TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut
Võrgutarkvara õppetool

Liikumissensorite andmete kuvamine ja selle sünkroniseerimine videoga javaskripti ja HTML5 API abil

Bakalaureusetöö

Üliõpilane:	Vadim Semenov
Üliõpilaskood:	112768IAPB
Juhendaja:	Alar Kuusik
Kaasjuhendaja:	Jaagup Irve

Tallinn
2014

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.

(*kuupäev*) (*allkiri*)

Annotatsioon

Käesoleva bakalaureusetöö eesmärgiks on HTML5 vahendeid kasutades projekteerida rakendust, mis võimaldaks kuvada liikumissensoritest tulevaid sisendandmeid dünaamiliste graafikute viisil ning sünkroniseerida seda videosalvestusega.

Olulisemaid oma töös käsitletavaid probleeme: andmete lugemise failist realiseerimine javaskripti ja HTML5 API abil. Teiseks probleemiks on videosalvestuse sünkroniseerimine graafikuga.

Töö käigus teostatakse rakenduse nõudmiste ja arhitektuuri analüüs, selle tükeldamine mooduliteks ja klassideks; rakenduse iga mooduli loomisega seotud probleemide kirjeldamine ning nende probleemide lahendamiseks API võimaluste leidmine; rakenduse moodulite ja klasside implementeerimine; valmis rakenduse katsetamine ja parandamine.

Töö põhitulemusteks on valmis rakendus, selle kood, rakenduses kasutatavate objektide ja andmeformaadi kirjeldus.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 61 leheküljel, 6 peatükki, 10 joonist, 3 võrrandit, 3 lisa.

Abstract

The main purpose of this thesis is to design an HTML5 API based application that can display input data incoming from motion sensors as dynamic graphs and to find out a way to synchronize this with the video recording.

The most important problems of this work are as follow: reading data from a local file using javascript and HTML5 API. Another one is the problem of the video recording's synchronization with graphs displayed.

During this research the application requirements and architecture are analyzed and this application is splitted into modules and classes. Also the description of problems connected to each application's module is provided and found some API opportunities to solve these problems. Then application classes and modules are implemented and after that the final software was tested and improved.

The main outcome of this work is ready application for motion sensors synchronization, it's code, description of objects and data formats used in the application.

The thesis is in Estonian and contains 61 pages of text, 6 chapters, 10 figures, 3 equations, 3 appendices.

Lühendite ja mõistete sõnastik

Lõuend	***Canvas*** Joonistuslõuend, DOM objekt ja sellele vastav html-element mis võimaldab kuvada pikseli- ja vektorgraafika elemente ja manipuleerida nendega.
Kohatäitja	***Placeholder*** Lõuend, mis kasutatakse FlotCharts teegi poolt graafiku joonistamiseks.
MEMS-sensor	***MEMS-sensor*** Sensor, mis kuulub mikroelektromehhaaniliste süsteemidesse
Faililugeja	***FileReader*** HTML5 objekt, mille atribuutide ja meetodite abil saab töötada lokaalsete failidega

Jooniste nimekiri

Võrrandite nimekiri

Sisukord

1. Sissejuhatus

IT-lahenduste kasutamine spordis on tänapäeval igati õigustatud. Sportlase tegevuste jälgimissüsteemid on selles kontekstis mängivad erilist rolli. Täpsemate mõõtmisvahendite kasutamine selles tegevuses võimaldab sportase tõhusaks teha. Mõõtmisriistvaraga koos arenevad ka neid kontrollivaid tarkvarasüsteeme. Enamik neist on päris kallid kommertstooted, mis on suunatud konkreetsetele spordiliikidele.

1.1 Taust ja probleem

Samas on sportlaste seas on väga teretulnud süsteemid, mis ei nõua keerulist seadistamist, aga seejuures võimaldavad sporditegevuste lihtsamat visualiseerimist.

Liikumissensorite kasutamine sporditreeningute käigus võimaldab treeneril ja sportlasel jälgida seda, kuivõrd efektiivsed on sportlase harjutused. Samas arstid ja treenerid vajavad sellest rakendust, mis oleks:

- kompaktne (oleks lihtne kopeerida ühest arvutist teisele);
- mitmeplatvormiline (et võiks kasutada erinevates OS, erinevates brauserites, nt Android, iPad jne.

Liikumissensoreid koos töös käsitleva rakendusega plaanitakse kasutada Tallinna Ülikooli Terviseteaduste ja Spordi Instituudis.

Arvestades neid vajadusi ELIKO Tehnoloogia Arenduskeskus OÜ, mis asub Tallinnas, otsustas alustada liikumissensori jälgitava sensorsüsteemi arendamist. See keskus tegeleb

1.2 Ülesande püstitus

Selle töö põhieesmärk on **HTML5 vahendeid kasutades projekteerida rakendust, mis võimaldaks kuvada liikumissensoritest tulevaid sisendandmeid dünaamiliste graafikute viisil ning sünkroniseerida seda videosalvestusega**.

Töö ülesanded on järgmised:

- Analüüsida rakenduse nõudeid ja arhitektuuri. Tükeldada kasutajaliides osadeks ja mooduliteks.
- Kirjeldada rakenduse iga mooduli loomisega seotud probleemi ning võimalusi, mida on API-s olemas.
- Implementeerida rakenduse mooduleid.
- Katsetada süsteemi testandmeid kasutades.

Selle eesmärgi saavutamisel tekivad järgmised probleemid:

- Andmete lugemine failist: javaskript kui kasutatav programmeerimiskeel ei võimalda otseselt failidega töötada.

- Videosalvestuse sünkroniseerimine graafikuga: selleks on vaja saada teada, kuidas informatsioon salvestuse ajast salvestatakse mpeg-4 konteineri sisse ning kuidas sisse kirjutada sisendfailisse info salvestusajast.

1.3 Ülevaade tööst

Töö on jaotatud kuueks peatükiks. Esimeses peatükis antakse ülevaade tarkavarasüsteemidest, mis kasutatakse spordis sensorite andmete töötlemiseks. Samuti selles peatükis vaadeldakse töös kasutatud riistvara.

Teises peatükis käsitletakse rakenduse põhiline pragmaatika, nõudeid, kontseptuaalmudel ning arhitektuur, muuseas kirjeldatakse rakenduse põhilised andmeobjektid. Samuti antakse rakenduse programmeerimiskeele ülevaade seoses rakenduse pragmaatikaga.

Kolmandast peatükis räägitakse rakenduse sisendandmetest. Antakse kasutatud sensorite lühikirjeldus ning nende poolt saadavaid andmete sisust (lühidalt). Kuidas andmeid tekitatakse, saadetakse rakendusse, mis viisil ja formaadis, kirjeldatakse rakenduse parser-it, mis teisendab andmeid rakenduse andmeobjektiks.

Neljas peatükk käsitleb seda, kuidas toimub saadud andmete visualiseerimine rakenduses. Käsitletakse javaskripti (jQuery) teegi FlotCharts, mida kasutatakse graafiku kuvamiseks. Kirjeldatakse rakenduse andmeobjekti teisendamine FlotCharts objektiks. Kirjeldatakse ka kõverate haldamine: ajaline uuendamine.

Viiendas peatükis käsitletakse video haldamist rakenduses ning sensorandmete sünkroniseerimise probleem graafikute kuvamisel. Antakse ülevaade erinevate video konteinerite kohta HTML5 ja brauserite puhul. Kirjeldatakse sensorandmete ja video mittesünkroniseerimise põhjusi ja selle lahendamise võimalusi rakenduses.

Viimane peatükk kirjeldab seda, kuidas toimus autori poolt toodetud kasutajaliidese katsetamine ja testimine. Kirjeldatakse testimise käigus tekkinud probleeme ja nende lahendamisviise.

2. Liikumise jälgimise süsteemid ja nende kasutamine spordis

Viimasel ajal on ilmunud palju kommertstooteid, mis võimaldavad jälgida või tuvastada liikumist. Nad luuakse erinevate eesmärkidega ning kasutatakse robootikas, turvaäris, mängude tootmisel, meditsiinis, transpordis jne. Siin peab kohe eristama liikumisdetektoreid, mis tuvastavad liikumise kui fakti, liikumissensoritest, mis tuvastavad ja mõõdavad liikumise erinevaid parameetreid, nagu kiirus, kiirendus, ruumiline orientatsioon jt. Liikumisdetektoritest kõige vanemad on infrapunased (Passive Infrafed, PIR), mis toodetakse al. 50-ndatest aastatest (vt Keller, 2000). Praegu kasutatakse ka mikrolainete alusel töötavaid liikumisdetektoreid, mida võib kasutada ka liikumissensoritena. (Li, 2014)

See töö käsitleb liikumissensoreid. Liikumissensoreid võib omakorda jagada kahte rühma. Esimesse kuuluvad sensoreid, mis jälgivad liikumist ilma füüsilise kontaktita liikuva esemega. Selleks võib kasutada nt mikrolaine liikumissensorit või videokaamerat, mis on kasutusel videomängudes nagu Kinect. Tihti optilisi liikumissensoreid mängudes kombineeritakse mehhaanilistega, nagu PlayStation või Wii. (Greenwald, 2010)

Teise rühma kuuluvad liikumissensorid, mis kasutavad liikumise tuvastamiseks ja mõõtmiseks mehhaanikat. Need on nn MEMS-sensorid (MEMS = Microelectromechanical systems). MEMS sensorite põhimõte on see, et mikroskeemil kombineeritakse elektroonika ning mehhaanika seadmete kasutamist. (MEMS) Nihtianovi ja Luque monograafia kohaselt MEMS sensoreid jagunevad jõutundlikeks, optilisteks, inertsiaalseteks ja survetundlikeks. (Nihtianov, Luque, 2013) Seega MEMS-sensorite abil saab mõõta erinevaid liikumise parameetreid. Käesolevas rakenduses kasutatakse inertsiaalset MEMS-sensorit.

Need sensorid on väga levinud meditsiinis ja spordis. On olemas isegi eraldi seadmete rühm nagu Bio-MEMS, mis kasutatakse meditsiini erinevates valdkondades. Siia kuuluvad nii mõõtmisvahenditena kasutatavad sensorid, kui ka implanteeritatavad MEMS-seadmed, mis on suunatud ravimisele. (Bio-MEMS)

2.1 MEMS-sensorite kasutamine spordis

MEMS-sensorite kasutamine spordis saab üha populaarsemaks ja on täiesti õigustatud. Sportlase ettevalmistamisel treeneril on vaja täpselt teada sportlase liikumise parameetre erinevate harjutuste kaupa. Vanad mõõtmisvahendid siin tihti ei sobi, kuna ei võimalda nõutud tänapäevast täpsust. Sellest lähtuvalt on arusaadavam see, miks tervise ja spordi valdkonnas luuakse palju tarkvarasüsteeme, mis võimaldavad sensorandmeid töödelda ning esitada lõppkasutajale inimtajutaval viisil.

Väga hea spordis kasutatavaid MEMS-sensorite ülevaadet võib leida 2013 a. David DiPaola esitluses (DiPaola, 2013). Ta toob mõni näiteid spordis kasutatavate sensorite ja rakenduste kohta.

Rakendustest, mis ei kasuta otseselt just liikumissensoreid võib mainida nt **Mobile Eye-XG**[1] prillid, mis jälgivad pilgu suunda, fokusseerimispaiga ja soojakaardi. Siin on huvitav see, et sensoriandmeid ühendatakse salvestatud videoga. Kinda sisse sisseehitatud survesensoreid kasutatakse **Tekscan Grip™** süsteemis.

Smart Garment (tark rõivas) – sensoreid integreeritakse spordirõivaste sisse. Liikumissensoreid seal ei ole, mõõdetakse ainult kehatemperatuuri, hingamissageduse, südame elektrilist aktiivsust (EKG). Kasutatakse tensoandurit, mille elektritakistus muutub venitamise puhul.

Goal Line Tracking (GLT) System – elektrikaablid paigaldatakse jalgpalli väljakul, et tuvastada pallisattumist teatud aladesse. Pallis on ka paigaldatud sensor, mis tekitab magnetvälja muudatusi ja saadab sellest sõnumit juhtmevabalt arvutile.

Sensoreid kasutavad ka erinevad fitness-kellad, mis on väga levinud viimastel aastatel, mille ülevaade võib leida Leta Shy artiiklis (Shy, 2014). Lihtsad liikumissensorid on ka sisse ehitatud tänapäeva nutitelefonidesse. Nende seadmete peamiseks puuduseks on see, et nende eraldusvõime on liiga madal ning võendamissagedus kõigub. Seetõttu ei saa neid kasutada professionaalsetes spordirakendustes.

Tänapäeval arenevad lahendused, mis kasutavad erinevaid tüübi andureid korraga. Üks selline süsteem on **Bodymedia Fit**. See on komplekt, mis sisaldab sensorit, mis mõõdab naha juhtivust (skin conductance), temperatuuri, liikumist. Andmevõtu sagedus on ligi 5000 korda minutis (st umbes 80 Hz). See on suunatud ennekõike põletatud kalorite arvutamiseks. Sellega koos tuleb liides, mis koosneb lauaarvuti rakendusest Activity Manager ning rakendus mobiilsete rakenduste jaoks, mis võimaldab vaadata kalorite põletamist peaaegu reaalajas. See on kommertstoode, mis on päris kallis nii individuaalseks, kui ka professionaalseks kasutamiseks.

Liikumissensoreid integreeritakse erinevatele spordiriistadesse. Liikumissensorid kasutatakse nt süsteemis **Nike Hyperdunk+**. Siin kasutatakse 3-teljeline kiirendussensor ja neli tundlikut resistorit, mis paigaldatakse spordijalatsi sisse. Andmed saadetakse Bluetooth kaudu arvutisse, kus andmeid kuvatakse spetsiaalse rakenduse abil. **VELOCITIP Arrow** süsteem ehitab 3-teljelist kiirendusmõõturit ja protsessorit noolte sisse. Korvpallisüsteemis **94Fifty** sensor paigaldatakse palli sisse ning saadab andmeid arvutisse või nutitelefonile. **Movea MotionPod** kasutatakse golfimängijate liikumise analüüsiks. Selles väikeses seadmes nimega liikumispakk (ingl. k. MotionPod) on ühendatud kiirendusmõõtur, güroskoop, magnetomeeter ja protsessor. Liikumispakki paigaldatakse golfikepisse. Sarnane sensor kasutatakse **Play & Connect** süsteemis, mis on toodetud Babolat kompani poolt koostöös Movea-ga. See on liikumispakk, mis paigaldatakse tennisereketi sisse ning mõõdab palli löökrõhu reketikeelte vastu ja reketi trajektoori enne ja pärast löögi. Sarnane mitmefunktsionaalne sensor toodetakse ka **Zepp** poolt. Seda sensorit võib kasutada tennisereketides, golfikepides ning pesapalli kurnikaigades.

[1] Lingid süsteemidele, mis on selles peatükis rõhutatud pakskirjaga vt kasutatud kirjanduse loetelu lõpus.

1-teljelist kiirendusmõõturit pannakse ka jalgratta sisse **Cannondale Simon Active Mountain Bike Suspension** süsteemis, mis mõõdab ratta kiiruse ning rattakahvli rippumise. Kõikides nendes rakendustes kasutatakse ainult üks sensor, video salvestust ei eeldata.

X2Impact Mouth Guard – 3-teljelist kiirendussensoriga ja güroskoobiga sensorit pannakse suukaitse sisse. See võimaldab mõõta löögi sportlase pea vastu mängu jooksul. Andmeid saadetakse juhtmevaba ühenduse kaudu baasjaamasse, mis on tehtud kui pilvandmebaas. Arvutirakenduse abil saab siis saadud andmeid kuvada ja analüüsida.

Veel üks DiPaola esitluses toodud näidetest on **Adidas MiCoach** süsteem. Sellesse süsteemi kuuluvad seade protsessoriga ja sammusensoriga (3-teljeline kiirendusmõõtur), südame löögisageduse mõõtur ning rakendus mobiiltelefoni / tahvelarvuti jaoks. Rakendus kasutab serveritehnoloogiad, mis saab kätte kasutajaandmeid, analüüsib neid ning saadab neid tagasi kasutajale. Uuenduse sagedus on 1 s. See on ka üsna kallis kommertstoode. Video salvestust siin ei kasutata.

XSENS MVN Motion Capture Suit – see on hästi arenenud toode, mis võimaldab luua liikumise 3D-mudelit. See koosneb 17 sensoritest, mis pannakse inimese erinevatele kehaosadele. Andmed nendest saadetakse arvutisse, kus on installitud MVN Studio rakendus. MVN Studio-s andmeid analüüsitakse ning konverteeritakse avatar-i liikumistesse, mida kasutaja võib näha reaalajas (andmeid saadetakse juhtmevaba ühenduse kaudu iga 8 ms). See lahendus oli kasutatud 2014 talveliste olümpiamängude lõputseremoonias.

Riddell InSite – sensorid (5 tk) paigaldatakse sportlase kiivris, mõõdetakse liikumise ulatuse, suunda, ja aega, mille jooksul mängija püsib mänguväljakul (*sustain on the field*). Samuti kasutab süsteem juhtmevaba ühendust tarkvara rakendusega.

Kui rääkida süsteemidest, mis kasutavad videosalvestust ja liikumissensoreid, siis on vaja esile tõsta **STT** poolt loodud rakendusi. See firma teeb erinevaid rakendusi spordi jaoks. Mõnedes kasutatakse liikumise analüüsi jaoks sensoreid (**Golf Studio, Cycling Studio**), teistes kasutatakse videosalvestust (nt **InThePool**, mis analüüsib ujumisvõistlusi, näidates korraga kogu basseini).

Sportlase liikumist võib jälgida ja analüüsida kasutades mobiiltelefoni sisseehitatud liikumissensori ja GPS-anduri. Näidisena võib tuua siin **Rowing in Motion** rakendust, mis on suunatud kõigepealt sõudmissportlastele. Nutitelefonil paigaldatud rakendust saab andmeid liikumise kohta ning saadab neid serverisse, kus neid töödeldakse.

Liikumissensorite andmeid ja videot sünkroniseeritavad rakendused

Sellised süsteemid on kasulikud spordis, sest võimaldavad siduda numbrilisi liikumisandmeid treenerile harjumuspärase visuaalse informatsiooniga.
Selles töös arendatava rakendusega kõige võrreldavaks süsteemiks peame nimetada **CAPTIV-L7000** süsteemi. See on professionaalne vähelevinud süsteem, mis võimaldab reaalajas hinnata sportlase liikumist. Süsteemis kasutatakse spetsiaalarvutit, seega süsteemi

maksumus on eriti suur. Süsteemi eeliseks on see, et see võimaldab sünkroniseerida filmitud videot sensoritest saadud andmetega ja siis detailselt neid andmeid analüüsida graafikute viisil. Samuti süsteem võimaldab kuvada liikumisstatistikat selle kestvuse, sageduse, korduvuse, siirde, sünkroonsuse vaatenurgast.
Sensorid, mis paigaldatakse kehale on ühendatud juhtmevabalt spetsiaalarvutiga – T-Log andmelogijaga (kuni 6 meetrini kaugusel). Sensorid saadavad logijale andmeid inimese ja keskkonna erinevate parameetrite kohta:

- füsioloogilised: elektromüograafia, südamelöögi sagedus, naha juhtivus, puhumine, oksümeetria;
- keskkonna kohta: temperatuur, surve, lendavad orgaanilised ühendid (Organic volatile compounds), tolm, valgus;
- biomeetrilised: jõu, goniomeetria / torsiomeetria, kiirendus, inklinomeetria, orientatsioon.

Meie eesmärgiks on realiseerida sarnane süsteem, mis oleks odavam ning lihtsam kasutaja jaoks.

2.2 Kasutatud seadmed

Antud töös arendatav süsteem põhineb ELIKO-s välja töötatud riistvaral. Seega autoril oli püstitatud ülesanne kasutades teatud liikumissensorit projekteerida rakendust, mis võimaldaks nendest sensorist andmeid visualiseerida.

