TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karmen Tamsalu, 155433IAPB

KIIRENDUSANDURI SIGNAALI EDASTAMINE JA TÖÖTLEMINE SAMMULUGEMISE EESMÄRGIL

Bakalaureusetöö

Juhendajad: Ardo Allik
MSc
Marko Kääramees
PhD

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Karmen Tamsalu

20.05.2018

Annotatsioon

Käesoleva bakalaureusetöö on suunatud sellele, et tutvustada kasutajatele keerulist sammulugemise protsessi visuaalselt lihtsa ja arusaadava liidese abil.

Töö eesmärgiks on kiirendusanduri signaali reaalajas edastamine ja töötlemine, graafikule kuvamine ja sammude lugemine. Eesmärgi täitmiseks valmivad kaks eraldi rakendust - mobiilirakendus kiirendusandurilt signaali kätte saamiseks ja veebirakendus signaali ning kokku loetud sammude arvu kuvamiseks.

Teiseks töö eesmärgiks on võimaldada algoritmi kasutamine alustala ja abivahendina uurimis- või magistritööde raames.

Tulemusena valminud rakendused täidavad püstitatud eesmärgid. Signaalipunktid kuvatakse graafikule reaalajas, sammulugemise algoritm loetleb kasutaja astutud sammud väikese hälbega ning töös on lahti seletatud algoritmi võimalikud arengusuunad.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 21 leheküljel, 6 peatükki, 8 joonist, 1 tabelit.

Abstract

The transmission and processing of accelerometer signal for step counting purposes

The objective of this bachelor thesis is to analyse the processing of accelerometer signal for real time signal visualising and step counting purposes.

The solution is made as a combination of mobile and web application. Mobile application connects to accelerometer via Bluetooth, receives and sends signal data to cloud database. Web application listens to database changes and processes signal data for further use. There are two background processes. One, to place the signal data points on to the graph and the other, to count steps based on accelerating signal.

The second aim of this thesis is to offer a base for further researches and development for master's thesis. One possibility is to evolve step counting algorithm so that it can detect user movement and/or calculate energy consumption.

As a result of this thesis the applications meet the set requirements. Data points are displayed on the graph in real time, step counting algorithm detects user's steps with a minor error and possible directions of further development are explained.

To test the accuracy of the algorithm a series of experiments were carried out, which concluded that the written algorithm can be described by high sensitivity. Based on additional testing it turned out, that it also can be described by low specificity.

Completed applications are great help to introducing complicated step counting principles with a simple and understandable user interface to people of interest. The applications are completed in collaboration with the Department of Health Technologies to which they are assistance.

The thesis is in Estonian and contains 21 pages of text, 6 chapters, 8 figures, 1 table.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakendusliides
REST	*Representational State Transfer*, tarkvaraarhitektuuri stiil, mis määrab, kuidas üle võrgu ressursside poole pöörduda
JSON	*JavaScript Object Notation*, lihtsustatud andmevahetusvorming
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastamise protokoll
SDK	*Software Development Kit*, tarkvaraarenduspakett
XML	*Extensible Markup Language*, laiendatav märgistuskeel
URL	*Uniform Resource Locator*, internetiaadress ehk universaalne ressursilokaator

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Töö autorile ja tema tutvusringkonnale meeldib kasutada erinevaid kiirendusanduritega seadmeid, et hoida pilku peal oma kehalisel liikumisel. Ette on tulnud olukordi, kus sammude päevanormi saavutamiseks vehitakse käega või plaksutatakse. Tervise edendamise huvides ei saa keskenduda ainult numbritele. Seetõttu ongi antud lõputöö raames valmiv rakendus suunatud sellele, et tutvustada kasutajatele keerulist sammulugemise protsessi visuaalselt lihtsa ja arusaadava liidese abil.

Lõputöö käigus valmib rakenduste komplekt, mille eesmärgiks on reaalajas kuvada kiirendusanduri signaalkuju ning arvutada kasutaja astutud sammude arv. Rakendused on valminud koostöös Tervisetehnoloogiate instituudiga, millele need on heaks abivahendiks. Täpsemalt kooliõpilastele ja huvilistele sammulugemise põhitõdede ja kiirendusanduri töö tutvustamiseks.

Lisaks rakenduste komplektile on lõputöö üheks alamülesandeks leida kasutaja poolt astutud sammude arv. Selle leidmiseks kirjutas autor lihtsa, kuid samas võimalikult täpse sammulugemise algoritmi, mille täpsuse analüüsimiseks viidi lõputöö raames läbi katseseeria.

Töö teiseks põhieesmärgiks on pakkuda alustala ja abivahend, mida edaspidi saaks kasutada uurimistöödes või magistritöö raames. Näiteks edasiarenduse puhul annab see võimalused kasutaja tegevuse tuvastamiseks kiirendusanduri signaali põhjal ja energiakulu arvutamiseks.

Töö koosneb neljast sisulisest peatükist, sissejuhatusest ja kokkuvõttest.

- Teises peatükis kirjeldab autor rakendustele seatud funktsionaalseid ja mittefunktsionaalseid nõudeid, millest lähtutakse rakenduste arendamisel. Lisaks annab autor ülevaate tervikrakenduste struktuurist ja arhitektuurist.

- Kolmandas peatükis toob autor välja rakenduste arendamise jaoks kasutatud tehnoloogiad ehk mida ja kuidas kasutati rakenduste valmistamiseks. Samuti seletab autor lahti sammulugemise algoritmi tööpõhimõtted.

- Neljandas peatükis räägib autor lahti suuremad rakenduse arendamisega tekkinud probleemid ning analüüsib sammulugemise algoritmi täpsuse mõõtmise katsetulemusi.

- Viiendas peatükis kirjeldab autor ülevaatlikult, kuidas oleks võimalik lõputöö raames valminud rakendusi edasi arendada. Täpsemalt, kuidas muuta sammulugemise algoritmi paremaks ning mida oleks veel võimalik kiirendusanduri signaalkuju töötlemisel arvutada.

2 Nõuete analüüs

Selles peatükis kirjeldab autor lõputöö raames valmivate rakenduste funktsionaalseid ja mittefunktsionaalseid nõudeid ning selgitab töö planeerimist. Antud rakendused on mõeldud kiirendusanduri tööpõhimõtete selgitamiseks huvigruppidele ning suunatud lõputöö põhijuhendajale kasutamiseks abivahendina. Sellest tulenevalt on nõuded pandud paika juhendaja soovide ning autori varasema kogemuse põhjal.

Kasutatud tehnoloogiaid kirjeldab autor lähemalt peatükis 3.

2.1 Mobiilirakendus

2.1.1 Funktsionaalsed nõuded

- Kasutaja saab ühendada oma Androidi seadme Shimmeri anduriga.
- Kasutajal on kaks valikut: *Start streaming* ja *Stop streaming*, mis vastavalt alustab andurilt andmete edastamise ning lõpetab andmete edastamise.

2.1.2 Mittefunktsionaalsed nõuded

- Kasutajale peab olema arusaadav, millal on Shimmeri andur valmis andmeedastuseks ning millal esineb tõrkeid.

2.2 Veebirakendus

2.2.1 Funktsionaalsed nõuded

- Kasutaja peab nägema signaali võnkumisi reaalajas.
- Kasutaja saab valida, milliseid signaale graafikul näha soovib. Valikus on x-, y-, z-telje signaalid ning nende signaalide ruutkeskmine. Vaikimisi kuvatakse rakenduses x-, y-, ja z- teljed.

- Kasutaja saab valida sobiva andmete edastamise sageduse vastavalt Shimmeri anduril sätestatud diskreetimissagedusele. Vaikimisi on rakenduses määratud 50,33Hz.

- Kasutaja näeb veebilehekülje reaalajas signaalkuju ning lisaks mitu sammu on autori poolt kirjutatud sammulugemise algoritm tuvastanud.

2.2.2 Mittefunktsionaalsed nõuded

- Veebirakenduses peab olema signaali kuvamine sujuv.

- Veebirakendus peab kasutajale olema lihtne ja arusaadav, kuna selle eesmärgiks on tutvustada inimestele kiirendusandurite tööpõhimõtteid.

