VENTSPILS AUGSTSKOLA
INFORMĀCIJAS TEHNOLOĢIJU FAKULTĀTE

BAKALAURA DARBS

GPS IZMANTOŠANA TRANSPORTA ORGANIZĀCIJĀ

Autors

Ventspils Augstskolas
Informācijas tehnoloģiju fakultātes
bakalaura studiju programmas
„Datorzinātnes”
3. kursa studente
Katrīna Zvaigzne
Matr.nr. 2011020289

_____
(paraksts)

Fakultātes dekāns asoc.prof., Dr. math. Gaļina Hiļķeviča

_____
(paraksts)

Zinātniskais vadītājs asoc.prof., Dr. hab. phys. Juris Žagars

_____
(paraksts)

Recenzents _____

_____
(paraksts)

Ventspils
2014

SATURS

Lpp

Saīsinājumu un nosacīto apzīmējumu saraksts

NNSS - Navy Navigation Satellite System

GPS - Global Positioning System

NAVSTAR - NAVigation System with Timing And Ranging

SA - Selective Avaibility

AS- Anti Spoofing

VLBI - Very Long Base Interferometry

GNSS – Global Navigation Satellite System

UT – Universal Time

GKS – Gaisa kara spēki

CSOC - Consolidated Space Operation Center

PRN – pseudo random noice

BPSK – bifāzu modulācija

PAL – Phase Alternating Line

RTK – Real Time Kinematic

DOP – Dilution of Precision – precizitātes "šķīšana"

GDOP – Global DOP

PDOP – Positional DOP

TDOP – Time DOP

HDOP – Horizontal DOP

VDOP – Vertical DOP

GPRS - General packet radio service

SMS – Short Message Service

CTA - Chicago Transit Authority

LAMP – Linux, Apache, MySQL, PHP

IP – Internet Protocol

PHP – Hypertext Preprocessor

XML – Extensible Markup Language

1. GPS TEHNOLOĢIJA UN GNSS

1.1 GPS rašanās un vēsture

1.1.1 Kosmiskais un tehniskais apskats

Lai nodrošinātu ASV Jūras kara spēku navigāciju, 20. gadsimta septiņdesmito gadu sākumā sākās satelītnavigāciju sistēmu izveide. It īpaši tas bija nozīmīgi atomzemūdeņu flotei, kam ilgstoši nācās atrasties zem ūdens un kuras navigācijai bija jābūt īpaši precīzai, jo vajadzēja nodrošināt augstu precizitāti izmantojot kodolieročus. Šo atrašanās vietu arī nevarēja noteikt atomzemūdenei uznirstot, jo tad iespējamam tuvējam pretiniekam būtu zināma tās atrašanās vieta. Sešdesmitajos gados jau eksistēja vairākas radionavigācijas sistēmas, bet tās izmantoja raidītājus, kas atradās uz Zemes un izmantoja radiodiapozona vidējos un garos viļņus, arī to viļņa garums bija ierobežots, jo, lai uztvertu šādus radioviļņus vajadzēja izmantot samērā lielas antenas. Atrodoties gaisā, piemēram, aviācijā, situācija bija nedaudz labāka, jo gaisā radioviļņa signāls bija spēcīgāks un tā uztveršanai nebija nepieciešamas tik lielas antenas. Zemes sfēriskais izliekums ierobežoja mikroviļņu izmantošanu, jo mikroviļņu virziens ir taisns un tie neliecas. Antenu izmērs arī bija viens no noteicošajiem faktoriem, lai esošās sistēmas neizmantotu zemūdeņu atrašanās vietas noteikšanai, kā arī tās darbojās lokāli, nevis globāli. Viena no tā laika pazīstamākajām radionavigācijas sistēmām LORAN-C darbojās tikai Ziemeļatlantijas reģionā, jo raidošās antenas bija izvietotas Kanādas un ASV piekrastē, gar Grenlandi, Eiropas piekraste un vēl dažos atsevišķos reģionos.

Radionavigācijas sistēmu daudzie trūkumi, kā piemēram, samērā lielās antenas, ierobežotie viļņu garumi un sistēmu lokālās izmantošanas iespējas bija arī galvenie iemesli, kāpēc 20. gadsimta septiņdesmito gadu sākumā radās pirmā globālās navigācijas satelītsistēma TRANSIT, saukta arī par NNSS. Tā sastāvēja no 5 līdz 7 satelītiem, kas rinķoja pa rinķveida orbītām 1100 kilometrus virs Zemes virsmas. Tā kā satelītus vēl neprata aprīkot ar atompulksteņiem, kas būtu ekspluatācijā droši, relatīvi lēti un pietiekami droši, lai veiktu navigācijas mērījumus izmantoja Doplera efektu nevis pseidoattālumu mērījumus. Lai arī TRANSIT navigācijas sistēma sastāvēja no neliela skaita satelītiem un tādēļ tā nespēja nodrošināt nepārtrauktu pozicionēšanu, sistēmu jau uzskatīja par globālu. Atkarībā no pozicionēšanai pieejamo satelītu konstelācijas navigācijas precizitāte bija ļoti mainīga. Meklējot iespēju, kā nodrošināt nepārtrauktu pozicionēšanu un analizējot orbitālo situāciju, tika modelēts minimālais satelītu skaits konstelācijās, kas visur uz zemeslodes nodrošina vismaz četrus satelītus virs almukantarata $h = 15°$. Almukantarats ir jebkurš iedomājams debess sfēras mazais riņķis, kas atrodas paralēli horizontam. [1] Lai arī aprēķini norādīja,

ka to iespējams paveikt izmantojot 21 satelītu rinķveida orbītās ar aprinķošanas periodu $P = 12\ h$ un inklināciju $i = 55°$, tomēr, pamatojoties uz to, ka šādas konstelācijas izrādījās ar zemu drošību, ja kāds no satelītiem nefunkcionē normāli un grūti sinhronizējamas, konstelācijas ar 24 satelītiem ar iepriekšminētajiem orbītu parametriem tika atzītas par uzticamākām un stabilākām.

1973. gadā tika radīts vēlākais GPS. Tolaik ASV Aizsardzības departaments (Department of Defence) uz TRANSIT sistēmas pieredzes bāzes pieņem lēmumu veidot jaunu, globālu satelītnavigācijas sistēmu NAVSTAR. NAVSTAR ļoti būtiski atšķīrās ar to, ka šīs sistēmas satelītus paredzēja aprīkot ar ļoti precīziem atompulkteņiem, kas nodrošinātu vienvirziena attāluma (pseidoattāluma) mērīšanas principu. Uz satelītiem Block-2, kas veidoja pirmo GPS kosmiskā segmenta satelītu pamatkonstelāciju, atradās divi cēzija un divi rubīdija frekvenču standarti – atompulksteņi. Šo atompulksteņu frekvenču stabilitāte, ko aprēķina pēc formulas 1.1. bija 10^{-13}. Vēlākajos satelītos Block-2R jau tika izmantoti ūdeņraža frekvenču standarti – atompulksteņi, kuru frekvenču stabilitāte bija vēl par 1-2 kārtām augstāka.

Tā kā NAVSTAR bija militāra sistēma, lai ierobežotu neautorizētas lietošanas precizitāti un mazinātu sistēmas slāpēšanas iespējas, sistēmā tika iestrādāti divi aizsardzības mehānismi – SA un AS.

SA izpaužas kā atbalsta frekvences iesvārstīšana atbilstoši izstrādātam algoritmam, kas noved pie degradētas signālu navigācijas informācijas precizitātes, ja vien šīs frekvences svārstības uztvērējā netiek kompensētas.

AS izpaužas kā slepena un ļoti sarežģīta militāra koda izmantošana kā rezultātā tiek izslēgta iespēja raidīt māņu signālus no potenciālo pretinieku satelītiem.

Izmantojot šos aizsargmehānismus Kuveitā Līča kara laikā, šie aizsargmehānismi pierādīja to neefektivitāti.[2]

1.1.2 Lietotāju apskats.

Lai arī sākumā NAVSTAR tehnoloģiju izmantoja tikai militārām vajadzībām (ASV armijas, jūras un gaisa kara spēku navigācijas vajadzībām) , attīstoties interferometriskās mērīšanas tehnoloģijām, tika secināts, ka šādas tehnoloģijas arī ir izmantojamas navigācijā. Interferometriskās mērīšanas tehnoloģijas ir visas mērīšanas tehnoloģijas, kas mērīšanā izmanto iekārtas, kas mērījumus veic izmantojot interferenci. Interference ir fizikāla parādība, kas rodas divu vai vairāku koherentu viļņu (ar vienādām vai dažādām frekvencēm) savstarpējas pārklāšanās un mijiedarbības procesos.[1] Interferometrisko mērīšanas tehnoloģiju pielietošanai nav jāizmanto kods, kā tas bija izmantojot militārās navigācijas

metodes, un šīs tehnoloģijas salīdzinoši ar militārās navigācijas metodēm ir par 2-3 kārtām precīzākas, bet arī sarežģītākas. Pamatojoties uz to 1964. gadā I. Smith patentē interferometriskās navigācijas ideju. Pēc tam 1970. gadā to attīsta un patentē R. Easton. Būtisks pagrieziens ir 1972. gadā Chuck Counselman no Masačūsetsas Tehnoloģiskā Institūta (MIT) veiktā kosmisko kuģu "Apollo-16" un "Apollo-17" Mēness stacionāro un pārvietojamo moduļu atsekošana, izmantojot interferometriskās navigācijas metodes, kas bija aizgūtas no VLBI tehnoloģijām. Izmantojot šobrīd zināmo informāciju par GPS un GNSS, 1972. gadā Chuck Counselman veiktais darbs patiesībā ir balstīts uz fāzu pozicionēšanas metodes idejām. Vēlāk no 1978. gada līdz 1979. gadam Chuck Counselman un citi publicē aptuveni visas galvenās interferometriskās jeb fāzu pozicionēšanas idejas un metodes.

Interferometrisko jeb fāzu pozicionēšanas ideju attīstība nebija vienīgais iemesls, kādēļ GPS sistēma jeb NAVSTAR kļuva slavena. Tas saistāms arī ar to, ka šī sistēma ne tikai pilnībā attaisno uz to liktās cerības, bet krietni pārspēj cerētos precizitātes un citus veiktspējas kritērijus, atšķirīgi no visbiežāk pieredzētās situācijas, kad šādas sarežģītas un rūpīgi izstrādātas sistēmas neattaisno uz tām liktās cerības, kad sistēma jau ir realizēta. Papildus NAVSTAR bija īpaša ar to, ka pozitīvus sasniegumus sasniedza jau vien tās eksperimentālās konstelācijas satelīts Block-1, bet, ka sistēmas pozicionēšanas reālo precizitāti vēl iespējams uzlabot tūkstoškārt. Fāzu pozicionēšanas ideju, algoritmu un metožu pielietošana būtiski ļāva NAVSTAR militārajai globālajai navigācijas sistēmai uzlabot tās turpmāko nākotni.

Pirmie reālu ieguvumu no globālās navigācijas sistēmas, ja tā spētu nodrošināt pozicionēšanas precizitāti līdz decimetram vai centimetram, saskatīja ASV ģeodēzisti un pasaules ģeodēzistu savienība. Viņi saskatīja lielu apvērsumu savā profesionālajā darbībā, ja būtu iespēja izmantot šo satelītsistēmu. Lai arī GPS kosmiskā daļa vēl nebija pilnībā izstrādāta, pārstāvji no ASV ģeodēzistu organizācijām vērsās ar lūgumu pie ASV varas iestādēm, un pat pie ASV prezidenta, izmantot GPS satelītu signālus ģeodēzisku darbu veikšanai, jo precīzās interferometriskās navigācijas metodes pielietotas GPS satelītu signāliem izteikti uzlabotu produktivitāti ģeodēziskajos mērījumos un homogenizācijas jomā. Homogenizācija ir visas tās metodes un procesi sistēmas viendabīguma radīšanai.[1] Vēlāk pēc šī ASV ģeodēzijas organizāciju lūguma, kad izrādījās, ka GPS sistēma būtiski uzlabo situāciju ne tikai militārajā sfērā, bet arī citās, tika pieņemts lēmums tolaik militāro un īpaši slepeno GPS sistēmu padarīt pieejamu arī civilai izmantošanai ne tikai globāli visā pasaulē, bet arī pilnībā bezmaksas, lai arī tās kosmiskā daļa joprojām nebija līdz galam pilnībā izstrādāta.

Sakarā ar GPS sistēmas ienākšanu ikdienā, ģeodēzijas, astronomijas un ģeofizikas speciālisti visā pasaulē, izmantojot interferometriskās navigācijas jeb fāzu pozicionēšanas idejas, radīja dažādas metodes GPS mērījumu apstrādei un izmantošanai. To pamatā pārsvarā bija divu vai vairāku uztvērēju vienlaicīga izmantošana, jo bija vieglāk nomērīt pseidoattālumu diferences ΔR nekā pašus pseidoattālumus R. Lai izstrādātu šīs metodes, tika attīstītas statiskās, kinemātiskās, pseidokinemātiskās un citas metodes. Pateicoties tam, ka ģeodēzisti vēlējās padarīt savus risinājumus līdz arvien augstākai precizitātei, neskaitot jau tā ļoti precīzos satelītu signālus, izmantoja arī citus signālus un tika izveidotas principāli jaunas metodes.

Pārsteidzošais uzplaukums GPS satelītu signālu apstrādes metožu izstrādē un tām pastiprināti pievērstā uzmanība rezultējās GPS pozicionēšanās izmantošanas iespēju paplašināšanos ārpus militārās jomas, piemēram tādās saimnieciskās dzīves sfērās, kā ģeodēzija, aviācija, loģistika un citās. 20. gadsimta astoņdesmitajos gados tika uzsākta arī vairāku GPS uztvērēju ražošana. Ar to nodarbojās tādas kompānijas, kā Ashtech, Trimble, AOA, Leica, Magelan un citas. Papildus GPS uztvērējiem attīstījās arī programmatūras izstrāde. 20. gadsimta deviņdesmitajos gados par līderi izvirzījās Bernes Astronomijas Institūts ar programmu kompleksu "Bernes programma". Jau sākotnēji "Bernes programma" bija nepārspēts līderis ģeodēzijas un ģeofizikas problēmu risināšanā izmantojot GPS mērījumu precīzo apstrādi. Pārsteidzoši ir tas, ka tā joprojām arī mūsdienās nav zaudējusi savu augsto reputāciju.[2]

1.2 Sistēmu apskats

Pasaulē šobrīd ir vai atrodas izveides stadijā vairākas lielas globālās navigācijas satelītu sistēmas. Šobrīd izveidotas ir ASV GNSS GPS un Krievijas GNSS GLONASS. Drīz vien šīm GNSS pievienosies Eiropas Savienības GNSS GALILEO, kurai tuvākajā nākotnē sekos arī Ķīnas COMPASS un Indijas GNSS. Bez šīm lielajām GNSS ir eksistējušas un joprojām eksistē arī vairākas mazākas pilnībā izstrādātas satelītnavigācijas sistēmas, kā piemēram, INMARSAT un citas. Šo mazo satelītnavigācijas sistēmu pielietošanas iespējas lai gan ir ierobežotas. Tomēr apskatot visas GNSS nākas secināt, ka to darbības principi ir faktiski vienādi. Katrai GNSS atšķiras tikai neprimāri parametri, kā piemēram, frekvenču diapozoni, modulācijas tipi, signālu skaits un to kombināciju variantu iespējas, kodējumi un citi. Tāpēc, lai nebūtu jāapskata katras GNSS minimālās atšķirības, tālāk apskatīsim GPS sistēmu, jo šī ir vienīgā pozicionēšanas sistēma, kuras darbības un izmantošanas pieredze ir uzkrāta 20 gadu garumā.[2]

1.2.1 Pozicionēšanas metode

Pārfrāzējot oficiālo GPS sistēmas definīciju, kas ir šāda, "GPS is an all-weather, space based navigation system under responsibility of the Department of Defence to satisfy the requirements for the military forces to acurately determine their position, velocity and time in a common reference system, anywhere on or near the Earth on a continuous basis", jāpiebilst, ka GPS ir pozicionēšanas sistēma, kas balstīta uz satelītu vienvirziena attālumu mērīšanu. Katra satelīta atrašanos telpā ir viegli aprēķināt, jo GPS satelīti pārraida signālus, kurus izmantojot tas izdodas ātri un precīzi. Lai aprēķinātu attālumu no katra no satelītiem līdz GPS uztvērēja antenai, pamatojoties uz to, ka pārraidīto signālu paketes tiek uzmanīgi sinhronizētas ar laiku, un izmērot aizkavēšanās periodu signālu paketēm, šo aizkavēšanās laiku reizina ar gaismas ātrumu c. GPS uztvērēja antenas atrašanās vietu iespējams noteikt, ģeometriski apskatot trīs sfēru, kuru centri atrodas satelītu atrašanās vietu koordinātēs, bet rādiusi jau iepriekš aprēķināti kā attālumi no katra no satelītiem līdz GPS uztvērēja antenai, krustpunktā.