2.2.1 MEMS-sensor

Käsitleva rakenduse poolt kasutatav sensori mikroskeem MPU6050 on toodetud Invensense poolt. Selle tundlikkus on 0,2 ms joonkiirendus ja 0,05 kraadi/s nurkkiirus, 16 bitti eraldusvõime, kuni 1000 võendit sekundis. (InvenSense Inc., 2013)

Võendamissageduse vaatenurgast kasutatavad sensorid võivad töötada sagedustel kuni 1000 Hz. Antud rakenduse nõuete vaatenurgast on optimaalne neid kasutada 100 Hz ja 50 Hz sagedustel.
MPU6050 sisaldab tegelikult 2 mõõtmisseadmet sees:

1) 3-teljeline (3-axis) MEMS güroskoop;
2) 3-teljeline (3-axis) MEMS kiirendusmõõtur

Tihti see sensor nimetatakse 6-teljeliseks, sellepärast, et selles on ühendatud kaks 3-teljelist sensorit. Seega selle poolt mõõdetavaid parameetre on 6:

1) kiirendus X-teljel;
2) kiirendus Y-teljel;
3) kiirendus Z-teljel;
4) nurgamuutus X-teljel;
5) nurgamuutus Y-teljel;
6) nurgamuutus Z-teljel.

Arvutiga neid sensoreid ühendatakse USB kontsentraatori abil. Sensori üldine vaade võib näha Joonis 1 ja selle paigaldamine kehaosal võib näha Joonis 2.

Joonis 1. Sensor MPU6050

Joonis 2. Sensori paigaldamine kehaosadel

2.2.2 Videokaamera

Käesolevas rakenduses kasutatakse tavalist veebikaamerat, mis võimaldab videosalvestust 25-60 fps kaadrisagedusega. Veebikaamera ühendatakse arvutiga usb-pordi kaudu ning juhitakse juhtimistarkvara poolt (st kasutaja ei pea selle eraldi sisse lülitama). Camcorder-ite kasutamine videokaamerana selles rakenduses ei ole realiseeritud, kuigi selle arendamine võiks olla rakenduse perspektiiviks, samas aga see kuulub pigem juhtimistarkvara skoobi.

3. Rakenduse üldine kirjeldus

Järgnevalt kirjeldatakse rakendust, kogu rakenduse ülesandeid ning selle üldist arhitektuuri. Ennekõike on vaja teha märkust selle kohta, et kui edaspidi räägitakse javaskripti klassidest, siis peab pidada silmas, et see mõiste on tinglik, sest javaskript ei käsitle klasse. Põhjalikumalt sellest vt p.3.4.

Liikumissensorite kasutamise stsenaarium. Kogutud nõuetest ja kasutusjuhtudest lähtuvalt sensorite andmete kogumine ja video salvestamine toimub järgmiselt:

1. Sportlane paneb enda kulge (erinevatele kehaosadele) liikumissensoreid.
2. Treener käivitab juhtimistarkvara, mis paneb videokaamerat käima.
3. Sportlane (või treener) paneb sensoreid käima (samme 2 ja 3 võib teostada suvalises järjekorras).
4. Pärast liikumissensorite sisselülitamist nende abil üle kantakse sportlase kehaosade liikumise erinevaid parameetreid: kiirendus, nurga muutmine jms. Andmed salvestatakse csv-faili.
5. Sportlase harjutusi salvestatakse veebikaamera abil videofaili.
6. Pärast harjutuse lõppemist sportlane / treener lülitab liikumissensoreid välja.
7. Treener peatab juhtimistarkvara (samme 6 ja 7 võib teostada suvalises järjekorras).
8. Videosalvestust ja sensorandmeid haldab back-endi rakendus, mis paneb kinni salvestatud csv-faili ning konverteerib salvestatud videovoolu sobivasse konteinerisse.
9. Juhtimistarkvara töö tulemuseks on üks videofail ning üks sensoriandmetega csv-fail.

Juhtimistarkvara on kirjutatud C keeles, ning see käivitatakse opsüsteemis kui desktop-rakendus.

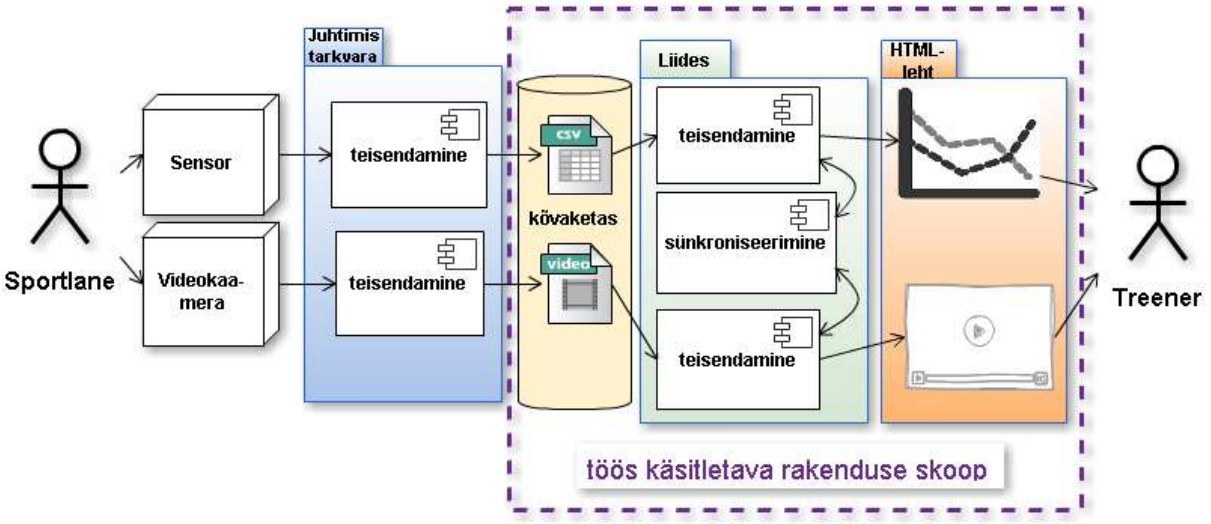
Sensorite andmete ja video sünkroniseeritud kuvamine toimub järgmiselt:

1. Treener avab veebilehitsejas rakenduse algusfaili *index.html*.
2. Brauser kuvab liidese, kus on võimalik valida videofail ning andmetega csv-faili.
3. Treener valib vastava videofaili.
4. Treener valib vastava andmefaili (sammude 3 ja 4 järjekord on suvaline).
5. Liidese vasakul pool tekkib videoaken, liidese paremal pool tekib graafik.
6. Treener paneb video käima videoakna all oleva start-nupu abil või mängija sisseehitatud start-nupu abil.
7. Graafiku aknas kuvatakse vastavaid andmeid.
8. Videofaili võib sirvida tõmmates mängija sisseehitatud sirvimisriba kasutades.
9. Videofaili võib sirvida kaaderhaaval liidese vastavatele nuppudele klõpsates, kaadrite arvu võib sätestada vastavas sisend-aknas.
10. Videofaili võib sirvida tõmmates graafiku kõveraid (kui on sisse lülitatud kõverate kuvamise režiim).
11. Treener lõpetab töö antud failidega valides teisi video- ja andmefaile.
12. Treener lõpetab töö rakendusega sulgedes veebilehitsejat.

Käesoleva töö põhiskoobiks on kasutajaliidese[2] osa, mille nõudeid käsitletakse järgmises punktis.

3.1 Töös realiseeritava tarkvara nõuded

Kogu tarkvara lahendus koosneb kahest osast (vt Joonis 3), millest esimene on sensorite ja kaamera juhtimistarkvara (joonisel vasakul). See oli realiseeritud enne käesolevat tööd. Käesolevas töös kirjeldatakse autori poolt realiseeritud kasutajaliidest (joonisel paremal).



Joonis 3. Süsteemi üldine skeem

Pärast seda, kui harjutus on lõppenud ning video ja sensorandmete fail on salvestatud, tekib vajadus seda faili läbi vaadata ning samal ajal analüüsida erinevate parameetrite muutmist visualiseeritud graafiku viisil: kas kõveratena või tulpadena.

Rakendus peab võimaldama järgmisi tegevusi:

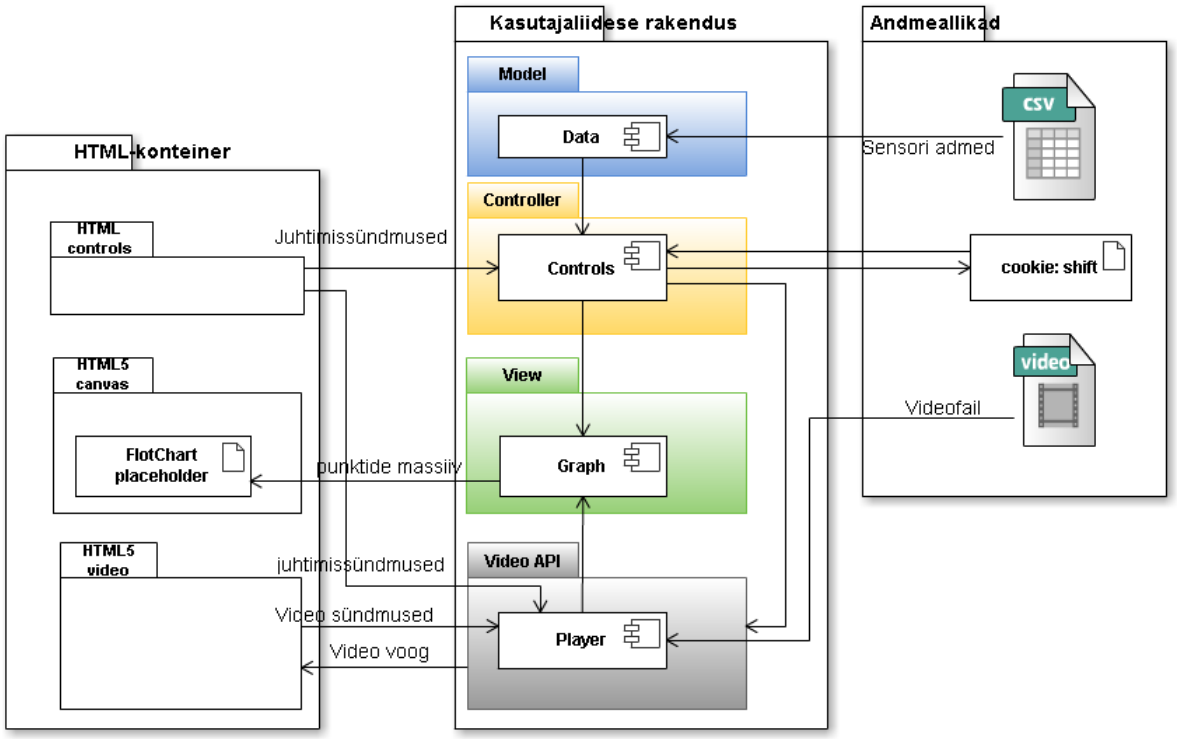
- Videofaili valik mängimiseks.
- Videofaili mängimine, peatamine ja mängimise lõpetamine
- Video sünkroniseeritud liigutamine ja peatamine koos graafikuga.
- Videofaili sirvimine kaaderhaaval (kaadri arvu sirvimiseks saab seadistada)
- Andmefaili valik graafiku kuvamiseks
- Andmefaili teisendamine graafikuks
- Graafiku viisi valik (kas tulbad või kõverad)
- Sensorite valik kuvamiseks
- Sensori parameetri valik selle kuvamiseks graafikul
- Graafiku skaleerimine X-teljel ja Y-teljel
- Graafiku liigutamine X-teljel ja Y-teljel
- Video sünkroonne liigutamine koos graafiku liigutamisega X-teljel mõlemas suunas

[2] Kasutajaliidese mõiste antud töös on tinglik, sest autori poolt tehtud rakendus realiseerib ka teatud äriloogikat.

3.2 Kasutajaliidese üldine ülesehitus

Liidese realiseerimiseks kasutatakse dünaamiline HTML-leht, ilma serverita. Kuna rakendus võtab andmeid andmefailist, ilma serverita, seega klient-serveri tüübi arhitektuuri pole kasutatud. Samal ajal rakenduse jaoks oli valitud ikka veebibrauser kui platvorm. Selle eelistamiseks on mitu põhjust.

- Veebilehitseja kasutamine oli tellija ja tarkvarafirma nõue. Seega töö autoril oli brauseri kasutamine tehniliseks ülesandeks
- Rakenduse mitmeplatvormilisus: see ei pea sõltuma konkreetsest opsüsteemist.
- See võimaldab ka rakenduse skaleerimist võrguühenduste kasutamise puhul.



Joonis 4. Kasutajaliidese ülesehitus

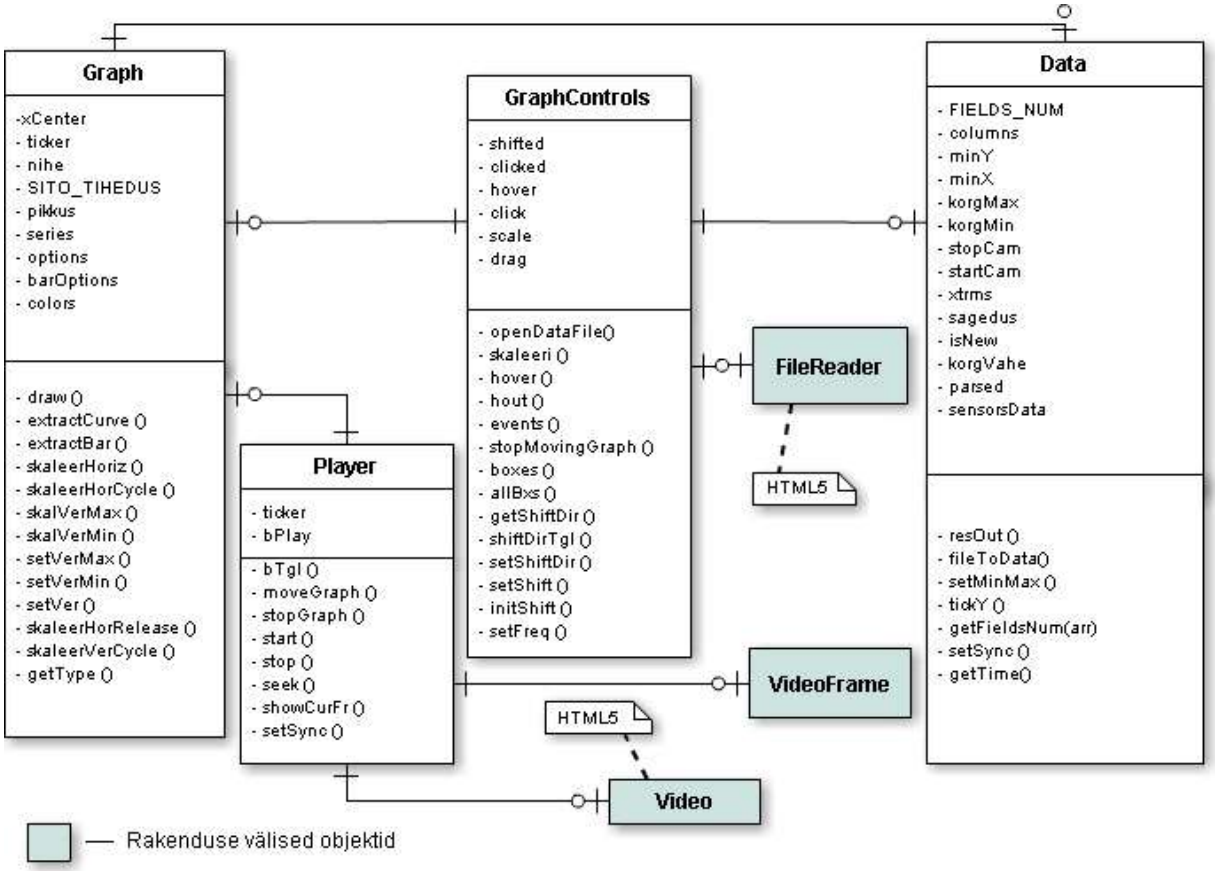
Arhitektuuris on osaliselt arvestatud MVC (Model-View-Controller) model. MVC on välja töötatud Martin Fowleri poolt (vt. Fowler, 2014a) ning eeldab rakenduse tükeldamist kolmele kihudele:

1. Mudel – tarkvara osa, mis teisendab sisendandmeid rakenduse sobivaks viisiks. Antud rakenduse skoobis objekt *Data*, mis teisendab andmeid CSV failist objektiks.
2. Kontroller – tarkvara osa, mis töötleb sündmusi. Käesolevas rakenduses seda ülesannet täidab *GraphControls* objekt, mis töötleb nt videofaili sirvimist, video mängimise algust jt sündmusi, sidudes oma vahel mudeli ja vaadet.
3. Vaade – tarkvara osa, mis genereerib lõpliku veebilehe. Need ülesanded kasutajaliidese rakenduses täidab *Graph* objekt, mille raames toimub graafiku andmete saatmine *FlotCharts API* abil HTML

Aga samas osaliselt antud rakendus ei klapi MVC mustriga, kuna antud rakenduse komponendid täidavad erinevaid ülesandeid (sensoriandmete töötlemine, video mängimine, nende sünkroniseerimine). Nimelt töö videoga realiseeritakse kõigepealt HTML5 Video API abil, seega selle jagunemine MVC-mustriks pole õigustatud, sest põhiline videofaili juhtimisloogika on juba realiseeritud HTML5 video API-s.

3.3 Rakenduse kontseptuaalne skeem

Vaatame kasutajaliidese rakenduses loodud javaskripti erinevate klassi objektide kontseptuaalse skeemi (vt Joonis 5). Selle skeemi järgi võib näha, et rakenduse käivitamisel kõigepealt luuakse objekt, mis kuulub klassi *Data* (andmete mudel), see objekt on sõltumatu teiste objektide olemasolust. Siis luuakse *Graph* (andmete kuva) klassi objekt, mis on seotud ühe *Data* objektiga. Kui on olemas *Graph* objekt, siis võib luua sellega seotud *Player* klassi objekti. Kõige viimasena luuakse objekt *GraphControls* (kontroller), mis on seotud ühelt poolt mudeliga (*Data*), teiselt poolt aga kuvaga (*Graph*).



Joonis 5. Rakenduse kontseptuaalne skeem

Üks märkus, mis puudutab paljude antud rakenduse objekte. Iga klassi sees luuakse üks sisemine muutuja *_this*, kuhu pannakse viide sellele samale objektile. Selle muutuja kasutamiseks tavalise *this* indeksi asemele põhjuseks on see, et javaskriptil on üks iseärasus. Nimelt see, et *this* väärtus ei ole staatiline, mis omistatakse objekti loomisel. See

väärtustatakse funktsiooni kutsumise ajal ja sellepärastki sõltub kontekstist. Seega, kui objekti meetod kutsutakse mingi liidese sündmusega või kontekst muutub mingil teisel moel (nt uue Interval klassi objekti loomisel), siis *this* väärtustatakse selle kutsutava objekti viidega.

Vaatame järgmist fragmenti:

```
1       var Graph = function(data){
2               var _this = this;
        ...
98              this.skaleerHorCycle = function(x){
99                              this.ticker= setInterval(function (){
100                                             _this.skaleerHoriz(x);
101                                     }, 100);
102             }
        ...
144     }
```

Kui pöörduda 100. reas tavalise *this* objektile siis selle viideks pannakse *Window* klassi objekt, sest funktsioon *setInterval()* kutsutakse *Windows* klassi kontekstis. Järgnevalt kui püütakse välja kutsuda selle klassi *.skaleerHoriz()* meetodit, siis interpretaator tekitab veateadet, sest Windows klassil sellist meetodit ei ole. Et seda „polüseemiat“ vältida autor kasutab surrogaatmuutujad *_this*, mis luuakse staatiliselt selle klassi sees. See probleem on tingitud sellise javaskripti kui skriptimiskeele omaduse poolt nagu prototüüp-orienteeritavus.
Seda, milliseid objektid luuakse käesolevas kasutajaliidese rakenduses, millistele klassidele nad kuuluvad, nende atribuudid ja meetodid võib leida Lisas 3.