2.3 Tervikrakenduse struktuur

Ülevaatlikult toimib tervikrakendus järgnevalt:

1. Kasutaja ühendab mobiilirakenduse kaudu Androidi seadme Shimmeri anduriga.

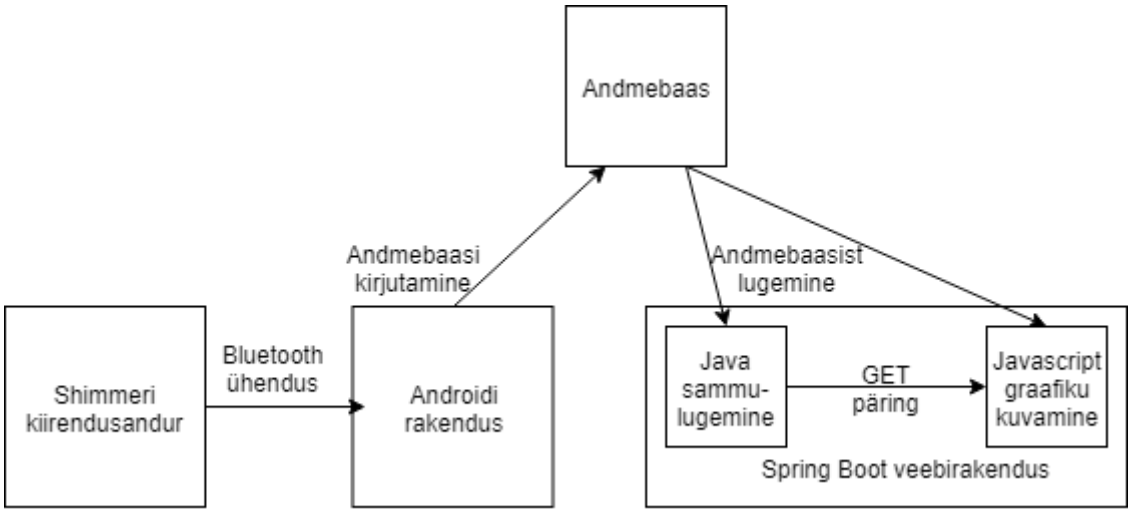
2. Mobiilirakenduses avaneb kasutajale võimalus „*Start streaming“*, mis alustab kiirendusandurilt andmete edastamist mobiilirakendusele ning „*Stop streaming“*, mis lõpetab andmete edastamise. Rakenduses on loodud ühendus andmebaasiga, kuhu kiirendusanduri signaal edasi saadetakse.

3. Kasutajal on lahti veebirakendus, mis kuulab andmebaasi muudatusi ning andmerea lisandumisel, kuvab selle signaalipunkti ka graafikul.

4. Paralleelselt töötab ka sammulugemise algoritm, mis kuulab samuti andmebaasi muudatusi, ning nende signaalide põhjal arvutab kasutaja poolt tehtud sammud. Algoritmi kokku arvutatud sammud saadetakse veebirakendusele päringu vastusena.

5. Veebirakenduses näeb kasutaja reaalajas vaikimisi kolme teljelist kiirendussignaali ning sooritatud sammude arvu.

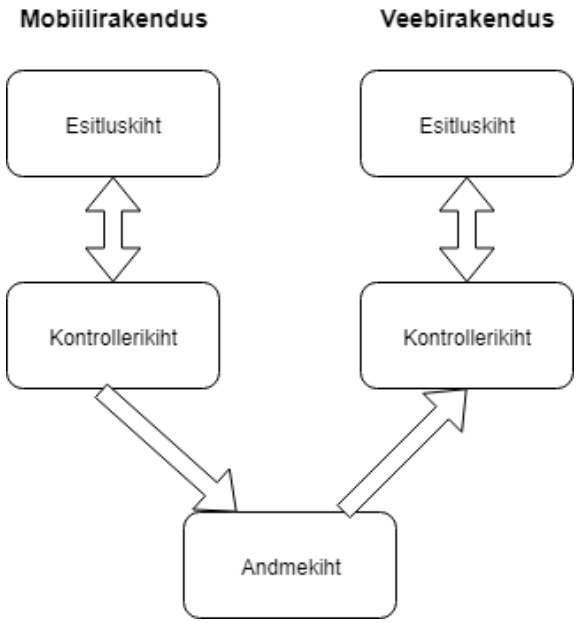
Süsteemi tervikrakenduse struktuur on kujutatud Joonisel 1



Joonis 1. Rakenduste struktuur, mille osade vahel edastatakse kiirendusanduri signaalkuju noolega näidatud suunas. Andmebaas on vahelüli signaalkuju saatmisel Androidi rakendusest veebirakendusesse.

2.4 Rakenduse arhitektuur

Mõlemas rakenduses on kasutusel kolmekihiline arhitektuur, mis on mõeldud olema võimalikult lihtne ja kergesti edasi arendatav. Kolm kihti on selgelt defineeritud: esitluskiht, kontrollerikiht ja andmekiht. Andmekiht ühtib nii veebi- kui ka mobiilirakendusel. Ülesehitust illustreerib Joonis 2.



Joonis 2. Rakenduse arhitektuur. Mobiili- ja veebirakendusel on ühine andmekiht, mille kaudu mobiilirakendus suhtleb veebirakendusega. Kontrolleri- ja esitluskihid on mõlemal rakendusel eraldi, kuna täidavad eri funktsioone.

Esitluskihti kuulub see osa rakendustest, mida kasutajale kuvatakse. Antud töös on see mobiilirakenduses ressursside kaustas *Layout* ning veebirakenduses *Spring Boot*'i

staatiliste ressursside kaustas olevad failid. Kasutaja tegevuste peale pöördub esitluskiht kontrollerikihi poole, kuhu antakse edasi kasutaja käsud. Sealt võib kontrollerikiht pöörduda tagasi esitluskihi poole uute andmetega või hoopis uute sisenditega andmekihi poole. Lõputöös valminud mobiilirakenduse näitel toimub kihtide vaheline suhtlemine järgnevalt:

- Kasutaja vajutab mobiilirakenduse esitluskihis nuppu „*Start streaming*“. Esitluskiht pöördub selle käsuga kontrollerikihi poole, mis käivitab funktsiooni kiirendusandurilt signaali mõõtmiseks.

- Kontrollerikiht omakorda saadab kiirendusanduri signaali edasi andmekihile, mis salvestab andmed *Firebase* andmebaasi. Andmete edastamine toimub seni, kuni esitluskihist saadakse uus käsk „*Stop streaming*“, mis tähistab andmeedastuse lõpetamist.

Veebirakenduses toimub suhtlemine vastupidiselt – andmekiht kuulab muudatusi andmebaasis ja edastab uued andmed kontrollerikihti, mis töötleb signaali ja arvutab sammud. Sealt edastatakse töödeldud andmed omakorda esitluskihti signaali ja sammude arvu kuvamiseks graafikul.

3 Rakenduse realiseerimine

Selles peatükis kirjeldab autor täpsemalt lõputöö rakenduste valmimiseks kasutatud tehnoloogiaid. Samuti põhjendab autor tehnoloogiate ja algoritmi valikuid ning annab ülevaate alternatiivsetest variantidest.

3.1 Ülevaade kasutatud tehnoloogiatest

Antud lõputöö raames kasutati mobiilirakenduse loomiseks *Android Studio* keskkonda ning Shimmeri *Java/Android API*'t. Veebirakenduse loomiseks kasutati *IntelliJ* keskkonnas *Spring Boot*i, Javat, Javascripti ja *CanvasJS*i ning andmebaasiks valiti *Firebase Realtime Database*.

Sellised tehnoloogiad said valitud kombinatsioonina isiklikest eelistustest tarkvara arendamisel, sarnaste rakenduste kallal töötavate professionaalide arvamusest ning lõputöö juhendajaga konsulteerimisel leitud lahendustest.