Šāda ideāla situācija, kad tik viegli iespējams noteikt GPS uztvērēja antenas atrašanās vietu, iespējama tikai tādā gadījumā, ja GPS uztvērējs arī tiek apgādāts ar pulksteni ar ļoti augstu precizitāti, lai būtu pilnvērtīgi iespējams noteikt signālu pakešu aizkavēšanās perioda garumu. Bet tā kā šāda situācija reālajā dzīvē ne vienmēr ir iespējama un GPS uztvērēji tiek nodrošināti ar vienkāršiem kvarca pulksteņiem, tad, lai līdzsvarotu šo neprecizitāti jeb pulksteņa korekciju, lai noteiktu šo korekciju, krustpunkta noteikšanai izmanto nevis vien trīs, bet vismaz četras sfēras, ko iegūst no GPS satelītiem un ar kuriem vienlaicīgi veic mērījumus. Lielākā daļa GPS uztvērēja darbojas izmantojot informāciju no 6-12 GPS satelītiem. Tādējādi, lai iegūtu vismazāko izkliedi ar uztvērēja atrašanās vietas koordinātām, izvēlas tādu pulksteņu korekciju, kas iegūtu izmantojot 6-12 sfēru krustpunktu. Apskatot visu šos faktorus iespējams secināt, ka, lai noteiktu precīzu GPS uztvērēja atrašanās vietu, nepieciešams ņemt vērā trīs ietekmējošus faktoru.

- Izmantoto sfēru centru precizitāte, ko nosaka satelītu koordināšu aprēķinu precizitāte.
- Izmantoto sfēru rādiusu garumu precizitāte, ko, galvenokārt nosaka mērīšanas precizitāte, ko izmanto, lai izmērītu signālu pakešu aizkavēšanās laiku.
- Pašu satelītu sistēmas izkārtojums – konstelācijas ģeometrija, kas ietekmē matemātisko algoritmu precizitāti.[2]

1.2.2 GPS satelītu sistēma

Pati GPS satelītu sistēma satur 30-32 satelītus. No šiem sateļītiem 24 veido sistēmas funkcionējošo daļu (6 dažādās orbītu plaknēs pa 4 satelītiem(1.1. att.)), bet atlikušie vai nu ir bojāti, vai atrodas rezervē, gadījumā, ja tiek bojāts kāds no funkcionējošās daļas satelītiem.

1.1. att. GPS konstelācija.[3]

Tabulā 1.1. redzams, kādi satelīti atrodas 17.05.2014. kādās plaknēs un to aprīkojums.[4]

1.1.tabula

Orbītā esošo satelītu izkārtojums pa plaknēm un to aprīkojums 17.05.2014.

Plakne	Satelīta nr.	SVN	PRN	Satelīta sērija	Atompulkstenis
A	1	65	24	IIF	CS
A	2	52	31	IIR-M	RB
A	3	38	8	IIA	CS
A	4	48	7	IIR-M	RB
A	5	39	9	IIA	CS
A	6	64	30	IIF	RB
B	1	56	16	IIR	RB
B	2	62	25	IIF	RB
B	3	44	28	IIR	RB
Plakne	Satelīta nr.	SVN	PRN	Satelīta sērija	Atompulkstenis

B	4	58	12	IIR-M	RB
C	1	57	29	IIR-M	RB
C	2	66	27	IIF	RB
C	3	59	19	IIR	RB
C	4	53	17	IIR-M	RB
C	5	33	3	IIA	CS
D	1	61	2	IIR	RB
D	2	63	1	IIF	RB
D	3	45	21	IIR	RB
D	4	34	4	IIA	RB
D	5	46	11	IIR	RB
E	1	51	20	IIR	RB
E	2	47	22	IIR	RB
E	3	50	5	IIR-M	RB
E	4	54	18	IIR	RB
E	5	23	32	IIA	RB
E	6	40	10	IIA	CS
F	1	41	14	IIR	RB
F	2	55	15	IIR-M	RB
F	3	43	13	IIR	RB
F	4	60	23	IIR	RB
F	5	26	26	IIA	RB

GPS satelīti tiek ievadīti riņķveida orbītās ar apriņķošanas periodu $P \approx 12h$, kam atbilst GPS satelīta riņķveida orbītas lielā pusass $a = 20200\,km$, un inklināciju $i = 55°$. Šāda satelītu sistēma ar šādiem parametriem nodrošina 4-8 vienlaicīgu satelītu, reizēm 12, mērīšanas iespēju virs almukantarata $h = 15°$ jebkurā brīdī jebkurā vietā uz Zemes. Attiecīgi virs almukantarata $h = 10°$ nodrošina vienlaicīgi ap 10 satelītu, bet virs $h = 5°$ ap 12 satelītu. Šāda konstelācija atkārtojas ik pa 12 stundām, kas attiecīgi pret novērotāju no Zemes ir ik pēc 24 stundām ar 4 minūšu laika nobīdi atšķirīgo laika skalu UT (pasaules laiks) un ST(zvaigžņu laiks) dēļ.

Uz katra no GPS satelītiem atrodas 4 atompulksteņi – frekvenču standarti, datori un raidītājs. Katrs satelīts aptuveni sver 1500 kilogramu, bet tā finansiālā vērtība ir gandrīz 50 miljoni ASV dolāru. Katra satelīta plānotais funkcionēšanas laiks ir 10 gadu, bet viena daļa

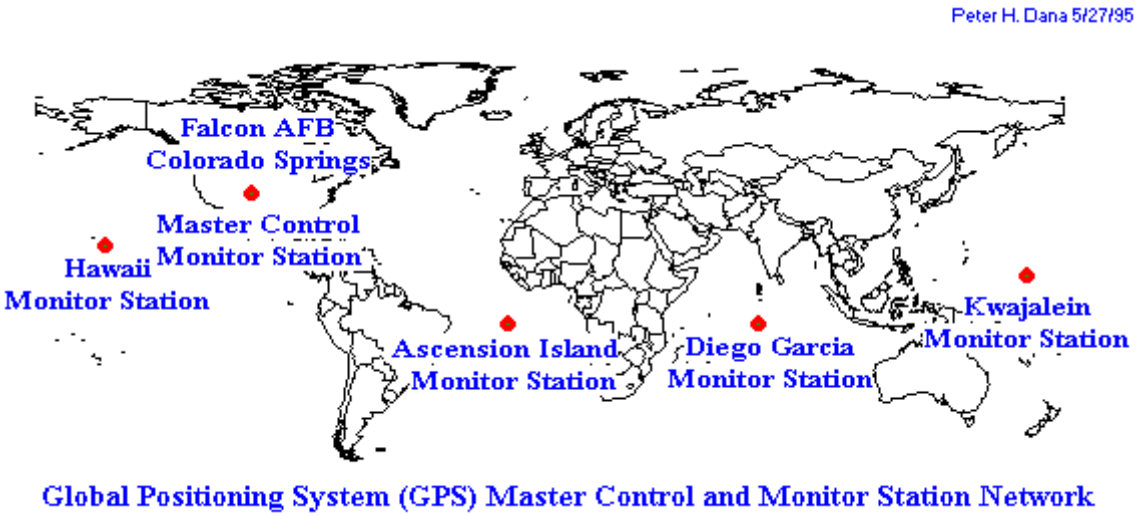
satelītu šo laiku ievērojami pārsniedz. Tabulā 1.2. redzams cik un kādi satelīti kādos laika periodos orbītās tikuši ievadīti.

1.2. tabula

Orbītā palaistie un plānotie satelīti

Laika periods	Satelītu sērijas nosaukums	Skaits
1978-1985	Block-1	10
1989-1990	Block-2	9
1990-1997	Block-2A	19
1997-2004	Block-2R	12
2005-2009	Block-2R-M	8
2010	Block-2F	(nomaina Block-2R-M satelītus)
2014	Block-3A	plānots

Lai satelīti nedegradētos un ar tiem tiktu nodrošināta visaugstākā pozicionēšanas precizitāte, satelīti no Zemes tiek regulāri kontrolēti un ne retāk kā reizi 8-12 stundās tajos tiek uzlabota informācija, tādējādi tie var teikt, ka tiek daļēji pārprogrammēti ikdienā. Kontrole un informācijas atjaunināšana notiek izmantojot piecas kontrolstacijas uz Zemes – Hawai, Colorado Springs, Ascension Island, Diego Garcia un Kwajailen (1.2. att.), kas nepārtraukti uzrauga satelītus. Galvenā kontrolstacija, kas agrāk atradās Vandenbergas gaisa kara spēku bāzē Kalifornijā, tagad ir pārcelta uz Šriveras, agrākās Falkonas, GKS bāzi Koloradospringsā. Šo kontrolstaciju sauc par CSOC. [2]



1.2. att. GPS kontrolstaciju izvietojums.[5]

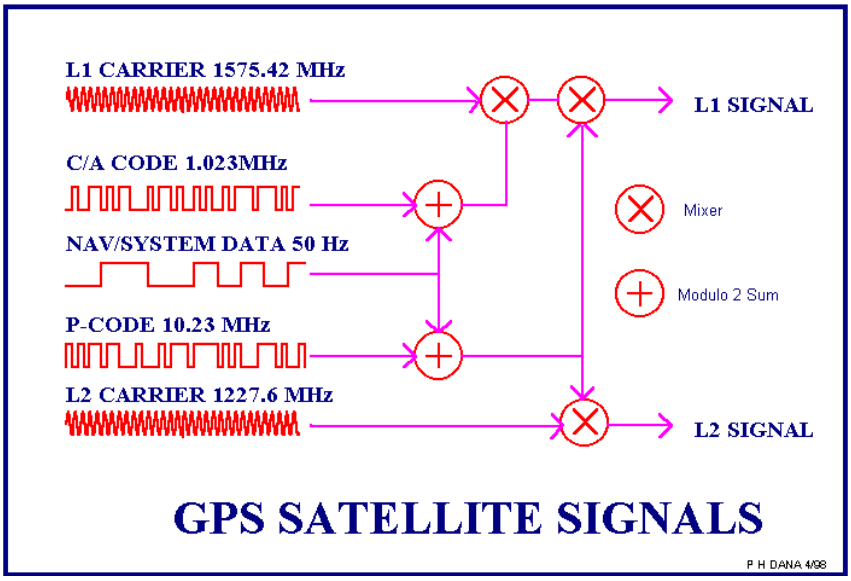
1.2.3 GPS satelītu signāli

GPS satelītu signāls ir dīvains signāls, kas ir līdzīgs binārajam "baltajam troksnim", tas ir līdzīgs haotiskai impulsu virknei, ko sauc par PRN vai pseidotroksni. Šis GPS satelītu raidītais signāls ar tā saucamo bifāzu modulācijas (BPSK) palīdzību uzmodulēts divām nesējfrekvencēm L1 un L2 ar šādiem parametriem kā frekvences lielums $f_{L1} = 1575{,}42\ MHz$, viļņa garumu $\lambda_{L1} = 0{,}19\ m$, $f_{L2} = 1227{,}6\ MHz$ un $\lambda_{L2} = 0{,}244\ m$.

Šādu neparastu un sarežģītu signāla uzbūvi izvēlējās vairākās vairāku praktisku apsvērumu dēļ, kā piemēram:

- izmantojot tik neparastu struktūru, jebkādas nesankcionētas pieejas gadījumā šāds signāls apmulsinātu pētniekus un tie no šiem signāliem nespētu izgūt nekādu informāciju;
- pateicoties pseidotroksņa jeb Golda koda savdabībai to ir neiespējami apstrādāt ar interferences slāpētājiem, ko iespējams varētu mēģināt pielietot pret GPS sistēmu gadījumā, ja rastos militārs konflikts;
- tikpat piemērotas, cik ir regulāras impulsu ķēdītes aizkavēšanās laiku mērījumiem ir arī PRN impulsu frontes.

Lai izveidotu GPS signāla kodu sākumā izmantoja divus pseidotrokšņu kodus. "Raupjais" kods C/A tika modulēts uz nesējfrekvences L1, bet "smalkais" kods P(Y) tika modulēts gan uz L1, gan uz L2 nesējfrekvencēm. (1.3. att.) Vēlāko paaudžu GPS satelīti sākot no 2010. gada izmanto trīs nesējfrekvences L1, L2 un L5. Abu pseidotrokšņu kodu modulācija izmantojot nesējfrekvenci L1 notiek ortogonālos fāzu ofsetos – kvadratūrās. Līdzīgi norisinās krāsu signāla modulācija televīzijas PAL sistēmā.



1.3. att. GPS satelītu signāla uzbūve. [6]

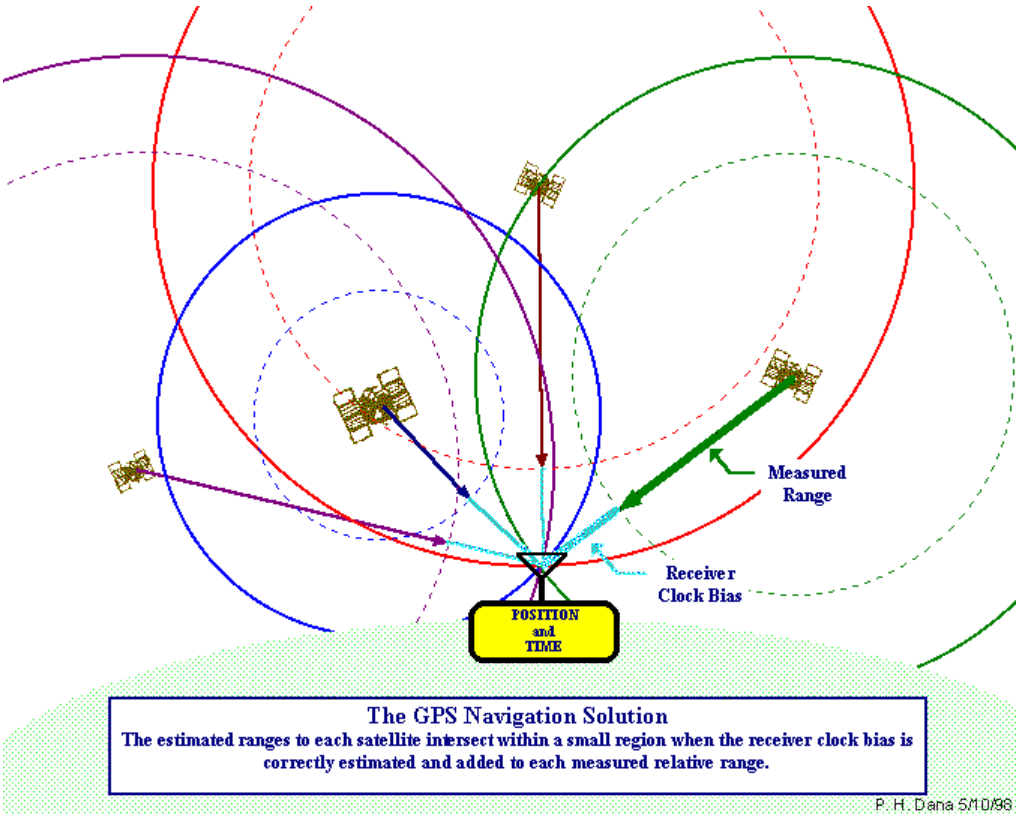
Par spīti tam, ka šis pseidotroksnis šķiet haotisks un pamatā paredzēts tikai aiztures laika noteikšanai, no kura iespējams izmērīt pseidoattālumus, tas tomēr satur arī informāciju

par satelīta atrašanās vietas koordinātām, laika signālus un pat dažādas korekcijas, kā piemēram, par jonosfēras stāvokli, un informāciju, kas satur statusu par satelīta sistēmu funkcionēšanu. Šo informāciju uzklāj virs pseidotrokšņa signāla un tā tiek pārraidīta ļoti lēni – ar takts frekvenci $50\ Hz$. Šo informāciju sauc par navigācijas paketi. Lai šo informāciju izdalītu, katrs GPS uztvērējs satur mikroshēmu, kas attiecīgi ģenerē vai nu visus vai tikai "raupjos" C/A pseidotrokšņu kodus. Lai iegūtu navigācijas paketes signālu, tiek salīdzināts GPS uztvērēja uztvertais signāls ar pseidotrokšņa ģeneratora ģenerēto signālu. Attiecīgi no pirmā signāla tiek atņemts otrais. Navigācijas paketēm arī ir noteikts datu pārraides protokols, kas nosaka, ka katra pakete satur 5 kadrus, kur katrs no tiem ir 300 bitu garš. Informācija, ko satur navigācijas paketes atkārtojas ik pēc 25 paketēm. Tādēļ nepieciešamas 12,5 minūtes, lai uztvertu pilnu sistēmas informāciju. Lai gan papildus katrs GPS satelīts 4. un 5. kadrā pārraida informāciju par visu GPS sistēmu kopumā.[2]