Välisobjektid, mis on kasutatud :

- *FileReader* (Faililugeja) – HTML5 API kaudu. Kasutatakse faili lugemiseks (selle API kohta vt p.4.2).
- *Video* – HTML5 API kaudu.
- *VideoFrame* – kolmanda poole teek, mis võimaldab tükeldada videofaili kaadriteks ning sirvida video kaaderhaaval.

3.4 Javaskripti iseärasused ja HTML5 võimalused Javaskripti puuduste kõrvaldamiseks

Javaskript oli loodud 1995. aastal firmas Netscape. Javaskriptil kui skriptimiskeelel on oluline rõhutada järgmisi tunnusi:

1. Javascript on ECMAScript-i dialekt ning selles osas vastab ECMAScript-i standardile.
2. See käib veebilehitseja kontekstis, seega see on suhteliselt sõltumatu opsüsteemist, pigem sõltub brauseri APIst.
3. See käib kliendi poolel, iseenesest see ei nõua serveri käivitamist ja sellega ühendust.
4. See on skriptimiskeel, mille lähtekood on igati kättesaadav. Hetkel kõige tuntumates brauserites kasutatakse JIT meetodit (*Just-in-time compilation*). Mozillas on selle meetodi realisatsiooniks JaegerMonkey (JaegerMonkey 2010), mis lisandub selle

brauseri javaskripti mootorile (SpiderMonkey 2014). Google Chrome-is see meetod on ka realiseeritud selle v8 javaskripti mootori raames (Wilson 2012).

5. See on objektorienteeritud keel.
6. Javaskriptis puuduvad klassid. Selle asemel pärimine toimub prototüüpobjektide abil. Tegelikult javaskriptis on olemas kõik vajalikud mehhanismid klassi loomiseks (vt. Introduction to Object-Oriented JavaScript, 2014)[3]. Antud rakenduses selleks kasutatakse konstruktorfunktsioone, mis asendavad klasside kirjeldused teistes objektorienteeritud keeltes.
7. See on dünaamilise tüüpimisega keel. Muutujate tüübid omistatakse programmi käivitamisel ning nad võivad muuta programmi käigus. Sellest lähtuvalt arendaja peab kogu aeg ise kontrollida muutuja tüübi – kas see ei muutunud.
8. Javaskriptil iseeneselt ei ole stdin-i ega stdout-i: see sõltub kontekstis, milles see käivitatakse ning on selle kontekstiga piiratud.

Viimasest iseärasusest lähtuvalt tekstifailide haldamine javaskriptis on natuke keerulisem. "Puhas" javaskript ei võimalda failidega tööd.

Üldlevinud võimalus failide lugemise jaoks on *XmlHttpRequest* objekti kasutamine (nn AJAX). Selle abil võib lugeda andmeid HTTP protokolli kaudu. See aga eeldab seda, et rakendus pöördub käivitatud serveri poole — kas kaudselt või lokaalserveri poole, mis antud rakenduse jaoks ei sobi, sest selle arhitektuur välistab klient – server mudeli kasutamist.

HTML4 standardis puudus API failidega tööks, see põhjustas erinevate jQuery teekide kasutamist.

HTML5 on tuumikkeel, mille struktuur oli võrreldes HTML4.01 versiooniga täiendatud, kohati lihtsustatud ning mille API oli rikastatud, mis võimaldas luua keerulisemaid veebirakendusi. Eriti oli arenenud see API multimeedia suunas: nimelt töö videoga, audioga, geolokatsiooniga jms arvelt.

Probleem on selles, et HTML5 siiani ei ole standartiseeritud. Hetkel kehtiv W3C versioon on W3C Candidate Recommendation (W3C, 2013a); oodatakse, et W3C Recommendation toimub 2014 aastal. Hetkel viimase versiooni mustand on ka avalikustatud (W3C, 2014).

Praeguse API võimalustest on antud rakendusel vajalikud järgmiseid:

1. HTML5 API video tööks. HTML4 standardis ei olnud video sisseehitamise html-lehele võimalusi. Selle puudujäägi kõrvaldamiseks kasutati erinevaid võimalusi (shokwave flash plugin-id, operatsioonsüsteemide standardsete meediamängijate sisseehitamine jne. HTML5 võimaldas brauserile iseseisvalt videofaile käsitleda, kuigi mõned standartiseerimisega seotud küsimused on jäänud. Üksikasjalikult neid probleeme vt peatükis 6.
2. Lõuend kui element, mis on kirjeldatud <*canvas*> tag-i abil. See element on ilmunud HTML5 API-s ning võimaldab otseselt kuvada graafikaelemente kasutades

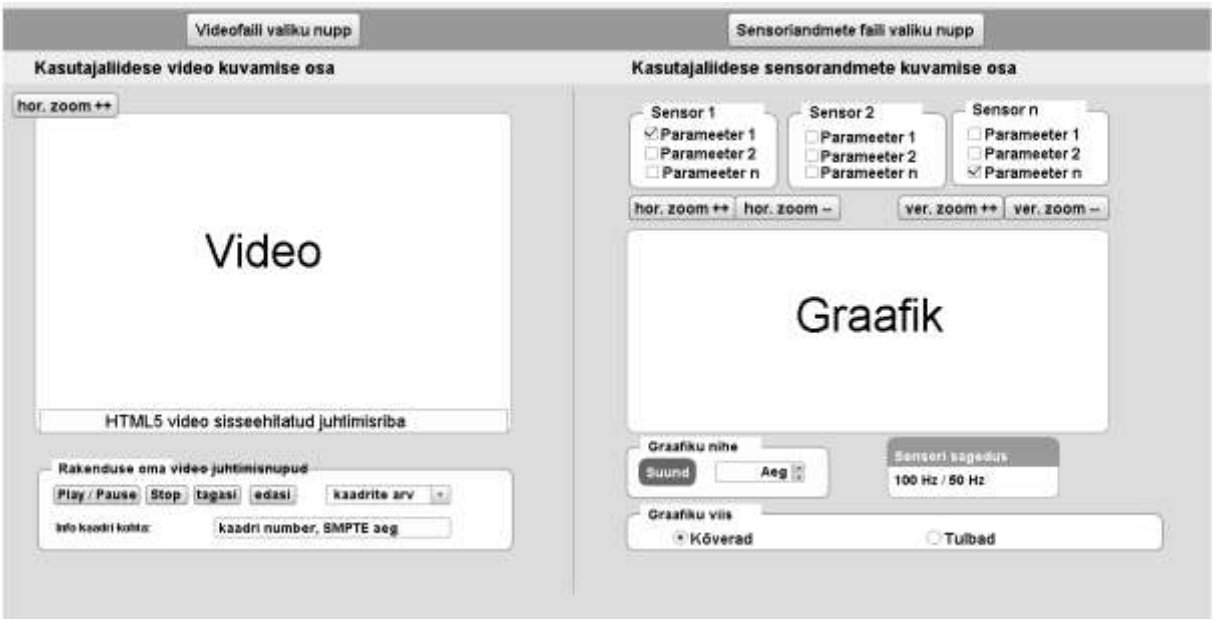
[3] Vt siin ka väga üksikasjalik selgitus prototüüp-orienteeritud programmeerimisele javaskripti puhul (Walsh, 2013). Muuseas Walsh pakub Javascript-i definitsiooni kui „ainult objektide" (objects-only) programmeerimiskeelt. Ta mainib sellele, et javaskriptis pärimine toimub mitte staatiliselt, vaid dünaamiliselt, seega iga klassi eksemplaar on tegelikult iseseisev objekt (hiline seostamine, late binding). Sellega on seotud probleem, kuidas klassi eksemplaris viidata selle samale eksemplaarile. Probleem on kirjeldatud p.3.3.

skriptimiskeelt. Antud rakenduses see element luuakse kasutatud FlotCharts jQuery teegi poolt. Graafiku joonistamine toimub selle elemendi vahendusel.

3. Lokaalsete failide lugemine. See võimalus on täiesti uus HTML5 API-s ning ilma selleta on antud kasutajaliidese rakenduse funktsionaalsus ei saa hakkama.

3.5 Kasutajaliidese prototüüp

Kasutajaliidese üldist prototüübi võib näha joonisel (vt Joonis 6). Nõuete kohaselt, kasutajaliides koosneb kahest poolest: vasakpoolne sisaldab videoakent koos juhtimisnuppudega, parempoolne sisaldab ühest või mitmest sensoritest saabuvaid signaaligraafikuid koos nende graafikute juhtimisvahenditega.



Joonis 6. Kasutajaliidese prototüüp

4. Liikumissensorite andmete import

Järgnevalt käsitletakse seda, kuidas rakenduses toimub andmete kättesaamine failist ja nende teisendamine andmeobjektiks, mida kasutatakse graafikute moodustamisel. Kõigepealt juttu on CSV formaadist, mis kasutatakse rakenduses andmete hoidmiseks, siis faililugeja objektist, mis on HTML5 API osa. Peatüki põhiosa on pühendatud andmete lugemise realiseerimisele liidese rakenduses.

4.1 Algandmete sisu ja formaat

Järgnevalt anname ülevaadet töös kasutatavast CSV formaadist ja selle erinevustest CSV standardist.

4.1.1 CSV standardi kirjeldus

CSV formaat (*comma-separated values*, komaga eraldatud väärtused) on laialt levinud formaat äri- ja teadusrakendustes andmevahetuse jaoks. See on avalik formaat. Praegu kehtib 2013 aasta standart (Hudson, 2013).
Selle formaadi kõige oluliseks eeliseks on struktuuri lihtsus ning seega ka lihtne parsimine ning andmete töötlemine. Sellest lähtuvalt rakenduses on kiirem andmetöötlus ning faili suurus väiksem kui näiteks XML ja JSON puhul.
Meie rakenduses kasutatav CSV fail põhijoontes järgneb CSV-1203 reeglitele (vt Hudson, 2013: 8-18, 21) .
Faili tasemel selle laiendus on *.csv*, selle tähestik on Windows-1252 (ANSI), failis kasutatavad kontrollmärgid on ainult rea(kirje)vahed ning faililõpud, fail ei tohi olla tühi.
Kirje tase. Üks kirje koosneb kirje sisust (*record payload*) ning kirje lõpust (*end of record, EOR*), iga kirje sisu suurus on rohkem kui üks.
Välja tase. Iga kirje väli ei ole fikseeritud suurusega. Iga kirje sisaldab sama arvu väljasid kui eelmine. Iga kirje sisaldab vähemalt kaks väljasid, mis on komaga eraldatud. Iga väli on eraldatud teisest ainult ühe märgiga. Väli ei tohi sisaldada tühikuid alguses ega lõpus. Selle kohta antud rakenduses on erandid, mida kirjeldatakse allpool, kui räägitakse video sünkroniseerimise ridade kohta
Väljade eraldaja tase (*Field separator*). Eraldaja on üks märk, see on seadistatud juhtimistarkvara poolt faili algusest alates ning see kestab kogu faili ulatuses. See märk peab olema koma (siin tuleb ka üks erand allpool). Kasutajaliidese rakendus peab eraldajana vastu võtma koma, tabulatsiooni märku, püstkriipsu (*vertical bar, pipe*, |) või semikooloni. Eraldaja märk ei tohi esineda kui välja sisuline osa.
Kirje lõpu tase (*EOR*). Rakenduse reavahe jaoks kasutatakse CR+LF kombinatsiooni (\r\n), see EOR kasutatakse pidevalt kogu faili ulatuses, EOR ei tohi esineda kui mõne välja sisuline osa.

Faili lõpu tase (*EOF*). Kui failis tekib faili lõpu märk, siis see käsitletakse kui faili lõpp. EOF-i ei saa kaitsta jutumärkidega. EOF ei tohi esineda kui mõne välja sisuline osa.
Faili pealkirja tase (*header*). Pealkiri on faili esimene kirje. See osutub ainult üks kord failis. Väljade märgised ei tohi olla tühjad. Märgistes ei või olla muutuvaid andmeid. Tavaliselt märgis sisaldab vihjet sellele, mis moodi info selles väljas on.
Muide standardis on kirjeldatud mõned nüanssid, mis rakenduses ei ole pidevalt kehtivad, kuigi see soovitatakse teha standardi poolt. Nimelt rakenduses märgistes kasutatakse tühikuid. See on seotud sellega, et rakendus kasutab neid märgiseid kõverate nimetuste kuvamiseks ning inimloetavuse eesmärkides ei asendata tühikuid allkriipsudega.
Andmete tase (*Data Records*). Failis võib olla null või rohkem andmeid. Tüübi identifikaatoreid siin ei kasutata.
Andmete sisu kaitse tase (*Field Payload Protection*). Komad, üksikud jutumärgid, reavahemärgid rakenduse andmete sisus ei kasutata (standardi järgi nad peavad olema kaitstud paarjutumärkidega). Üldiselt paarjutumärkidega andmete kaitsmist siin ei kasutata.
Excel kaitse tase antud rakenduse jaoks ei ole oluline, sest siin ei kasutata Exceli andmete kuvamise ega interpreteerimise jaoks.

4.1.2 Töös kasutatava CSV-formaadi täpsustamine

Andmete semantika
Andmed pealkirjas on loetletakse järgnevalt:

1. *absolute timestamp* — siia salvestatakse kellaaeg süsteemist (millisekundites), millega on liikumissensorid ühendatud. Selle absoluutse kellaaja poolt identifitseeritakse iga kirje – see on unikaalne kogu faili ulatuses.
2. *node id* — see on sensori identifikaator.
3. *Acc.x, Acc.y,Acc.z,Gyr.x,Gyr.y,Gyr.z* — selle sensori abil mõõdetavad parameetrid

Kuna arvutiga võib olla samal hetkel ühendatud üks või rohkem sensoreid, siis punktid 2. ja 3. võivad korduda.
Järgnevalt on faili pealkirja näide:

```
absolute timestamp,node id,Acc.x, Acc.y,Acc.z,Gyr.x,Gyr.y,Gyr.z,node id,Acc.x,Acc.y,Acc.z,Gyr.x,Gyr.y,Gyr.z
```

Järgnevalt on faili regulaarse kirje näide:

```
1377090933159,1,498,76,-9406,-180,-63,-207,2,376,170,-7276,-259,-185,-59
```

Formaadi erinevused CSV standardiga võrreldes rakenduse kontekstis
1. CSV üks oluline iseärasus, mis võib olla andmevahetuse takistuseks on andmete homogeensus. Kui on vaja edastada hierarhilisi ja / või heterogeenseid andmeid, siis see formaat ei sobi. Kuna rakenduses kasutatavad andmed on osaliselt hierarhilised, siis neid tuleb juhtimistarkvara tasemel lineaarseks teha. Rakenduses võib olla 1 või rohkem sensoreid, igaühel on omakorda võib olla üks või mitu parameetrit. Selleks pealkirjas kasutatakse väli *node id*. Kõik parameetrite väljad, mis selle väljale järgnevad (kuni järgmise *node id*) väljani kuuluvad siis sellele sensorile. Siis järgneb järgmine *node_id* väli, ja seejärel järgmise sensori parameetrid ja nii edasi.

2. Veel üks CSV-1204 standardist kõrvalekaldumine on see, et failis tekib kaks rida, mis ei kirjelda sensori andmeid, vaid kajastab video salvestamise algus- ning lõpuaega. Need read on vajalikud video ja sensorandmete sünkroniseerimise jaoks. Nende reade formaat on järgmine:

- alguses on ’@’ märk, mis näitab parserile, et tegemist on mitte regulaarkirjega (sensori oma) vaid video sünki kirjega.
- Siis tuleb täht ‘s’, kui tegemist on video salvestamise algusajaga või täht ‘e’ video salvestamise lõpuaja puhul.
- Pärast seda veel üks ‘@’ märk, mis tähendab, et järgnevalt on ajatempel.

Ajatempel võetakse arvutist juhtimistarkvara poolt ning pannakse kohe pärast seda kahe sensorandmete ridade vahele. Sünkroniseerimisega tekitavaid küsimusi käsitletakse peatükis 6.
Järgnevalt on video sünkroniseerimise kirjete näiteid:

```
@s@1377090933227
@e@1377090933687
```

Neid kirjed kontekstis võib vaadata Lisa 1. Algandmete näide (vt 6. ja 30. read). See iseärasus tekitab seda, et rakenduses kasutatav csv-fail ei vasta formaadile välja tasemel (vähemalt kaks ridu, kus väljade arv ei ole sama, mis kõikidel ülejäänutel ridadel), väljade eraldaja tasemel (koma ei kehti eraldajana kogu faili ulatuses). Lisaks siin toimub andmete homogeensuse printsiibi rikkumine.
Selle lahenduseks võiks olla video sünkroniseerimise andmete eristamine eraldi failisse. Aga sel lahendusel on ka oma miinuseid. Kõige suurem probleem on seotud HTML5 Faililugeja API realiseerimisega – nimelt see, et ta ei võimalda programmeeritult viidata failile, mida on vaja avada. Üksikasjalikult sellest vt järgmises punktis. See piirang põhjustab seda, et kasutaja peab ise valida iga fail, mida kasutatakse lugemiseks rakenduse poolt. Kui tehakse video sünkroniseerimise eraldi failis, siis see mõjutab halvasti rakenduse kasutatavuse, sest peale videofaili otsimist ning andmefaili otsimist kasutajal tuleb ka valida video sünkroniseerimise faili.
Sellest asjaolust lähtuvalt oli otsustatud, et csv-faili homogeensuse deformatsioon selle rakenduse kontekstis on odavam kui kasutatavuse keerustamine.

4.2 Faililugeja API

Enne faili parsimist rakendus peab olema võimeline lähteandmetega faili kätte saama.
Kuna javaskriptis ei ole sisseehitatud stdin-i ega stdout-i, seega välisandmete kättesaamine ja väljastamine sõltub kontekstist. Javaskripti loomulik kontekst on dokument, millega Javaskripti rakendus pöördub DOM (*document object model*) kaudu, niimoodi ta võib saada, muuta ja lisada dokumendi erinevaid elemente. Välistele andmeallikatele andmepääsu saamiseks on javaskriptil võimalused piiratud.
Üks nendest võimalustest on saada andmeid failist HTTP kaudu, kasutades XMLHttpRequest objekti API (XMLHttpRequest). See kasutatakse AJAX tehnoloogias ja eeldab, et rakendus

kui klient pöördub serveri poole. Selline lähenemine antud rakendusele ei sobi, sest kasutaja peaks olema võimeline kasutama rakenduse ühenduseta ega lokaalserveri installeerimata.
Teine võimalus on kolmanda poole teekide kasutamine.
Kolmas võimalus on HTML5 faililugeja API (vt W3C, 2013b; Using files from web applications, 2014; Shmitt, Simpson, 2012; Bidelman, 2010).
Kirjeldatud API järgi on ette nähtud järgmised failidega töö võimalused, mis on meie jaoks olulised:

1. Brauser peab toetama vähemalt DOM4 API (koos Event Target-iga)
2. Faili valik. See toimub ainult kasutaja poolt. Rakendus ei saa ise genereerida linke lokaalfailidele. Kõik failiteed näitab kasutaja ise.
3. Faili lugemiseks peab looma Faililugeja objekti.
4. Kasutaja võib valida üks või rohkem faile. Faili valiku põhjal genereeritakse FileList'i tüübi interface kui konteiner failide nimekirja (listi) jaoks.
5. Faili valiku lõppemine on sündmus, mida jälgitakse faililugeja *onload* sündmuse kuulaja atribuudi poolt.
6. Iga valitud faili sisu võib saada kas *Blob* või *File* interface-ide kaudu. *File* pärineb *Blob*-ist. Selle *result* atribuut annab võimaluse saada faili sisu.
7. See, mis viisil faili sisu loetakse sätestatakse ühe eelnevalt käivitatud faililugeja meetodi abil. Need võivad olla kas *readAsDataURL*, *readAsText* või *readAsArrayBuffer* meetodid. Varem oli ka *readAsBinaryString* meetod, aga see on praegu ebasoovitatav ning praeguses standardi mustandis seda enam ei ole.
8. Kui kasutada *readAsText* meetodit, siis *result* atribuudi tulemuseks on tekstiandmed (String, vaikimisi UTF-8).