3.1.1 Shimmeri andurid

Shimmer on väike juhtmevaba andur, mis on suunatud uurimustööde ja nende põhiste rakenduste arendamiseks. Shimmeri seadmetesse integreeritud kinemaatilised andurid ning suur mäluruum on peamiselt mõeldud liikumise tuvastamiseks, pikaajaliste andmete kogumiseks ja nende reaalajas jälgimiseks. Antud töö raames töötles autor sammude tuvastamiseks Shimmeri anduri (edaspidi kiirendusanduri) x-, y-, z-telje signaale ja signaalide salvestamise hetke aega. [1]

Joonis 3. Shimmeri andur külg- ja eestvaates [2].

Joonis 4. Shimmeri andur käe ümber [2].

3.1.2 Shimmer Java/Android API

Selleks, et kiirendusandurilt signaalid kätte saada, kasutas autor arendajatele mõeldud teeki *Shimmer Java/Android API* [2]. Selle abil on võimalus *bluetooth* ühenduse kaudu edastada, kuvada ja logida andmeid Androidi seadmes, millest autor kasutas ainult signaali edastamise võimalust.

Teek on üles ehitatud mitmest projektist, milledest antud töös olid kasutusel kaks osaprojekti. Esimene osaprojekt *ShimmerAndroidInstrumentDriver* deklareerib *bluetooth* ühenduse loomise Androidi seadmega. Teine antud lõputöös kasutatud osaprojekt *efficientDataArrayExample* võimaldas reaalajas signaalide edastamise Androidi seadmele ning pakkus lihtsa rakenduse kujunduse, mida autor kohendas vastavalt projekti vajadustele.

3.1.3 Spring Boot

Lõputöö raames oli vaja valmis saada veebilehekülg, mis kuvaks signaalkuju graafikut ning sammude arvu. Veebilehe kirjutamiseks oli loogiline valik Javascript, kuid sammulugemise algoritmi otsustas autor kirjutada Java programmeerimise keeles selle laialdaste võimaluste tõttu. Java ja Javascripti omavaheliseks suhtlemiseks uuris autor erinevaid võimalusi. Üheks variandiks oli *.NET* raamistik, mis võimaldab arendada laialdasi rakendusi, kuid põhilisteks keelteks, mida selle raamistiku puhul kasutatakse

on C# ja C. Valituks osutus siiski *Spring Boot* just tema uudsuse ning lihtsuse poolest. Valikut mõjutasid ka XML konfiguratsioonifailide puudumine ja Java keele kasutamise võimalus.

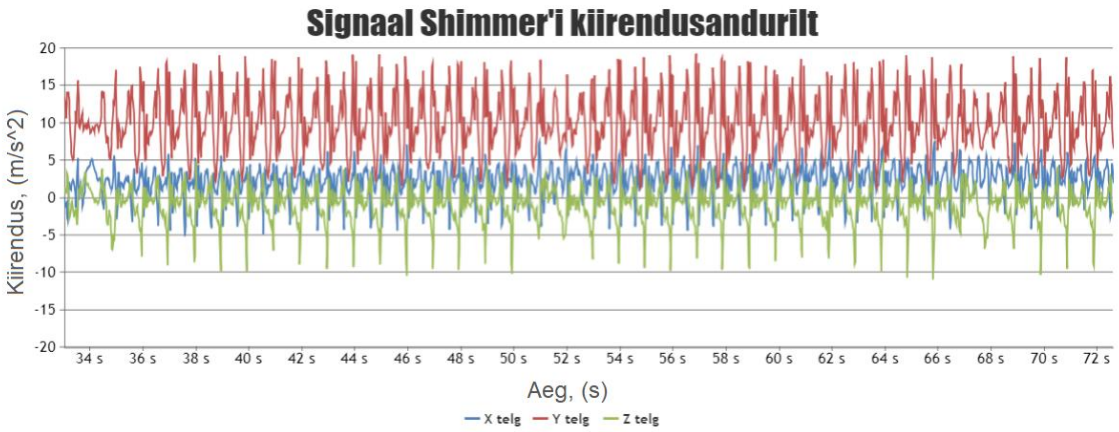
Spring Boot on raamistik rakenduste ehitamiseks, mis võimaldab arendada rakendusi nii REST, Java kui ka veebi põhistena [3]. Peamiseks eesmärgiks on arendajatele pakkuda kiiret ja laialdast võimalust rakenduste püsti panemiseks [4]. *Spring Boot* raamistiku kasutamine käib sarnaselt iga teise Java teegiga. Antud projekti raames kasutas autor *Spring Boot'i* seadistamiseks *Gradle* installatsiooni, kus *Spring*'i deklaratsioon tuleb teha projekti *build.gradle* failis.

Spring Boot raamistiku abil tegi autor nii veebilehe ja selle kuvamise loogika kui ka Java osa sammulugemise funktsionaalsusega.

3.1.4 CanvasJS

Lõputöö veebirakenduse üks olulisemaid funktsionaalsusi oli reaalaja graafiku kuvamine. Siinkohal osutus raamistiku valimine lihtsaks, kuna alternatiivsed võimalused puudusid. Valitud *CanvasJS* sobis rakenduse jaoks väga hästi, kuna see pakkus dünaamilise graafiku kuvamist ning kiiret jõudlust isegi 50 000 andmepunkti kuvamiseks.

CanvasJS pakub arendajatele 30 erinevat tüüpi graafikuid, mida saab kuvada erinevatel platvormidel telefonides ning arvutites [5]. Antud lõputöö raames kasutas autor *CanvasJS* graafiku tüüpi *Line chart.* See kuvab informatsiooni graafikul andmepunktidena, mis on omavahel ühendatud sirge joonega [6]. Igal andmepunktil on nii x- kui ka y-telje väärtus. Antud projekti raames on horisontaalse telje väärtus signaali kiirendus vastava telje suunas ja vertikaalse telje väärtus signaali salvestamise ajahetk. Selleks, et signaal graafikul oleks reaalajas muutuv kombineeris autor *Line chart*'i *Dynamic chart*'iga [7]. Selline graafik uuendab oma andmeid kindla ajavahemiku tagant. Andmete uuendamise korral andmepunkte lisatakse ning kustutatakse, mille tulemusena kuvatakse graafikul alati kindla arvu sekundite jagu signaali punkte.



Joonis 5. Sammude kolme telje signaalkuju kuvatuna *CanvasJS* dünaamilisel joon-graafikul.

3.1.5 Firebase

Andmete edastamiseks Androidi seadmest veebirakendusesse oli vaja leida võimalikult kiire ja lihtne lahendus, mille abil oleks võimalik kuvada signaali reaalajas. Üheks võimalikuks lahenduseks oli serverisse püsti panna andmebaas. See oleks aga vajanud serveri koodi kirjutamist rakenduses. Lisaks ei tundunud see autorile kõige kiirem variant signaali kuvamiseks graafikule. Seetõttu asuti uurima erinevaid alternatiivseid võimalusi, mis oleksid lisaks vajalikule kiirusele ka lihtsamad. Alternatiivsete võimalustena olid valikus *Parse Server* ja *Firebase*. Autori varasema kogemuse tõttu *Google Firebase*'ga osutuski valituks *Firebase Realtime Database*. Samuti kiitsid valiku heaks Androidi arendajad, kellega autor soovituste saamise eesmärgil konsulteeris.

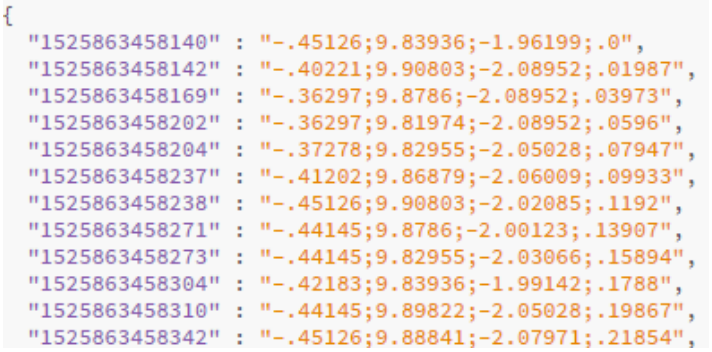
Firebase Realtime Database pakub nii andmebaasi kui ka *backend* teenuseid [8]. See on pilvepõhine NoSQL andmebaas, kus andmeid hoitakse JSON formaadis. HTTP päringute asemel sünkroniseeritakse andmed iga ühendatud kliendiga millisekundite jooksul, kui andmetes toimub muudatusi.