1.2.4 Pseidoattālumu mērīšana

Kā jau iepriekš uzzinājām katrs GPS uztvērējs spēj atdalīt pseidotrokšņa signālu no navigācijas paketes un attiecīgi no tā iegūt informāciju, lai aprēķinātu satelīta tekošās koordinātes, kas apzīmē sfēru centrus līdz 1 metra precizitātei. Procedūru, kad tiek salīdzināts GPS uztvērēja uztvertais un no navigācijas paketes atdalītais pseidotrokšņa signāls ar uztvērējā iebūvētā PRN ģeneratora signālu, izmantojot kodu korelatoru un tam tiek noteikta nobīde laikā attiecīgi pret otru signālu, sauc par pseidoattāluma mērīšanu. Tā kā abu PRN signālu frontes tiek uzmanīgi sinhronizētas ar laiku, bet ar atšķirīgiem frekvenču standartiem – pārraidītajam pseidotrokšņu signālam izmantojot satelītā atrodošos 4 atompulksteņu frekvenču standartus un GPS uztvērējā ģenerētajam PRN signālam izmantojot kvarca pulksteņus, tad arī nomērītā pseidotrokšņu signālu nobīde tiek veikta izmantojot atšķirīgus standartus un ar atšķirīgu precizitāti. Šādus mērījumus sauc par pseidoattālumiem, jo, kamēr nav noteikta uztvērēja pulksteņa korekcija, tie nav pietiekami precīzi un tos nevar saukt par attālumiem.

1.4. att. redzama atšķirība starp patiesajiem attālumiem un pseidoattālumiem. Šos attālumus - pseidoattālumus (pārtrauktās līnijas) un patiesos attālumus (nepārtrauktās līnijas) pēc satelītu signāliem mēģina noteikt GPS uztvērējs.



1.4.att. Pseidoattālumu un patieso attālumu atšķirības. [7]

Tāpat jāņem vērā vairāki traucējoši faktori, kas var traucēt noteikt ne tikai pareizos attālumus, bet arī pseidoattālumus. Kā piemēram:

- jonosfēras refrakcija, kas rada kļūdas attālumu mērījumos, kas var sasniegt 40-50 metrus;
- troposfēras refrakcija, kas rada kļūdas mērījumos ar kārtu 1-3 metri;
- relatīvistiskie efekti, kas rada kļūdas mērījumos ar kārtu 0,2 metri;
- multirefleksija, kas rada ne vien kļūdas mērījumos lielākas par 10-20 metriem, bet arī var pilnībā šo procesu sabojāt.

Refrakcija ir gaismas stara noliekšanās, tam ejot cauri Zemes atmosfērai, kā cēlonis ir Zemes atmosfēras nehomogenitāte [1]. Lai apkarotu visievērojamāko no mērījumus apgrūtinošajiem faktoriem – jonosfēras refrakciju, kas ir atkarīga no elektronu vertikālās koncentrācijas jonosfērā, izmanto dažādas metodes, kā piemēram:

- speciālās jonosfēras observatorijās mēra šo elektronu vertikālo koncentrāciju jonosfērā, un pieejamie dati par atsevišķiem reģioniem ir pieejami internetā, piemēram, par Japānu;
- paildzinot mērīšanas procesu, lai iegūtu papildus informāciju par procesiem jonosfēra, mērot GPS mērījumus, iegūst modeļa parametrus, ko izmanto, lai modelētu jonosfēras refrakciju;

- izmantojot navigācijas paketē iekļauto informāciju, jonosfēras refrakciju tuvināti aprēķina;
- izmantojot nesējfrekvences L1 un L2, tas ir realizējot "matemātisko hetorodinu" jeb no mērījumiem izrēķinot starpfrekvenci, jonosfēras refrakciju izslēdz, jo starpfrekvencei ir nulles jonosfēras refrakcija.

Pēdējā metode arī vislabākā no visām metodēm, ar kurām cenšas novērst jonosfēras refrakcijas nelabvēlīgo iespaidu, un arī pierāda kādēļ jau sākumā GPS signāls tika pārraidīts pa divām nesējfrekvencēm.

Otrs apgrūtinošākais faktors ir multirefleksija, kura ir atkarīga no uztvērējam tuvumā esošām atstarojošām virsmām pret ko atstarojas GPS signāli, un, kas nodrošina signāla vairākkārtēju nonākšanu uztvērējā. Šāda situācija ir vēl bīstamāka un sarežģītāka, jo var pilnībā sabojāt GPS uztvērēja darbību. Lai no šādas situācijas būtu iespējams izvairīties, nepieciešams uztvērēju uzstādīt tālu no atstarojošām virsmām. Vēl viens variants ir antenu ekranēt no apakšas un nedaudz no sāniem. Vēl pēdējā laikā aktuāla kļuvusi polarizācijas filtru lietošana, jo tiešais un atstarotais signāls ir cirkulāri pretēji polarizēti. Tāpat, protams, ir izstrādātas vairākas sarežģītas metodes, kas balstās uz to, ka efekts signāliem ar atšķirīgām nesējfrekvencēm L1 un L2 ir atšķirīgs.[2]

1.2.5 Pozicionēšanas matemātiskie modeļi

Lai aprēķinātu sfēru krustpunktus, pēc kuriem GPS uztvērēja procesors noteiktu antenas koordinātes, vispirms no nomērītā pseidotrokšņa signāla nobīdes vienam pret otru izved formulu

$$\overline{\Delta t} = \overline{t_r} - \overline{t_s} = (t_r - \delta_r) - (t_s - \delta_s) = \Delta t + \Delta\delta \qquad (1.1.)$$

kur

$$\Delta \mathrm{t} = t_r - t_s \ , \qquad (1.2.)$$

$$\Delta\delta = \delta_s - \delta_r \approx -\delta_r \ , \qquad (1.3.)$$

$\overline{\Delta t}$ – nomērītā PRN signāla nobīde vienam pret otru,

$\overline{t_r}$ – uztvērējā reģistrētais PRN signāla uztveršanas moments pēc uztvērēja pulksteņa,

$\overline{t_s}$ – PRN signāla emisijas moments no satelīta antenas pēc satelīta pulksteņa,

t_r – uztvērējā reģistrētais PRN signāla uztveršanas moments pēc absolūtā laika,

δ_r – uztvērēja pulksteņa korekcija,

t_s - PRN signāla emisijas moments no satelīta antenas pēc absolūtā laika,

δ_s – satelīta pulksteņa korekcija, ko iespējams uzzināt no informācijas navigācijas paketē.

Pēc tam, izmantojot izvesto formulu (1.1), un pareizinot to ar gaismas izplatīšanās ātrumu iegūstam formulu, kas aprēķina satelīta pseidoattālumu.

$$R = c\overline{\Delta t} = c(\Delta t + \Delta\delta) = c\Delta t + c\Delta\delta = \rho + c\Delta\delta, \qquad (1.4.)$$

kur

$\rho = c\Delta t$ – satelīta patiesais attālums no GPS uztvērēja antenas,

R – satelīta pseidoattālums,

$\Delta\delta$ – starpība starp satelīta un uztvērēja pulksteņa korekciju,

c – gaismas (mikroviļņu) izplatīšanās ātrums.

Vienkāršības labad pieņemam, ka mērījumu sistemātiskās kļūdas un citi traucējošie faktori tiek ņemti vērā un ir iekļauti pseidoattālumu aprēķinos. Tā kā formula (1.4) tiek izmantota katram satelītam, tiek pievienots indekss j tiem parametriem, kas ir atkarīgi no katra konkrētā satelīta, bet patiesais attālums satelītam no GPS uztvērēja antenas atkarībā no laika rakstāms formā

$$\rho_j(t) = \sqrt{(x_j(t) - X)^2 + (y_j(t) - Y)^2 + (z_j(t) - Z)^2} = \rho_j(X, Y, Z), \qquad (1.5.)$$

kur

$x_j(t), y_j(t), z_j(t)$ – no laika atkarīgas satelīta koordinātes, kuras iespējams aprēķināt pēc informācijas, kas atrodama navigācijas paketē,

X, Y, Z – nezināmas, konstantas GPS uztvērēja antenas koordinātes.

Pēc formulas (1.5) redzams, ka patiesais attālums no kāda satelīta līdz GPS uztvērēja antenai ρ_j ir gan no laika, gan no GPS uztvērēja antenas koordinātēm atkarīga funkcija. Ja katram no j satelītiem atbilstošajā formulā (1.4) ievietojam tam atbilstošo vienādojumu, kas apzīmē patieso attālumu no uztvērēja antenas līdz satelītam, iegūstam šādu vienādojumu sistēmu

$$\begin{cases} \rho_1(X, Y, Z) + c\Delta\delta = R_1 \\ \rho_2(X, Y, Z) + c\Delta\delta = R_2 \\ \vdots \\ \rho_j(X, Y, Z) + c\Delta\delta = R_j \end{cases}, \qquad (1.6.)$$

kas satur j vienādojumus, kur katrs no tiem satur 4 nezināmos – GPS uztvērēja antenas koordinātes X, Y, Z un laika korekciju $\Delta\delta$. Lai aprēķinātu šos četrus nezināmos, nepieciešama vienādojumu sistēma ar vismaz $j = 4$, kas nozīmē, ka nepieciešami vismaz 4 satelīti, lai noteiktu uztvērēja antenas atrašanās vietu un laika korekciju. Šādiem vienādojumiem bieži tiek veikta vienādojumu linearizācija, jo vienādojumu sistēma (1.6.) atkarībā pret nezināmajām uztvērēja antenas koordinātēm ir nelineāra. Lai veiktu linearizāciju tiek meklētas korekcijas $\Delta X, \Delta Y, \Delta Z$ attiecībā pret tuvinātajām antenas koordinātēm X_0, Y_0, Z_0. Šādu pielietoto matemātisko modeli sauc par absolūto kodu pozicionēšanas modeli.[2]

1.2.6 Fāzu pozicionēšana

Pateicoties bifāzu modulācijas, ko izmanto GPS signālu formēšanā, īpašībai, kas izpaužas tajā, ka paceļot signālu kvadrātā jeb reizinot signālu pašu ar sevi, modulācija pazūd, iespējams iegūt gandrīz sinusoidālu signālu ar dubultotu nesējfrekvenci. Šo signālu iespējams izmantot, lai iegūtu fāzu diferences pseidoattālumu noteikšanai uztvērējā ģenerētā atbalsta signāla un uztvertā demodulētā signāla starpā. Tas nozīmē, ka ir iespējams pielietot signālu interferometriskās apstrādes metodes, kuras, starp citu, izceļas ar ļoti augstu precizitāti. Vēl pierādījums šo metožu efektivitātei un precizitātei ir tas, ka tās ir pamatā precīzajiem ģeodēziskajiem un ģeofizikālajiem mērījumiem, kā piemēram, populārajai RTK metodei. Pamatojoties uz šo fāzu pozicionēšana izceļas ar tās priekšrocību augsto precizitāti, kā arī nedrīkst nepieminēt tās trūkumu cikliskās nenoteiktības (integer ambiguity) parādīšanos. Cikliskā nenoteiktība ir tāds fenomens, kas notiek interferometriskai ainai identiski atkārtojoties, izmainoties attālumam starp GPS antenu un satelītu par nesējfrekvences viļņa garumu. Tādējādi ir grūti noteikt to, cik daudz pilna viļņu garuma intervālu aizpilda attālumu no satelīta līdz antenai. Fāzu pozicionēšanas gadījumā, lai noteiktu satelīta pseidoattālumu izmanto formulu

$$R = N\lambda + \phi\lambda\,, \tag{1.7}$$

kur

R – satelīta pseidoattālums,

N – cikliskā nenoteiktība (vesels skaitlis),

λ – demodulētā signāla viļņa garums,

$\phi\lambda$ – pēc fāzu nobīdes interferometriski nomērītā pseidoattāluma korekcija,

ϕ – fāzu nobīde ($0 \leq \phi < 1$). Ievietojot (1.6) vienādojumu sistēmā formulu (1.7) iegūst fāzu pozicionēšanas vienādojumu sistēmu

$$\begin{cases} \rho_1(X,Y,Z) + c\Delta\delta = N_1\lambda + \phi_1\lambda \\ \rho_2(X,Y,Z) + c\Delta\delta = N_2\lambda + \phi_2\lambda \\ \vdots \\ \rho_j(X,Y,Z) + c\Delta\delta = N_j\lambda + \phi_j\lambda \end{cases},$$

$$\begin{cases} \rho_1(X,Y,Z) + c\Delta\delta - N_1\lambda = \phi_1\lambda \\ \rho_2(X,Y,Z) + c\Delta\delta - N_2\lambda = \phi_2\lambda \\ \vdots \\ \rho_j(X,Y,Z) + c\Delta\delta - N_j\lambda = \phi_j\lambda \end{cases}, \tag{1.8}$$

kas satur j vienādojumus, bet $j + 4$ nezināmos $X, Y, Z, \Delta\delta, N_1, N_2, \ldots, N_j$. Ja tiek veikts tikai viens fāzu nobīdes ϕ_j mērījums līdz katram no satelītiem, tad šādai sistēmai, diemžēl, nav viennozīmīga atrisinājuma. Arī jaunu satelītu pievienošana mērīšanas procesam situāciju

neuzlabo, pamatojoties uz to, ka cikliskās nenoteiktības vērtība N_j katram satelītam ir atšķirīga. Līdz ar to pievienojot jaunus satelītus, vienādojumu sistēmai (1.8) tiek pievienoti jauni vienādojumi un līdz ar to arī pa vienam jaunam nezināmajam – katra satelīta cikliskajai nenoteiktībai N_j.

