusten mukaisesti.
 Säännöllisen lausekkeen perusteet löytyvät insinöörityöstä ja ne voi
 katsoa sieltä tarkemmin.
*****************************************************************************/

   pcrecpp::RE navpoint ("((<[\\d|\x3f]{1,4}>)?([[:upper:]][\\w|\\xc380-\\xc3bf| |;|\\-
|,]{1,49})(\\[[\\w|\\xc380-\\xc3bf| |;|\\-|,]{2,50}\\])? ?(\\(\\d{5}\\))?([\\w|\\xc380-\\xc3bf|
|;|\\-|,|\x0d|\\(|\\)|\\+|:]{2,80})?\x0a\\[((\\d{2}).){4}(\\x00)([\\x00-\\xff]{8}))", options);

   if (navpoint.error().length() > 0) {
      cout << "PCRE compilation failed with error: " << navpoint.error() << endl;
   }

      // etsitään hakuosumia niin kauan, kuin luettavaa tiedostoa riittää
   while (navpoint.FindAndConsume(&input, &hit)) {

      len = hit.length();

      // koordinaatit ovat int32-muuttujassa
      float lat = inter(hit, 4, len - 1);
      float lon = inter(hit, 4, len - 5);

      lon /= 100000;
      lat /= 100000;
      hit.resize (len - 9);

      // valmistellaan loput hakuosumasta tulostettavaan muotoon
      hit = tabber (hit);

      output << hit << '\t' << fixed << setprecision(5) << lat << '\t' << lon << endl;
      hitNum++;
   }
   output.close();

   cout << hitNum << " hits." << endl;

}
```

Mioa ja Navmania varten tehty ohjelmakoodi

```
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int search_pos = 0;
int image_size = 0;
int all_miss = 0;

/*****************************************************************************
 find_text-funktio etsii tekstidataa, joka sijaitsee xml-tagien välissä. Funktio saa
tekstimuuttujan, eli < > -sulkujen sisällä olevan sanan ja funktio lisää sulut ja tuottaa alku-
ja lopputagien väliin jäävän saman funktion palautusarvona.
*****************************************************************************/


string find_text (string image, string tag) {
    char result[255] = "";
    int start_pos, end_pos, searchlength;
        string search = "<" + tag + ">";
        searchlength = search.length();
    start_pos = image.find(search, search_pos);


/*****************************************************************************
 Alla oleva vertailu on ns. hakurajoitin, eli jos tietue katkeaa kesken, ei ohjelma etsi seuraavaa
osumaa seuraavaksi vastaan osuvasta tietueesta vaan palauttaa tyhjän.
*****************************************************************************/
    if (start_pos == string::npos)
    {
        all_miss++;
        return "";
        }

    search = "</" + tag + ">";
    end_pos = image.find(search, search_pos);
    if ((end_pos - start_pos) > 100)
        return "";
        search_pos = end_pos;
    image.copy(result, (end_pos - start_pos - searchlength), (start_pos + searchlength));
    return (result);
}

// find_number-funktio on sama kuin find_text, mutta luvuille, jotka voivat olla desimaalilukuja

float find_number (string image, string tag) {
    char result[255] = "";
    float point;
    int start_pos, end_pos, searchlength;
        string search = "<" + tag + ">";
        searchlength = search.length();
```

```
    start_pos = image.find(search, search_pos);

    if (start_pos == string::npos)
    {
        all_miss++;
        return 0;
        }

    search = "</" + tag + ">";
    end_pos = image.find(search, search_pos);
    if ((end_pos - start_pos) > 30)
        return 0;
        search_pos = end_pos;
    image.copy(result, (end_pos - start_pos - searchlength), (start_pos + searchlength));
    point = atof(result) / 100000;
    return (point);
}


main (int argc, char* argv[]) {

        string  image, line;
    string  displayname, behaviour,
            lastused, stickyorder, name, name2,
            entryname, entryhouseno, entrypostcode,
            xtypename, phonenum, description, type;

    char*   input = argv[1];
    char    outputFile[64];

    float   lat, longi, entrylat, entrylongi,
            fspos, fimage, percent;

        ifstream inputFile( input );

    cout << "Luetaan muistiin... " << endl;