Firebase rakendused töötavad edasi ka võrguühenduse katkemise korral, kuna *Firebase Realtime Database* SDK salvestab andmed seadme kõvakettale. Uuesti võrku ühendudes sünkroniseerib klientrakendus kõik vahepeal toimunud muudatused. Antud lõputöö raames on siiski oluline võrguühenduse olemasolu, et oleks võimalik kiirendusanduri signaali reaalajas graafikul kujutada ning tuvastada samme.

Firebase Realtime Database'i üheks eeliseks on ka see, et andmebaasiga ühendamiseks ei ole vaja serveri rakendust. Andmebaas on kättesaadav otse klientrakendusest läbi

mobiiliseadme või veebibrauseri. Serveri rakenduse puudumise tõttu on ka turvalisuse ja andmete valideerimise reegleid võimalik muuta otse andmebaasis. Andmebaasile pääseb ligi läbi *Firebase Console* veebilehekülje, kus lisaks turve seadetele on võimalik vaadata ning muuta ka andmebaasis olevaid andmeid.

Antud lõputöös kasutatud andmebaasi struktuuri kirjeldab Joonis 6.



Joonis 6. Andmebaasi struktuur näidisandmetega. Andmed on salvestatud võti-väärtus paaridena, kus võtmeks on andmebaasi kirjutamise hetke aja teisendus ning väärtuseks kiirendussignaali x-, y-, z-telje suunalised väärtused ja mõõtmise hetke ajaline väärtus eraldatuna semikoolonitega.

Andmed salvestatakse *Firebase* andmebaasi Androidi rakendusest. Andmebaasi muudatusi kuulavad mõlemad veebirakenduse graafiku kuvamise osa ja Java sammulugemise algoritmi osa. Autor proovis rakenduse struktuuri lihtsustamise mõttes panna veebirakenduse Javascript'i koodi edastama POST meetodiga Java sammulugemise koodile signaali koordinaate, kuid sellisel viisil jõudsid signaalid Java koodi asünkroonselt ehk vales järjekorras. Katsetati ka signaalide saatmist grupeerituna 10 kaupa massiivides, aga ka selline lahendus ei osutunud efektiivseks. Kuna signaalide õige järjekord on sammulugemise algoritmis oluline aspekt, siis otsustas autor jääda esimese variandi juurde, kus nii Javascript kui Java kuulavad andmebaasi muudatusi eraldi. Selline lahendus osutus kõige vahetumaks ning kiiremaks.

3.2 Sammulugemise algoritm

Selles peatükis kirjeldab autor lähemalt erinevaid sammulugemise algoritme, mida töö raames kaalus kasutada. Lahti on seletatud ka lõputöö jaoks kirjutatud algoritmi põhifunktsionaalsus.

3.2.1 Algoritmi metoodika valimine

Lõputöö üheks põhiliseks funktsionaalsuseks oli sammulugemise algoritmi kirjutamine kiirendusanduri signaali põhjal. Algoritmi kirjutamisel võttis autor arvesse lõputöö püstitusega sätestatud tingimusi: algoritm peab olema võimalikult lihtne ja täpne. Lihtsuse kriteeriumi põhjuseks oli autori eriala, mille raames on varasemalt signaalitöötlusega kokku puututud vaid minimaalselt. Seetõttu on väga suur kaal just antud lõputöö rakendustel, mitte parima signaalitöötlus algoritmi arendamisel. Täpsuse kriteerium seisneb selles, et algoritm suudaks anduriga kõndija astutud sammud arvutada kokku võimalikult väikese erinevusega reaalsuses astutud sammudest.

Algoritmi valimiseks uuris autor erinevaid teadusartikleid sarnaste katsete ja projektide kohta. Autor valis algoritmi kirjutamiseks Java programmeerimiskeele, kuna see keel oli kõige tuttavam ning kirjandusülevaate põhjal oli Java keeles ka varasemalt mitmeid sammulugemise algoritme kirjutatud [9].

Algoritmi metoodika valimiseks olid erinevad võimalused. Üheks variandiks oli *Pan-Tompkins*'i meetod, mis kasutas ainult maaga risti olevat kiirendussignaali komponenti, lävendi printsiipi müra eemaldamiseks, mitmeid filtreid ning seejärel tuvastada sammud eeltoodud signaalitöötluse järel. Antud meetod ei osutunud praktikas siiski väga täpseks [9]. Täpsemaks variandiks osutus *Wolf*'i meetod, kus signaali müra lävendit kohandati vastavalt parasjagu töötlusel olevate signaalide aknale (*Sliding window* meetod) [9]. Antud töö autor kasutas algoritmi kirjutamisel *Wolf*'i meetodile sarnast taktikat, kuid kohandatuna.

Sammulugemise algoritmi tulemusena peaks autori poolt koostatud sammulugemise algoritm tuvastama kõik kõndija poolt sooritatud harilikud sammud. Sellisteks sammudeks saab lugeda sammu, mis ei ole marssimine ega ka hiiliv astumine. Sarnaselt nutiseadmete sammulugemise algoritmidele ei pruugi ka autori kirjutatud algoritm tuvastada väga tasaseid või väikeseid sammusid, kuna kiirendusanduri signaalis esinev müra raskendab sammude tuvastamist.

3.2.2 Algoritmi realiseerimine

Algoritmi kirjutamise aluseks võttis autor M. Mladenov'i ja M. Mock'i artikli „*A Step Counter Service for Java-Enabled Devices Using a Built-In Accelerometer*" [10]. Erinevalt artiklis kasutatud nutiseadmetesse sisseehitatud kiirendusanduritest kasutas

antud lõputöö autor eelmainitud Shimmeri kiirendusandurit, mis mõõtis kiirendussignaali kolme telje sihis.

Sammude tuvastamiseks võib katsealune hoida kiirendusandurit taskus, rihmaga käel või seljal. Anduri paiknemisest tingitud vigade vähendamiseks kasutatakse algoritmis kolme kiirendusvektori geomeetrilist summat ehk kiirenduse magnituudi. Algoritmi esimeseks sammuks ongi leida magnituud kolme telje kiirendussignaalide geomeetrilise summana *Pythagorase* valemi (1) abil, kus m tähistab magnituudi ning x, y, z vastava telje suunalist kiirendussignaali.

$$m = \sqrt{x^2 + y^2 + z^2} \quad (1)$$

Järgmise sammuna tuleb signaali magnituudide seast välja filtreerida ebavajalik müra. Kasutati lihtsat filtrerimise meetodit, mis võtab teatud arvust signaali magnituudidest keskmise. Signaale filtreerib autor 5 kaupa, kuna erinevate katsetuste järeldusena oli selline massiivi suurus võimalikult suur, et saaks lahti üleliigsest mürast, aga samas piisavalt väike, et algoritm oleks võimalikult täpne.

Iga 2 sekundi tagant liigub algoritm magnituudidega edasi sammude tuvastamise juurde. Sammude tuvastamiseks läbib algoritm kaks erinevat tsüklit, millest esimesega arvutatakse signaalitippude keskmine väärtus. Esimene tsükkel on välja toodud Joonisel 7.

```
public double getPeakMean() {
    int peakCount = 0;
    double peakAccumulate = 0;

    for (int i = 1; i < magnitudes.size() - 1; i++) {
        double forwardSlope = magnitudes.get(i + 1) - magnitudes.get(i);
        double backwardSlope = magnitudes.get(i) - magnitudes.get(i - 1);
        if (forwardSlope < 0 && backwardSlope > 0 && magnitudes.get(i) > K) {
            peakCount++;
            peakAccumulate += magnitudes.get(i);
        }
    }

    return peakAccumulate/peakCount;
}
```

Joonis 7. Signaalitippude keskmise väärtuse arvutamine. Signaalitipuks loetakse magnituudi, mis on suurem temale eelnevast ja järgnevast magnituudist ning müra lävendist K.