4.3 Rakendus: andmete import ja parsimine

Meie rakenduses andmete lugemine on jagatud kahe klasside vahel. Kuna faili valiku väli (*input type="file"*) kuulub kasutajaliidesesse, siis selle väljaga seotud sündmusi on pandud klassi *GraphControls*. Selle klassi meetod *openDataFile* töötleb sündmuse, mis on seotud selle välja väärtuse muutmisega.

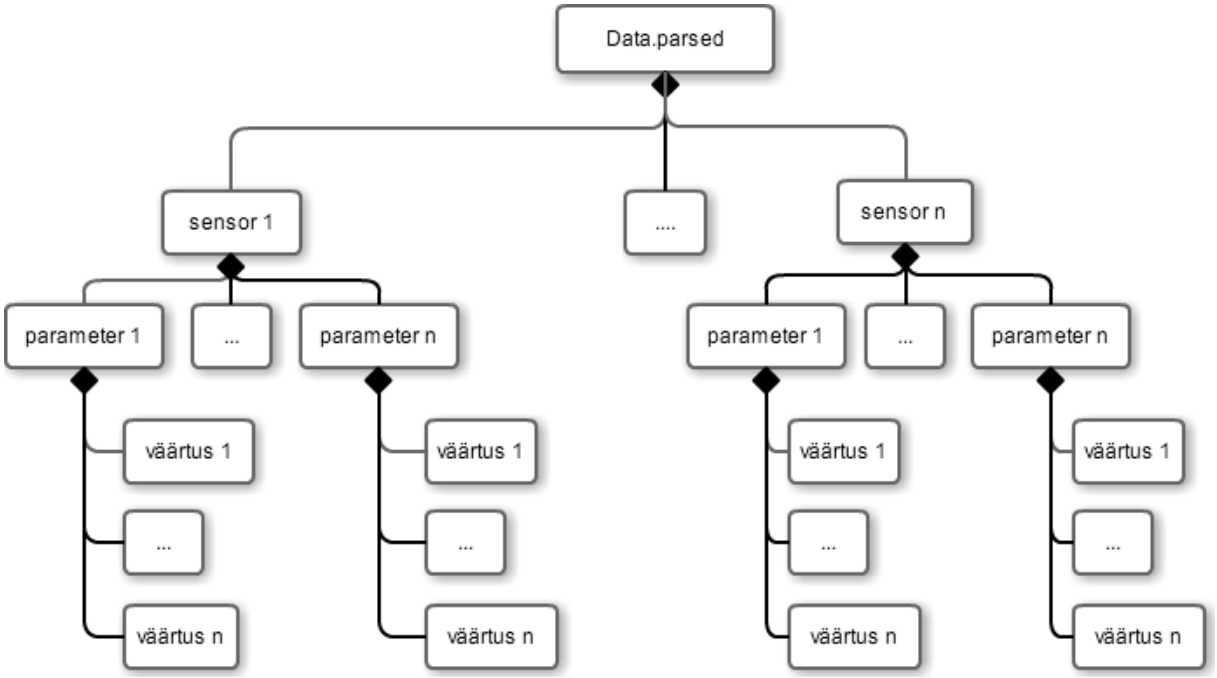
Liidese tasemel kirjeldatakse see väli jQuery selektor-ite kaudu. DOM struktuuris faili valiku välja id on *#sensFile*, selle muutmisel käivitatakse eelnevalt loodud *GraphControls* tüübi objekti meetodi *.openDataFile()*. Argumendina see meetod saab sündmuse (*event*), mille sihiks on valitud failide nimekiri (*FileList interface*). Kuna on teada, et kasutaja valib ainult üht faili, siis võetakse sellest nimekirjast ainult esimest elemendi.

Pärast seda luuakse uue faililugeja tüübi objekti ning selle objekti *onload* sündmuse seostame selle faili sisu parsimisega. Selleks luuakse anonüümne funktsioon, kus see fail loetakse massiivi ridade kaupa, siis see parsitakse selle objektiga seotud Data objekti resOut meetodi abil.

Data objekti *.resOut()* meetod omakorda salvestab saadud massiivi *Data* objekti *sensorsData* atribuudi väärtuseks. Siin toimub ka kontroll, et faili sisu pole tühi ja lülitakse sisse ka uue faili trigerit (*isNew*).

4.3.1 Põhidata teisendamine objektiks

Siis kutsutakse välja Data objekti meetod *.fileToData()*, kus toimubki parsimine. Kõigepealt väärtustatakse selle objekti *columns* atribuut (see loeb .csv faili esimese rea ning jagab selle massiiviks tulbade nimetustega. Meetod *.getFieldsNum()* tuvastab iga sensori parameetrite arvu. Siis arvutatakse sensorite arvu ning luuakse *obData* objekt, mis omistatakse *Data.parsed* atribuudile. Selle objekti struktuuri võib vaadata joonisel (vt Joonis 7):



Joonis 7. Faili parsimisel loodud Data.parsed objekti struktuur

Joonisel on näha, et loodud objekt on põhimõtteliselt võrreldav kolmemõõtelise massiiviga, mille ülemine tase on tavamassiiv, kus loetletakse sensoreid failis tekkimise järjekorras. Seega *Data.parsed[0]* on esimese sensori massiiv, *Data.parsed[1]* on teise sensori massiiv jne.

Sensori parameetrid on assotsiatiivne massiiv, mille võtmeteks on parameetrite nimetused. Iga sensori esimeseks parameetriks on absoluutne ajatempel (*absolute timestamp*). Seega nt. *Data.parsed[1]['absolute timestamp']* on teise sensori ajatemplite massiiv, *Data.parsed[0]['Acc.x']* on esimese sensoriga saadud x-teljel kiirenduse väärtuste massiiv jne.

Omakorda iga parameeter on tavaline massiiv, kus on kõik parameetrite väärtused loetletud selles järjekorras nagu nad esinevad andmefailis. Iga sensori massiivide järjekorranumbrid vastavad iga parameetri massiivile. Teiste sõnadega, kui meil on *Data.parsed[0]['absolute timestamp'][19]* väärtus on 1377090933539, siis x-teljel kiirenduse parameetri 20. väärtus (*Data.parsed[0]['Acc.x'][19]*) on 428 ning see vastab ülalnimetatud ajahetkele. Samuti y-

teljel kiirenduse 20. väärtus massiivis (*Data.parsed[0]['Acc.x'][19]*), mis on 34 on fikseeritud sensori poolt samal ajahetkel. Kuna meil on andmete edastamise sagedus (S, Hz) konstantne, siis iga järgmine väärtus tuleb ajavahemikuga (1/S)*1000. Nüüd selleks, et leida iteratsiooni numbri (I), kus on kirjas esimene sensori väärtus, mis tuli esimesena pärast X ms peale andmete edastamise algusest, siis on meil vaja arvutada järgmiselt (*absolute timestamp = at*):

$$\mathbf{I} = \frac{\boldsymbol{S}*(\boldsymbol{Data.parsed}[\mathbf{0}][\boldsymbol{'at'}][\boldsymbol{X}]-\boldsymbol{Data.parsed}[\mathbf{0}][\boldsymbol{'at'}][\mathbf{0}])}{\mathbf{1000}} \quad \textbf{(1)}$$

Pärast seda, kui rakenduse poolt on see objekt loodud, siis läbitakse failist saadud ridade massiivi. Kõigepealt kontrollitakse, kas käesolevas reas ei ole kirjas video sünkroniseerimise andmeid. Sel juhul kutsutakse välja *.setSync()* meetodit. Ülejäänud read töödeldakse kui sensori parameetrite väärtused. Iga rida jagatakse tulpadeks (*split* funktsioon ", " separaatorina), siis iga tulpa puhul arvutatakse sensori number, millele see kuulub (jooksev tulpa number jagatakse väljade arvuga), siis arvutatakse parameetri indeks (jooksva tulpa number jagatakse moodulil väljade arvuga).

4.3.2 Min ja max andmete leidmine, arvutamine ja teisendamine objektideks

Data.parsed objekti kõrval luuakse veel kaks objekti sarnase struktuuriga. Esimene objekti nimi *maxMin* – siia pannakse info kõikide kõverate minimaalse ja maksimaalse punktide kohta. Teine objekt on on *nodeX* – siia salvestatakse info ekstreemumite kohta.

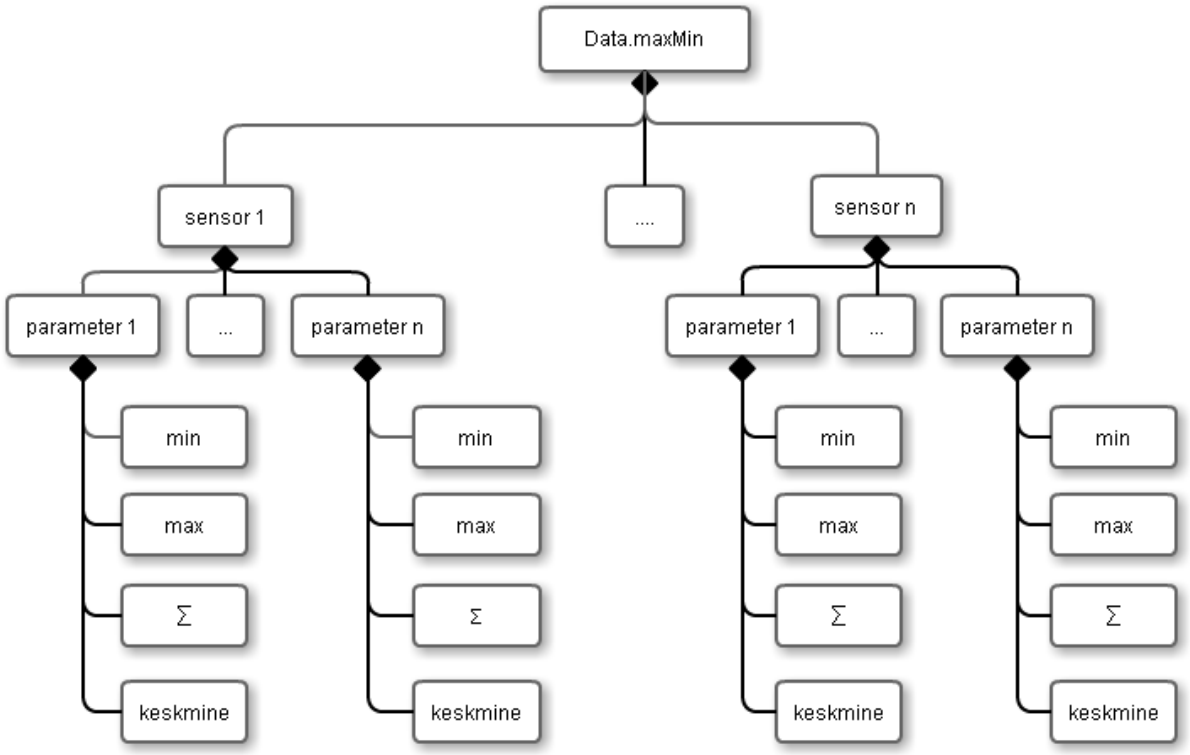
Objekt *Data.maxMin*. Selle objekti struktuur võib näha skeemil (vt Joonis 8). Parsimisel luuakse jälle kolmemõõteline massiiv, kus on ülemine tase – sensorite tavamassiiv. Omakorda iga sensori massiiv (nt *Data.maxMin[0]*) on assotsiatiivne massiiv, mille võtmeteks on füüsikalise parameetrite andmed. Selle objekti puhul andmete parsimisel ajatempli väli (*absolute timestamp*) ignoreeritakse. Massiivid luuakse ainult reaalsete füüsiliste parameetrite jaoks. Iga sensori parameetri väärtused on omakorda tavamassiiv (nt *Data.maxMin[0][Acc.x]*), mille suurus on 4 elementi. Esimeseks elemendiks on selle parameetri maksimaalväärtus, teiseks on minimaalväärtus, 3-ndaks on väärtuste summa (vahetulemus vajalik keskmise arvutamiseks) ning 4-ndaks on väärtust aritmeetiline keskmine. Näited (arvutatud Lisa 1. Algandmete näide toodud näidisandmete põhjal):

Data.maxMin[0][Acc.x][0] = 498

Data.maxMin[0][Acc.x][1] = 318

Data.maxMin[0][Acc.x][2] = 12584

Data.maxMin[0][Acc.x][3] = 405,9355



Joonis 8. Faili parsimisel loodud Data.maxMin objekti struktuur

Andmed maksimaalse ja minimaalse väärtuse kohta on vajalikud, et tuvastada, mis on selle graafiku mastaap (minimaalse ja maksimaalse kõrguse mõõt), mis rakenduses sätestatakse alguses automaatselt. Kui andmete parsimine on lõppenud, siis funktsioonist *Data.fileToData* kutsutakse välja meetod *Data.setMinMax(),* kus vaadatakse läbi kõikide sensorite kõik kasutaja poolt kuvamiseks aktiveeritud parameetrid, ning nende seast valitakse objektist *Data.maxMin* suurim ja vähim väärtused.

Andmed keskmiste väärtuste kohta on vajalikud, et tuvastada filtri, mis võimaldab vältida liigsete iteratsioonide graafiku andmestruktuuri lisamist graafiku kuvamisel (see tehakse jõudluse huvides).

5. Andmete visualiseerimine

Järgnevalt kirjeldatakse seda, kuidas kasutajaliides teostab andmete kuvamist kasutades FlotCharts jQueri teegi. Ennekõike kirjeldatakse FlotCharts-i API põhivõimalused, mis on antud rakenduse kontekstis olulised ja siis käsitletakse andmete kuvamise mehhanisme kasutajaliideses.

5.1 jQuery

DOM elementidele ligipääsu lihtsustamiseks ja unifitseerimiseks kasutatakse meie rakenduses jQuery 1.9.0 teegi. jQuery on javaskripti teek, mis laiendatakse MIT litsentsi alusel (License). On olemas mitu raamatuid, kus käsitletakse seda teegi. Üks parimaid on (Bibeault, Katz, 2010).

Siin kasutatakse juba suhteliselt vana jQuery versiooni (vt Methvin, 2012), sest rakenduse tuumikfunktsionaalsus oli loodud 2013. a. suvel. Kuna selle töö maht on piiratud, siis jQuery-t siin monograafiliselt käsitlema ei hakka. Antud rakenduses kasutatakse selle baasomadusi, mis võimaldavad DOM-elemente välja valida, nende omadusi dünaamiliselt määrata ja muuta ning siduda nendega sündmusi, mis tekitavad äriloogika mehhanismide käivitamist.

Samuti jQuery kasutamise vajadus meie rakenduse kontekstis on tingitud selle poolt, et selle põhjal tegutseb FlotCharts teek, mida käsitletakse allpool. Sõltub jQuery-st ka Twitter Bootstrap raamistik, mida siin kasutatakse liidese küljendamise ja kujundamise parandamiseks.

5.2 Kasutatud teegid: FlotCharts ja selle API

FlotCharts on javaskripti ja jQuery teek, mis võimaldab joonistada graafikuid kasutades HTML5 canvas elemendi. (FlotCharts). Selle API dokumentatsiooni võib leida siit: (Kleine-König, 2014). Meie rakenduse kontekstis mitte kõik FlotChart API omadused on vajalikud. Allpool on nende kirjeldus.

Selleks, et joonistada graafikut, on vaja kutsuda jQuery meetodit *jquery.plot(placeholder, data, options)* See meetod tagastab objekt *plot*. Selle meetodi argumendid on järgmised:

- *placeholder* – kohatäitja, jQuery objekt, mis on seotud HTML5 lõuendiga (*canvas*), kuhu graafik joonistatakse. Kohatäitja kirjeldamise näide jQuery abil: *$("#placeholder")*
- *data* — On võimalik seda esitada kahel viisil.

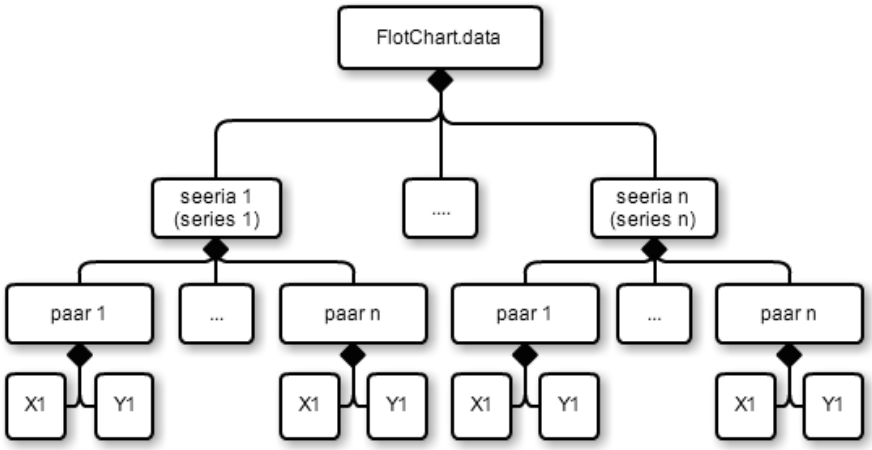
a. data kui javaskripti kolmemõõteline tavamassiiv. Selle struktuur saab näha skeemil (vt Joonis 9) *Data* massiivi kirjeldamise näide javaskripti abil:

- [[[1, 10],[2,32],[3, 12],[4, 55]],[[0.5, 22],[0.7, 15]]]

b. data kui objekt. Sel juhul selle objekti (valikulised) omadused on kirjeldatud nagu järgmiselt (punasega on esile tõstetud rakenduses kasutatavad atribuudid):

```
{
    color: color or number
    data: rawdata
    label: string
    lines: specific lines options
    bars: specific bars options
    points: specific points options
    xaxis: number
    yaxis: number
    clickable: boolean
    hoverable: boolean
    shadowSize: number
    highlightColor: color or number
}
```

- Nendest omadustest meie rakenduse kontekstis on *data* kui argumendi jaoks olulised värvi omadus (*color*), andmete väärtused (*data*, vt punkt a.) ning kõverate (*lines*) või tulpade (*bars*) kuvamise omadus. Need omadused meil on vaja sätestada iga andmete seeria jaoks. X- ja Y-telje omadused meil sätestatakse kogu graafiku jaoks.



Joonis 9. FlotCharts plot funktsiooni data argumendi struktuur

- *options* — objekt, mille atribuutide abil saab seadistada graafikute kuvamise omadusi. Nendest omadustest kogu graafi meie jaoks kõige olulisemad on:

a. *yaxis* – see on ise objekt, millel on FlotCharts API vastavalt 22 atribuute (nende seas nt *timezone*, *transform*, *autoscaleMargin* jt). Nendest meie rakenduse kontekstis on olulised järgmised atribuudid:

- *min, max* – graafiku alam- ja ülemine piir
- *tickSize* – interval, millega näidatakse skaalat Y-teljel.

b. *xaxis* – samuti objekt. Kokku atribuute on ka 22. Selle rakenduse kontekstis meil ei ole vaja selle telje märgistusi näidata. X-teljel meil mõõdetakse aega ning meil on oluline teada vaid jooksva hetke millisekundite arvu alates video alustamisest. Seda tehakse *grid* atribuudi abil (vt. allpool). Seega sätestatakse ainult atribuudi *show* ning pannakse selle väärtuseks *false-i*.

- *grid* – on objekt, millel on 15 atribuute. Meie jaoks on oluline markeeringute (*markings*) atribuut, mille abil luuakse vertikaaljoone x-teljel, mis näitab jooksva ajahetke graafikul. Joone all kuvatakse jooksva hetke kellaaega (millisekundites, alates video algusest). *grid.markings* atribuudiks pannakse järgmine objekt: {*color: „#000“*, *lineWidth:* 1, *xaxis:{to: , from:}*

Objekti *xaxis:* atribuudiks pannakse intervall, mis on 1 piksel paks ning asub x-telje keskel (vt.4.3.1).