// tiedosto voidaan lukea rivi kerrallaan, koska esimerkiksi rivinpäättymismerkkejä ei tarvita

        while (getline (inputFile, line)) {
            image.append(line);
                    line.erase();
        }

    cout << "...Valmis." << endl;
    cout << "Prosenttia tutkittu:" << endl;

    strcpy(outputFile, argv[1]);
    strcat(outputFile, ".csv");

    ofstream output;
    output.open (outputFile, ios::out);

    if (!output)
    {
```

```
        cerr << "File could not be opened." << endl;
        exit(1);
    }

    image_size = image.size();
    fimage = (float)image_size;

    output <<
"Name\tName2\tLat\tLong\tEntryname\tEntryLat\tEntryLong\tEntryHouseNo\tEntryPostcode\tT
ypename\tPhoneNum\tDescription\tType\tLastUsed" << endl;

    while (search_pos < image_size) {

        // haetaan järjestyksessä tallennetun paikkapisteen tietueen eri osat
        lastused = find_text(image, "LastUsed");
        name = find_text(image, "name");
        name2 = find_text(image, "name2");
        lat = find_number(image, "lat");
        longi = find_number(image, "long");
        entryname = find_text(image, "entryName");
        entrylat = find_number(image, "entryLat");
        entrylongi = find_number(image, "entryLong");
        entryhouseno = find_text(image, "entryHouseNo");
        entrypostcode = find_text(image, "entryPostcode");
        xtypename = find_text(image, "typeName");
        phonenum = find_text(image, "phoneNum");
        description = find_text(image, "description");
        type = find_text(image, "type");

        fspos = (float)search_pos;
        percent = (fspos / fimage) * 100;
        cout << percent << "\r" << flush;

        // jos kaikki tai osa on haettavista tietueen osista on löytynyt, tulostetaan nämä osat
        if (all_miss < 14) {
            output << name << "\t" << name2 << "\t" << lat << "\t" << longi << "\t" <<
entryname << "\t";
            output << entrylat << "\t" << entrylongi << "\t" << entryhouseno << "\t" <<
entrypostcode << "\t";
            output << xtypename << "\t" << phonenum << "\t" << description << "\t" << type
<< "\t" << lastused << endl;
        }
        // jos yhtään tietueen osaa ei löydy, ohjelmarungon ehto (epäsuorasti) puretaan
        if (all_miss == 14){
            search_pos = image_size;
            cout << "100.0  " << endl;
        }
        all_miss = 0;
    }
}
```

Navigonia varten tehty ohjelmakoodi

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <pcrecpp.h>

using namespace std;

main(int argc, char *argv[])
{
    int len, pos_m, pos_n, hitNum = 0;
    string output, hit, lat, lon, post_code, number, town, temp;
    stringstream ss;
    char outname[64];

    ifstream file(argv[1], ios::in | ios::binary);
    string sample ( (istreambuf_iterator<char>(file.rdbuf())), (istreambuf_iterator<char>()));

    strcpy(outname, argv[1]);
    strcat(outname, ".csv");

    ofstream outputFile;

    outputFile.open (outname, ios::out);
    if (!outputFile)
    {
        cerr << "File could not be opened." << endl;
        exit(1);
    }

        // Taulukon otsikkotiedot
    outputFile << "NIMIMERK-
KI\tKATU/PISTE\tNUMERO\tPOSTINUMERO\tKAUPUNKI/ALUE\tLAT\tLONG\tEKSTRA DATA" <<
endl;