Teise tsükli käigus vaadatakse kõik magnituudid uuesti läbi ning leitakse sellised signaalitipud, mis on suuremad temale eelnevast ja järgnevast, keskmisest signaalitipu

suurusest ning müra lävendist. Nendele tingimustele vastavat signaalitippu saab arvestada sammuna. Teine tsükkel on välja toodud Joonisel 8.

```
public void stepCountingAlgoritm(List<Double> mags) {
    double peakMean = getPeakMean();

    for (int i = 1; i < mags.size() - 1; i++) {
        double forwardSlope = mags.get(i + 1) - mags.get(i);
        double backwardSlope = mags.get(i) - mags.get(i - 1);
        if (forwardSlope < 0 && backwardSlope > 0 && mags.get(i) > C * peakMean && mags.get(i) > K) {
            steps++;
        }
    }
}
```

Joonis 8. Signaalitipp loetakse sammuks, kui magnituud on suurem temale eelnevast ja järgnevast magnituudist, keskmisest signaalitippude kõrgusest korrutatuna konstandiga C ning müra lävendist K.

Müra lävendi K leidis autor erinevate katsetamiste tulemusena, millest selgus, et sammu magnituud on tavaliselt suurem kui 11,5 m/s^2. Eelnevalt leitud keskmine signaalitipp korrutatakse läbi konstandiga C(=0,8), kuna leidub ka samme, mille magnituud on väiksem keskmise signaalitipu magnituudist.

Tsüklite lõppedes tühjendatakse filtreeritud magnituudide loend ning alustatakse juba järgmiste signaalide hulga filtreerimist ning nendest sammude välja lugemist.

4 Analüüs

Selles peatükis kirjeldab autor rakenduse valmimisega tekkinud suuremaid probleeme. Lisaks analüüsitakse lõputöö raames läbiviidud sammulugemise algoritmi testimise katseseeria tulemusi.

4.1 Tehniliste probleemide lahendamine

Lõputöö raames valmis rakenduste komplekt, mis koosneb erinevatest väiksematest osadest. Need osad tuli omavahel integreerida, et tervik töötaks ühtse ja loogilise rakendusena. Siin peatükis kirjutab autor lahti erinevate rakenduse osade omavahelise suhtlemise ning selle lahendamise.

4.1.1 Mobiili- ja veebirakendus

Veebirakenduse funktsionaalsetes nõuetes on paika pandud, et kasutaja peab nägema signaali võnkumisi reaalajas. Selle jaoks oli autoril vaja leida lahendus, mis edastaks kiirendusanduri signaali, mille logimine Androidi rakendusele oli eelnevalt defineeritud *Shimmer Java/Android API*'ga, vahetult kohe veebirakendusse. Seega tuli leida andmete hoiustamiseks selline koht, et need oleksid mobiilirakendusest üles laadides kohe kättesaadavad ka veebirakendusele. Põhiliseks probleemiks oli aga kiirendusanduri andmete edastamise sagedus. Nimelt minimaalne signaali edastamise sagedus kiirendusandurilt mobiilirakendusele on 50,33Hz, seega andmebaasi tuleb kirjutada umbes 50 korda ühe sekundi jooksul. Selline edastamise sagedus teeb ka andmemahu küllaltki suureks, kuna signaali graafikul peaks korraga olema näha umbes 40 sekundi jagu andmeid.

Antud lõputöö rakendustes toimibki parima lahendusena leitud *Firebase* andmebaas vahelülina mobiili- ja veebirakenduse omavahelises suhtlemises. Mobiilirakenduse projektis deklareeritud *Firebase* viite abil on võimalik kirjutada kiirendusanduri signaali koordinaadid pilvepõhisesse andmebaasi. Veebirakenduses (nii Javascriptis kui Javas) on defineeritud muudatuste kuulajad, mis andmerea lisandumisel andmebaasi asuvad täitma edasisi funktsioone vastava signaali koordinaatidega. Selline pilvepõhise

andmebaasi lahendus on täitnud kõik nõuded ning andmed jõuavad millisekunditega mobiilirakendusest veebirakendusele töötlemiseks.

4.1.2 Veebirakendus ja sammulugemise algoritm

Veebirakenduses on kaks eraldi töötavat komponenti: graafiku kuvamine Javascriptiga ning sammulugemise algoritm Javaga. Nagu eelpool mainitud, siis lahendus, kus mõlemad veebirakenduse komponendid kuulavad eraldi andmebaasi muudatusi osutus kõige efektiivsemaks ja kiiremaks. Alternatiivse variandi puhul, kus autor saatis signaalid Java algoritmile POST päringu kaudu, muutus signaalide kohale jõudmine Javasse asünkroonseks ning algoritm ebatõeseks.

Sellegipoolest oli vaja sammude arvu edastamiseks Java algoritmist kasutajaliidesele need kaks poolt omavahel suhtlema panna. Lahendus oli *Spring Boot* raamistiku poolt olemas – GET päring [11], mille funktsionaalsuse võib lugeda võrdseks veebiteenustes kasutatava samanimelise pärinuga. *Spring Boot*'i puhul tähistatakse GET päringut Java koodis annotatsiooniga „@GetMapping(value = "/steps")". Javascripti poole pealt kutsutakse päringut välja *JQuery Ajax*'i funktsiooniga, mis edastatakse samasuguse URL aadressiga, nagu Java annotatsioonis (window.location + "steps"). Päringule vastatakse Java *SignalResponse* objektitüüpi vastusega, mis koosneb sõnest, kas päring õnnestus või mitte ning selleks ajahetkeks kokku loetud sammude arvust. Päringu õnnestumise korral kirjutatakse sammude arv kasutajaliideses vastavasse kohta. Päringut GET kutsutakse Javascriptis välja iga kord, kui ka graafikut uuendatakse ehk iga 100 millisekundi tagant.

Erinevalt POST päringust, toimib GET päring sammude edastamiseks ideaalselt. Põhjuseks võib olla see, et GET päringut kutsutakse sekundis välja 10 korda samal ajal kui POST'i oli vaja saata minimaalselt 50 korda sekundi jooksul. Sammulugemise algoritmi puhvri suuruse tõttu muutub sammude arv aga iga 2 sekundi tagant, seega ka asünkroonsuse puhul jõuab kasutajani õige sammude arv. Rakenduse lihtsuse põhimõtet järgides ei teinud autor GET päringu jaoks Javascripti eraldi ajaintervalli funktsiooni. Seda valikut mõjutas ka Javascripti keele lihtsus ning asjaolu, et selles keeles puudub lõimede funktsionaalsus.

4.2 Valminud rakenduste analüüs

Lõputöö raames valminud rakendused täidavad kõiki püstitatud funktsionaalseid ja mittefunktsionaalseid nõudeid.

Mobiilirakendus ühendub Shimmeri kiirendusanduriga Bluetooth ühenduse abil. Mobiilirakendusest saab kasutaja alustada ning lõpetada signaali edastamist kiirendusandurilt ning kasutajale on ära näidatud ühendusel esinevad tõrked Andoridi *Toast* sõnumitega. Mobiilirakenduse ekraanipilt on välja toodud Lisas 1.

Veebirakenduses on graafikule kujutatud signaali kuju reaalajas ning algoritmi poolt kokku loetud sammude arv. Veebirakenduse graafiku vaade on näidatud Lisas 2. Rakenduse kujundamisel on rõhutatud lihtsusele, et signaalkuju graafik oleks kasutajale võimalikult arusaadav. Graafikule kuvatakse vaikimisi x-, y- ja z- telje suunalised kiirendussignaalid, kuid kasutaja saab kuvatavaid telgi menüüst ise muuta. Lisaks vaikeväärtustele on valikus ka signaalide ruutkeskmine ehk magnituud. Veebirakenduse menüü ekraanipilt on näidatud Lisas 3. Samuti saab kasutaja määrata menüüs Shimmeri seadistusele vastava andmete edastamise sageduse ning graafikule kuvatavate andmete ajavahemiku.

4.3 Algoritmi täpsuse analüüsimiseks läbi viidud katseseeria

Lõputöö raames viis autor läbi katseseeria, et testida kirjutatud sammulugemise algoritmi täpsust. Selles peatükis kirjeldab autor katse läbiviimise tingimusi ning analüüsib selle tulemusi.