Selliseid *options*-objekti antud rakenduses luuakse kaks. Esimene kasutatakse graafikus juhul, kui kasutaja on valinud andmete kuvamist kõverate viisil, teine aga kui on valitud andmete kuvamine tulpade viisil. Viimasel juhul meie jaoks *grid* objekt pole vajalik, sest andmete diakroonilist esitamist ei toimu ning vajadust jooksva ajahetke fikseerida ei teki.

Pärast jQuery objekti *plot* meetodi välja kutsumist flotCharts API kaudu luuakse objekt Plot ning kutsutakse välja selle objekti atribuut *draw()*, mis joonistab graafiku ettekirjutatud lehe elemendis, mille *id* on fikseeritud selle lehe DOM-is kui kohatäitja. Selle kohatäitja elemendi sees luuakse HTML5 lõuendi element, kuhu joonistatakse graafik ise.

5.3 Graafik kui objekt. Graafikute kuvamine rakenduses

Antud rakenduse raames on vaja teisendada saadud .csv failist *Data* objekti FlotCharts API aktsepteeritavaks andmeteks. Selleks meie rakenduse raames luuakse objekt nimega *graph*, mille jaoks me kirjeldame javaskripti klassifunktsiooni *Graph*. Selle klassi atribuudide vahendusel me seadistame graafiku parameetreid, ning kuvame graafiku selle meetodite abil. *Graph* objekti atribuutide kirjeldust vt Lisas 3.

5.3.1 Graafiku joonistamine

Ülaltoodud FlotChars API kirjeldusest järgneb, et kui rakendusel on vaja kuvada mõni hetke graafiku, siis tal on vaja saada teistest objektidest lähteandmeid õigesti vormistatud massiivina või objektina ning objekti graafiku omadustega. Graafiku põhiomadusi sätestatakse juba *Graph* objekti konstruktoris (vt p. 5.3). Aga mõned nendest omadustest peavad dünaamiliselt muuta – seega peab olema ligipääs nendele ka hiljem.

Graafiku joonistamine toimub selle klassi *Graph.draw()* meetodi sees. Aga iga konkreetse hetke vajalikud seadistused toimuvad *Graph.extractCurve()* meetodi abil, mis kutsutakse välja *Graph.draw()* meetodi seest. *Graph.extractCurve()* meetod tagastab massiivi, mis on

aktsepteeritav FlotCharts-i API poolt kui andmed kuvamiseks. Aga samal ajal see meetod väärtustab *Graph* objekti mõni atribuute.

Kõverate massiivi koostamine. Põhiandmete massiivi koostamine toimub kasutajaliideses *Graph.extractCurve()* meetodi abil. Selle meetodi signatuur on järgnevalt.

1. Argument *sensor* on sensori number, mille järgi leitakse *data.parsed* objektist vajaliku objekti (vt. p. 4.3.1)
2. Argument *fys* on võti, mille järgi leitakse *data.parsed* leitud parameetri objektist vajalikku parameetri massiivi (vt. p. 4.3.1)
3. Argument *start* on hetke video täis ajatempel (Unix timestamp[4] * 1000). Seda aega saab kätte data-objektist, kasutades selle meetodit *.getTime*, mis tagastab video algusaega andmefailist ning video hetke aega summat (üksikasjalikult vt Peatükis 6).
4. Argument *pikkus* on graafiku punktide iteratsioonide arv. Miks ei saa võtta Graph objekti oma *pikkus* atribuudi väärtust? Sest skaleerimise käigus, kui kasutaja muudab graafiku pikkuse hiirega või nuppudega see väärtus omistatakse atribuudile alles siis, kui nupp on lahti (released). Aga tahaks kasutajale dünaamiliselt näidata graafiku pikkuse muutmist, seega autor otsustas iga kord selle funktsioonile seda väärtust eraldi muutuja abil saata.

Kõigepealt siin luuakse *offset* muutuja (**I**) ja arvutatakse selle väärtus. See väärtus on esimene iteratsioon, mida pannakse graafikusse x-teljel. Jooksva hetke teadasaamiseks meil on vaja lahutada video jooksva hetke täisajatemplist (**Video ts**) andmete alguse täis ajatemplit (**Data ts**). Saadud arv on aga millisekundites, aga meie iteratsioonid tulevad harvem sõltuvalt sensori sagedusest (**S,** Hz). Järelikult tuleb jagada saadud arvu sensori andmete intervalliga[5] (**dt =** S / 1000, ms). Siis, kuna alustatakse graafiku kuvamist enne jooksva hetke (jooksev hetk on graafiku keskel), siis meil on vaja määrata graafiku esimeseks iteratsiooniks seda iteratsiooni, mis tuleb pool pikkuse varem. Selleks tuleb lahutada saadud iteratsiooni numbrist graafiku poolpikkuse (**len / 2**). Peale selle siin läheb arvesse nihe (**shift**), mida kasutaja võib ise määrata, et tagada sünkroniseerimise täpsust (vt sellest üksikasjalikult Peatükis 6). See nihe liidetakse lõpparvuga. Lühikokkuvõttes toimub graafiku algiteratsiooni (**I**) leidmine järgmise foormula vastavalt:

$$\mathrm{I} = \frac{Video\ ts - Data\ ts}{dt} - \frac{len}{2} + shift \qquad (2)$$

Kui graafiku algusiteratsioon on leitud, siis rakendust võtab *data.parsed* objekti, leiab sellest vajaliku sensori objekti (võtmeks on meetodi *Graph.extractCurve* esimene argument), siis selle sensori objekti vajaliku parameetri massiivi (võtmeks on meetodi *Graph.extractCurve* teint argument) ning teisendab selle massiivi *pikkus* argumendina etteantud väärtusi alates leitud algiteratsioonist. Siis andmeid teisendatakse *plot.data* poolt vastuvõetavaks massiiviks.

[4] Unix time või POSIX time – aeg al. 1.01.1970-st sekundites (vt Unix time).
[5] Nagu on juba öeldud eelmises peatükis sageduse asemel on lihtsam kasutada andmete tuleku intervalli, mis on konstantne ja arvutatakse millisekundites kui sensori sagedus jagatud 1000 ms-tega.

Seejuures selle meetodi raames arvutatakse ka piirang iteratsioonide kuvamiseks. Selleks meil on muutuja nimega *sito*. Selle arvutamiseks määratakse mingi konstandi (*TIHEDUS*), mis on graafiku pikkuse lähedal (meil on see 600). Siis *sito = Math.round(pikkus / TIHEDUS)*. See muutuja kasvab proportsionaalselt graafiku pikkusega. Siis lisatakse graafiku massiivi ainult neid iteratsiooni, mille järjekorranumber on *sito* kordne. Selle filtri arvutamine on tingitud jõudluse parandamisega.

Lisaks selle meetodi raames toimub graafiku keskjoone (mis kujundab jooksva hetke) koordinaatide uuendamine. Selleks pannakse *plot.options.grid.markings[0].xaxis.from* ja *plot.options.grid.markings[0].xaxis.to* väärtuseks jooksva algiteratsiooni ning graafiku poolpikkuse vahet.

Tulpade massiivi koostamine. Põhiandmete massiivi koostamine toimub kasutajaliideses

Graph.extractBar() meetodi abil. Selle meetodi signatuuri on järgmised:

1. Argument *sensor* on sensori number, mille järgi leitakse *data.parsed* objektist vajaliku objekti (vt. p. 4.3.1)
2. Argument *fys* on võti, mille järgi leitakse *data.parsed* leitud parameetri objektist vajalikku parameetri massiivi (vt. p. 4.3.1).
3. Argument *htk* on hetke video täis ajatempel, millisekundites (Unix timestamp * 1000). Seda aega saab kätte data-objektist, kasutades selle meetodit *.getTime*, mis tagastab video algusaega andmefailist ning video hetke aega summat (vt. üksikasjalikult Peatükis 6).

Siin on meil vaja teada ainult sensori numbri, füüsilise parameetri võtme ning video jooksva täieliku ajahetke (Unix timestamp * 1000). Jooksva iteratsiooni teadasaamiseks meil on vaja leida video ajahetke (**Video ts**) ja andmete alguse ajahetke (**Data ts**) vahet ning jagada seda ajaintervalliga (**dt**). Kuna kogu kuvamine on sünkroonne, seega mingit aega arvutusi tegema ei pea. Ainult sellele iteratsioonile on vaja lisada kasutaja poolt määratud nihet (**shift**). Seega jooksva iteratsiooni arvutatakse välja järgmise foormula järgi:

$$\mathbf{I} = \frac{\boldsymbol{Video\ ts - Data\ ts}}{\boldsymbol{dt}} + \boldsymbol{shift} \quad (3)$$

Selles meetodis määratakse ka graafiku väljanägemine. Siin on oluline ainult y-teljel mõõt. Seega graafiku maksimaalseks ja minimaalseks väärtuseks y-teljel pannakse *Data* objektis salvestatud minimaalsed ja maksimaalsed väärtused. Miinimum ja maksimum x-teljel on suvalised (meil on 0-st kuni 50-ni). Järgnevalt ka andmepaaris oluliseks väärtuseks on ainult y-väärtus. X-väärtusena pannakse järgmise tulpa koordinaat, mis arvutatakse nii, et massiivi pannakse uue tulpa tähendus, siis jooksva massiivi pikkus korrutatakse 5-ga (mis on tulpade laius).

Meetod tagastab massiivi [x,y] andmepaariga plot objekti jaoks.

Andmete joonistamine toimub Graph.draw() meetodi sees. Selle meetodi argumendiks on kasutaja poolt määratud graafiku nihe. Kõigepealt see meetod kutsub selle meetodiga seotud

Data.tickY(), mille abil saab teada graafiku y-telje märgistusi. Kuna üldine mõõtmisintervall luuakse dünaamiliselt, meil on vaja neid märgistusi ka dünaamiliselt ümber joonistama.

Seejärel seadistatakse kaks objekti *options* ja *series* plot objekti jaoks. Siis saab teada video jooksva hetke, kasutades *Data* objekti *getTime()* meetodid. Kui kasutaja poolt on pandud mingi nihe, siis lisatakse seda hetke väärtuseni.

Järgmine eesmärk on saada teada, millised liidese elemendid (kastid), mis on seotud teatud sensoritega ja teatud füüsiliste parameetritega on kasutaja poolt valitud. Tuleb omistama viide selle massiivile muutujale *curvBxs*. Selleks võib kasutada jQuery selectorit: `$("#curveControls :checkbox[name=curve]:checked");` ning tulemusena saab jQuery objektide massiivi.

Kui on teada, palju füüsilist parameetreid kuvamiseks on valitud, siis on teada ka seda, kui palju kõveraid / tulpe meil on vaja graafiku peale kuvada. Siis rakendus jookseb *curvBxs* massiivi läbi ning arvestades seda, kas kasutaja poolt valitud kõverate või tulpade valimine (seda saab teada *Graph.getType()* meetodi abil, mis kontrollib, kumb raadio ring on valitud liidesel) valib vastavat ümberlüliti haru (*switch-case*).

1. **Kõverad**. Selle valiku puhul kutsutakse välja *.extractCurve()* meetod. Selle meetodi poolt tagastatud massiiv on *plot.data* argumendina. Sensori ja füüsilise parameetri võtmeid saab kätte jooksva kasti *id* atribuudist[6]. Jooksva kõvera värv võetakse *Graph.colors* massiivist, võtmeks on ülalmainitud kolmeliikmelise massiivi kolmas stringi-tüübi element. Plot andmeobjekti *lines* atribuudiks pannakse objekti *{show: true}*.
2. **Tulbad**. Graafiku kuvamine tulpade viisil toimub analoogselt. Tulpade kuvamiseks kutsutakse välja ülalkirjeldatud *.extractBars()* meetod Sel juhul *plot* andmeobjekti *bars* atribuudiks pannakse objekti *{show: true}*. Lisaks atribuut *pikkus* muutuja siin pole enam vaja.

Graafiku kuvamiseks luuakse *plot* objekti, mille kohatäitjaks pannakse ettenähtud elemendi, andmeobjektiks pannakse eelnevalt seadistatud *series* element ning valikuteks pannakse *opts* muutujad. Kui video ei ole laaditud või tekib mingi probleem videoga, siis sünkroniseerimist ei toimu ning jooksvaks ajahetkeks graafikus on 0 ms.

5.3.2 Sensorite füüsikaliste parameetrite valimine kuvamiseks graafikul

GraphControls.boxes() meetod käivitatakse pärast seda, kui valitud andmefail on laaditud ja andmed on teisendatud *Data* objekti poolt *Data.parsed* objektiks (vt 4.3) mille *.length* atribuut tagastab salvestatud sensorite arvu. Siis iga sensori jaoks kutsutakse välja *GraphControls.boxes()* meetod.

Kuna ei ole eelnevalt teada sensorite arvu ning igaühe sensori parameetrite arvu, seega ei saa luua sensorite ja parameetrite kontrollitava liidese staatiliselt.

[6] Nende DOM elementide loomine ja nende *id* nimetuste koostamine vt p. 5.2.2.

Kui käsitletakse uut andmefaili, siis pannakse meetodi atribuut *.boxes().count*[7] nulliks. Igaühe sensori jaoks HTML-lehel luuakse uut <*div*> elemendi. Siis iga sensorile luuakse kasti kõikide selle sensori füüsiliste parameetrite valimiseks või tühistamiseks. Siis selle sensori iga parameetri jaoks luuakse kast selle valimiseks. Iga selle elemendi puhul on oluline *id* omadus, mis koosneb 3-st osast: 1) füüsikaliste parameetri võti; 2) sensori number; 3) kasti järjekorranumber. Iga kastiga seostatakse sündmus *.onchange*, mis kutsub välja järjest kaks meetodit:

1) *Data.setMinMax()* – kuna uus kõver on erinevate minimum ja maksimum väärtustega, siis võib tekkida vajadus lõuendi masstaapi kohendada (vt p. 4.3.2).
2) *Graph.draw()* – graafik peab olema ümber joonistatud muudatusi arvestades (vt 0)

5.3.3 Graafiku pikkus. Kõverate skaleerimine x- ja y-teljel

Kõik sündmused, mis tingivad seda, millist andmeid rakenduses kuvatakse jaotakse kahte rühma sündmuste allikate järgi. Esimene allikas on kasutaja (treener), teine on video jooks, selle algus ja lõppemine. Rakenduse kasutaja võib graafiku ja videot sirvida, ta võib ka graafiku skaleerida x-teljel või y-teljel. Kõik need sündmusi töödeldakse objektis *GraphControls*, siis se võtab Data objektist nõutud andmed ja saadab neid Graph objekti graafiku plottimiseks. Järgnevalt vaatame, milliseid sündmusi see *GraphControls*

Atribuutide väärtusest rääkides tuleb märkida, et vaikimisi pannakse graafiku pikkuse 1000 iteratsiooni. Kuna iga iteratsioon tuleb sõltuvalt sagedusest kas 10 ms intervalliga (100 Hz sensorid) või 20 ms intervalliga (50 Hz sensorid), seega vaikimisi kõverate pikkuseks, mida on igal hetkel graafikul näha on vastavalt ligi 10 või 20 s. Seda on piisavalt, et näha sportlase jooksva harjutuse lähima konteksti. Soovi korral kasutaja võib graafiku pikkuse muuta (skaleerida x-teljel), tehes seda kas suuremaks või väiksemaks. Loomulikult kohatäitja suurus jääb seejuures sama.

GraphControls objekt jälgib liidese sündmusi, mis tekitavad graafiku pikkuse muutmist. Graafilise liidese elemendid, mis asuvad html-lehel paremal pool otse graafiku kohal võimaldavad graafiku skaleerida vertikaalselt või horisontaalselt (vt. Lisa 2. Kasutajaliidese näide. G3 ja G4 elemendid). *GraphControls.skaleeri()* meetodis kirjeldatakse, milliseid DOM elementidega seotud sündmusi kasutatakse graafiku x-teljel ja y-teljel skaleerimiseks. Skaleerimise saab teha ka hiire vahendusel skrollimisratast kasutades kui kursor asub graafiku lõuendi alal (vt Lisa 2. Kasutajaliidese näide G5). Neid sündmusi kirjeldatakse *GraphControls.events()* meetodis. Järgnevalt on loetletud neid elemente ja sündmusi.

Skaleerimine x-teljel. 4 HTML-elemendi (Vt Lisa 2. Kasutajaliidese näideG3 ja G4 ikoonid) kuuluvad css-klassile *.skal* ning neid saab valida jQuery selektori abil. Horisontaalse skaleerimise eest vastavate elementide (G3) DOM-identifikaatorid on vastavalt *#skalXzOut* ja

[7] Javaskriptis ei kasutata staatilisi muutujaid funktsiooni skoobis, mis luuakse prototüübi objektist sõltumata. Selle asemel võib kasutada javaskripti funktsioonide omadusi, mis omistatakse funktsiooni prototüübile (vt Flanagan 2011:177). Omistatakse *.boxes().count* atribuudi staatilise muutujana.

#skalXzIn. *GraphControls* klass jälgib järgmiseid sündmusi, mis on seotud nende elementidega:

1) *.hover* (kasutatakse aktiivse ikooni märgistamiseks),
2) *.mouseup* ja *.mouseout* mõlemate sündmuste puhul kutsutakse välja *Graph.skaleerHorRelease()* meetodit. Selle meetodi sisuks on eelnevalt loodud Interval-tüübi objekti *Graph.ticker* kustutamine. See objekt luuakse, et perioodiliselt kutsuma välja skaleerimismeetodit (vt allpool). Kui nupp vabastatakse, siis *ticker* tühistatakse ning skaleerimine katkestatakse.
3) *.click*. Elemendi *#skalXzIn* puhul kutsutakse välja *Graph.skaleerHoriz()* meetod, mille argumendiks on inverteeritud *GraphControls.scale.x* väärtus[8]. Elemendi *#skalXzOut* puhul kutsutakse välja sama meetod, sama kuid inverteerimata argumendiga. Meetod *skaleerHoriz()* suurendab graafiku pikkuse selle argumendi võrra. Argumendi absoluutsuurus vähendatakse proportsionaalselt graafiku pikkuse vähenemisele. Pikkuse muutmine toimub ühekordselt.
4) *.mousedown*. Elementide *#skalXzIn* ja *#skalXzOut* puhul kutsutakse välja *Graph.skaleerHorCycle()* meetod, mille argumendiks on *GraphControls.scale.x* positiivne või negatiivne väärtus. Klõpsatud hiire puhul iga 100 ms kutsutakse välja *.skaleerHoriz()* meetodit, mis muutub graafiku pikkuse.

Skaleerimine y-teljel. DOM-elementide *#skalYzIn* ja *#skalYzOut* (vt Lisa 2. Kasutajaliidese näide G4) sündmusi jälgitakse rakenduse poolt vertikaalseks skaleerimiseks *GraphControls.skaleeri()* meetodis. Kuna need elemendid samuti kuuluvad css-klassile *.skal*, siis nende puhul ka kehtivad üleval kirjutatud pp. 1-2.

5) *.click* . Elemendi *#skalYzIn* ja *#skalYzOut* puhul kutsutakse välja järjest meetodeid *Graph.skalVerMax()* ja *Graph.skalVerMin()*, mille argumentideks on *GraphControls.scale.y* vaikimisi väärtused. *#skalYzIn* elemendi puhul see väärtus inverteeritakse. *Graph.skalVerMax()* meetod muudab graafiku maksimaalset väärtust saadud argumendi võrra. *Graph.skalVerMin()*, meetod omakorda niipidi muudab graafiku minimaalset väärtust. Pärast selle väärtuse muutmist graafik joonistatakse uuesti.
6) *.mousedown*. Elementide *#skalYzIn* ja *#skalYzOut* puhul kutsutakse välja *Graph.skalVerCycle()* meetod positiivse või negatiivse argumendiga. Kuni hiir on klõpsatud, siis luuakse Interval-tüübi objekt *Graph.ticker*, mis iga 100 ms kutsub välja ülalmainitud *.skalVerMax()* ja *.skalVerMin()* meetodeid ning see pidevalt muutub graafiku kõrguse. Kui hiir vabastatatkse, siis töödeldakse selle CSS-klassi elementidega seotud sündmusi *.mouseup* ja *.mouseout* (vt. p. 2).