        // Määritellään pcre:n asetukset
    pcrecpp::RE_Options options;
    options.set_caseless(true).set_ungreedy(true);
    pcrecpp::StringPiece input(sample);

/*****************************************************************************
     Luodaan haettava säännöllinen lauseke (grep), aiemmin luotujen asetusten mukaisesti.
Säännöllisen lausekkeen ”perusteet” löytyvät Teemu Hokkasen insinöörityöstä ja ne voi katsoa
sieltä tarkemmin.
*****************************************************************************/

    pcrecpp::RE navpoint ("((\\[[\\w| |\\-
]{0,30}\\|\\|\\])((\\[\\d{1,3}\\])){2})((.\\d{1,3}\\.\\d{5}){2})?(\\|[\\w|\xc0-\xdd| |\\-
]{2,50})?\\|(\\d{5})?((\\|\\d{1,3}\\.\\d{5}){2})[\\x00-\\xff]{0,200}\\|\\|15\x0d\x0a)", opti-
ons);
```

```
    if (navpoint.error().length() > 0) {
        cout << "PCRE compilation failed with error: " << navpoint.error() << endl;
    }

    while (navpoint.FindAndConsume(&input, &hit)) {

/*****************************************************************************
 Yhden tietueen rakennetta ei ole täysin takaisinmallinnettu. Tietueesta poimitaan
 omiin sarakkeisiinsa oleellisin, eli varsinaisen osoitteen tai paikan osoitetieto
 ja koordinaatit. Loputkin tiedot olisi mahdollista tietueesta poimia, mutta ne eivät ole tärkeitä ja
hidastavat ohjelmaa. Tämän olisi voinut pistää myös omaan funktioonsa,mutta tämän ohjelman
kohdalla jätin sen suoraan ohjelmaa pyörittävään silmukkaan.

 Ylipäätään tietuetta purkava rutiini ei ole erityisen ”nätti”, mutta huomattavasti siistimpi ja sel-
keämpi kuin C:llä kirjoitettuna ja char:ia käytettämällä.
*****************************************************************************/

        pos_m = hit.find("|", 0);
        output.assign(hit, 1, pos_m - 1);
        output.append("\t");
        pos_m = hit.find("]|", pos_m);
        pos_n = hit.find("|", pos_m + 2);
        output.append(hit, pos_m + 2, pos_n - pos_m - 2);
        pos_m = hit.find("|", pos_n + 1);
        post_code.assign(hit, pos_n + 1, pos_m - pos_n - 1);
        pos_n = hit.find("|", pos_m + 1);
        lon.assign(hit, pos_m +1, pos_n - pos_m - 1);
        pos_m = hit.find_first_of("[|", pos_n + 1);
        lat.assign(hit, pos_n + 1, pos_m - pos_n - 1);

        if (hit.compare(pos_m, 1, "|") == 0)
            number = "";
        else {
            pos_n = hit.find("]|", pos_m);
            pos_m = hit.find("|", pos_n + 2);
            temp.assign(hit, pos_n + 2, pos_m - pos_n - 2);
            if (47 < temp[0] && temp[0] < 58) {
                number.assign(temp);
                town = "";
            }
            else {
                town.assign(temp);
                number = "";
            }
        }

        output = output + "\t" + number + "\t" + post_code + "\t" + town + "\t" + lat+ "\t" +
lon + "\t";

        pos_n = hit.length();

        temp.assign(hit, pos_m, pos_n - pos_m);

        std::replace(temp.begin(), temp.end(), '[', ' ');
        std::replace(temp.begin(), temp.end(), ']', ' ');
```

```
        std::replace(temp.begin(), temp.end(), '|', ' ');

        output.append(temp);

        outputFile << output;
        hitNum++;

    }
    outputFile.close();
    cout << hitNum << " hits." << endl;
}
```