Tāpēc tiek izstrādāts liels daudzums dažādu metožu, no kurām nav iespējams noteikt, kura no tām ir labākā un precīzākā, lai noteiktu nezināmos $X, Y, Z, \Delta\delta$ un atrisinātu vienādojumu sistēmu (1.8), kā arī, lai noteiktu visas cikliskās nenoteiktības $N_1, N_2, \dots, N_j$, jo visas metodes ir ne tikai ar priekšrocībām, bet arī ar trūkumiem. Metožu izstrādi nevar nosaukt par pabeigtu, jo literatūrā vēl arvien tiek piedāvātas arvien jaunas metodes, lai atrisinātu dažādus konkrētus navigācijas vai pozicionēšanas uzdevumus. Apskatīsim tikai vienu no šīm metodēm, kuras pamatideja ir vairākkārtēji mērījumi, lai izmērītu pseidoattālumu līdz katram satelītam. Ņemot vērā faktu, ka satelīti nepārtraukti atrodas kustībā attiecībā pret izmantoto koordinātu sistēmu (parasti WGS-84), tas nozīmē, ka viena un tā paša satelīta dažādos laika momentos t izmērītie pseidoattālumi $R_j$vai to korekcijas $\phi_j\lambda$ būs savstarpēji atšķirīgi, šo procesu apzīmēsim ar indeksu t. Tiek veikti t pseidoattālumu korekciju mērījumi j satelītiem ar nekustīgu GPS uztvērēja antenu. Šādā gadījumā vienādojumu sistēma (1.8) sastāvēs no tj vienādojumiem un nezināmajiem, ko sastādīs j cikliskās nenoteiktības N_j plus t pulksteņa korekcijas $\Delta\delta_t$, plus trīs antenas koordinātes X, Y, Z. Tāpēc, lai atrisinātu šo vienādojumu sistēmu (1.8) nepieciešams vairāk vienādojumu nekā nezināmie

$$jt \geq j + t + 3\,, \tag{1.9}$$

kur

j – satelītu skaits,

t – pseidoattālumu mērījumu skaits.

Ja līdzīgi kā iepriekš, lai atrisinātu vienādojumu sistēmu ir nepieciešami 4 satelīti, tad tagad no nosacījuma (1.9) izriet, ka ar katru no satelītiem jāveic 3 vai vairāk pseidoattālumu mērījumi. Lai papildus visam katru reizi klāt nenāktu arī atšķirīgas antenas koordinātes X, Y, Z katrā pseidoattālumu mērījumu reizē, mērījumi obligāti jāveic ar nekustīgu GPS uztvērēja antenu. Tādā veidā netiek palielināts nezināmo skaits vienādojumu sistēmā (1.8). Antenas kustību, nenojaucot pozicionēšanas procesu, iespējams uzsākt tikai pēc tam, kad ir noteiktas visas cikliskās nenoteiktības N_j. Tieši šāda iemesla dēļ šo algoritmu nav iespējams pielietot aviācijā un tas rada problēmas arī izmantojot ar citiem kustīgiem objektiem. Tā kā negaisa, satelītu konstelācijas izmaiņu vai citu iemeslu dēļ satelītu iespējams uz īsu brīdi pazaudēt, matemātiskais process nojūk, un, lai to atjaunotu bez GPS

antenas un kustīgā objekta, uz kura tā atrodas, apstādināšanas, pietiekami komplicēti un diez gan rafinēti algoritmi.[2]

1.2.7 Precizitātes "šķīšana" jeb DOP faktori

Kā jau iepriekš tika minēts vienādojumu sistēmas (1.6) un (1.8) bieži tiek linearizētas pirms to risināšanas un, tādējādi, tiek reducētas matricu formā

$$\underline{Ax} = \underline{l}, \tag{1.10}$$

kur

$\underline{A}$ – koeficientu matrica,

$\underline{x}$ – parametru vektors,

$\underline{l}$ – mērījumu vektors, tas ir,

$$\underline{x^T} = \{\Delta X, \Delta Y, \Delta Z, c\Delta\delta\}, \tag{1.11}$$

kur

^{T} – transponēšanas operācija, bet GPS antenas koordinātes tiek aprēķinātas pēc formulām

$$X = \Delta X + X_0,$$
$$Y = \Delta Y + Y_0,$$
$$Z = \Delta Z + Z_0,$$

kur

X_0, Y_0, Z_0 – antenas tuvinātās koordinātes.

Lai būtu iespējams atrisināt sistēmu (1.10) jāizpildās nosacījumam $|\underline{A}| > \varepsilon > 0$. Attiecīgi aprēķinātais parametru vektors $\underline{x}$ ir neprecīzs, ja šis nosacījums neizpildās. Vektora $\underline{x}$ precizitāti lineārā tuvinājumā korelāciju teorija ļauj raksturot ar korelāciju matricu. Matricas diagonāle satur aprēķināto parametru dispersijas (standartnoviržu kvadrātus) $\sigma^2_{\Delta X}$, $\sigma^2_{\Delta Y}$, $\sigma^2_{\Delta Z}$, $\sigma^2_{c\Delta t}$.

Lai raksturotu GPS sistēmā pozicionēšanas precizitāti, izmanto normalizētas standartnoviržu funkcijas, kuras sauc par DOP faktoriem. Izšķir šādus DOP faktorus: GDOP, PDOP, TDOP, HDOP, VDOP.

$$GDOP = \sqrt{\frac{\sigma^2_{\Delta X}}{\sigma^2} + \frac{\sigma^2_{\Delta Y}}{\sigma^2} + \frac{\sigma^2_{\Delta Z}}{\sigma^2} + \frac{\sigma^2_{c\Delta t}}{\sigma^2}}$$

$$PDOP = \sqrt{\frac{\sigma^2_{\Delta X}}{\sigma^2} + \frac{\sigma^2_{\Delta Y}}{\sigma^2} + \frac{\sigma^2_{\Delta Z}}{\sigma^2}}$$

$$TDOP = \sqrt{\frac{\sigma_{c\Delta t}^2}{\sigma^2}}$$

kā arī HDOP un VDOP, kurus iegūst transformējot uz horizontālo koordinātu sistēmu atbilstošās standartnovirzes no WGS-84 koordinātu sistēmas. GDOP, PDOP un TDOP aprēķinos ar σ tiek apzīmēta pseidoattālumu mērījumu standartnovirze jeb vidējā kvadrātiskā kļūda. Pamatojoties uz korelācijas teoriju, no kuras izriet, ka DOP faktori ir tikai atkarīgi no satelītu sistēmas (konstelācijas) ģeometrijas nevis paša mērīšanas procesa, DOP faktorus ir iespējams aprēķināt iepriekš. Kas nozīmē, ka laika periodos, kad DOP faktori ir apmierinoši nelieli, ir iespējams plānot GPS novērojumu seansus. Lai arī lielākoties par apmierinošiem tiek uzskatīti tādi GPS mērījumi, kad atbilstošie DOP faktori nav lielāki par 6, nereti tiek izmantotas arī mazākas DOP faktoru vērtības. PDOP faktoram ir vienkārša ģeometriskā interpretācija – faktors ir apgriezti proporcionāls prizmas tilpumam, kur viena virsotne atrodas GPS uztvērēja antenas koordinātēs, bet pārējās virsotnes veido visi vienlaikus mērāmie satelīti.[2]

2. TRANSPORTA KONTROLES IEKĀRTU UN SISTĒMU PĀRSKATS

2.1. Autoparka kontroles sistēmas

Tā kā daudzos uzņēmumos, kas balstās uz transporta izmantošanu – loģistikas uzņēmumos, taksometru pakalpojumu sniedzējos, sabiedriskā transporta uzņēmumos vai jebkurā uzņēmumā, kura īpašumā atrodas lielāks vai mazāks autoparks, nepieciešama atvieglota un uzticama autoparka pārraudzība, arvien populārāki kļūst uzņēmumi, kas piedāvā automatizētas autoparka pārvaldes pakalpojumus.

2.1.1 Seko.lv sekošanas sistēma

Seko.lv ir uzņēmuma SIA "Karšu izdevniecība Jāņa sēta" veidota sekošanas sistēma, kas piedāvā GPS sekošanu transportam, GPS sekošanu personām un GPS navigāciju. Izmantojot GPS sekošanu transportam, sistēma piedāvā samazināt autotransporta un nodokļu izmaksas, palielināt autoparka izmantošanas efektivitāti, kā arī atvieglot loģistikas plānošanu un grāmatvedības uzskaiti. Transporta kontroles sistēmas mērķis ir atvieglot autoparka pārvaldību un vadītāju ikdienas darbu, nodrošinot attālinātu autotransporta izmantošanas uzskaiti, plānošanu, komunikāciju un kontroli. Tāpat seko.lv sekošanas sistēma veic automašīnas datu analīzi, kā rezultātā pārvaldītājs spēj novērtēt auto ekspluatācijas efektivitāti un uzlabot to, piemēram, salīdzinot nobraukumus, plānojot maršrutus un degvielas patēriņu vai novērst transporta izmantošanu nesankcionētos laika intervālos. Izmantojot šo sistēmu tiek atvieglota komunikācija ar vadītāju un strīdu risināšana.

Transporta kontroles sistēma piedāvā šādas pamatfunkcijas: operatīva auto atrašanās vietas noteikšanu, detalizētas nobraukuma un autoparka izmantošanas atskaites, degvielas atskaites, degvielas rēķinu importu un datu apstrādi, notikumu un brīdinājumu ziņojumus par pārkāpumiem (ātrums, kustība, teritorija u.c.), tādus loģistikas pakalpojums, kā maršruta plānošanu, pieturvietu optimizēšanu un izpildes kontroli, ziņojumu apmaiņu un pieturvietu nosūtīšanu auto vadītājam uz Garmin navigāciju.

"Karšu izdevniecība Jāņa sēta" piedāvā divu veidu sekošanas sistēmas - Interneta versiju un Stacionāro. Interneta risinājuma sekošanas sistēma sastāv no vairākām komponentēm. Galvenā komponente ir sekošanas iekārta, ar ko tiek aprīkota pati automašīna, lai varētu noteikt ne tikai tās atrašanās vietu, bet arī citus parametrus kā piemēram aizdedzes stāvokli, durvis un citus. Bez sekošanas iekārtas sistēma sastāv no SIM

kartes un sistēmas programmatūras risinājuma. SIM karti ievieto sekošanas iekārtā, lai nodrošinātu datu pārsūtīšanu un saņemšanu no datu bāzes. Programmatūras risinājums sevī ietver saņemto datu attēlošanu uz kartes un satura analīzi.

Sistēmas darbības pamatā esošā sekošanas iekārta izmantojot tajā ievietoto SIM karti un SMS vai GPRS pārsūta datus uz Internetu, kur tie tiek tālāk nosūtīti datu bāzei. Datu bāze, pamatojoties uz WEB aplikāciju servera pieprasījumu, nosūta datus, no kurienes tie tālāk tiek attēloti lietotājam pēc veiksmīgas autentifikācijas sistēmā. Lietotājs pēc pieprasījuma var izgūt gan informāciju par automašīnas atrašanās vietu, gan dažādas atskaites un cita veida informāciju par automašīnu. Sekošanas iekārta savu atrašanās vietu nosaka izmantojot GPS satelītus. Labos laika apstākļos, sekošanas iekārtas precizitāte ir no 2 līdz 10 metriem, praksē norādītā precizitāte visbiežāk ir 3 metri. Atrodoties starp augstām mājām un debesskrāpjiem, signāls var atstaroties pret sienām, kas rezultējas nobīdē pret faktisko atrašanās vietu.

Stacionārais varianta darbības princips ir līdzīgs Interneta risinājuma sekošanas sistēmai. Tam ir raksturīgi un no Interneta risinājuma atšķiras ar to, ka datu bāze un programmatūra atrodas klienta serveros un darbstacijās, maršrutu atvēršanas un transporta kontroles saskarne ir labi pārskatāma un viegli uztverama, tajā var veikt ne tikai kontroles, bet arī transporta kustības plānošanas pasākumus - plānoto maršrutu optimizēšanu, risinājums ir piesaistīts vienam vai vairākiem datoriem atkarībā no programmas "JS Baltija 2 Professional" licenču skaita, klients var pasūtīt individuālu risinājumu programmas funkcionalitātei un karšu materiāls un funkcionalitāte tiek piedāvāta atjaunotās programmas versijās ik pēc diviem gadiem.

Stacionārais sekošanas sistēmas risinājums vairāk paredzēts tieši atskaišu izveidei un apskatei. Stacionārā sekošanas sistēmas režīmā šīs iespējas piedāvā "JS Baltija 2" Professional aplikācija.

2.1.2 SIA "Trackpoint" autoparka vadības sistēma

SIA "Trackpoint" autoparka vadības sistēma piedāvā tiešsaistē sekot līdzi sava autoparka aktivitātēm: apskatīt kartē visu automašīnu atrašanās vietu kopskatu, kā arī katru automašīnu atsevišķi, uzzināt, vai tās pārvietojas vai stāv, kā arī noskaidrot, ar kādu ātrumu tās pārvietojas.

Trackpoint piedāvā izveidot vairāku veidu atskaites, kas sniedz visplašāko informāciju par transportlīdzekļa ekspluatāciju jebkurā laika periodā. Sistēma darbinieku vietā automātiski aizpilda ceļazīmes ar faktisko informāciju. Papildu esošajam atskaišu klāstam Trackpoint piedāvā bez maksas izstrādāt arī specializētas atskaites un grafiskos

pārskatus. Atšķirībā no "Karšu izdevniecības Jāņa sēta" seko.lv sekošanas sistēmas Trackpoint piedāvā ar SMS, e-pasta vai Trackpoint sistēmas palīdzību saņemt nepārtrauktus un tūlītējus paziņojumus un atgādinājumus par būtiskiem notikumiem. Piemēram, ja transportlīdzeklis pārsniedz atļauto ātrumu, ir nolieta degviela, automašīnu lieto nepiemērotā laikā un vietā vai arī nepieciešams veikt tehnisko apskati. Trackpoint sistēmā visa svarīgā informācija tiek saglabāta automātiski un ir pieejama jebkurā laikā. Lai veiktu degvielas kontroli Trackpoint piedāvā vairākus risinājumus degvielas patēriņa un izmaksu kontrolei – gan izmantojot papildu mērierīces, gan esošo pludiņu un transportlīdzekļu borta datoru. Autoparka pārvaldības sistēmā izmantotā Garmin savienojumu sistēma ļauj autovadītāja navigācijas ierīci lietot kā ērtu saziņas līdzekli. Iespējams nosūtīt ziņu, darba uzdevumu vai galapunktu uz Garmin iekārtu, un autovadītājam ir iespējams atbildēt, izmantojot transportlīdzekļa navigācijas ierīci.

Trackpoint darbības princips balstās uz to, ka GPS satelīts nosaka ar Trackpoint sistēmu aprīkotā transporta līdzekļa atrašanās vietu, Trackpoint GPS iekārta apkopo visus nepieciešamos datus, koordinātes un GPS iekārtas apkopotie dati tiek nosūtīti uz serveri informācijas pārstrādei. Pēc tam sistēmas lietotāji ar individuālu lietotāja vārdu un paroli no jebkuras pasaules vietas, izmantojot datoru ar interneta pieslēgumu, var ērti iegūt visu nepieciešamo informāciju.

Trackpoint iespējas iespējams paplašināt pieslēdzot GPS iekārtai dažādas papildierīces, kā piemēram, degvielas mērītāju, signalizāciju, termometrus kravas telpai, GPS navigāciju, vadītāja autorizācijas iekārtu.

2.1.3 KurTuEsi GPS sekošanas sistēma

KurTuEsi atšķirībā no iepriekšminētajām sekošanas un autoparka pārvaldības sistēmām piedāvā ne tikai GPS iekārtas, kartes un pašu izstrādāto programmatūras risinājumu, bet ir iespējams arī piesaistīt jau esošas GPS iekārtas, kas atbalsta datu pārraidi ar GPRS pie KurTuEsi servera, izmantojot KurTuEsi sniegtos parametrus uzstādījumiem un reģistrējoties sistēmā, kur iespējams pievienot savu ierīci datubāzei, aizpildot nepieciešamos laukus.

Līdzīgi kā SIA "Karšu izdevniecība Jāņa sēta" piedāvātā sekošanas sistēma seko.lv arī KurTuEsi autoparka uzraudzībai izmanto pasaulē plaši izmantotās GNSS tehnoloģijas, mijiedarbībā ar GSM/GPRS datu pārraides tehnoloģijām. Piedāvātā GNSS sistēma, kas darbojas NAVSTAR (ASV) tīklā, kas plašākam lietotāju lokam pazīstama kā GPS, nepārtraukti fiksē automašīnu atrašanās vietu, bet GSM/GPRS tehnoloģija nodrošina atrašanās vietas informācijas (ģeogrāfiskās koordinātes) tūlītēju nosūtīšanu uz centrālo datu

glabātuvi, kur dati tiek tūlītēji apstrādāti un publicēti internetā bāzētā kartē, nodrošinot autoparka ērtu pārraudzību no jebkuras vietas pasaulē, kur pieejami interneta sakari. Pārraidītā informācija no automašīnas satur ģeogrāfisko atrašanās vietu, laiku, kad automašīna bijusi konkrētā vietā, kā arī informāciju par automašīnas dzinēja stāvokli (ieslēgts/izslēgts), u.c.