Skaleerimine x-teljel ja y-teljel selle rakenduse raames on võimalik ka **hiire abil.** Selleks on *GraphControls.events()* meetodis on kirjeldatud millised graafiku lõuendi sündmused töödeldakse milliste meetodite abil. Skaleerimiseks siin toimub lõuendi hiire skrollimisega

[8] Selle rakenduse skoobis see atribuut on konstaantne ning pandud =100

seondumine jQuery meetodi *.bind('mousewheel', function(){})* abil. Järgnevalt kutsutakse välja anonüümne funktsioon, mis

1) tuvastab skrollimise suunda,
2) kui shift on vajutatud, siis teeb y-teljel skaleerimist (kutsub välja järjest ülalnimetatud meetodeid *Graph.skalVerMax()* ja *Graph.skalVerMin()*.
3) kui shift ei ole vajutatud, siis tehakse x-teljel skaleerimist (kutsutakse välja ülalnimetatud meetod *Graph.skaleerHoriz()*).

5.4 Raamistikud liidese kujundamiseks. Twitter Bootstrap

Liidese küljendamise ja kujundamise lihtsustamiseks rakenduses kasutatakse sellist tundliku raamistikku (*responsive framework*) nagu Twitter Bootstrap 2.3.2. Kuna antud kasutajaliidese rakenduse tuumik oli loodud 2013. a. suvel, siis kasutatakse Bootstrap-i vana versiooni. See ei mõjuta funktsionaalsust ja jõudlust, sest siin on kasutusel ainult Bootstrap-i baasomadusi, mis on seotud küljendamisega ning fontide trükkimisega, sh *iconglyph* ikoonide kasutamine video mängimise juhtimise nuppude kuvamisel.

6. Video haldamine. Video ja graafiku sünkroniseerimine

Antud peatükis kirjeldatakse rakenduse nõuded video suhtes, vaadeldakse erinevaid videokonteinereid, mis kasutatakse HTML5 API-s, kirjeldatakse javaskripti lisateegi, mis kasutatakse kaaderhaaval sirvimise jaoks, käsitletakse video ja andmefaili sünkroniseerimise küsimust ning kirjeldatakse, kuidas toimub töö videoga kasutajaliidese rakenduses.

6.1 Nõuded video formaadile

Käesoleva rakenduse nõuetest lähtudest oli vaja valida sobivat video formaadi. Rakendusest lähtuvalt see pidi vastama järgmistele nõuetele:

1. See peab olema aktsepteeritava HTML5 API poolt ja integreeritav veebirakendusesse.
2. Videopildi suurus on rakenduses pandud kui 5 / 12 veebilehitseja akna laiusest. Näiteks, kui ekraani mõõt on 1600*900 px, siis video laius on vähem kui 635*476 px, kui ekraani mõõt on 1024*768, siis video laius on 406*305 px. Seega standardne VGA lahutusvõime (640*480) on selle rakenduse jaoks on täiesti sobiv.
3. Kuigi kasutatavate sensorite andmeedastuse sagedused on 100 Hz või 50 Hz, siiski kasutatavuse vaatenurgas on sobiv sagedus vastab levinud veebikaamerate ka selle eelistuse põhjused antud rakenduses on järgmised:
    a. paljudes veebikaamerates see on vaikimisi pandud
    b. see on optimaalne jõudluse vaatenurgast.
4. Nõuded videofailide suuruse kohta sõltuvad treeneril kasutusel olevast riistvarast. Kuna põhirõhk tehakse mobiilsete seadmete kasutamisele (nt sülearvutid, tahvelarvutid, nutitelefonid), siis video peaks ikka olema pakitud.
5. Video peab saama sirvida kaaderhaaval. Selleks on vaja teada saada iga kaadri täpne kellaaeg.

6.2 Video konteinerid ja koodekid, mida HTML5 API kasutab

HTML5 API video formaat ei ole standardiseeritud. Selleks on hulk põhjusi. On olemas kolm videoformaadi, mis on kandidaadid HTML5 video standardiks.

1. **MP4** – see on formaat, mille tuumikuks on proprietaarne koodek AVC / H.264. H.264 standart on kirjeldatud artiiklis (Sullivan jt, 2004). AVC (*advanced video codec*) on standardiseeritud ISO poolt kui ISO/IEC IS 14496-10[9]. Koodek oli arendatud ITU-T kompanii poolt.

[9] Standardi täis nimetus on ISO/IEC IS 14496-10: Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding and ITU-T Rec. H.264, March 2005: Series H: Audiovisual and Multimedia

2. **WebM** – avatud lähtekoodiga (open source) videoformaat, mis on matroska-tüübi konteiner (WebM Container Guidelines). Selle koodek on VP8, arendatud On2 Technologies poolt. See koodek on täiesti vaba. 2010 a. On2 oli ostetud Google poolt (vt Google Closes On2 Technologies Acquisition) ning 2013. a. Google ja MPEG-LA leppisi Aga jõudluse testid näitavad, et see formaat on kehvem, kui H.264 (vt Ozer, 2010; Kulikov, 2011).
3. **Ogg** on alternatiivne avatud lähtekoodiga formaat (vt The Ogg container format), millesse on ehitatud sisse videokoodek Theora (mis on sama firma On2 Technologies VP3 koodeki edasiarendus). Selle testimise tulemused (Ozer 2010a) näitavad samuti, et video kvaliteet on kehvem kui H.264 oma.

MP4 formaat võimaldab parimat balanssi pildi kvaliteedi, andmete pakkimise ja video dekodeerimisel nõutud arvutijõudluse vahel. Seetõttu praegu see formaat on de-facto standart internet-video valdkonnas.

Samas selle formaadi kasutamine HTML5 API-s tekitab diskussiooni. See on tingitud selle formaadi litsentsiküsimustega. Selle koodeki litsents eeldab, et arendajal, kes loob rakendusi, mis kodeerivad ja dekodeerivad H.264 video ning ettevõtetel, kes laiendab videoid interneti kaudu, peab olema ostetud litsents selle tegevuse jaoks (AVC Patent Portfolio License Briefing). Litsentsi ei pea ostma video lõppkasutaja. Aga veebilehitseja arendaja selle litsentsi järgi peab ikka ostma. Diskussiooni käigus on väga tugev seisukoht see, et HTML5 video ei pea olema litsentseeritud, nagu ei pea olema litsentseeritud teised HTML standardis käsitletavad komponendid, nagu CSS ja Javascript (vt Shaver 2010). Seetõttu erinevatel veebilehitsejate tootjatel on seisukoht erinev. Mõned ei nõustu üldse H.264 toetada (Opera), teised piiravad seda toetust (nt Safari toetab H.264 kui audio koodek on AAC, mitte MP3).

Internet Explorer toetab H.264, kuna Microsoft on üks MPEG-LA patenti kandjatest. Aga samas Internet Explorer piirab toetust teiste formaatide jaoks: Ogg ega WebM formaadi ametlikult ei toeta, kuigi al. 9. versioonist kasutaja võib ise installida plugin-id, mis võimaldavad neid videoformaate Internet Explorer-is mängida (vt Hachamovitch 2011).

Mozilla Firefox ja Google Chrome otsustati jätkata H.264 toetust, kuigi Mozilla on väidetud, et mittevaba formaadi kasutamine avatud lähtekoodiga projektis on nende ideoloogiaga vastuolus (Eich, 2012) ning Chrome on kuulutanud, et kuna nende prioriteediks on VP8 arendamine, seega H.264 toetus tulevikus eemaldatakse Chrome brauserist (Paul, 2011).

Ülaltoodud lähtuvalt võib teha järeldust, et antud kasutajaliidese rakenduse kontekstis H.264 formaat on kõige sobivam pakkimise, jõudluse ja videokvaliteedi vaatenurgast. Litsentsiküsimused autorit ei puuduta, sest kasutajaliides ega juhtimistarkvara ei tegele video (de)kodeerimisega (selleks kasutatakse olemasolevate rakendusi ja API-d). Kuna see ei ole veebirakendus, seega pole vaja siin ka arvestada litsentsi video laiendamisega seotud küsimusi: sportlaste videod on suunatud privaatseks kasutamiseks nende ja nende treenerite poolt.

Systems: Infrastructure of audiovisual services – Coding of moving video: Advanced video coding for generic audiovisual services (vt ISO/IEC 14496-10:2012).

6.3 HTML5 API ja lisateegid: VideoFrame.js

See teek on kasutajal jaoks oluline selleks, et saaks kaaderhaaval videot sirvida.
HTML5 video API on kirjeldatud raamatus (Pfeiffer, 2010). See API on realiseeritud kui vastava <video> HTML-elemendi omaduste (atribuutide) ja sündmuste kogum. Samuti on olemas mõned selle HTML-elemendi kui DOM objekti javaskripti meetodid. Käesoleva rakenduse kontekstis on kasutusel järgmised sündmusi, omadusi ja meetodeid.
Sündmused. Neid on HTML5 API-s palju (vt Jeremie, 2014). Nendest rakenduse jaoks on olulised järgmised:

- *ended* – toimub mängimise lõppemisega.
- *loadedmetadata* – see sündmus toimub kui video andmed on täiesti alla laaditud ning video on mängimiseks valmis. Siis saab teada video pikkust, failteed jt. (Pfeiffer, 2010: 85).
- *pause* – toimub, kui video on ajutiselt peatatud.
- *play* – toimub, kui mängimine algab pärast pausi.
- *seeking* – toimub, kui video sirvimine algab.

Kui DOM-objekt HTML5 video kuulub HTMLVideoElement interface-i, mis omakorda pärineb HTMLMediaElement interface-ist (vt Flanagan, 2014). Sel elemendil on järgmised rakenduse jaoks olulised omadused:

- *currentTime* – hetke mängimise aeg video algusest alates, sekundites. Seda omadust võib muuta, siis mängimine jätkub uuendatud ajast.
- *duration* – video pikkus, sekundites. Kui videot pole, siis 0. Kui video pikkuse pole teada, siis NaN. Striimi puhul on see Inf.
- *ended* – boolean, näitab kas mängimine on lõppenud.
- *paused* – boolean, näitab kas mängimine on pandud pausile.
- *seeking* – boolean, näitab, kas hetkel toimub video sirvimine.
- *src* – tee meediaobjektile. Võib olla local path-ks või URL.

Meetodid:

- *canPlayType*() – stringi tagastav, kas seda tüübi faili saab mängida. Väärtused:{‘probably’, ‘maybe’, ‘’}.
- *load*() – hakkab faili allalaadimist. Ei tagasta midagi.
- *play*() – hakkab videot mängima. Ei tagasta midagi.
- *pause()* – peatab video mängimist. Ei tagasta midagi.

6.4 Video ja sensorandmete sünkroniseerimine: juhtimistarkvara osa

Kui juhtimistarkvara on käivitatud, siis kasutaja võib selle liides kaudu panna käima kaamerat, mille abil salvestatakse videot. Kui kasutaja liidese kaudu annab juhatuse lülitada kaamerat sisse, siis klient paneb kirja selle hetke täis ajatempel (Unix timestamp * 1000), mis salvestatakse andmefaili sisse järgmises formaadis: @s@[*ajatempel*] (vt sellest 4.2.2 ning

näideks on Lisas 1. toodud andmefaili väljatrükki 6. rida). Samamoodi, kui kasutaja annab juhatuse kaamerat peatada, siis salvestatakse andmefailisse täis ajatempel formaadis @e@[*ajatempel*].

Siin tekib selline probleem, et kaamera käivitamise juhatusandmise hetk ning video salvestamise alguse reaalne hetk ei ole lange kokku. Nende sündmuste vahel on kindlasti ajavahemik, mille suurus sõltub kasutatava arvuti parameetritest ning kaamera tüübist. Paljude katsete järel erinevatel arvutitel ning erinevate kaameratega selle ajavahemiku suurus oli vahemikus 2-5 s. Sellest järgneb üldine andmete sünkroniseerimine probleem: rakendus ei saa seisaku aega automaatselt tuvastada. Seega juhtimistarkvaras see pandud automaatselt 2 s. Aga täpsemaks kuvamiseks kasutaja peab olema võimeline ise graafiku nihutama, et rakendus arvestaks seda tegurit. Sellest räägitakse pp.6.5.2 – 6.5.3.

6.5 Video ja sensorandmete sünkroniseerimine: rakenduse osa

Player kui objekt. Et rakendus võiks lihtsam opereerida videofailiga rakenduses luuakse *Player* prototüübi objekt. Selle atribuudid ja meetodid on kirjeldatud Lisas 3. Kui kasutajaliidese rakendus käivitatakse, siis algatatakse ka aknale defineeritud meetod *window.localFileVideoPlayerInit()* (vt fail html5video.js), kus seostatakse faili valiku sisendväli meetodiga *playSelectedFileInit()*. Selle meetodi sees omakorda võetakse aknast (*window*) kui objektist valitud failide nimekirja, sellest nimekirjast võetakse esimese välja ning kontrollitakse, kas seda tüübi fail saab olla mängitud HTML5 mängija poolt. Ebaõnnestuse puhul kuvatakse veateadet ja rakendus peatub.

Kui faili tüüp sobib, siis selle faili tee omistatakse <video> kui HTML-elemendi *src* atribuudile, mis on rakenduses seotud *HTMLMediaElement* tüübi DOM-objektiga (“*#video*”).

Pärast seda DOM-struktuuris automaatselt luuakse muutuja *video*, millele väärtustatakse viide ülalmainitud HTMLMediaElement-ile. Edaspidi kõik operatsioonid *video* objektiga rakendus teeb selle muutuja vahendusel. Kui *video* on saadud kätte, siis toimub sündmus *loadedmetadata*, millega *Player* objektis on seotud anonüümne funktsioon, mis paneb video mängima ning kohe seda peatab. See on tehtud sellepärast, et mõned videofailid pärast *loadedmetadata* sündmuse ei tagastanud korrektseid video atribuutide väärtusi (nt. *video.duration* aeg-ajalt oli endiselt null).

Graafiku sünkroniseemine videoga. Nagu on juba eelnevalt räägitud (vt p. 5.3.1), selleks, et saada teada, millist graafiku osa on vaja joonistada, on vaja teada, milline on antud hetke video täis ajatempel millisekundites. See ajatempel on Unix-i aeg korrutatud 1000-ga (see peab olema millisekundites). HTML5 Video API ei võimalda rakendusel saada teada absoluutse kellaja. Seega kasutatakse selleks infot, mis on kirja pandud andmefaili sisse juhtimistarkvara poolt (vt p. 4.1.2). Sellele väärtusele lisatakse jooksva hetke kellaaega alates videoalgusest. Selle arvutamine toimub *Data.getTime()* meetodis. Kui *video.duration* väärtus on numbriline (st video on edukalt laaditud ja ei ole striim), siis ta võtab *Data.startCam*

atribuudi (kus on kirjas andmefailist võetud kaamera algusaeg) ning liidab selle arvuga *video.currentTime * 1000* (kuna *currentTime* atribuut on sekundites). See summa ongi tagastatav väärtus. Seda väärtust saadetakse meetodisse *Graph.draw()* kui hetke ajatempel (vt. 5.3.1).

Video sündmusega *play* rakenduses on seotud *Player.moveGraph()* meetod. Kõigepealt selles meetodis vahetatakse video mängimise ja peatamise kipplüliti liideses. Siis kui video on lõppenud, kutsutakse välja *Player.stopGraph()* meetod, mille sees veelkord kuvatakse graafik ning tühistatakse Player.ticker objekt, mis kuulub javascript Interval-i tüübi.

Kui video ei ole pausi pandud, siis *Player.ticker* atribuudile omistatakse uus Interval-i tüübi objekt, mis iga 33 ms kutsub välja *Graph.draw()* meetodi.

Nihe käsitsi määratlemine. P. 6.4. oli räägitud sellest, et kasutaja peab olema võimeline silma järgi määrata graafiku ja video vastavust. Selleks rakenduses on ettenähtud numbri sisestamise kast, mille vahendusel saab määrata graafiku nihutamise suurust ning nupp suunda muutmiseks. (vt Lisa 2. Kasutajaliidese näide G7).

Kui nende nihe suurus on muudetud, siis käivitatakse *GraphControls.setShift()* meetod, mis loeb kastis sisestatud numbri, kontrollib sisestamise korrektsust ning paneb selle *Graph.nihe* atribuudi väärtuseks. Selleks, et kasutaja võiks lehekülge uuendada ilma nihe suurust uuesti sisestamata kasutatakse selles rakenduses *localStorage* muutujat, mis on HTML5 Web Storage interface-i realisatsioon[10]. Siis toimub graafiku ümberjoonistamine *Graph.draw()* meetodi abil.

Kui kasutaja muudab nihe suunda, vastava nupule vajutades, siis kutsutakse välja *GraphControls.shiftDirTgl()* meetod. Meetod kontrollib, kas nupu 'alt'-atribuudi väärtus on 0 ja sel juhul paneb selle 1-ks, kui ei, siis paneb selle nulliks. Samuti ta valib vastava ikooni ning kutsub välja *GraphControls.setShift()* meetod (vt ülalpool).

Kui sensori andmetega fail on esmakordselt alla laaditud, siis kutsutakse välja *GraphControls.initShift()* meetod, mis loeb nihe suurust, mis on salvestatud Web Storage-is ning paneb kõik vaikimisi väärtused nihe kastidele.

Graafiku ja video sirvimine. Videoframe.js. Selles rakenduses saab sirvimist teostada kolme tüübi tegevuste kaudu:

1. Ekraanil HTML5 video sisseehitatud liidese osa on allpoolne riba, mille abil saab video sirvida (vt Lisa 2. Kasutajaliidese näide V2). Kui see riba kasutatakse, siis toimub *video* sündmus *seeking*, mis on meie rakenduses seotud javaskripti funktsiooniga. See viimane kutsub välja *Graph.draw()* meetodit, mis uuesti joonistab graafikut uue ajast alates (vt p. 5.3.1 kuidas see meetod saab video jooksva kellaaja).

[10] Vt (HTML5 Web Storage). Rakenduses ei kasutata cookie ega Indexed DB tehnoloogiaid andmete säilitamiseks lokaalselt, sest Google Chrome nende kasutamisel eeldab, et leht on avatud http-serveril, mitte lokaalselt. Kuna rakendus ei eelda serveri kasutamist, seega see ei sobi.

2. Sirvimisnupudega. Rakenduses on ette nähtud nupud, mis võimaldavad video täpsemat sirvimist. Nende abil võib määrata kaadrite hulka, mille kaupa sirvida video edasi või tagasi. (vt Lisa 2. Kasutajaliidese näide V5, V6). Siin tekkib üks probleem seoses sellega, et HTML5 API ei võimalda pöörduda video teatud kaadrile. Selle omadus *currentTime* selles mõttes on ebadiskreetne: see võtab vastu aeg sekundites (see võib olla kümmendmuru viisil) ning võtab kuvamiseks selleks ajaks lähima kaadri. Selleks, et lihtsustada kaadritele pöördumist, kasutatakse kolmanda poole teegi *Videoframe.js* (Sarkisyan, 2013). Kui rakenduses luuakse *Player* objekt, siis selle atribuudina luuakse ka *VideoFrame* prototüübi objekt *frames*, mille jaoks määratakse html-video elemendi id-d ning kaadrisadedust, mida pannakse 30 (=*FrameRates.web* konstant selle teegi sees). Siis kui vajutatakse ühele sirvimisnupudest, siis kutsutakse välja *VideoFrame.seekForward* meetodit, mille atribuudiks pannakse kaadrite arvu. See arv on korrutatud suunateguriga: kui vajutatud tagasisirvimise nupule, siis se tegur on -1.