iGO-laitteita varten tehty ohjelmakoodi

```
#include <iostream>
#include <string>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <pcrecpp.h>

using namespace std;

/*****************************************************************************
 inter funktio muuttaa string-muuttujaan luetut tavut int-muuttujaksi.
 Funktio voi ottaa vastaan 1:stä n:ään tavua ja käsittelee ne tavu kerrallaan.
 Tavut luetaan väliaikaiseen int-muuttujaan ja ensin varmistetaan, että muuttujaan
 ei jää muuta kuin tämä yksi tavu. Tämän jälkeen tavu siirretään oikealle paikalleen
 ja kopioidaan oikealle kohdalle palautettavaa int-muuttujaa.
*****************************************************************************/

int inter (string hit, int rp, int bytes) {
    int rots = bytes - 1, temp, sum = 0;
    for (int i = 0; i < bytes; i++) {
        temp = hit [rp--];
        temp |= 0xffffff00;
        temp ^= 0xffffff00;
        if (rots > -1) {
            temp = temp << (8 * rots);
            sum |= temp;
            rots--;
        }
    }
    return (sum);
}
/*****************************************************************************
 timer funktio muuttaa int32-muuttujassa olevan arvon kellonajaksi. Muuttujassa oleva arvo on
millisekunteja päivän alusta ja funktion tehtävä on laskea ja erotella tunnit, minuutit ja sekun-
nit. Vaikka aika voidaan ilmoittaa millisekunnin tarkkuudella, laite ei näin tee.
(Eikä sellaista tarkkuutta tarvitakaan.) Lopputulos palautetaan
 string-muuttujassa.
*****************************************************************************/

string timer (int timeInt) {
    float temp = 0;
    int hour, min, sec;
    stringstream result;
    temp = timeInt / (1000.0 * 60.0 * 60.0); // jaetaan millisekunneilla, sekunneilla ja minuuteil-
la, jää tunnit
    hour = temp;    // poimitaan kokonaiset tunnit float-muuttujasta int-muuttujaan
    temp -= hour;   // vähennetään float-muuttujasta kokonaisluvut
    temp *= 60.0;   // lasketaan minuutit, tämän jälkeen tehdään sama sekunneilla
    min = temp;
    temp -= min;
    temp *= 60.0;
    sec = temp;
```

```
    result << hour << ":" << min << ":" << sec;
    return result.str();
}

/*****************************************************************************
 cleaner-funktio siivoaa tallennetusta datasta, erityisesti osoitteen kohdalta, turhat
 NULL-arvot pois.
*****************************************************************************/

string cleaner (string add) {
    int pos = 0, len;
    len = add.length();
// muutetaan hakuosumissa olevat kohdat, joissa on kolme peräkkäistä NULL-arvoa välilyönniksi
    do {
        pos = add.find("\x00\x00\x00", pos, 3);
        if (pos > -1)
            add.replace(pos, 3, " ");
    } while (pos > -1);
    // poistetaan loput NULL-arvot, jotta teksti voidaan tulostaa tiedostoon
    pos = 0;
    do {
        pos = add.find("\x00", pos, 1);
        if (pos > -1)
            add.erase(pos, 1);
    } while (pos > -1);
    return add;
}

main(int argc, char *argv[])
{
    int len, pos, hitSum = 0, year, month, day, timeInt;
    float lat, lon, timeFloat;
    string hit, timeString, add;
    stringstream ss;
    char outname[64];

    ifstream file(argv[1], ios::in | ios::binary);
    string sample ( (istreambuf_iterator<char>(file.rdbuf())), (istreambuf_iterator<char>()));

    strcpy(outname, argv[1]);
    strcat(outname, ".csv");

    ofstream output;
    output.open (outname, ios::out);
    if (!output)
    {
        cerr << "File could not be opened." << endl;
        exit(1);
    }

        // luodaan taulukon otsikkorivi
    output << "PÄIVÄYS\tAIKA (GMT)\tLAT\tLONG\tOSOITE" << endl;

        // määritellään pcre:n asetukset
    pcrecpp::RE_Options options;
```

```
    options.set_caseless(true);
    pcrecpp::StringPiece input(sample);

/*************************************************************************
  Luodaan haettava säännöllinen lauseke (grep), aiemmin luotujen asetusten (options) mukaisesti.
  Säännöllisen lausekkeen perusteet löytyvät insinöörityöstä ja ne voi katsoa sieltä tarkemmin.
*************************************************************************/

    pcrecpp::RE navpoint ("([\\x00-\\xff]\\x07[\\x00-\\xff]{6}(\\xff{4})?[\\x00-\\xff]{8}h\\x00i\\x00s\\x00t\\x00_\\x00[\\w|\\x00]{5,15}\\x00{3}s\\x00y\\x00s\\x00{3}(\\[\\x00([N|S|E|W]\\x00\\x00[\\d|\\x00]{2,6}\\xb0\\x00[\\d|\\x00]{2,4}'\\x00[\\d|\\x00]{2,4}\\.\\x00[\\d|\\x00]{2,4}\"[,| |\\x00|x|\\]]{0,7}){4})?(([\\x00-\\xff]\\x00{3})?[\\w|\\xc0-\\xff|\\x00|\\.|,| |/]{4,100}(_[[:upper:]]{3}\\x00{6})?[\\w|\\xc0-\\xff|\\x00|\\.|,| |/]{0,100})?[\\x00|\\x01]{6})", options);


    if (navpoint.error().length() > 0) {
        cout << "PCRE compilation failed with error: " << navpoint.error() << endl;
    }

        // etsitään hakuosumia niin kauan, kuin luettavaa tiedostoa riittää
    while (navpoint.FindAndConsume(&input, &hit)) {

        // hist_flag
        if ((hit.compare(8, 4, "\xff\xff\xff\xff")) == 0)
            hit.erase (8, 4);

        // vuosi on int16-muuttujassa
        year = inter (hit, 1, 2);
        month = hit[2];
        day = hit[3];

        // kellonaika on int32-muuttujassa
        timeInt = inter (hit, 7, 4);
        timeString = timer (timeInt);

        // koordinaattipisteet ovat int32-muuttujissa, mutta vaativat vielä pari laskutoimitusta
        lon = inter (hit, 11, 4);
        lon = lon * 1.1920929 / 10000000;
        lat = inter (hit, 15, 4);
        lat = lat * 1.1920929 / 10000000;

        // haetaan hakuosumasta kohta jonka jälkeen alkaa varsinainen osoite
        pos = hit.find("s\x00y\x00s", 16, 5);
        len = hit.length();
        add.assign (hit, pos + 8, len - pos - 4);

        add = cleaner (add);

        output << year << "-" << month << "-" << day << "\t" << timeString << "\t" << fixed << setprecision(7) << lat << "\t" << lon << "\t" << add << endl;

        hitSum++;
    }
```

```
    output.close();
    cout << hitSum << " hits." << endl;
}
```