Uzkrājot no GPS sekošanas iekārtām saņemto informāciju, kurtuesi.lv sistēma nodrošina klientam iespēju ģenerēt un saņemt mūsu izstrādātās standarta atskaites par katru transporta vienību atsevišķi vai arī par autoparku kopumā. Iespējams arī veidot specifiskas, klienta vajadzībām pielāgotas atskaites.

Atsevišķas izsekošanas iekārtas ir aprīkotas ar tā saucamo SOS drošības funkciju, kuras aktivizācijas gadījumā uz kartes tiek attēlota vieta un laiks, kad tā aktivizēta.

KurTuEsi sistēma atšķiras no iepriekšminētajām arī ar to, izmantojot izstrādāto aplikāciju viedtālruņiem, iespējams mobilo tālruni piesaistīt serverim, tādējādi iespējams pārvaldīt savu autoparku, neiegādājoties atsevišķas GPS sekošanas iekārtas. Lai gan jāpiebilst, ka šī iespēja pieejama tikai to viedtālruņu īpašniekiem, kuru viedtālruņi darbojas Windows Mobile 6 vidē.

2.2 Tiešsaistes sabiedriskā transporta sekošanas sistēmas

Ikdienas ritmam kļūstot arvien straujākam un arvien lielāku lomu ikdienas ritmā ieņemot pēkšņām plānu maiņām, arī sabiedriskajam transportam ir jābūt kā sabiedrotajam nevis kā traucēklim. Līdz ar to pasažieriem ir svarīgi uzzināt viņiem aktuālā sabiedriskā transporta atrašanās vietu, lai vajadzības gadījumā nekavējoties būtu iespējams mainīt plānus un nokļūt paredzētajā vietā paredzētajā laikā. Šādas sistēmas pasaulē ir plaši izplatītas un tādas ir ne tikai daudzām lielpilsētām sabiedriskajam transportam, bet arī starppilsētu sabiedriskajam transportam. Šādas sistēmas ļoti izplatītas ir tieši ASV un Kanādas pilsētu sabiedriskā transporta tīklā.

2.2.1 Mountain Line Bus Tracker sabiedriskā transporta sekošanas sistēma

Mountain Line Bus Tracker sistēma ir veidota Misulas (Missoula) pilsētas, Montanas štatā, ASV sabiedriskā transporta tīklam. Mountain Line aplikācija ir bezmaksas un tā ir pieejama gan tiešsaistes režīmā Mountain Line mājaslapā, gan arī viedtālruņu lietotājiem, kas izmanto Android un iOS operētājsistēmas. Tādējādi tiek aptverts plašs lietotāju tīkls, jo

aplikācija ir pieejama gan stacionāro un portatīvo datoru lietotājiem neatkarīgi no operētājsistēmas, gan viedtālruņu un planšetdatoru lietotājiem, kuru ierīces darbojas izmantojot Android un iOS operētājsistēmas.

Līdzīgi autoparka kontroles sistēmām arī Mountain Line Bus Tracker darbojas balstoties uz to, ka GPS koordinātas periodiski tiek sūtītas uz serveri, no kurienes tās ar aplikācijas palīdzību tiek attēlotas lietotājam saprotam veidā kartē. Mountain Line piedāvā papildus arī maršruta plānotāju. Visas šīs iespējas iespējamas jebkuram lietotājam, bet reģistrējoties iespēju klāsts paplašinās pievienojot iespēju saņemt atgādinājumus par interesējošiem maršrutiem un to pienākšanas laikiem interesējošās pieturās.

2.2.2 StarTran Bus Tracker sabiedriskā transporta sekošanas sistēma

StarTran Bus Tracker sekošanas sistēma paredzēta Nebraskas universitātes Linkolnā, Nebraskas štatā, ASV, studentiem un Linkolnas iedzīvotājiem pilsētas sabiedrisko autobusu tīklam. Tās lietotāji redz savos viedtālruņos tiem interesējošā autobusa atrašanās vieta un paredzamo laiku, līdz tas ieradīsies viņiem interesējošajā pieturā.

Aplikācija izmanto līdzīgu tehnoloģiju kā citas šāda veida sistēmas – attiecīgi GPS dati tiek pārsūtīti uz serveri, no kurienes tie tiek attēloti aplikācijā lietotājam saprotamā formā uz kartes.

Tāpat kā Mountain Line Bus Tracker, arī StarTran Bus Tracker pieejams Android un iOS lietotājiem un sistēma pieejama bez maksas.

2.2.3 CTA Train Tracker vilcienu sekošanas sistēma

CTA piedāvā ne tikai autobusu sekošanas sistēmu, bet arī vilcienu sekošanas sistēmu. Šobrīd sistēmai pieejama beta versija, kas nozīmē, ka tās lietotāji var aktīvi piedalīties sistēmas testēšanā un uzlabošanā, nosūtot savu viedokli aptaujas veidā aplikācijas izstrādātājiem.

Līdzīgi kā autobusu sekošanas sistēmas, arī CTA Train Tracker piedāvā sistēmu, kas noteiks aptuveno laiku, kad vilciens ieradīsies lietotājam interesējošā stacijā. Šī iespēja tiek realizēta, pamatojoties uz to, kāds ir bijis iepriekšējais un vidējais vilciena ātrums veicot iepriekšējo noteiktā garuma ceļa posmu. Papildus CTA izveidotajā sistēmā iespējams pievienot lietotāja atsauksmi gadījumā, ja vilciens ir aizkavējies vai atstājis staciju agrāk kā plānots, kā arī meklēt staciju ne tikai pēc nosaukuma, bet arī balstoties uz lietotāja atrašanās vietu, tādējādi atrodot tuvāko iespējamo pieturu konkrētajā maršrutā.

Lietotājiem tiek piedāvāta iespēja pievienot arī stacijas iecienītāko staciju sarakstā, lai vēlāk varētu piekļūt tām ērtāk un ātrāk. CTA piedāvā ar teksta servisu, kas nozīmē, ka iespējams uzzināt iespējamo vilciena pienākšanas laiku stacijā izmantojot SMS, kas atvieglos tos gadījumus, kad nav iespējams interneta savienojums. Iespējama arī vilcienu sekošana tās klasiskajā izpratnē, kad izvēloties kādu konkrētu vilcienu, iespējams sekot tam, kamēr vilciens pārvietojas, tādējādi vienmēr esot drošam par to, ka vilciens netiks nokavēts. Šāda iespēja pieejama tikai kustībā esošiem vilcieniem.

3. SISTĒMAS “BUSSIE” IZSTRĀDE

Sistēma “Bussie” sastāv no MySQL datu bāzes, kas atrodas uz virtuālā LAMP servera, aplikācijas pārvadātājiem, mobilās aplikācijas šoferiem un mobilās aplikācijas lietotājiem. Datu bāze satur piecas tabulas, kas satur datus par šoferiem, maršrutiem, kursēšanas laikiem, reisiem un punktiem, kuros atradies autobuss. Aplikācijā pārvadātājiem iespējama jaunu šoferu, maršrutu un kursēšanas laiku pievienošana. Aplikācija šoferiem pēc šofera autentifikācijas, izmantojot tam piešķirto identifikatoru un paroli, liek izvēlēties maršrutu un kursēšanas laiku. Balstoties uz kursēšanas laiku un šofera identifikatora tiek reģistrēts jauns reiss un ir iespējama jaunu punktu pievienošana. Punkti tiek pievienoti ik 30 sekundes un šoferiem nepieciešams tikai nospiest pogu, kas sāk punktu pievienošanu. Aplikācija lietotājiem pēc maršruta izvēles, līdzīgi kā aplikācija šoferiem, piedāvā izvēlēties kursēšanas laiku no sākuma pieturas. Tad jāizvēlas lietotājam interesējošā diena un, balstoties uz šo informāciju tiek atrasts reiss, kuram tiek nolasīts pēdējais atrastais punkts šofera aplikācijā, kas pēc tam tiek atlikts kartē. Punkts tiek automātiski pārlādēts pēc 30 sekundēm.

3.1 Servera uzstādīšana un datu bāzes izveide

Lai uzstādītu serveri uz izvēlētās bāzes iekārtas tiek instalēta lietojumprogramma Oracle VM VirtualBox 4.3.10. Bāzes iekārta darbojas izmantojot Windows 7 64 bitu operētājsistēmu. Kā virtuālais serveris tika izvēlēts Ubuntu 12.04.4 LTS serveris. VirtualBox vidē tiek izveidota jauna ierīce UbuntuServ. UbuntuServ tiek palaista un pie jautājuma, kā instalēt jauno operētājsistēma, izvēlas CD\DVD-ROM iekārta un izvēlas lejupielādēto servera ISO failu. Pēc izvēles tiek uzstādīti vajadzīgie uzstādījumi, pie instalējamiem serveriem izvēloties LAMP serveri, kas nodrošinās ar nepieciešamo Apache, MySQL un PHP un OpenSSH serveri, lai būtu iespējams pieslēgties serverim arī izmantojot SSH.

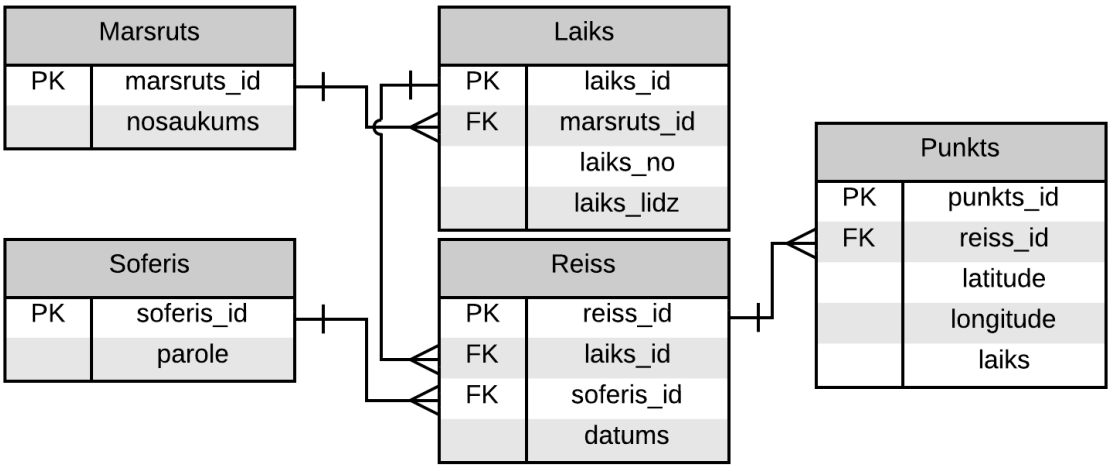
Lai būtu iespējams piekļūt serverim no lokālās ierīces, VirtualBox galvenajā virtuālo mašīnu izvēlē izvēlas UbuntuServ, kam izvēlas “Settings”, jaunajā logā izvēlas no saraksta kreisajā pusē “Network”. Pie “Attached to:” izkrītošajā izvēlnē izvēlas Bridged Adapter, bet pie “Name:” lokālās iekārtas izmantojošā adaptera nosaukumu. Pārstartējam serveri un, lai pārbaudītu, vai servera instalācija bijusi veiksmīga, izmantojot komandu “ifconfig” uzzinām IP adresi, kuru ievadām lokālās mašīnas interneta pārlūkprogrammas adrešu joslā aiz tās pievienojot porta numuru 8888. Ja lapa izvada paziņojumu “It works!”, servera instalācija ir bijusi veiksmīga un pārejam pie tālāko uzstādījumu veikšanas.

Lai pārbaudītu, vai PHP darbojas, izmantojot komandu “cd /var/www” pārvietojamies uz noklusējuma web direktoriju. Ar nano teksta redaktora palīdzību izveidojam jaunu PHP failu “test.php”, kas satur

```
<?php
phpinfo();
?>
```

Ja atkal adrešu joslā ievadot IP adresi, bet aiz tās “/test.php” atveras PHP informācijas lapa, tad serveris ir pareizi konfigurēts, lai uz tā darbotos PHP.

Izmantojot komandu “sudo apt-get install phpmyadmin”, uz servera uzinstalējam PhpMyAdmin. Šī, no interneta pārlūka darbināmā programma, ļauj izveidot datu bāzi ne komandrindas režīmā. Izveidojam datu bāzi Points, kas sastāv no 5 tabulām – Soferis, Marsruts, Laiks, Reiss un Punkts. Tabulu savstarpējās attiecības un lauki redzami 3.1. att.



3.1.att. Datu bāzes “Points” ERD diagramma.

3.2 Aplikācijas pārvadātājiem “Bussie” izveide

Aplikācija “Bussie” pārvadātājiem ir programmēšanas valodā JAVA rakstīta darbvirsmas lietojumprogramma, kas veidota programmēšanas vidē IntelliJ IDEA 12.1.6, kas paredzēta sabiedriskā transporta uzņēmumiem, lai tiem būtu iespējams pievienot jaunus maršrutus, jaunus šoferus un jaunus kursēšanas laikus. Lietojumprogramma sastāv no piecām klasēm, no kurām četras, “ParvadatajiemGUI”, “JaunsSoferisGUI”, “JaunsMarsrutsGUI”, “JaunsLaiksGUI”, ir paredzētas grafiskajam lietotāja interfeisam, bet atlikusī klase “Main” aplikācijas palaišanai.

Klase “Main” sastāv no tukša konstruktora un vienas metodes “public static void main(String[] args)”. Metode tiek automātiski izsaukta palaižot aplikāciju. Vispirms, izsaucot metodi, tiek ielādēts draiveris, kas palīdz veidot savienojumu ar datu bāzi.

```
try {
     System.out.println("Loading driver...");
     Class.forName("com.mysql.jdbc.Driver");
     System.out.println("Driver loaded!");
} catch (ClassNotFoundException e) {
     throw new RuntimeException("Cannot find the driver in
the classpath!", e);
}
```

Veiksmīgas draivera ielādes gadījumā, konsolē tiek izvadīts paziņojums “Driver loaded”. Ja konsolē tiek izvadīts paziņojums “Cannot find the driver in the classpath!”, draiveris jāpievieno projektam. IntelliJ IDEA to iespējams izdarīt augšējā izvēlnē izvēloties File, tad “Project Structure…”. Jaunatvērtajā logā pie kreisajā pusē pie “Modules” izvēlas moduli, kam nepieciešams pievienot savienotāju, uzspiež uz plus ikonas un izvēlas “1 Jars or directories…”, sistēmā atrod vajadzīgo JAR failu un apstiprina nospiežot “OK”, pēc tam vēlreiz apstiprina modulī veiktās izmaiņas, aizverot logu ar “OK”. Lai izveidotu savienojumu ar datu bāzi izveido klases Connection objektu, definē, izmantojot String, savienojuma adresi. Izmantojot klases DriverManager statisko metodi “getConnection”, izveido savienojumu ar datu bāzi. Veiksmīga savienojuma gadījumā konsolē izvada paziņojumu “Connected to database”.

```
Connection conn = null;
try {
     String url = "jdbc:mysql://"+IP+":3306/Points";
     conn = DriverManager.getConnection(url, user, pass);
     System.out.println("Connected to database");
} catch (SQLException e) {
     e.printStackTrace();
}
```

Pēc veiksmīga savienojuma izveides ar datu bāzi atveras klases “ParvadatajiemGUI” logs un tā konstruktoram, kā arguments tiek padots izveidotais savienojums.

```
ParvadatajiemGUI pGUI = new ParvadatajiemGUI(conn);
pGUI.setVisible(true);
```

“ParvadatajiemGUI” klase paredzēta, kā sākuma logs, no kura iespējams pievienot jaunus maršrutus,šoferus un kursēšanas laikus. Klase manto “JFrame” klasi un satur vienu “JPanel” objektu, kas satur trīs “JButton” klases objektus. “ParvadatajiemGUI” klase satur konstruktoru, kam kā arguments tiek padots “Connection” klases objekts, kas izveidots Main klasē, un implementēto metodi, kas nosaka, kāda poga ir nospiesta.