3. Graafiku hiirega tõmmates kohatäitja alal (vt Lisa 2. Kasutajaliidese näide G5). Kui kasutaja klõpsab hiirega kohatäitja alal, siis töödeldakse selle DOM-elemendi *.mousedown* sündmus, nagu see on kirjeldatud *GraphControls.events()* meetodis, mis kutsutakse välja selle objekti loomisel. *.mousedown* sündmuse töötlemiseks on anonüümne funktsioon, mis paneb mällu kursori koordinaate klõpsu hetkel, omistab *GraphControls.clicked* atribuudile 1, ning paneb *video.seeking* omadus tõeseks.

   Hiire tõmbamine kohatäitja alal on ka *GraphControls.events()* meetodis kirjeldatud sündmus, mis kutsub välja funktsiooni, mis esialgu kontrollib, kas klõpsu koordinaadid on olemas ja kas hiir on klõpsatud. Kui jah, siis kui shift ei ole vajutatud, ta arvutab välja liigutamise kiiruse tegurit (Graph.pikkus / 1000), siis paneb mällu jooksva kellaja, ning iga 5 ms ta arvutab vahe jooksva kursori ja selle algpositsiooni. Siis meetod kutsub välja *Graph.draw()* meetodit, kuhu argumendina saadetakse see vahe. Kui sellele meetodile on esimene argument saadetud, siis ta liidab selle väärtuse video jooksva hetke väärtusele.

   Hiire vabastamine ning hiire graafiku kohatäitja alalt eemaleviimine kui sündmused tekitavad *GraphControls.stopMovingGraph()* meetodi väljakutsumist. Kui shift mitte vajutatud, siis *video.currentTime* omadusele liidetakse hiire liigutamise pikkus x-teljel jagatud 1000-ga, graafik joonistatakse uuesti (nüüd ilma argumendita: algushetkeks võetakse muudetud *video.currentTime* omadus).

7. Rakenduse katsetamine

Rakenduse arendamine toimus iteratsiooniliselt. Iga iteratsiooni käigus tehtud etappe oli testitud ELIKO-s töötavate testijate poolt. Järgnevalt me vaatame lõpliku rakenduse kasutajaliidest ja kommenteerime seda, millised vead ja probleemid olid tuvastatud selle testimise käigus ja nende lahendamisviisid.

7.1 Valmis rakenduse kasutajaliides

Lõpliku rakenduse kasutajaliides vt joonisel allpool (Joonis 10).

Joonis 10. Kasutajaliidese pilt

7.2 Valmis kasutajaliidese rakenduse testimine

Rakenduse automaattestimist ei toimunud. Testimise käigus olid tuvastatud järgmised vead ja probleemid:

1. Esialgselt primaarseks veebilehitsejaks oli kavandatud kasutada Ms Internet Explorer-it HTML5 toetava veebilehitsejate asemel. (Internet Explorer-i aktuaalne versioon oli IE8). Videofaili mängimiseks plaaniti kasutada ActiveX objekti, mis käivitaks opsüsteemi sisseehitatud MediaPlayer-it. Sel lahendusel aga olid osutunud puudused:

a. sõltuvus opsüsteemist: rakendus saab käivitada ainult arvutitel, kuhu on Ms Windows paigaldatud.

b. turvalisuse seadistused olid liiga keerulised kasutaja jaoks: selleks, et IE-s lubada ActiveX objektide kasutamine on vaja muuta vaikimise määranguid veebilehitsejas, mis on tavakasutajal päris raske leida ning see tekitab palju hoiatussõnumeid, mis ehmatavad kasutajat.

c. samas see turvalisusseadistuste muutmiste tegi arvuti kaitsmine lahjaks. Tulemusena ELIKO-s otsustati Internet Explorer-it mitte kasutada.

2. Testimise käigus selgus, et Mozilla Firefox-i 2013. a. versioonid ei toeta mp4 formaadi (mis oli arutatud Peatükis 6). Kui kasutati tahvelarvuti, kus oli paigaldatud Android 4.3 opsüsteem, siis Firefox mängis video ainult .ogv formaadis. Seega oli otsustatud keskenduda Google Chrome-i kasutamisele.

3. Samas Android 4.3 opsüsteemiga tahvelarvutis ei õnnestunud Chrome brauseriga videot käivitada. Selle põhjused on tundmatu.

4. IPad tahvelarvutis ei õnnestunud rakendust käivitama, mille põhjuseks on piirangud lokaalfailidega tööks selles seadmes.

5. Rakenduse esialgse variandi järgi oli plaanis, et luuakse kaks eraldi faili, kuhu juhtimistarkvara paneb eraldi sensoriandmeid ja videosalvestuse algus- ja lõppaega. Testimise käigus otsustati, et see variant on kasutajale ebamugav, sest tal tuleb kokku rakenduse käivitamisel avada kolm faili: videofaili, sensorite andmefaili ning video sünkroniseerimisfaili.

6. Testimise käigus oli tuvastatud parsimise viga: kui kasutaja valis uue andmefaili vana järjest, siis rakendus ei kustutanud vana andmeid ära (andmed olid lisatud olemasolevatele *Data.parsed* objekti andmetele. Viga oli parandatud.

7. Rakendus oli katsetatud erinevate usb-kaameratega k.a. kõrgresolutsiooniga (HD) kaameraga, mille kaadrisagedus oli 100 fps. Salvestus ja mängimine läks edukalt. Olid mõned probleemid ühe Lenovo sülearvutiga sisseehitatud veebikaameraga – pilt tuli, opsüsteemis paigaldatud videomängija poolt video avanes, aga HTML5 API abil brauseris seda mängida ei õnnestunud. Põhjus oli tundmatu.

8. Töö oli katsetatud mitu Eesti treenerite poolt. Rakendus oli katsetatud maksimaalselt 3 sensoriga. Katsetamise käigus selgus, et kasutajal peab olema võimalus käsitsi graafiku nihe määrata. Rakendus oli täiendatud nihe suuruse ja suunda sisestamiskastiga (vt p. 6.5).

8. Kokkuvõte

Selle töö põhieesmärgiks oli projekteerida rakendust, mis võimaldaks kuvada liikumissensoritest tulevaid andmeid ning sünkroniseerida selle videoga, kus on salvestatud liikuva sportlase tegevused. Andmeid peaksid olema kuvatud dünaamiliste graafikute viisil. Ülesanne püstitamisel põhinõudena oli arvestatud see, et rakendus käivituks veebilehitsejas ilma serveriühendust HTML-lehega. Sellega oli tingitud töövahendite valik: skriptimiskeelega javaskript / jQuery, raamistikuna HTML5 API.

Järgnevalt autori poolt on tehtud järgmisi samme eesmärkide saavutamiseks.

1. Olid analüüsitud rakenduse nõudeid ja skoobi ning läbi mõeldud rakenduse arhitektuurile. Sellest lähtuvalt kasutajaliidese tarkvara oli tükeldatud neljaks klassideks, mille ülesanded põhijoontes vastavad MVC mustrile.
2. On kirjeldatud rakenduse iga klassi loomisel tekkinuid probleeme ning võimalusi, mida HTML5 API-s olemas nende probleemide lahendamiseks. Olulisemateks probleemideks on lokaalsete failide lugemine javaskripti abil, video avamine ja mängimine.
3. Rakenduse klassid olid implementeeritud. Implementeerimise käigus olulisimaks probleemiks oli videosalvestuse sünkroniseerimine graafikuga. Selleks oli vaja saada teada, kuidas saab kätte ajatempli mpeg-4 konteinerist ning kuidas sisse kirjutada sisendfailisse info salvestusajast. Probleem oli lahendatud, aga sünkroniseerimise viga on jäänud ikka, sest juhtimistarkvara jaoks oli võimatu tuvastada videosalvestuse alguse reaalse ajahetke.
4. Süsteem oli katsetatud kergejõustikutreenerite ja –sportlaste poolt.

Töö põhitulemuseks on see, et projekteeritud rakendus oli loodud, testitud ja katsetatud. Töö käigus tekkinud probleeme, nagu lokaalsete failide lugemine ja video sünkroniseerimine graafikuga olid enamasti lahendatud.

Vaatamata sellele, et sageli javaskripti vaadeldakse kui skriptimiskeelt, mis kasutatakse liidese osana ja mille abil ei saa äriloogikat teostada, meie rakenduses on esistletud täisfunktsionaalne rakendus, mis põhikeelena kasutab javaskripti.

Sensoriandmete kuvamissüsteemide nõudmine on suur. Seega võimalike edasiarendusi on palju. Mõned on nö “laiuti” edasiarendused, nt

1. Graafikute võrdlemine: erinevate sporditegevuste andmeid võib kõrvuti panna, et võrrelda sportlas(t)e tegevusi vajalikutel füüsikalistel parameetritel.
2. Graafikute kuvamine reaalaajas: mitte failist, vaid pesast (*socket*) tuleva video- ja andmevoogu töötlemine nõuab erirakendust.
3. Harjutuste kuvamine ühe kaupa ja nende võrdlemine. Oleks kasulik rakendus, mis võimaldaks eristada lühikesed harjutused, mis on üksteisega sarnased ning võrrelda neid omavahel.

Võib esile tõsta ka nö “sügavuti” edasiarendusi.

1. Kõigepealt see on suunatud video sünkroniseerimise probleemi lahendamisele. Siin võiks pakkuda lahendusi, selleks, et sünkroniseerimine toimuks ilma kasutaja vahenduseta.
2. Üks probleem, mis jäi hetkel lahendamata on video kaadrisageduste automaatne tuvastamine. Praegu see on vaikimisi pandud 30 fps ja selle muutmiseks tuleb lähtekoodi muuta. See ei mõjuta graafiku ja video sünkroniseerimist, aga video sirvimine kaaderhaaval teistel kaadrisagedustel saab ebakorrektseks.
3. Veel üks edasiarendusnõutav aspekt on jõudluse optimeerimine. Kui andmesalvestus läheb pikaks, siis andmeobjekt läheb suureks ja see mõjutab jõudlust negatiivselt. Selle lahendamiseks tuleb realiseerida andmete osadena lugemist. Veel üks jõudlust mõjutavaid teguritest on andmefailist salvestatud sensorite arv. Kui nende hulk on suur, siis FlotCharts objekti töökiirus langeb, mis mõjutab graafikute kuvamist ning manipuleerimist.

Teiseks edasiarendamise suunaks on teiste tüübi kaamerate, esialgu käekaamerate kasutamine.

Kokkuvõttes saab väita et autoril õnnestus realiseerida töötavat rakendust, mis vastab ette pandud nõuetele, mis on kompaktne ja käib veebibrauseris.

Summary

The title of thesis: **„Displaying Data Received from Motion Sensors and Synchronising this with Video through Use of HTML5 API and Javascript“**

The main purpose of this thesis is to design an HTML5 API based application that can display input data incoming from motion sensors as dynamic graphs and to find out a way to synchronize this with the video recording.

This application will be used in the athletes' training. There is a need to compare the video of exercises with data received from the motion sensors. The application receives the data from the *.csv* file. Then it should be displayed in a comprehensible form, as a graphs. The trainer must be able to see the data obtained from the sensor with an accuracy of one videoframe. One of the requirements for this application is that it should be opened in a web browser and still operate as a standalone application.

One of most important problems was reading data from a local file using javascript and HTML5 API. Since native javascript has no standard input nor output, we have to use other means. For the standalone application the HTML5 FileReader API is most appropriate solution.

Another one is the problem of the video recording's synchronization with graphs displayed. The application should relate timestamps obtained from a sensors data file with video timecodes. This is done using the synchronization record in the data file. This solution is partial, because this record has a slight shift in time because we can not determine the exact time of the camera starting.

During this research the application requirements and architecture are analyzed and this application is splitted into modules and classes. Also the description of problems connected to each application's module is provided and found some API opportunities to solve these problems. Then application classes and modules are implemented and after that the final software was tested and improved.

The main outcome of this work is ready application for motion sensors synchronization, it's code, description of objects and data formats used in the application. The application has been tested by in-house developers and tested by practitioners coaches.

This work may be extended to improve this app. This work can be extended to improve the app. In particular, to achieve precise synchronization, the frame rate automatically determining and performance optimizing with increasing amounts of input data.

In addition, this application can be taken as a basis for developing similar products, which would allow to compare graphics, to display video and sensors data graphs in real-time, consistently show and compare individual exercises etc.

Kasutatud kirjandus

1. AVC Patent Portfolio License Briefing. *MPEGLA*. [WWW]. http://www.mpegla.com/main/programs/avc/Documents/avcweb.pdf (22.05.2014).
2. Bibeault, B., Katz, Y. (2010). jQuery in Action: Second Edition. Stamford: Manning Publications Co.
3. Bidelman, E. (2010). Reading files in JavaScript using the File APIs. [WWW]. http://www.html5rocks.com/en/tutorials/file/dndfiles/ (06.03.2014).
4. Bio-MEMS. *Wikipedia, the free Encyclopedia*. [WWW]. http://en.wikipedia.org/wiki/Bio-MEMS (27.05.2014).
5. Bootstrap. [WWW]. http://getbootstrap.com/2.3.2/ (27.05.2014).
6. DiPaola, D. (2013). BIOMEMS 2013: The Revolutionary Change in Sports from MEMS and Sensor Enabled Products. [WWW]. http://www.dceams.com/Assets/biomems%202013%20mems%20and%20sensor%20enabled%20products%20for%20sporting%20applications%20db%202-5-2012.pdf (22.05.2014).
7. Eich, B. (2012). Video, Mobile, and the Open Web. Mozilla Hacks. [WWW]. https://hacks.mozilla.org/2012/03/video-mobile-and-the-open-web/ (22.05.2014).
8. Flanagan, D. (2011). JavaScript: The Definitive Guide, 6th Edition. Activate Your Web Pages. Sebastopol: O'Reilly Media, Inc.
9. Flanagan, D. (2014). HTMLMediaElement. *Mozilla Developer Network.* [WWW]. https://developer.mozilla.org/en-US/docs/Web/API/HTMLMediaElement (22.05.2014).
10. Flotcharts. [WWW]. http://www.flotcharts.org/ (27.05.2014).
11. Fowler, M. (2014a). Model View Controller. [WWW]. http://martinfowler.com/eaaCatalog/modelViewController.html (05.05.2014).
12. Google and MPEG LA Announce Agreement Covering VP8 Video Format. (2013). [WWW]. http://www.mpegla.com/Lists/MPEG%20LA%20News%20List/Attachments/88/n-13-03-07.pdf (22.05.2014).
13. Google Closes On2 Technologies Acquisition. (2010). *Google Investor Relations.* [WWW]. http://investor.google.com/releases/2010/0219.html (22.05.2014).
14. Greenwald, W. (2010). Kinect vs. PlayStation Move vs. Wii: Motion-Control Showdown. [WWW]. *PCMag.* http://www.pcmag.com/article2/0,2817,2372244,00.asp (23.05.2014).

15. Hachamovitch, D. (2011). HTML5 Video Update—WebM for IE9. *IEBlog*. [WWW]. http://blogs.msdn.com/b/ie/archive/2011/03/16/html5-video-update-webm-for-ie9.aspx (22.05.2014).

16. HTML5 Web Storage. *W3Schools.com.* [WWW]. http://www.w3schools.com/html/html5_webstorage.asp (22.05.2014).

17. Hudson, C. (2013). CSV-1203. CSV File Format Specification. United Kingdom: mastpoint.com. [WWW] http://mastpoint.curzonnassau.com/csv-1203/csv-1203.pdf

18. Introduction to Object-Oriented JavaScript. (2014). *Mozilla Developer Network.* [WWW] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Introduction_to_Object-Oriented_JavaScript (5.05.2014).

19. ISO/IEC 14496-10:2012 Information technology — Coding of audio-visual objects — Part 10: Advanced Video Coding. [WWW]. http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=61490

20. Jack, K. (2001). Video demystified: a handbook for the digital engineer. Eagle Rock: LLH Technology Publishing.

21. JaegerMonkey. (2010). MozillaWiki. [WWW] https://wiki.mozilla.org/JaegerMonkey (12.03.2014).

22. Jeremie. (2014). Media events. *Mozilla Developer Network.* [WWW]. https://developer.mozilla.org/en-US/docs/Web/Guide/Events/Media_events

23. Keller, H. (2000). 30 Years of Passive Infrared Motion Detectors – a Technology Review. *Reprint from plenary talk at OPTO/IRS2 Erfurt Germany May 11, 2000.* [WWW]. http://www.kube.ch/downloads/pdf/kube_irs2paper.pdf (23.05.2014).

24. Kleine-König, U. (2014). Flot Reference. *GitHub*. [WWW]. https://github.com/flot/flot/blob/master/API.md (27.05.2014).

25. Kulikov, D. (2011). MPEG-4 AVC/H.264 Video Codecs Comparison. CS MSU Graphics&Media Lab, Video Group. [WWW]. http://www.compression.ru/video/codec_comparison/h264_2011/mpeg-4_avc_h264_video_codecs_comparison.pdf (21.05.2014).

26. Li, C. (2014). Microwave noncontact motion sensing and analysis. Hoboken: John Wiley & Sons.

27. License. jQuery foundation. [WWW]. https://jquery.org/license/ (20.05.2014).

28. Methvin, D. (2012). jQuery Core: Version 1.9 and Beyond. [WWW]. http://blog.jquery.com/2012/06/28/jquery-core-version-1-9-and-beyond/ (20.05.2014).

29. InvenSense Inc. (2013). MPU-6000 and MPU-6050 Product Specification Revision 3.4. [WWW]. http://www.invensense.com/mems/gyro/documents/PS-MPU-6000A-00v3.4.pdf

30. Nihtianov, S., Luque, A. (2013). Smart Sensors and MEMS: Intelligent Devices And Microsystems For Industrial Applications. Cambridge: Woodhead Publishing Limited.

31. Ozer, J. (2010). VP8 vs. H.264. [WWW]. http://www.streamingmedia.com/conferences/west2010/presentations/SMWest-2010-H264-VP8.pdf (21.05.2014).

32. Ozer, J. (2010a). Ogg vs H264 - Round One. [WWW]. http://www.streaminglearningcenter.com/articles/ogg-vs-h264---round-one.html (21.05.2014).

33. Pfeiffer, S. (2010). The Definitive Guide to HTML5 Video. New York: Springer Science+Business Media, LLC.

34. Paul, R. (2011). Google Reveals Plan To Remove H.264 Support From Chrome. *Wired.* [WWW]. http://www.wired.com/2011/01/google-reveals-plan-to-remove-h-264-support-from-chrome/ (21.05.2014).

35. Sarkisyan, A. (2013). VideoFrame. [WWW]. https://github.com/allensarkisyan/VideoFrame (22.05.2014).

36. Shaver. (2010). HTML5 video and codecs: [WWW]. http://shaver.off.net/diary/2010/01/23/html5-video-and-codecs/ (06.03.2014).

37. Shmitt, C., Sympson, K. (2012). HTML5 Cookbok. Sebastopol: O'Reilly Media, Inc.

38. Shy, L. (2014). Band Together: 10 Gadgets That Track Your Fitness Stats. [WWW]. http://www.fitsugar.com/Comparison-Nike-FuelBand-FitBit-Jawbone-Up-More-21429516#photo-21429522 (25.05.2014).

39. Silverwind. (2014). Media formats supported by the HTML audio and video elements. [WWW]. https://developer.mozilla.org/ru/docs/HTML/Supported_media_formats (22.05.2014).

40. SpiderMonkey. (2014). Mozilla Developer Network. [WWW] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/SpiderMonkey (12.03.2014).

41. Sullivan, G. J., Topiwala, P., Luthra, A. (2004). The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions. *SPIE Conference on Applications of Digital Image Processing XXVII Special Session on Advances in the New Emerging Standard: H.264/AVC, August, 2004*. [WWW]. http://www.fastvdo.com/spie04/spie04-h264OverviewPaper.pdf (22.05.2014).

42. The Ogg container format. [WWW]. http://www.xiph.org/ogg (21.05.2014).

43. Unix time. *Wikipedia, the free Encyclopedia*.

44. Using files from web applications. (2014). Mozilla Developer Network. [WWW] https://developer.mozilla.org/en-US/docs/Using_files_from_web_applications (19.03.2014).