4.3.1 Katse kirjeldus

Katses osalesid 5 tervet ja noort inimest, kes igaüks pidid ühe katse jooksul astuma 200 sammu. Testija sammusid lugesid kaasa ka kaks vaatlejat, et tagada astutud sammude arvu korrektsus. Kõik testijad pidi katse läbima 3 korda, mille käigus kinnitati kiirendusandur iga kord testijale erinevasse kohta: käe ümber, seljale ehk keha raskuskeskme lähedale ja taskusse. Testijal oli katse läbimiseks kaasa antud ka Androidi seade, mis oli *Bluetooth* kaudu ühendatud kiirendusanduriga. Töö autoril oli katse läbiviimiseks üles seatud sülearvuti, kus jooksis lõputöö raames kirjutatud veebirakendus signaali kuvamiseks ning sammude arvu loendamiseks algoritmi poolt. Androidi seadmel ja sülearvutil oli loodud ühendus internetiga.

Katseseeria alguses kinnitati kiirendusandur testija kehale. Kui kiirendusandur sai stabiilselt kinnitatud, seisis testija paigal ning töö autor fikseeris sammulugemise algoritmi algse sammude arvu. Seejärel anti testijale märguanne, et võib alustada katsega. Kui testija sai astutud umbes 200 sammu, jäi ta seisma. Juhul kui kaks vaatlejat lugesid sama arvu sammusid, arvestati katse loetuks ning töö autor fikseeris algoritmi poolt loetud sammud. Iga testija kordas katset 3 korda, kus andur oli paigutatud erinevasse asukohta ehk kokku viidi katset läbi 15 korda.

4.3.2 Katse tulemuste analüüs

Täpsed katsete tulemused on välja toodud tabelis 1.

Tabel 1. Sammulugemise algoritmi sammude lugemise täpsuse katse tulemused. Astutud sammude lahtrisse on märgitud kontrollijate poolt kokku loetud sammud ning algoritmi poolt loetud sammude lahtrisse on märgitud sammulugemise algoritmi poolt kokku arvutatud sammud. Erinevuse lahtris on nende väärtuste vahe absoluutväärtus.

Kiirendusanduri asukoht	**Testija**	**Astutud sammud**	**Algoritmi poolt loetud sammud**	**Erinevus**	**Keskmine viga 200 sammu kohta (sammudes)**
Käe ümber	1	201	210	9	5,4
	2	202	200	2	
	3	201	200	1	
	4	197	184	13	
	5	201	199	2	
Seljal	1	200	200	0	1,4
	2	201	200	1	
	3	203	201	2	
	4	200	196	4	
	5	209	209	0	
Taskus	1	200	209	9	7,4
	2	201	200	1	
	3	201	197	4	
	4	212	208	4	
	5	207	226	19	

Tulemustest on selgelt näha, et kõige väiksem keskmine viga 200 sammu kohta oli kiirendusanduri paiknemisel testija seljal. Selle põhjenduseks võib olla asjaolu, et

kiirendusandur oli paigutatud testija raskuskeskme lähedale ning see asukoht oli kiirendusandurile kõige stabiilsem. Kõige suurem keskmine viga 200 sammu kohta oli testija taskus, kuna see asukoht on vastupidiselt seljale, kõige ebastabiilsem ning võis esineda sammusarnaseid mürasignaale.

Autori kirjutatud sammulugemise algoritmi saab iseloomustada tundlikkuse ja spetsiifilisuse abil [12]. Tundlikkus kirjeldab kui palju astutud sammudest tuvastatakse algoritmi poolt sammudena ning spetsiifilisus kirjeldab kui palju erinevaid liigutusi tuvastatakse algoritmi poolt sammudena. Vastavalt sooritatud katse tulemustele on antud lõputöös kirjutatud algoritmi puhul tundlikkus hea, kuna sammulugemise algoritm loeb sammud kokku väikese veaga. Samas algoritmi algelisuse tõttu loetakse ka vägapalju mittesammusid sammude alla ning see väljendabki spetsiifilisuse kriteeriumit, mis algoritmi puhul on kehv. Näiteks kui kiirendusanduriga käes lihtsalt viibata kiirelt, loeb algoritm kõik viiped sammudeks, kuigi tegelikult võib isik lihtsalt istuda ning kirjutada paberile.

5 Võimalikud arengusuunad

Antud bakalaureuse lõputööd sobiks väga hästi edasi arendada magistri tööks. Valminud veebirakendusele on võimalik juurde lisada erinevaid funktsionaalsusi, mis täiustaksid rakendust ning annaksid kasutajatele lisavõimalusi. Selles peatükis räägib autor nende arendamise võimalustest.

5.1 Lõputöö raames kirjutatud algoritmi edasi arendamine

Lõputöö käigus läbiviidud katsete seeria kinnitas seda, et autori kirjutatud sammulugemise algoritm on hea täpsusega, kuid võib loetleda ka mittesammusid sammudena. Üks võimalusi, kuidas lõputööd edasi arendada oleks täiustada sammulugemise algoritmi välja filtreerides signaalitipud, mis tegelikult ei ole sammud. Sellise filtreerimise üheks võimaluseks oleks hoida vektorina mälus 3 järjestikust väärtust. Nende väärtuste järjestikusel suurenemisel on võimalus, et signaalitipp on järgnemas. Kui selle vektori esimene väärtus ületab ka müra lävendit, siis salvestatakse see vektor signaalitipu vektorina. Signaalitipule peab järgnema signaali väärtuste vähenemine, täpsemalt vaadeldakse kahte järgnevat väärtust ning kui need vähenevad, siis loetaksegi eelnev signaalitipp sammuks. Hiljem võrreldakse salvestatud signaalitipu vektoriga ka teisi tippe, et kas nende signaali väärtus on salvestatud vektoriga sarnane ning selle alusel filtreeritakse välja signaalitipud, mis ei osutunud tegelikult sammuks. [13]

5.2 Energiakulu arvutamine

Lõputöö käigus valminud rakenduste üks potentsiaalseid lisafunktsionaalsusi oleks kasutaja energiakulu arvutamine. Üks täpsemaid viise inimese energiakulu arvutamiseks on sisse- ja väljahingatava gaasi analüüsimine [14]. Sellegi poolest on tänapäeval väga populaarne viis kalorikulu jälgimiseks nutiseadmete vahendusel. Selle toimimiseks on kirjutatud algoritme, mis töötavad kiirendusanduri signaali analüüsides, täpselt nagu sammude tuvastamise algoritm, mis on kirjutatud antud lõputöö raames.

Lõputöös kirjutatud sammulugemise algoritmi põhimõtted on sarnased energiakulu arvutamise algoritmile. Üks peamisi erinevusi seisneb selles, et energiakulu leidmiseks ei tule esmalt arvutada kolme telje signaali magnituudi. Siiski sarnaselt tuleb rakendada filtreid müra vähendamiseks. Edasi arvutuste juurde minnes, on üheks variandiks võtta integraal iga telje signaalist ning alles siis arvutada kolme telje signaalide integraalist magnituud. Tulemuseks on signaalkuju, mida edasi töödeldes ongi võimalik arvutada energiakulu [14].

5.3 Kasutaja tegevuse tuvastamine

Teine variant lõputöö käigus kirjutatud sammulugemise algoritmi edasi arendamiseks oleks kasutaja tegevuste tuvastamine. See nõuaks põhjalikumat signaali töötlemist ning erinevate uute meetodite rakendamist. Üks võimalusi kasutaja tegevuse tuvastamiseks oleks masinõppe abil [15]. Kasutades CNN (*Convolutional Neural Networks*) klassi on võimalik väikestele andmehulkadele rakendada filtreid, mis salvestavad mällu erinevaid signaali kuju andmete mustreid ning variatsioone. Kasutaja tegevusi jälgides ja õppides ongi masinõppe koodil võimalik tuvastada kasutaja tegevust, näiteks kõndimist, sörkimist, jooksmist.