Konstruktors nosaka loga izmērus, tā izkārtojumu un tiek noteikts, kā rīkoties, ja logs tiek aizvērts.

```
public ParvadatajiemGUI(Connection conn) {
     this.setLayout(null);
     this.setBounds(50, 50, 400, 200);
     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Vispirms tiek definēti paneļa izmēri un tā izkārtojums. Panelis tiek pievienots logam.

```
panel1 = new JPanel(null);
panel1.setBounds(0, 30, 400, 120);
this.add(panel1);
```

Pēc tam tiek definētas trīs pogas, to izmēri un uzraksti uz tām. Visām pogām tiek pievienots "klausītājs", kas pēc tam ļauj pievienot pogai notikumus, kas notiks pēc tās nospiešanas, un tās tiek pievienotas iepriekš izveidotajam panelim.

```
jaunsSoferis = new JButton("PIEVIENOT JAUNU ŠOFERI");
jaunsSoferis.setBounds(10, 10, 360, 30);
jaunsSoferis.addActionListener(this);
panel1.add(jaunsSoferis);

jaunsMarsruts = new JButton("PIEVIENOT JAUNU MARŠRUTU");
jaunsMarsruts.setBounds(10, 50, 360, 30);
jaunsMarsruts.addActionListener(this);
panel1.add(jaunsMarsruts);

jaunsLaiks = new JButton("PIEVIENOT JAUNU LAIKU MARŠRUTAM");
jaunsLaiks.setBounds(10, 90, 360, 30);
jaunsLaiks.addActionListener(this);
panel1.add(jaunsLaiks);
```

Pēc pogu izveidošanas "Connection" klases objektam, kas definēts pie "ParvadatajiemGUI" mainīgajiem tiek piešķirta vērtība, ka tas ir vienāds ar konstruktoram padotā argumenta vērtību.

Implementētā interfeisa "ActionListener" metodē "actionPerformed" tiek izveidots "Object" klases objekts un pēc tā nosaka, kura no pogām ir nospiesta.

```
@Override
public void actionPerformed(ActionEvent actionEvent) {
     Object o = actionEvent.getSource();
```

Ja nospiestā poga ir "jaunsSoferis", tiek izveidots klases "JaunsSoferisGUI" objekts un atvērts logs. Konstruktoram kā arguments tiek padots "Connection" klases objekts.

```
if (o.equals(jaunsSoferis)) {
    JaunsSoferisGUI jaunsSofGUI = new JaunsSoferisGUI(conn);
    jaunsSofGUI.setVisible(true);
}
```

Ja nospiesta poga “jaunsMarsruts”, tiek izveidots klases “JaunsMarsrutsGUI” objekts un atvērts logs. Konstruktoram kā arguments tiek padots “Connection” klases objekts.

```
if (o.equals(jaunsMarsruts)) {
     JaunsMarsrutsGUI jaunsMarsGUI = new
     JaunsMarsrutsGUI(conn);
     jaunsMarsGUI.setVisible(true);
}
```

Ja nospiesta poga “jaunsLaiks”, tiek izveidots klases “JaunsLaiksGUI” objekts un atvērts logs. Konstruktoram kā arguments tiek padots “Connection” klases objekts

```
if(o.equals(jaunsLaiks)){
     JaunsLaiksGUI jaunsLaiksGUI= new JaunsLaiksGUI(conn);
     jaunsLaiksGUI.setVisible(true);
}
```

“JaunsSoferisGUI” klase paredzēta, lai būtu iespējams pievienot jaunu šoferi, izvēloties tam paroli. Identifikators tiek automātiski ģenerēts no datu bāzes. Tā vērtība ir vesels skaitlis (1,2,3…). Klase satur paneli uz kura izveidots ar “JLabel” palīdzību veidots virsraksts un teksts pie paroles lauka un “JTextField” klases objekts paroles ierakstīšanai. Bez paneļa klasē vēl ir divas “JButton” komponentes apstiprināšanai un, lai atgrieztos uz sākumu.

Līdzīgi kā “ParvadatajiemGUI” klasē, arī šīs klases konstruktoram, kā arguments tiek padots “Connection” klases objekts. Izsaucot “JaunsSoferisGUI” objektu, vispirms konstruktorā tiek noteikti loga izmēri, “Connection” klases objekta vērtība tiek pielīdzināta konstruktoram argumentā padotā objekta vērtībai un izveidots panelis, kam arī tiek noteikti izmēri.

```
public JaunsSoferisGUI(Connection conn) {
        this.setLayout(null);
        this.setBounds(50, 50, 400, 200);
        this.conn = conn;

        mainPanel = new JPanel(null);
        mainPanel.setBounds(0, 0, 380, 125);
        this.add(mainPanel);
```

Panelim tiek pievienoti divi “JLabel” klases objekti, kam tiek definēti to izmēri un virsrakstu vērtības.

```
lbl1 = new JLabel("JAUNA ŠOFERA PIEVIENOŠANAS FORMA");
lbl1.setBounds(5, 5, 370, 30);
mainPanel.add(lbl1);

lbl2 = new JLabel("Ievadiet jaunā šofera paroli");
lbl2.setBounds(5, 40, 370, 30);
mainPanel.add(lbl2);
```

Bez virsrakstiem panelim pievieno arī "JTextField" klases objektu, kam tiek definēti izmēri. Zem paneļa tiek izveidotas divas pogas. Viena no tām paredzēta apstiprināšanai, bet otra, lai atgrieztos sākumlapā. Abām pogām tiek pievienoti klausītāji, kas nosaka, vai poga ir nospiesta.

```
pass = new JTextField();
pass.setBounds(5, 75, 200, 30);
mainPanel.add(pass);

okBtn = new JButton("PIEVIENOT JAUNU ŠOFERI");
okBtn.setBounds(5, 125, 180, 30);
okBtn.addActionListener(this);
this.add(okBtn);

backBtn = new JButton("ATPAKAĻ UZ SĀKUMU");
backBtn.setBounds(195, 125, 180, 30);
backBtn.addActionListener(this);
this.add(backBtn);
```

Implementētā interfeisa "ActionListener" metodē "actionPerformed" tiek izveidots "Object" klases objekts un pēc tā nosaka, kura no pogām ir nospiesta.

```
@Override
public void actionPerformed(ActionEvent actionEvent) {
     Object o = actionEvent.getSource();
```

Ja nospiesta poga "okBtn", tiek izveidots jauns ieraksts datu bāzē un par to tiek paziņots izvadot attiecīgu paziņojumu, bet, ja tiek nospiesta poga "backBtn", aizveras jauna šofera pievienošanas logs.

```
@Override
public void actionPerformed(ActionEvent actionEvent) {
     Object o = actionEvent.getSource();
     if (o.equals(okBtn)) {
          this.setVisible(false);
          try {
               String sql = "INSERT INTO Soferis (parole)
               VALUES ('" + pass.getText() + "')";
               statement = conn.createStatement();
               statement.executeUpdate(sql);
               JOptionPane.showMessageDialog(this,
               "Pievienots ieraksts");
          } catch (SQLException e) {
                e.printStackTrace();
          }
     }
     if (o.equals(backBtn)) {
          this.setVisible(false);
     }
}
```

"JaunsMarsrutsGUI" klasē lietotājs var pievienot jaunu maršrutu ievadot tā sākuma un gala pieturu. Logs sastāv no paneļa un divām pogām. Uz paneļa atrodas loga virsraksts un JLabel objekti blakus diviem JTextField objektiem. JTextField objekti paredzēti, lai ievadītu sākuma un gala pieturu. Klase sastāv no konstruktora, kuram kā arguments tiek padots klases "Connection" objekts un implementētā interfeisa "ActionListener" metodes "actionPerformed".

Izveidojot jaunu klases objektu vispirms konstruktorā tiek definēts loga izmērs un tā izkārtojums, klases "Connection" objekta vērtība tiek pielīdzināta konstruktoram padotā argumenta vērtībai un tiek izveidots panelis, kam tiek noteikts izkārtojums un definēti izmēri.

```
public JaunsMarsrutsGUI(Connection conn) {
        this.conn = conn;
        this.setLayout(null);
        this.setBounds(50, 50, 400, 200);

        panel = new JPanel(null);
        panel.setBounds(0, 0, this.getWidth() - 10, 3 *
        this.getHeight() / 5);
        this.add(panel);
```

Pēc tam tiek izveidoti trīs "JLabel" klases objekti piešķirot teksta vērtībā virsrakstus, objektiem tiek definēti izmēri un tie tiek pievienoti panelim.

```
virsraksts = new JLabel("JAUNA MARŠRUTA PIEVIENOŠANA");
l1 = new JLabel("Vieta, no kurienes atiet maršruts:");
l2 = new JLabel("Vieta, līdz kurienei ir maršruts:");

virsraksts.setBounds(0, 0, 2 * panel.getWidth() / 3,
panel.getHeight() / 5);
l1.setBounds(0, panel.getHeight() / 5, 2 *
panel.getWidth() / 3, panel.getHeight() / 5);
l2.setBounds(0, 2 * panel.getHeight() / 5, 2 *
panel.getWidth() / 3, panel.getHeight() / 5);

panel.add(virsraksts);
panel.add(l1);
panel.add(l2);
```

Panelim vēl tiek pievienotas divas klases "JTextField" komponentes un tām definēti izmēri.

```
no = new JTextField();
lidz = new JTextField();
no.setBounds((2 * panel.getWidth() / 3) - 15,
panel.getHeight() / 5, panel.getWidth() / 3,
panel.getHeight() / 5);
lidz.setBounds((2 * panel.getWidth() / 3) - 15, 2 *
panel.getHeight() / 5, panel.getWidth() / 3,
panel.getHeight() / 5);
```

```
panel.add(no);
panel.add(lidz);
```

Logam tiek pievienotas divas pogas, ar kurām iespējams pievienot jaunu maršrutu un atgriezties sākumlapā. Pogām tiek definēti izmēri, teksti uz pogām un pievienoti klausītāji, kas nosaka, kā rīkoties pogas nospiešanas gadījumā.

```
ok = new JButton("JAUNS MARŠRUTS");
back = new JButton("ATPAKAĻ UZ SĀKUMU");

ok.setBounds(0, (3 * this.getHeight() / 5) + 5,
this.getWidth() / 2 - 3, this.getHeight() / 5 - 5);
back.setBounds(this.getWidth() / 2 - 3, (3 *
this.getHeight() / 5) + 5, this.getWidth() / 2 - 18,
this.getHeight() / 5 - 5);

ok.addActionListener(this);
back.addActionListener(this);

this.add(ok);
this.add(back);
```

Implementētajā metodē tiek izveidots klases "Object" objekts, kas nosaka, kura no pogām tiek piespiesta. Ja piespiestā poga ir "ok", tiek pievienots jauns ieraksts datu bāzē un izvadīts attiecīgs paziņojums, bet, ja nospiestā poga ir "back", notiek atgriešanās uz sākumlapu.

```
@Override
public void actionPerformed(ActionEvent actionEvent) {
     Object o = actionEvent.getSource();
     if (o.equals(ok)) {
          this.setVisible(false);
          try {
          String sql = "INSERT INTO Marsruts (nosaukums)
          VALUES ('" + no.getText() + " - " + lidz.getText()
          +"')";
          statement = conn.createStatement();
          statement.executeUpdate(sql);
          JOptionPane.showMessageDialog(this,
          "Pievienots ieraksts");
          } catch (SQLException e) {
               e.printStackTrace();
          }
     }
     if (o.equals(back)) {
          this.setVisible(false);
     }
}
```

Klase "JaunsLaiksGUI" paredzēta tam, lai lietotājs varētu pievienot jaunu kursēšanas laiku maršrutam. Logs sastāv no paneļa un divām pogām. Uz paneļa izvietoti četras JLabel komponentes, divi teksta lauki, lai varētu ievietot sākuma laiku un beigu laiku un JComboBox komponente, kurā iespējams izvēlēties maršruta nosaukumu. Klase sastāv no konstruktora, kuram kā arguments tiek padots "Connection" klases objekts un implementētas metodes "actionPerformed" no "ActionListener" interfeisa.

Izveidojot "JaunsLaiksGUI" klases objektu, tiek izsaukts konstruktors, kas nosaka loga izmērus, tā izkārtojumu un izveido paneļa objektu, kam arī definē izmērus un nosaka izkārtojumu. "Connection" klases objektam tiek piešķirta vērtība, kas tiek padota klases konstruktoram kā arguments.

```
public JaunsLaiksGUI(Connection conn) throws
HeadlessException {
        this.conn = conn;
        this.setLayout(null);
        this.setBounds(50, 50, 400, 200);

        panel = new JPanel(null);
        panel.setBounds(0, 0,400, 120);
        this.add(panel);
```

Pēc tam panelim tiek pievienoti divi "JLabel" klases objekti un divi "JTextField" klases objekti. Visām komponentēm tiek definēti izmēri.

```
l3 = new JLabel("Laiks, cikos atiet autobuss: (hh:mm:ss) ");
l4 = new JLabel("Laiks, cikos pienāk autobuss: (hh:mm:ss) ");

l3.setBounds(0, 3 * panel.getHeight() / 5, 2 *
panel.getWidth() / 3, panel.getHeight() / 5);
l4.setBounds(0, 4 * panel.getHeight() / 5, 2 *
panel.getWidth() / 3, panel.getHeight() / 5);

panel.add(l3);
panel.add(l4);

cikos = new JTextField();
lidzCikiem = new JTextField();

cikos.setBounds((2 * panel.getWidth() / 3) - 15, 3 *
panel.getHeight() / 5, panel.getWidth() / 3,
panel.getHeight() / 5);
lidzCikiem.setBounds((2 * panel.getWidth() / 3) - 15, 4 *
panel.getHeight() / 5, panel.getWidth() / 3,
panel.getHeight() / 5);

panel.add(cikos);
panel.add(lidzCikiem);
```

Pēc teksta lauku un virsrakstu izveides tiek izveidotas divas pogas, kas tiek pievienotas logam. Pogām tiek definēti izmēri un uzraksti virs tām, kā arī tiek pievienoti klausītāji, kas nosaka, kas notiek pēc pogas nospiešanas.

```
okBtn = new JButton("PIEVIENOT LAIKU");
okBtn.setBounds(5, 125, 180, 30);
okBtn.addActionListener(this);
this.add(okBtn);

backBtn = new JButton("ATPAKAĻ UZ SĀKUMU");
backBtn.setBounds(195, 125, 180, 30);
backBtn.addActionListener(this);
this.add(backBtn);
```

Lai izveidotu "JComboBox" klases objektu, kurā automātiski tiktu pievienoti visi datu bāzē esošie maršruti, maršrutu nosaukumus un identifikatorus ielasa no datu bāzes un ievieto divos "ArrayList<String>" tipa sarakstos. Pēc tam šo sarakstu ar maršrutu nosaukumiem padod kā argumentu, veidojot "JComboBox" klases objektu, definē komponentes izmērus un pievieno to panelim.

```
Statement stmt = null;
try {
     stmt = conn.createStatement();
     ResultSet rs = stmt.executeQuery("SELECT * FROM
     Marsruts;");
     while(rs.next()){
          nosaukumi.add(rs.getObject(2).toString());
          ids.add(rs.getObject(1).toString());
     }
} catch (SQLException e) {
     e.printStackTrace();
}
comboBox = new JComboBox(nosaukumi);
comboBox.setBounds(panel.getWidth() / 3, 2 *
panel.getHeight() / 5-10, 2 * panel.getWidth() / 3-20,
panel.getHeight() / 5);
panel.add(comboBox);
```

Nobeigumā tiek izveidoti divi atlikušie "JLabel" klases objekti, uzstādītas to vērtības un izmēri un pievienoti panelim.