45. Full Flight Technology. Velocitip ballistic system. [WWW]. http://www.velocitip.com/ (23.05.2014).
46. W3C. (2013a). HTML5. A vocabulary and associated APIs for HTML and XHTML. W3C Candidate Recommendation 6 August 2013. [WWW] http://www.w3.org/TR/2013/CR-html5-20130806/ (06.03.2014).
47. W3C. (2013b). File API. W3C Last Call Working Draft 12 September 2013. [WWW] http://www.w3.org/TR/FileAPI/ (19.03.2014).
48. W3C. (2014). HTML5. A vocabulary and associated APIs for HTML and XHTML. Editor's Draft 6 March 2014. [WWW] http://www.w3.org/html/wg/drafts/html/CR/ (06.03.2014).
49. Walsh, D. (2013). JS Objects: Inherited a Mess. [WWW]. http://davidwalsh.name/javascript-objects (5.05.2014).
50. WebM Container Guidelines. *WebM: an open web media project.* [WWW]. http://www.webmproject.org/code/specs/container/ (22.05.2014).
51. Wilson, C. (2012). Performance Tips for JavaScript in V8. [WWW] http://www.html5rocks.com/en/tutorials/speed/v8/ (12.03.2014).
52. XMLHttpRequest (2012). Living Standard — Last Updated 14 February 2014. [WWW] http://xhr.spec.whatwg.org/ (13.03.2014).

Lingid sensorsüsteemide resurssidele

1. **94Fifty** – http://www.94fifty.com/
2. **Adidas Micoach** – http://micoach.adidas.com/
3. **ADXL202/ADXL210** – http://www.analog.com/static/imported-files/data_sheets/ADXL202_210.pdf
4. **Bodymedia** – http://www.bodymedia.com/
5. **Cannondale Simon Active Mountain Bike Suspension** –
6. **CAPTIV-L7000** – http://teaergo.com/drupal/en/products/manufacturers/tea/captiv-software
7. **Cycling Studio** – http://www.stt-systems.com/en/products/motion-capture/sports/cycling/
8. **Goal Line Tracking (GLT) System** – http://www.cairos.com/?page_id=28
9. **Golf Studio** – http://www.stt-systems.com/en/products/motion-capture/sports/golf/
10. **InThePool** – http://www.stt-systems.com/fotos/menus_archivos/sheet_inthepool.pdf
11. **Nike Hyperdunk+** – http://www.nike.com/us/en_us/c/basketball/nike-basketball-hyperdunk-plus
12. **Mobile Eye-XG** – http://www.asleyetracking.com/Site/Products/MobileEyeXG

13. **Movea MotionPod** – http://www.movea.com
14. **Play & Connect** – http://en.babolatplay.com/
15. **Riddell InSite** – http://www.riddell.com/innovation
16. **Rowing in Motion** – http://www.rowinginmotion.com/
17. **Smart Garment** – http://www.smartlifetech.com/technology/
18. **STT** – http://www.stt-systems.com/en/
19. **Tekscan Grip™** – http://www.tekscan.com/grip-pressure-measurement
20. **VELOCITIP Arrow** – http://www.velocitip.com/
21. **X2Impact Mouth Guard** – http://www.x2bio.com/x2_x_guard/
22. **XSENS MVN Motion Capture Suit** – http://www.xsens.com/products/mvn-biomech/
23. **Zepp** – http://www.zepp.com/

Lisa 1. Algandmete näide

faili nimi: kuus_kanalit Aug 21 16.15.csv

faili sisu:

```
1. absolute    timestamp,node    id,Acc.x,    Acc.y,Acc.z,Gyr.x,Gyr.y,Gyr.z,node
   id,Acc.x,Acc.y,Acc.z,Gyr.x,Gyr.y,Gyr.z\r\n
2. 1377090933159,1,498,76,-9406,-180,-63,-207,2,376,170,-7276,-259,-185,-59\r\n
3. 1377090933179,1,340,58,-9344,-175,-59,-207,2,298,106,-7220,-266,-185,-56\r\n
4. 1377090933199,1,406,136,-9432,-165,-44,-206,2,278,92,-7296,-264,-187,-58\r\n
5. 1377090933219,1,400,38,-9388,-164,-38,-211,2,250,136,-7330,-265,-188,-52\r\n
6. @s@1377090933227\r\n
7. 1377090933239,1,440,12,-9304,-161,-39,-205,2,292,80,-7292,-265,-194,-57\r\n
8. 1377090933259,1,358,34,-9390,-170,-46,-211,2,296,108,-7386,-269,-192,-55\r\n
9. 1377090933279,1,356,68,-9204,-174,-57,-206,2,312,114,-7408,-267,-194,-59\r\n
10. 1377090933299,1,460,22,-9286,-173,-56,-203,2,274,88,-7194,-267,-188,-56\r\n
11. 1377090933319,1,354,-20,-9472,-168,-57,-202,2,236,120,-7298,-267,-197,-55\r\n
12. 1377090933339,1,420,24,-9288,-166,-47,-202,2,278,88,-7318,-272,-201,-50\r\n
13. 1377090933359,1,348,6,-9442,-166,-40,-207,2,306,130,-7266,-268,-202,-56\r\n
14. 1377090933379,1,416,38,-9502,-168,-45,-207,2,262,66,-7286,-262,-192,-56\r\n
15. 1377090933399,1,382,-16,-9396,-170,-52,-206,2,210,56,-7260,-255,-173,-55\r\n
16. 1377090933419,1,318,40,-9444,-171,-54,-206,2,218,52,-7256,-253,-170,-58\r\n
17. 1377090933439,1,446,96,-9300,-170,-49,-207,2,270,106,-7260,-260,-176,-59\r\n
18. 1377090933459,1,374,62,-9384,-172,-52,-209,2,252,170,-7402,-273,-209,-59\r\n
19. 1377090933479,1,392,40,-9270,-171,-47,-204,2,286,36,-7234,-285,-224,-56\r\n
20. 1377090933499,1,374,52,-9438,-169,-46,-207,2,292,86,-7328,-280,-205,-51\r\n
21. 1377090933519,1,446,80,-9398,-168,-53,-208,2,184,0,-7300,-268,-197,-56\r\n
22. 1377090933539,1,428,34,-9310,-164,-45,-205,2,202,64,-7206,-259,-183,-57\r\n
23. 1377090933559,1,416,-38,-9304,-163,-44,-209,2,308,90,-7348,-246,-162,-57\r\n
24. 1377090933579,1,380,6,-9312,-164,-43,-205,2,348,98,-7258,-239,-147,-54\r\n
25. 1377090933599,1,436,2,-9358,-169,-51,-208,2,252,118,-7196,-247,-168,-54\r\n
26. 1377090933619,1,408,40,-9318,-178,-62,-205,2,242,86,-7278,-278,-205,-56\r\n
27. 1377090933639,1,384,10,-9226,-174,-63,-207,2,328,90,-7310,-298,-233,-56\r\n
28. 1377090933659,1,424,-2,-9390,-163,-45,-208,2,292,64,-7328,-287,-230,-56\r\n
29. 1377090933679,1,450,16,-9468,-156,-30,-197,2,298,130,-7330,-272,-197,-52\r\n
30. @e@1377090933687\r\n
31. 1377090933699,1,434,38,-9566,-153,-24,-202,2,252,96,-7298,-260,-180,-54\r\n
32. 1377090933719,1,434,0,-9430,-177,-58,-204,2,282,10,-7354,-260,-177,-53\r\n
33. 1377090933739,1,448,28,-9440,-191,-81,-207,2,270,10,-7154,-263,-182,-57\r\n
34. 1377090933759,1,414,90,-9372,-179,-65,-212,2,296,70,-7356,-253,-169,-55\r\n
```

Lisa 2. Kasutajaliidese näide

Lisa 2: leppemärgid

Video osa	**Graafiku osa**
V0 – videofaili valiku nupp	**G0** – andmefaili valiku nupp
V1 – sisseehitatud HTML5 video lõuend	**G1** – sensorite kuvamiseks valiku kastid
V2 – sisseehitatud HTML5 video juhtimisnupud	**G2** – füüsiliste parameetrite kuvamiseks valiku kastid
V3 – video start / pause nupp	**G3** – kõverate x-teljel skaleerimise nupud
V4 – video peatamisnupp	**G4** – kõverate y-teljel skaleerimise nupud
V5 – video tagasirullimise nupp kaaderhaaval	**G5** – lõend graafikuga
V6 – video edasirullimise nupp kaaderhaaval	**G6** – keskjoon ja jooksva hetke ajatempel
V7 – kaadrite arvu sätestamise sisend kaaderhaaval sirvimiseks	**G7** – kõverate nihutamise sisend
	G8 – sensori sageduse detektor
V8 – jooksva kaadri ja video aega detektor	**G9** – graafiku kuvamise viisi valiku kastid

Lisa 3. Rakenduse klasside, atribuutide ja meetodite kirjeldus

Data

- Selle klassi objektis säilitakse andmeid, mis on saadud failist. Kõik andmeid säilitakse operatiivmälus.
  Klassi objektidel on järgmised **atribuudid**:
  - *.FIELDS_NUM* – siia pannakse iga sensori parameetrite arv parsimisel. See arv on konstantne kõikide sensorite puhul ühe rakenduse seanssi jooksul.
  - *.columns* – .csv-faili põhiosa väljade massiiv.
  - *.minY*, *.maxY* – kõvera maksimaalne ja minimaalne väärtused y-teljel (andmefailist)
  - *.korgMax*, *.korgMin* – graafiku maksimaalne ja minimaalne väärtused y-teljel
  - *.stopCam* – kaamera peatamise aboluutne aeg (millisekundites)
  - *.startCam* – kaamera startimise absoluutne aeg (millisekundites)
  - *.xtrms* – massiiv kuhu salvestatakse kõik kõverate ekstreemumid
  - *.sagedus* – sensori signaalidevaheline intervaal
  - *.isNew* – uuesti avatud faili triger.
  - *.korgVahe* – vahe korgMax'i ja korgMin'i vahel
  - *.parsed* – objekt, kuhu salvestatakse kõik failist saadud andmeid. Selle objekti struktuurist vt
  - *.sensorsData* – massiiv, kuhu pannakse kogu sensorandmetefail ridade kaupa

  Selle klassi **meetodid**:
  - *.resout* – väärtustab *sensorsData* atribuudi argumendist saadud massiviga. Paneb *isNew* atribuudi õigeks ning kutsub parseri meetodit *FileToData*.
  - *.fileToData* – parsib andmeid massiivis *sensorsData*. Tulemusena on atribuut *parsed*. Samuti leiab sünkroniseerimise ridu ning kutsub *setSync* meetodit. Väärtustatakse atribuute: *columns, FIELDS_NUM, xtrms, minY, maxY*.
  - *.setMinMax* – sätestab graafiku maksimaalse ja minimaalse masstaapi. Väärtustab atribuute *korgMin* ja *korgMax*.
  - *.tickY* – arvutab, mis on graafiku horisontaalsete abijoontevaheline interval.
  - *.getFieldsNum* – arvutab ja väärtustab atribuudi FIELDS_NUM. Argumendina on välade nimede massiiv.
  - .setSync – argumendiks on tekstirida, millest parsitakse välja *startCam* ja *stopCam* atribuudi väärtuse
  - *.getTime* – tagastab video praeguse hetke algusest möödunud aega millisekundites.

Graph

- Graph – selle klassi objekt hoiab ning töötleb sensorite andmeid sel viisil, kui nad on vajalikud FlotChart teegi API jaoks. Selle meetodid võimaldavad joonte andmeid lisada, eemaldada ning ümber joonistada HTML Canvas'el.
  Klassi objektidel on järgmised **atribuudid**:
  - *_this* – pseudoatribuut viidega klassi eksemplarile (vt ülalpool lk 14).
  - *.xCenter* – koordinaat x-teljel, kus asub graafiku kesk. Seal joonistatakse keskjoon.
  - *.data* – viit andmeobjektile andmetega failist.
  - *.ticker* – id väärtus, mille abil kutsutakse ja peatatakse akna intervaal.
  - *.nihe* – atribuut, kuhu salvestatakse kõverate nihe. Algseisuks võtakse algne andmete ja video sünkroniseerimine (nihe = 0). Kasutaja võib seda parameetri muuta.
  - *.SITO_TIHEDUS* – see atribuut kasutatakse selleks, et vähendada kõverate iteratsioonide arvu (jõudluse optimiseerimiseks).
  - *.pikkus* – graafiku pikkus. Ühikuteks on kirjed andmefailis. Kuna sensorite sagedus erineb, siis selle atribuudi seos reaalse ajaga võib muutuda.
  - *.series* – massiiv, kuhu salvestatakse kõverate massiivid FlorCharti jaoks sobivas formaadis.
  - *.options* – masiiv, kus hoiakse info kõverate seadistuste kohta.
  - *.barOptions* – massiiv, kus hoiakse seadistused selleks, et kuvada graafiku tulbade viisil.
  - *.colors* – kõverate värvide massiiv

  Selle klassi **meetodid**:
  - *.draw* – kuvab selle hetke kõveraid javaskripti lõuendil (kasutatakse FlotChart teegi API). **Argumentideks** on mittekohustuslik nihe (shift) parameeter. Meetod kontrollib, millised sensorid ja nende parameetrid on sisse lülitatud, valmistab ette nende parameetride andmete massiive (järgmise meetodi *extractCurve* abil), valib andmete kuvamise viisi (kõverad / tulbad) ja värvi ning kuvab neid.
  - *.extractCurve* – see meetod tagastab antud sensori antud parameetri andmete massiivi kõvera kuvamiseks. Omab 4 kohustuslikku **argumendi**: sensor, parameeter, algusaeg, pikkus
  - *.extractBar* – see meetod tagastab antud sensori antud parameetri andmete massivi tulba kuvamiseks. Omab 3 kohustuslikku **argumendi**: sensor, parameeter, hetk, millal andmeid näidatakse.
  - *.skaleerHoriz* – meetod salvestab selle objekti uue pikkuse väärtuse ning kuvab uuendatud graafiku. **Argumendiks** on väärtus, mida liidetakse olemasolevaga kõvera pikkusega.
  - *.skaleerHorCycle* – kutsub javaskripti *setInterval* meetod ning selle abil regulaarselt kutsutakse välja selle Graph objekti *skaleerHoriz* meetod. **Argumendiks** on kõvera pikkuse liidetav.

- *.skalVerMax* – muudab graafiku ülemise väärtuse y-teljel. **Argumendiks** on selle väärtuse liidetav.
- *.skalVerMin* – muudab graafiku alumise väärtuse y-teljel. **Argumendiks** on selle väärtuse liidetav.
- *.setVerMax* – see kutsutakse kohe pärast andmefaili parsimist, et sätestama graafiku ülemise väärtuse y-teljel ning intervali horisontaalsete abijoonte vahel graafikul.
- *.setVerMin* – see kutsutakse kohe pärast andmefaili parsimist, et sätestama graafiku alumise väärtuse y-teljel ning intervali horisontaalsete abijoonte vahel graafikul.
- *.setVer* – kutsub välja järjest *setVerMax* ja *setVerMin* meetodit.
- *.skaleerHorRelease* – see puhastab intervalli, mis oli kutsutud meetodis *skaleerHorCycle* ning peatab *skaleerHoriz* meetodi regulaarset väljakutsumist.
- *.skalVerCycle* – kutsub javascripti setInterval meetod ning selle abil regulaarselt kutsutakse välja selle Graph objekti *skalVerMax* ja *skalVerMin* meetodid. **Argumendiks** on kõvera pikkuse liidetav.
- *.getType* – tuvastab seda, kas kasutaja poolt on valitud andmete kuvamine kõverate või tulbade viisil.

GraphControls

- **GraphControls** – see klass luuakse selleks, et seadistada elemente kasutajaliidese tasemel ning töödelda selle taseme sündmusi. Ainult ühe selle klassi objekti on võimalik luua rakenduse elutsükli jooksul.
  Klassi objektidel on järgmised **atribuudid**:
  - *.graph* – see on viide graafiku objektile, millele kuuluvad selle klassi juhtimisseadmed
  - *.data* – viide selle klassiga seotud andmeobjektile.
  - *.shifted* – boolean väärtus, mis väärtustatakse tõega kui *shift* nupp on vajutatud ning valega kui see vabastatakse.
  - *.clicked* – boolean väärtus, mis väärtustatakse tõega kui hiire vasak nupp on vajutatud ning valega kui see vabastatakse.
  - *.hover* – boolean väärtus, mis väärtustatakse tõega kui hiire kursoor asub on vajutatud ning valega kui see vabastatakse.
  - *.click* – objekt, kuhu salvestatakse info kliki jooksul. selle objekti atribuudiks on “dist”, kus salvestatakse klikitud ja tõmmatud tee pikkus.
  - *.scale* – objekt, kuhu salvestatakse info sellest, milline graafiku masstaap kasutatakse x-teljel ja y-teljel.
  - *.drag* – objekt, kuhu salvestatakse info hiire koordinaatide kohta.

  Selle klassi **meetodid**:
  - *.openDataFile* – see meetod kutsutakse välja kui muudetakse liidese file andmesisestamise väli (vt. 3.4). Argumendina on faili valiku sündmus.

- *.skaleeri* – see meetod kutsutakse välja kohe pärast lehe allalaadimist. Selles meetodis kirjeldatakse, kuidas töödelda skaleerimisnupude ja veel mõni sündmusi.
- *.hover*, *.hout* – kirjeldab skaleerimisnupude stiili
- *.events* – kirjeldab, kuidas töödeldakse erinevaid sündmusi (vt XXXX)
- *.stopMovingGraph* – peatab graafiku horisontaalse või vertikaalse (kui shift nupp on vajutatud) liigutamist. Argumendina on hiire vabastamine kui sündmus.
- *.boxes* – see funktsioon joonistab ruudud, mille vajutamisel võib kuvada või peita sensori parameetre. Argumentideks on sensori id ning sensorite arv.
- *.allBxs* – funktsioon lülitab sisse või välja (toggle) kõik antud sensori parameetrite kõveraid.
- *.getShiftDir* – tagastab kõverate nihe suuna.
- *.shiftDirTgl* – muudab nihe suuna vastupidi
- *.setShiftDir* – sätestab nihe suuna sõltuvalt argumendist *s*.
- *.setShift* – sätestab nihe väärtuseks kasutaja poolt liidese kaudu sisestatud väärtust.
- *.initShift* – sätestab nihe väärtuse ja suuna rakenduse käivitamisel. Viimase käivitamise nihet võetakse cookie'st.
- *.setFreq* – kuvab kasutajaliideses antud sensori sagedus (see arvutatakse Data objektis andmete parsimise jooksul).

Player

- Player – selle klassi objekti abil teostatakse video kuvamise juhtimist ning selle sünkroniseerimine graafikuga.
  Klassi objektidel on järgmised **atribuudid**:
  - *.graph* – viide aktiivsele graafiku objektile (Graph)
  - *.ticker* – muutuja setInterval-i meetodi jaoks (kasutatakse video mängimise ajal: vt meetodit *moveGraph*).
  - *.bPlay* – viide nupule, mis alustab ja lõpetab video mängimist.
  - *.frames* – uus VideoFrame klassi objekt, mis on vaja video sirvimiseks.

  Selle klassi **meetodid**:
  - *.bTgl* – meetod tuvastab, kas video hetkel mängitakse ning sellest sõltuvalt kuvab bPlay nupul sobivat pilti.
  - *.moveGraph* – iga 33 ms võtab video ajatemplit, leiab selle ajatemplile vastavaid andmeid data.parsed objektist ning kuvab seda graafikul.
  - *.stopGraph* – puhastab intervalli (vt ticker atribuut), ning peatab videot.
  - *.stop* – paneb video algusesse ning peatab selle.
  - *.start* – sõltuvalt sellest, kas video hetkel mängib, peatab selle või paneb käima.
  - *.seek* – sirvib video antud kaadrite arvu kaupa.
  - *.showCurFr* – kuvab kasutajale info praeguse kaadri kohta: selle numbri ja suhtelist aega.