Antud lõputöö käigus valminud rakendusele oleks lihtsam integreerida juurde järgnevalt kirjeldatud tegevuse tuvastamise meetodit. Nagu ka töös kasutatud sammude tuvastamiseks, on siin tegevuse tuvastamise aluseks võetud kolme telje magnituud [16]. Kasutati samamoodi akna meetodit, ehk korraga vaadeldi ühte puhvritäit signaale. Tegevuse muutumisel määratakse uueks tegevuseks tegevus, mis oli kestnud vähemalt kaks puhvritäit. Tegevusi klassifitseeriti otsustuspuu põhjal. Sarnaselt esimese näitega, on ka praeguse näite puhul vaja mudelit treenida, nimelt peab mudel otsustama, kas tegevus on aktiivne või mitte. Tegevuse teistkordsel töötlemisel otsustatakse edasi, mis tegevust kasutaja täpsemalt teeb. Kui tegevus otsustatakse aktiivseks, peab mudel aru saama, kas kasutaja jalutab, jookseb või sõidab rattaga. Mitteaktiivse tegevuse puhul peab mudel otsustama, kas kasutaja seisab või sõidab autoga. Enne mudelite reaalajas kasutusele võtmist peab neid treenima näidisandmetega.

6 Kokkuvõte

Käesoleva lõputöö üheks põhieesmärgiks oli luua rakendus, mille abil on võimalik näha kiirendusanduri signaalkuju reaalajas. Oluline aspekt on signaalide töötlemisel kokku lugeda ka kasutaja astutud sammud.

Eesmärgi täitmiseks tegi autor kaks eraldiseisvat rakendust, mis suhtlevad omavahel ühise andmebaasi kaudu. Mobiilirakenduse abil kirjutatakse kiirendusanduri kolme telje kiirendussignaalid ja signaali mõõtmise ajahetk andmebaasi. Veebirakenduses kuulatakse andmebaasi muudatusi, kus uue kiirendussignaali lisandumisel minnakse selle töötlemise juurde. Kiirendusanduri signaali töödeldakse kahel erineval meetodil, millest esimene kuvab uue signaalipunkti koordinaadid reaalajas muutuvale graafikule ning teine töötleb signaali vastavalt sammulugemise algoritmile, mille tulemusena arvutatud sammud kuvatakse samuti veebilehekülje.

Autori kirjutatud sammulugemise algoritmi soorituse hindamise testimiseks viidi läbi katseseeria, millest järeldus, et kirjutatud algoritmi iseloomustab kõrge tundlikkus. Lisakatsetuste põhjal selgus, et kahjuks ka madal spetsiifilisus. Kõrge tundlikkus väljendub selles, et algoritm loeb ära kõik sammud, mida kiirendusanduriga testija astub. Madal spetsiifilisus näitab aga seda, et algoritm võib lugeda ka paljud mittesammud sammudena, näiteks kiired viiped kiirendusanduriga.

Teiseks põhieesmärgiks seadis autor antud lõputööga rajada sobiv alustala uurimistööde ja ka näiteks magistritöö edasi arendamiseks, mis sai saavutatud lihtsa sammulugemise algoritmi kirjutamisega. Autori arvates on lõputööd hea edasi arendada näiteks algoritmi madalat spetsiifilisust parandades. Algoritmile on lihtne juurde lisada ka muid funktsionaalsusi, näiteks kasutaja energiakulu arvutamist sammude põhjal ja tegevuse tuvastamist kiirendussignaali põhjal.

Lõputöö käigus valminud mobiili- ja veebirakendus olid autorile isiklikult uudsed ja arendavad. Signaalitöötlusega ei olnud autoril varasemat kokkupuudet, kuid kirjutatud

sammulugemise algoritmi abil õppis selle kohta palju. Autor loodab, et valminud rakendused on edaspidi abiks Tervisetehnoloogiate instituudi poolsele juhendajale.

Kasutatud kirjandus

[1] „Shimmer Sensing,“ Realtime Technologies Ltd, 2017. [Võrgumaterjal]. Available: http://www.shimmersensing.com/images/uploads/docs/Shimmer_User_Manual_rev3p.pdf. [Kasutatud aprill 2018].

[2] „Shimmer Sensing,“ Realtime Technologies Ltd, 2018. [Võrgumaterjal]. Available: http://www.shimmersensing.com/products/shimmer-android-id. [Kasutatud aprill 2018].

[3] „Spring,“ Pivotal Software, Inc, 2018. [Võrgumaterjal]. Available: https://projects.spring.io/spring-boot/ . [Kasutatud aprill 2018].

[4] „Spring,“ Pivotal Sowtware, Inc, 2018. [Võrgumaterjal]. Available: https://docs.spring.io/spring-boot/docs/2.1.0.BUILD-SNAPSHOT/reference/htmlsingle/#boot-documentation-about. [Kasutatud aprill 2018].

[5] „CanvasJS,“ Fenopix, Inc., 2018. [Võrgumaterjal]. Available: https://canvasjs.com/ . [Kasutatud aprill 2018].

[6] „CanvasJS,“ Fenopix, Inc, 2018. [Võrgumaterjal]. Available: https://canvasjs.com/docs/charts/chart-types/html5-line-chart/ . [Kasutatud aprill 2018].

[7] „CanvasJS,“ Fenopix, Inc, 2018. [Võrgumaterjal]. Available: https://canvasjs.com/docs/charts/how-to/creating-dynamic-charts/ . [Kasutatud aprill 2018].

[8] „Firebase,“ Google, Inc, 2018. [Võrgumaterjal]. Available: https://firebase.google.com/docs/database/. [Kasutatud 2018 aprill].

[9] M. Marschollek, M. Goevercin, K.-H. Wolf, B. Song, M. Gietzelt, R. Haux ja E. Steinhagen-Thiessen, „A performance comparison of accelerometry-based step detection algorithms on a large, non-laboratory sample of healthy and mobility-impaired persons,“ IEEE, Vancouver, 2008.

[10] M. Mladenov ja M. Mock, „A Step Counter Service for Java-Enabled Devices Using a Built-In Accelerometer,“ COMSWARE, Dublin, 2009.

[11] „Spring,“ Pivotal Software, Inc, 2018. [Võrgumaterjal]. Available: https://spring.io/guides/gs/serving-web-content/. [Kasutatud mai 2018].

[12] A. J. Saah ja D. R. Hoover, „"Sensitivity" and "Specificity" Reconsidered: The Meaning of These Terms in Analytical and Diagnostic Settings,“ Annals of Internal Medicine, Philadelphia, 1997.

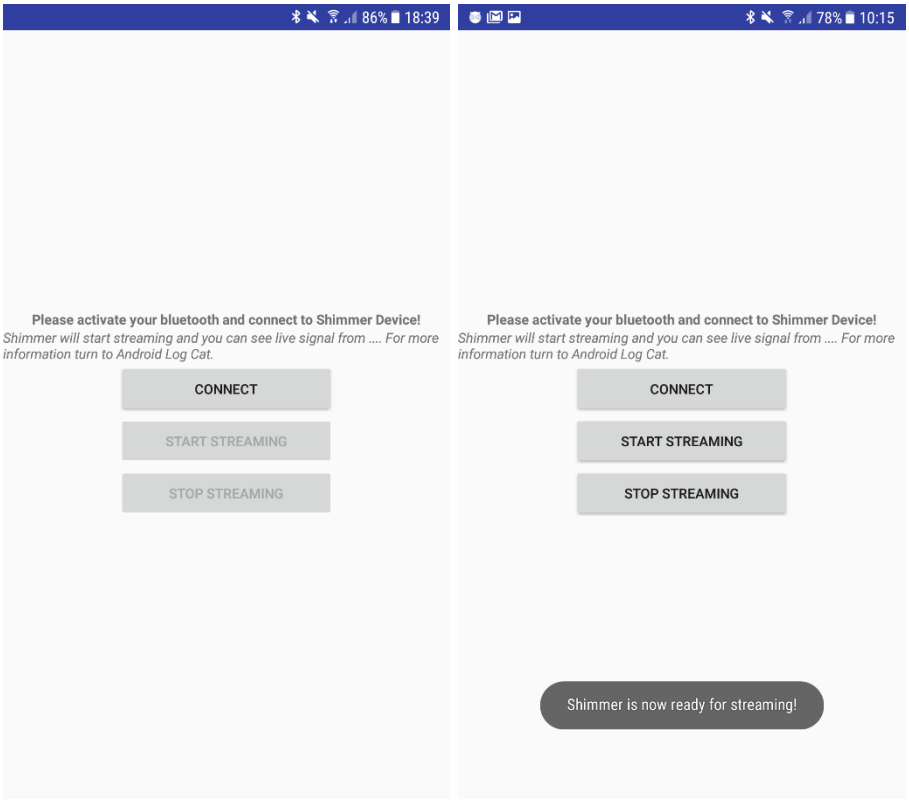
[13] A. Abadleh, E. Al-Hawari, E. Alkafaween ja H. Al-Sawalqah, „Step Detecton Algorithm For Accurate Distance Estimation Using Dynamic Step Length,“ IEEE, Daejeon, 2017.