```
virsraksts = new JLabel("JAUNA LAIKA PIEVIENOŠANA");
virsraksts.setBounds(0, panel.getHeight() / 5-10, 2 *
panel.getWidth() / 3, panel.getHeight() / 5);
panel.add(virsraksts);

virsraksts2 = new JLabel("Izvēlies maršrutu ");
virsraksts2.setBounds(0,  2*panel.getHeight()  /  5-10,
panel.getWidth() / 3, panel.getHeight() / 5);
panel.add(virsraksts2);
```

Implementētā metode “actionPerformed” satur “Object” klases objektu, kas nosaka, kura no pogām ir piespiesta. Ja piespiestā poga ir “ok”, tiek pievienots jauns ieraksts datu bāzē un izvadīts attiecīgs paziņojums, bet, ja nospiestā poga ir “back”, notiek atgriešanās uz sākumlapu.

```
@Override
public void actionPerformed(ActionEvent actionEvent) {
     Object o = actionEvent.getSource();
     if(o.equals(okBtn)){
          this.setVisible(false);
          String id = "";
          id = ids.get(comboBox.getSelectedIndex());
          try {
               Statement st = conn.createStatement();
               st.executeUpdate("INSERT INTO Laiks VALUES
               (NULL,'"+id+"','"+cikos.getText()+"','"+lid
               zCikiem.getText()+"')");
          } catch (SQLException e) {
               e.printStackTrace();
          }
     }
     if(o.equals(backBtn)){
          this.setVisible(false);
     }
}
```

3.3 Mobilās aplikācijas šoferiem “Bussie” izveide

Mobilā aplikācija “Bussie” šoferiem ir Android viedtālruņiem paredzēta mobilā aplikācija, kas veidota programmēšanas vidē IntelliJ IDEA 12.1.6. Aplikācija veidota, lai būtu iespējams uz datu bāzi nosūtīt autobusa atrašanās vietu ik 30 sekundes. Aplikācija satur piecus layout failus, kas rakstīti izmantojot XML programmēšanas valodu, trīs aktivitāšu failus, kas rakstīti JAVA programmēšanas valodā, piecus failus, kas nodrošina komunikāciju ar datu bāzi un vienu failu, kas izgūst lokācijas datus. Lai no datu bāzes izgūtu datus, uz servera atrodas astoņi PHP programmēšanas valodā rakstīti faili.

Lai varētu no ierīces nolasīt atrašanās vietu un izmantot internetu, “AndroidManifest.xml” failā tiek pievienotas sekojošas rindiņas:

```
<uses-permission
android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

Aplikācijas galvenā aktivitāte un tā, ar ko sākas visa darbība, ir aprakstīta "LogInActivity" klasē. Lai aplikācija darbotos, "AndroidManifest.xml" failā nepieciešams pievienot sekojošas rindiņas:

```
<activity android:name=".LogInActivity"
          android:label="@string/app_name">
     <intent-filter>
     <action android:name="android.intent.action.MAIN"/>
     <category
     android:name="android.intent.category.LAUNCHER"/>
     </intent-filter>
</activity>
```

Šīs rindiņas jāiekļauj starp "<application>" un "</application>" atslēgvārdiem bet iepriekšminētās atļaujas aiz aizverošā "</application>". "LogInActivity" klase manto "Activity" klasi un satur publisku statisku String tipa mainīgo "fullLink", kurā norādīta PHP failu atrašanās vieta uz servera un servera IP adrese. Šis mainīgais ir pieejams visām klasēm. Vēl arī šajā klasē atrodas publisks statisks ArrayList<String> tipa saraksts "parametri", kas sevī aplikācijas dzīves cikla laikā dažādos brīžos uzkrāj vajadzīgos parametrus, lai izveidotu datu bāzē jaunu reisa ierakstu. Tāpat katra aktivitāte satur arī "Context" klases objektu. "LogInActivity" klase satur onCreate un onStart metodes, kas tiek mantotas no superklases un startMarsruti metodi, kas izpildās gadījumā, ja tiek nospiesta poga.

OnCreate metode tiek izsaukta automātiski brīdī, kad aktivitāte tiek izveidota. Vispirms tiek izsaukta superklases onCreate metode, kam kā arguments tiek nodots klases "Bundle" objekts, tad tiek uzstādīts skats no XML failiem. Tajā ir izkārtoti un izveidoti redzamie objekti. Pēc tam "Context" klases mainīgajam tiek piešķirta vērtība.

```
@Override
public void onCreate(Bundle savedInstanceState) {
     super.onCreate(savedInstanceState);
     setContentView(R.layout.main);
     c = getBaseContext();
```

Lai būtu iespējams veikt darbības ar objektiem no skata, to vērtības ir jāpiešķir. Šīs darbības arī tiek veiktas onCreate metodē.

```
     id = (EditText) findViewById(R.id.editText2);
     parole = (EditText) findViewById(R.id.editText);
     btn = (Button) findViewById(R.id.button);
}
```

OnStart metodē tiek izsaukts tikai superklases onStart metode.

```
@Override
public void onStart(){
     super.onStart();
}
```

Nospiežot pogu "btn", tiek izsaukta metode "startMarsruti". Metode pārbauda, vai šofera identifikatora lauciņš nav tukšs. Ja lauciņš ir tukšs, tiek izvadīts attiecīgs paziņojums, ja nē, parametru sarakstam tiek pievienots pirmais parametrs – šofera identifikators un tiek izveidots jauns klases "readIdsActivity" objekts, kuram tiek izsaukta mantotā metode "execute", kam kā arguments tiek padots šofera identifikators. Pēc metodes izpildes, tiek izsaukta "onPostExecute" metode, kam kā arguments tiek padots metodes "execute" atgrieztais rezultāts. "OnPostExecute" metode ir šāda:

```
@Override
protected void onPostExecute(String result){
if(result.equals(parole.getText().toString())){
     Intent i = new Intent(c,ReadMarsrutiActivity.class);
     startActivity(i);
}
else{
     if(parole.getText().equals(""))
          Toast.makeText(c,"Ievadi
          paroli",Toast.LENGTH_LONG).show();
     else
          Toast.makeText(c, "Pazudis interneta
          savienojums vai nepareizs id/parole",
          Toast.LENGTH_LONG).show();
}
}
```

"ReadIdsActivity" klase manto "AsyncTask<String,Void,String>" klasi un tās metodi "doInBackground". Šī klase satur konstruktoru un mantoto metodi "doInBackground". "ReadIdsActivity" klase nodrošina saziņu starp aplikāciju un PHP failiem, kas izvietoti uz servera. Šajā gadījumā, kad metodei "execute" klasē "LogInActivity" tiek padots viens arguments – šofera identifikators, šī klase "doInBackground" metodē nodrošina to, ka tā rezultātā atgriezīs paroli atkarībā no izvēlētā identifikatora, lai pēc tam to varētu salīdzināt ar ievadīto. Vispirms tiek nolasīts padotais arguments un tas tiek ar GET metodes palīdzību nodots PHP failam.

```
@Override
protected String doInBackground(String... args) {
     String id=args[0];
     String link = LogInActivity.fullLink
     +"getSoferiIDS.php?soferis_id="+id;
```

"GetSoferiIDS.php" fails satur:

```
<?php
include 'connect.php';
$id=$_GET['soferis_id'];
$result = mysqli_query($connection,
"SELECT parole FROM Soferis WHERE soferis_id='$id'");
while($row = mysqli_fetch_array($result)){
        echo $row['parole'];
}
mysqli_close($connection);
?>
```

"Connect.php" fails tiek izsaukts, jo ar tā palīdzību tiek izveidots savienojums ar datu bāzi. Tālākais "readIdsActivity" klases "doInBackground" metodē ir vienāds ar visām pārējām klases metodēm, kas veic saziņu starp aplikāciju un PHP failu.

```
try {
     URL url = new URL(link);
     HttpClient client = new DefaultHttpClient();
     HttpGet request = new HttpGet();
     request.setURI(new URI(link));
     HttpResponse response = client.execute(request);
     BufferedReader in = new BufferedReader(new
InputStreamReader(response.getEntity().getContent()));
     while((line = in.readLine())!=null){
          buffer.append(line);
          break;
     }
     in.close();
     return buffer.toString();
} catch (Exception e) {
     return new String ("Exception:"+e.getMessage());
}
```

Pēc veiksmīgas šofera autentifikācijas atveras jauns logs un sākas jauna aktivitāte – "ReadMarsrutiActivity". Šī klase manto "Activity" klase un satur mantoto "onCreate" metodi, kā arī divas metodes, kas tiek izsauktas nospiežot pogu un vēl vienu metodi. Klase satur arī četrus "ArrayList<String>" tipa sarakstus, lai uzglabātu maršrutu nosaukumus un identifikatorus un kursēšanas laiku nosaukumus un identifikatorus.

Izveidojot klases objektu, tiek izsaukta "onCreate" metode. Izsaucot metodi, vispirms tiek izsaukta superklases onCreate metode, kam kā arguments tiek nodots klases "Bundle" objekts, tad tiek uzstādīts skats no XML failiem. Pēc tam "Context" klases mainīgajam tiek piešķirta vērtība.

```
@Override
public void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.marsruta_izvele);
    c = getBaseContext();
```

Šis skats sevī ietver arī divas pogas un divas izkrītošās izvēlnes, kas arī tiek inicializētas "onCreate" metodes izsaukšanas brīdī.

```
spin2 = (Spinner) findViewById(R.id.spinnerMarsruts);
laiksSpin = (Spinner) findViewById(R.id.laiksSpin);
btn2 = (Button) findViewById(R.id.buttonJaunsReiss);
btnLaiks = (Button) findViewById(R.id.buttonLaiks);
```

Izveidojot jaunu "readMarsruti" klases objektu, tiek izsaukta atkal klases "execute" metode, kam kā arguments tiek padots tukšs "String" tipa objekts.

```
new readMarsruti(){
    @Override
    protected void onPostExecute(String result){
        ReadMarsrutiActivity.this.readMarsruti(result);
    }
}.execute("");
```

Klase "readMarsruti" ir līdzīga "readIdsActivity" klasei. Arī šī klase manto "AsyncTask<String,Void,String>" klasi un tās metodi "doInBackground". Tāpēc minēsim tās vienīgo atšķirību – citu PHP failu, kas aplikāciju saista ar datu bāzi un ar kā palīdzību tiek nolasīti maršruti no datu bāzes.

```
String link = LogInActivity.fullLink+"getMarsruti.php";
```

"GetMarsruti.php" fails satur:

```
<?php
include 'connect.php';
$result = mysqli_query($connection,
"SELECT marsruts_id, nosaukums FROM Marsruts");
while($row = mysqli_fetch_array($result)){
echo $row['marsruts_id']."!!!".$row['nosaukums']."<br>";
}
mysqli_close($connection);
?>
```

Pēc "doInBackground" metodes izpildes, tiek izsaukta klases "ReadMarsrutiActivity" metode "readMarsruti", kam kā arguments tiek padots "doInBackground" atgrieztais String objekts. Objektu vispirms sadala pēc "
" un tad katru String tipa masīvu sadala pēc "!!!" un pievieno identifikatorus vienam sarakstam, bet nosaukumus otram.

```
public void readMarsruti(String result) {
    String[] marsruti = result.split("<br>");
    for(String s:marsruti){
        String[] idAndName = s.split("!!!");
        nosaukumi.add(idAndName[1]);
        marsruts_ids.add(idAndName[0]);
    }
```

Izmantojot XML failu, kas nosaka, kā lejupkrītošajā izvēlnē izskatīsies izvēle, un pievienojot maršrutu nosaukumu sarakstu, izveido "ArrayAdapter<String>" klases objektu. Izveidoto objektu uzstāda iepriekš inicializētajai "spin2" vērtībai.

```
ArrayAdapter<String> adapter = new ArrayAdapter<String>
(this,R.layout.simple_spinner_item, nosaukumi);
adapter.setDropDownViewResource(R.layout.simple_spinner_dr
opdown_item);
spin2.setAdapter(adapter);
}
```

Nospiežot pogu "btnLaiks", tiek izsaukta metode "addLaiks". Katru reizi nospiežot šo pogu, tiek izveidoti divi jauni "ArrayList<String>" tipa saraksti kursēšanas laiku un to identifikatoru uzkrāšanai. Pēc tam tiek izveidots jauns "readLaiki" klases objekts, kam mantotajai "execute" metodei kā arguments tiek padots izvēlētā maršruta identifikators. Klase "readLaiki" līdzības ziņā neatšķiras no "readMarsruti" un "readIdsActivity" klasēm. Vienīgā atšķirība arī šeit ir padotajā argumentā metodei "doInBackground" un PHP faila nosaukumā.

```
@Override
protected String doInBackground(String... args) {
     String id = args[0];
     String link = LogInActivity.fullLink+
     "getLaiki.php?marsruts_id="+id;
```

"GetLaiki.php" fails satur:

```
<?php
include 'connect.php';
$id=$_GET['marsruts_id'];
$result = mysqli_query($connection,"SELECT laiks_id,
laiks_no FROM Laiks WHERE  marsruts_id='$id'");
while($row = mysqli_fetch_array($result)){
echo $row['laiks_id']."!!!".$row['laiks_no']."<br>";
}
mysqli_close($connection);
?>
```

Pēc šīs metodes izpildes tiek izsaukta "onPostExecute" metode, kurai kā arguments tiek padots "doInBackground" atgrieztais String tipa objekts. Ar lejupkrītošo izvēlni rīkojas līdzīgi kā gadījumā, kad tika ielasīti maršruti.

```
protected void onPostExecute(String result){
    String[] laiki = result.split("<br>");
    for(String s:laiki){
        String[] idAndLaiks = s.split("!!!");
        laiki2.add(idAndLaiks[1]);
        laikiIds.add(idAndLaiks[0]);
    }
```

```
    ArrayAdapter<String> arrayAdapter = new ArrayAdapter
     <String>(c,R.layout.simple_spinner_item,laiki2);
     arrayAdapter.setDropDownViewResource(R.layout.simple_
     spinner_dropdown_item);
    laiksSpin.setAdapter(arrayAdapter);
    laiksSpin.setVisibility(View.VISIBLE);
}
```

Nospiežot pogu "btn2", parametru sarakstā tiek ievietota otrā vērtība – izvēlētā laika identifikators. Pēc tam tiek izsaukts klases "insertReiss" objekts, kura mantotajai metodei "execute" kā argumenti tiek padotas abas parametru sarakstā esošās vērtības. Arī "insertReiss" klase no iepriekšminētajām klasēm, kas saista aplikāciju ar PHP failu, atšķiras tikai ar saņemtajiem argumentiem un PHP faila nosaukumu.

```
@Override
protected String doInBackground(String... args) {
   String laiks = args[0];
   String soferis = args[1];
   String link = LogInActivity.fullLink+
   "insertReiss.php?laiks_id="+laiks+
   "&soferis_id="+soferis;
```

"InsertReiss.php" fails satur:

```
<?php
include 'connect.php';
$laiks_id=$_GET['laiks_id'];
$soferis_id=$_GET['soferis_id'];
$result = mysqli_query($connection,"INSERT INTO Reiss
(laiks_id, soferis_id, datums) VALUES
('$laiks_id','$soferis_id',CURDATE())");
$id = mysqli_insert_id($connection);
if(!$result){
        die('Error:'.mysql_error());
}
echo $id;
mysqli_close($connection);
?>
```

Pēc veiksmīgas reisa pievienošanas tiek izsaukta "onPostExecute" metode, kas parametru sarakstā pievieno trešo vērtību – reisa identifikatoru. Pēc tam ar "Intent" klases objekta palīdzību tiek izsaukta klases "AddPointsActivity" mantotā metode onCreate. Arī "AddPointsActivity" manto "Activity" klasi. Vēl šī klase satur mantoto "onStart" metodi, divas metodes, kas tiek izsauktas, nospiežot pogu, un interfeisa "Runnable" objektu. Izsaucot "onCreate" metodi vispirms tiek izsaukta superklases "onCreate" metode, kam kā arguments tiek nodots klases "Bundle" objekts, tad tiek uzstādīts skats no XML failiem. Pēc tam "Context" klases mainīgajam tiek piešķirta vērtība. "OnCreate" metodē tiek inicializēti arī trīs teksta lauki, kas paredzēti, lai izvadītu šofera, kursēšanas laika un reisa identifikatorus,

divas pogas, lai sāktu un apturētu atrašanās punktu nosūtīšanu uz serveri, un "Handler" un "GPSTracker" klašu objekti.

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.add_points);
    c = getBaseContext();

    s = (TextView) findViewById(R.id.soferaid);
    l = (TextView) findViewById(R.id.marsrutaid);
    r = (TextView) findViewById(R.id.reissid);

    b = (Button) findViewById(R.id.buttonStartPoints);
    b2 = (Button) findViewById(R.id.stopButton);

    handler = new Handler();
    gps = new GPSTracker(c);
}
```

Lai būtu iespējams nolasīt ierīces atrašanās vietu, pie moduļiem ir jāpievieno "google-play-services_lib" modulis un starp moduļiem jāuzstāda atkarības. Pēc tam, līdzīgi kā pievienojot pārvadātāja aplikācijai savienotāju ar datu bāzi, pievieno JAR failu. "GPSTracker" klase satur metodes, kas veic visu nepieciešamo, lai nolasītu ierīces atrašanās vietu. Metode "getLocation" atgriež ierīces atrašanās vietu, metode "stopUsingGPS" pārstāj izgūt ierīces atrašanās vietu, metodes "getLatitude" un "getLongitude" atgriež ierīces atrašanās vietas garuma un platuma vērtības, metode "canGetLocation" pārbauda, vai ierīce ļauj piekļūt savai atrašanās vietai un metode "showSettingsAlert" izvada dialoga logu, kas piedāvā ieslēgt iespēju, lai būtu iespējams nolasīt atrašanās vietu.

Pēc metodes "onCreate" izpildes automātiski tiek izsaukta metode "onStart",kas vispirms izsauc superklases metodi "onStart" un tad piešķir vērtības trīs teksta laukiem, kas tika inicializēti "onCreate" metodē.

Nospiežot pogu "b", tiek izsaukta metode "startPoints", kas pārbauda, vai iespējams iegūt atrašanās vietu. Ja tas ir iespējams "Handler" klases objektam tiek izsaukta metode "onPostDelayed", kam kā arguments tiek padots "Runnable" objekts.

```
private Runnable updateTimerThread = new Runnable(){
    @Override
    public void run() {
        gps = new GPSTracker(c);
        lat = ""+gps.getLatitude();
        lon = ""+gps.getLongitude();
```

Iegūtās atrašanās vietas vērtības, izmantojot klases "insertPunkts" objektu, tiek padotas metodei "execute" kā argumenti kopā ar parametros esošo reisa identifikatora vērtību. Tas

notiek reizi 30 sekundēs, kamēr vien nav nospiesta poga “b2”, kas izsauc metodi “stopPoints”.

```
public void stopPoints(View view){
    cancel=true;
    gps.stopUsingGPS();
}
```

“Run” metodes turpinājums:

```
        new insertPunkts(){
            @Override
            protected void onPostExecute(String result){
            }
        }.execute(LogInActivity.parametri.get(2),lat,lon);
        if(!cancel)
            handler.postDelayed(this,30*1000);
    }
};
```

“InsertPunkts” klase ir līdzīga visām iepriekšminētajām klasēm, kas savieno aplikāciju ar PHP failu. Vienīgā tās atšķirība ir padotajos argumentos un izmantotā PHP faila nosaukumā.

```
@Override
protected String doInBackground(String... args) {
    String reiss_id = args[0];
    String latitude = args[1];
    String longitude = args[2];
    String link = LogInActivity.fullLink+
     "createPunkts.php?reiss_id="+reiss_id+
     "&latitude="+latitude+"&longitude="+longitude;
```

“CreatePunkts.php” satur:

```
<?php
include 'connect.php';
$reiss_id=$_GET['reiss_id'];
$lat=$_GET['latitude'];
$lon = $_GET['longitude'];
$result = mysqli_query($connection,"INSERT INTO Punkts
(reiss_id, latitude, longitude, laiks) VALUES
('$reiss_id','$lat','$lon',CURTIME())");
$id = mysqli_insert_id($connection);
if(!$result){
        die('Error:'.mysql_error());
}
echo $id;
mysqli_close($connection);
?>
```

3.4 Mobilās aplikācijas pasažieriem “Bussie” izveide

Mobilā aplikācija “Bussie” pasažieriem ir Android viedtālruņiem paredzēta mobilā aplikācija, kas ik pēc 30 sekundēm uzzināt interesējošā reisa atrašanās vietu. Aplikācija satur piecus layout failus, kas rakstīti izmantojot XML programmēšanas valodu, četrus aktivitāšu failus, kas rakstīti JAVA programmēšanas valodā un trīs failus, kas nodrošina komunikāciju ar datu bāzi. Lai no datu bāzes izgūtu datus, uz servera atrodas astoņi PHP programmēšanas valodā rakstīti faili.

Lai varētu izmantot internetu, aiz “application” aizverošā atslēgvārda , “AndroidManifest.xml” failā tiek pievienotas sekojošas rindiņas:

```
<uses-permission
android:name="android.permission.INTERNET"/>
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.
READ_GSERVICES"/>
```

Lai varētu izmantot karti, pie moduļiem ir jāpievieno “google-play-services_lib” modulis un starp moduļiem jāuzstāda atkarības. Pēc tam, līdzīgi kā pievienojot pārvadātāja aplikācijai savienotāju ar datu bāzi, pievieno JAR failu. “AndroidManifest.xml” failā pirms “application” aizverošā atslēgvārda failā tiek pievienotas sekojošas rindiņas:

```
<meta-data
   android:name="com.google.android.gms.version"
   android:value="@integer/google_play_services_version" />
<meta-data
   android:name="com.google.android.maps.v2.API_KEY"
   android:value=API_KEY/>
```

kur pie android:value “API_KEY” vietā ir īpaši pakotnei ģenerētā atslēga. Pirms atļaujām tiek pievienotas sekojošas koda rindiņas:

```
<uses-feature
            android:glEsVersion="0x00020000"
            android:required="true"/>
```

Līdzīgi kā mobilajā aplikācijā “Bussie” autobusiem, arī šajā aplikācijā pirmajā aktivitātē, kas tiek palaista reizē ar programmu, ir divi publiski, statiski mainīgie – String tipa objekts “fullLink” un “ArrayList” klases String tipa saraksts “parametri”. Palaižot aplikāciju pirmā aktivitāte, kas tiek palaista ir “StartActivity”, kas manto “Activity” klasi. Tā satur “onCreate” metodi un metodi, kas izpildās gadījumā, kad tiek nospiesta poga “b”.

“OnCreate” metode:

```
public void onCreate(Bundle bundle){
    super.onCreate(bundle);
    setContentView(R.layout.start_screen);
    c = getBaseContext();
    b = (Button)findViewById(R.id.button);
}
```

Metode “atrastAutobusu”:

```
public void atrastAutobusu(View view){
    Intent i = new Intent(c,ReadMarsrutiActivity.class);
    startActivity(i);
}
```

Nospiežot pogu, atveras nākamā aktivitāte “ReadMarsrutiActivity”, kas ir pilnīgi tāda pati, kā “Bussie” autobusiem. Sekmīgas izpildes gaitā parametru sarakstā tiek ievietots pirmais mainīgais – kursēšanas laika identifikators un aplikācija pāriet pie aktivitātes, kurā tiek izvēlēts interesējošais reisa kursēšanas datums – “DateActivity”.

```
public void toDateActivity(View view){
     StartActivity.parametri.add(laikiIds.get(spin2.getSe
     lectedItemPosition()));
     Intent i = new Intent(c,DateActivity.class);
     startActivity(i);
}
```

“DateActivity” klase manto “Activity” klasi. Izsaucot klasi, tiek izsaukta “onCreate” metode. Bez “onCreate” metodes, klase vēl satur metodi, kas tiek izsaukta nospiežot pogu un metodi, kas uzstāda kalendārā esošās dienas datumu.

“OnCreate” metode:

```
@Override
public void onCreate(Bundle b){
    super.onCreate(b);
    setContentView(R.layout.date_layout);
    c = getBaseContext();
    setCurrentDate();
}
```

“OnCreate” metode beigās izsauc “setCurrentDate” metodi, kas uzstāda šīs dienas datumu.

```
public void setCurrentDate(){
    datePicker =(DatePicker)findViewById(R.id.datePicker);
    final Calendar c = Calendar.getInstance();
    year = c.get(Calendar.YEAR);
    month = c.get(Calendar.MONTH);
    day = c.get(Calendar.DAY_OF_MONTH);
    datePicker.init(year, month, day, null);
}
```

Nospiežot pogu, tiek izsaukta metode “nextActivity”, kas izgūst izvēlētās dienas datumu, pārvērš to vajadzīgajā formātā, pievieno datumu parametru sarakstam, kā otro parametru un palaiž “MapActivity” klasi.

```
public void nextActivity(View view){
    day = datePicker.getDayOfMonth();
    month = datePicker.getMonth();
    year = datePicker.getYear();
    Calendar cal = Calendar.getInstance();
    cal.set(Calendar.DAY_OF_MONTH,day);
    cal.set(Calendar.MONTH,month);
    cal.set(Calendar.YEAR,year);
    SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
    String date = sdf.format(cal.getTime());
    StartActivity.parametri.add(date);
    Intent i = new Intent(c, MapActivity.class);
    startActivity(i);
}
```

Klases izveidošanās brīdī tiek izsaukta “onCreate” metode, kas izsauc superklases “onCreate” metodi padodot tai “Bundle” klases objektu kā argumentu. Pēc tam tiek uzstādīts skats un izsaukta metode “checkMap”, kas pārbauda vai ir izveidota karte, un, ja tādas nav, to izveido. Pēc “checkMap” metodes, tiek izveidots “Handler” klases objekts un izsaukta metode “readStartCoordinates”.

```
@Override
protected void onCreate(Bundle b){
    super.onCreate(b);
    setContentView(R.layout.map_screen);
    checkMap();
    handler = new Handler();
    readStartCoordinates();
}
```

```
private void checkMap() {
    if (map == null) {
        map = ((MapFragment) getFragmentManager()
        .findFragmentById(R.id.map)).getMap();
        if (map != null) {
            options = new GoogleMapOptions();
            options.mapType(GoogleMap.MAP_TYPE_NORMAL)
                   .compassEnabled(false)
                   .rotateGesturesEnabled(true)
                   .tiltGesturesEnabled(false);
            MapFragment.newInstance(options);
        }
    }
}
```

Metode “readStartCoordinates” izveido jaunu klases “ReadLatLong” objektu, kura “execute” metodei kā argumentus padod kursēšanas laika identifikatoru un izvēlēto datumu no parametru saraksta.

“ReadLatLong” klase ir līdzīga visām pārējām klasēm, ko izmanto, lai aplikācija varētu sadarboties ar PHP failiem. Arī šī klase manto “AsyncTask<String,Void,String>” klasi un atšķiras tikai izmantojot savus argumentus un PHP faila nosaukumu.

```
@Override
protected String doInBackground(String... args) {
    String id = args[0];
    String date = args[1];
    String link = StartActivity.fullLink+
    "getLatLong.php?laiks_id="+id+"&datums="+date;
```

“GetLatLong.php” fails satur:

```
<?php
include 'connect.php';
$id=$_GET['laiks_id'];
$date=$_GET['datums'];
$result = mysqli_query($connection,
     "SELECT latitude, longitude, laiks FROM Punkts
      WHERE punkts_id=(SELECT MAX(punkts_id)FROM Punkts
      WHERE reiss_id =(SELECT reiss_id FROM Reiss
      WHERE laiks_id='$id' AND datums='$date'))");
while($row = mysqli_fetch_array($result)){
        echo $row['latitude']."!!!".$row['longitude'].
                                "!!!".$row['laiks'];
}
mysqli_close($connection);
?>
```

Pēc “doInBackGround” metodes izpildes String tipa atgrieztais objekts kā arguments tiek padots “onPostExecute” metodei. Metode sadala objektu pēc “!!!”, atdalot garuma, platuma un laika vērtības. Izmantojot garuma un platuma vērtības kartē tiek atlikts marķieris un pie tā tiek pielikts teksts, kas atspoguļo laiku, cikos autobuss ir atradies norādītajā vietā.

```
@Override
protected void onPostExecute(String result){
    String[] latLongTime = result.split("!!!");
    BUS = new LatLng(Double.parseDouble(latLongTime[0]),
                     Double.parseDouble(latLongTime[1]));
    time = latLongTime[2];
    bus = map.addMarker(new MarkerOptions().position(BUS)
             .title("Autobusa atrašanās vieta"));
    bus.setSnippet("Laiks: "+time);
    map.moveCamera(CameraUpdateFactory.newLatLng(BUS));
    map.animateCamera
          (CameraUpdateFactory.zoomTo(16),3000,null);
}
```

Pēc “onCreate” metodes automātiski sāk darboties “onStart” metode, kas vispirms izsauc superklases “onStart” metodi, nosaka, ka karte vēl nav izslēgta un “Handler” klases objektam tiek izsaukta metode “onPostDelayed”, kam kā arguments tiek padots “Runnable” objekts. Ja nospiež taustiņu uz atpakaļ, karte tiek izslēgta un to ir iespējams atkal ielādēt no jauna.

```
@Override
protected void onStart(){
    super.onStart();
    off = false;
    handler.postDelayed(updateTimerThread,0);
}
@Override
public void onBackPressed(){
    super.onBackPressed();
    off = true;
}
```

Izveidojot “Runnable” objektu, tiek atkal izveidots jauns “ReadLatLong” klases objekts, kas atgriež garuma, platuma un laika vērtības, tikai šajā gadījumā iepriekšējais marķieris tiek noņemts, tā vietā uzstādīts jauns un kamera pārvietota uz jaunā marķiera pusi. Tas atkārtojas ik pēc 30 sekundēm, ja vien nav nospiests taustiņš “atpakaļ”.

```
@Override
protected void onPostExecute(String result){
    bus.remove();
    String[] latLongTime = result.split("!!!");
    BUS = new LatLng(Double.parseDouble(latLongTime[0]),
                     Double.parseDouble(latLongTime[1]));
    time = latLongTime[2];

    bus = map.addMarker(new MarkerOptions().position(BUS)
              .title("Autobusa atrašanās vieta"));
    bus.setSnippet("Laiks: "+time);
    bus.showInfoWindow();
    map.moveCamera
          (CameraUpdateFactory.newLatLngZoom(BUS,16.0f));

}
```

IZMANTOTĀS LITERATŪRAS UN AVOTU SARAKSTS

1. *Zinātnes un tehnoloģijas vārdnīca.* Proj. vad. Ilze Kačevska, galv. red. Dainuvīte Guļevska, Rīga, Apgāds "Norden AB", 2001. 36., 269., 287., 558.lpp. ISBN 9984-9383-5-2
2. J.Žagars, J.Zvirgzds, J.Kaminskis, *Globālās navigācijas satelītu sistēmas (GNSS)*, 2014 (izdošanas procesā)
3. Peter H. Dana, GPS Nominal Constellation, [cites April 26, 2014]. Available: http://www.colorado.edu/geography/gcraft/notes/gps/gif/orbits.gif
4. GPS CONSTELLATION STATUS FOR 05/17/2014, [cites May 17, 2014]. Available: http://www.navcen.uscg.gov
5. Peter H. Dana, GPS Master Control and Monitor Network, [cites April 26, 2014]. Available: http://www.colorado.edu/geography/gcraft/notes/gps/gif/gpscont.gif
6. Peter H. Dana, GPS satellite signals, [cites April 26, 2014]. Available: http://www.colorado.edu/geography/gcraft/notes/gps/gif/signals.gif
7. Peter H. Dana, Intersection of Range Spheres, [cites April 26, 2014]. Available: http://www.colorado.edu/geography/gcraft/notes/gps/gif/figure09.gif