[14] J. H. Choi, J. Lee, H. T. Hwang, J. P. Kim, J. C. Park ja K. Shin, „Estimation of Activity Energy Expenditure: Accelerometer Appoach,“ IEEE, Shanghai, 2006.

[15] A. Ignatov, „Real-time human activity recognition from accelerometer data using Convolutional Neural Networks,“ Applied Soft Computing, Zurich, 2018.

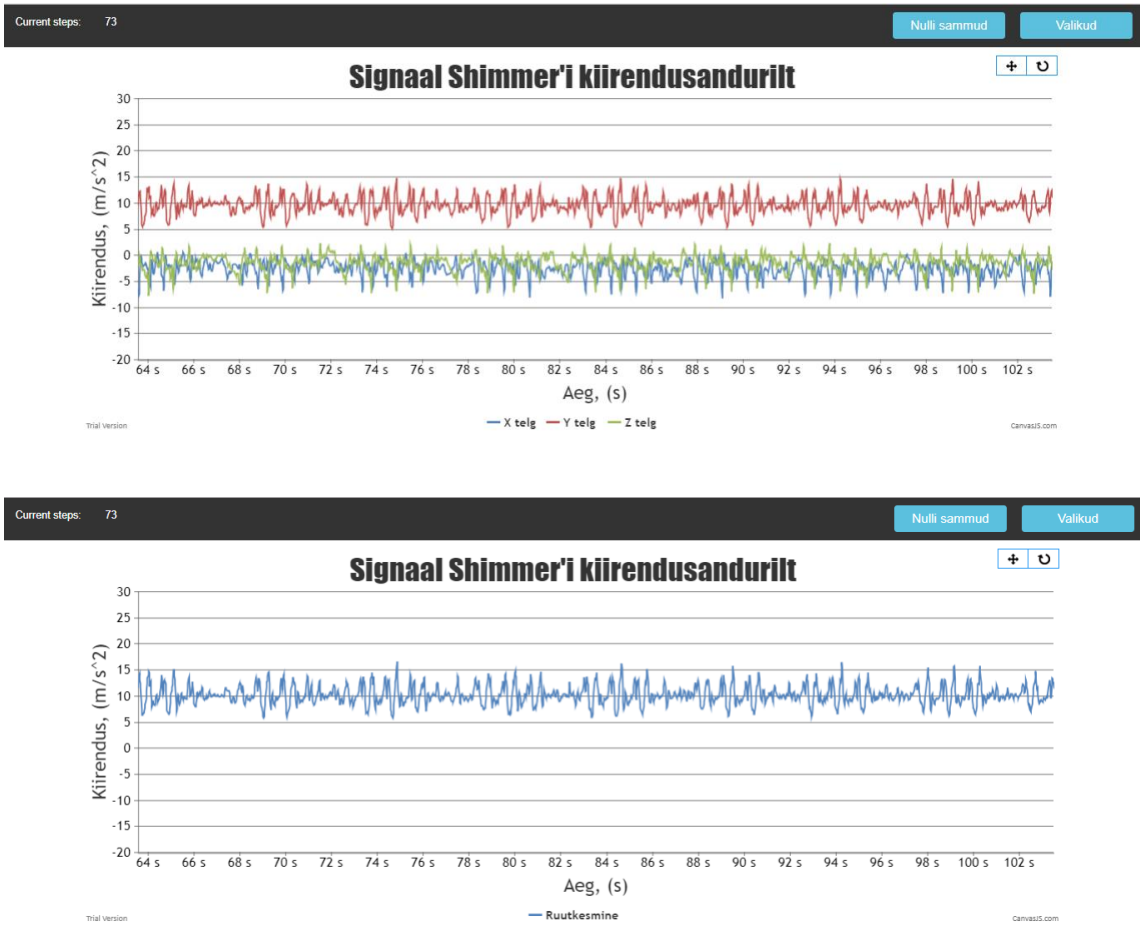
[16] P. Siirtola ja J. Röning, „Recognizing human activities user-independently on smartphone based on accelerometer data,“ IJIMAI, La Rioja, 2012.

Lisa 1 – Mobiilirakenduse ekraanipilt



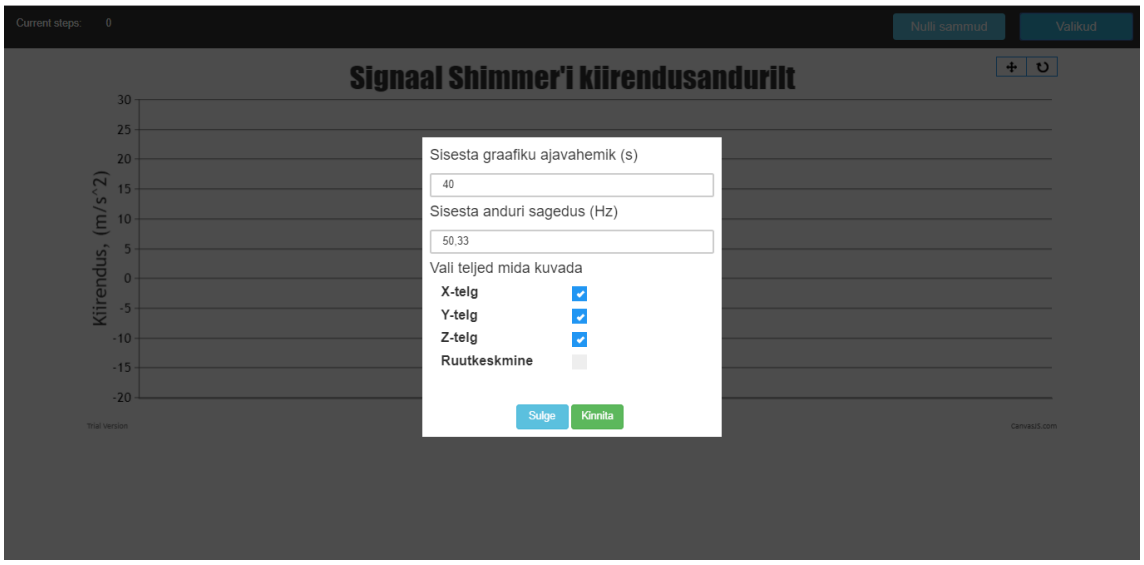
Joonis 9. Lõputöö käigus valminud mobiilirakenduse ekraanipilt. Nupud *Start streaming* ja *Stop streaming* ei ole vasakpoolsel pildil aktiivsed, kuna ühendust Shimmeri anduriga pole loodud. Ühenduse loomisel muutuvad nupud automaatselt aktiivseteks ning Android *Toast*'ga antakse märku, et Shimmeri andur on valmis andmeid edastama.

Lisa 2 – Veebirakenduse ekraanipilt



Joonis 10. Lõputöö käigus valminud veebirakenduse esimene vaade. Ülemisel pildil on graafikule kuvatud kõigi kolme telje suunalised signaalid ja teisel pildil on graafikule kuvatud signaalide ruutkeskmine.

Lisa 3 – Veebirakenduse menüü ekraanipilt



Joonis 11. Veebirakenduse valikute menüü, kus kasutaja saab määrata graafikul kuvatavate signaalipunktide ajavahemiku, Shimmeri kiirendusanduri andmete edastamise sageduse ning graafikul kuvatavad teljed.