Mikko Soininvaara

Sensoripohjaisten eleiden koodikirjasto Android-sovelluskehitysympäristöön

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

14.5.2013

Metropolia

Tekijä(t) Otsikko Sivumäärä Aika	Mikko Soininvaara Sensoripohjaisten eleiden koodikirjasto Android-sovelluskehitysympäristöön 37 sivua + 4 liitettä 14.5.2013
Tutkinto	Insinööri (AMK)
Koulutusohjelma	Tietotekniikka
Suuntautumisvaihtoehto	Ohjelmistotekniikka
Ohjaaja(t)	Lehtori Peter Hjort Koulutusohjelmavastaava Markku Karhu

Työssä suunniteltiin liike-eleitä Android-sovelluskehitysympäristöön sekä luotiin koodikirjasto, jonka avulla sovelluskehittäjät voivat lisätä nämä eleet omiin sovelluksiinsa kuten muut näytön asettelussa ilmoitetut komponentit.

Koska sensoreiden toimintaa sekä niiden pohjalta luotuja liike-eleitä ei ole liiemmin käsitelty eri kirjallisuuslähteissä, oli työ mielenkiintoinen ja haastava.

Eleet tunnistettiin käyttämällä kiihtyvyysanturia, gyroskooppia ja kiertovektorisensoria. Magnetometriä käytettiin laitteen asentoa laskettaessa kiertovektorisensorin toimesta.

Kun eleet sisältävä Gesture-luokka oli valmis, lisättiin kustomoidut attribuutit, kuten eleen valinta sekä Shake-eleen herkkyys XML-tiedostoon *attrs.xml*. Tämän jälkeen eleet pystyttiin esittelemään näkymän asettelussa ja etsimään ne omalla tunnisteellaan testisovelluksen aktiviteetissä.

Eleiden tunnistaminen ei ollut täysin aukotonta, mutta kokonaisuus toimi kuitenkin kelvollisesti ja vakaasti, joten tuloksiin oltiin kokonaisuutena tyytyväisiä. Tärkeimpänä asiana pidettiin että nyt rakennettu kehys eleiden helpolle käyttämiselle oli luotu ja siinä onnistuttiin hyvin. Valmis koodikirjasto käsittää *Gesture.java*- sekä *attrs.xml*-tiedoston.

Avainsanat	Android, sensorit, ele, Java, XML

Author(s) Title Number of Pages Date	Mikko Soininvaara Code library for sensor-based gestures in Android software development environment 37 pages + 4 appendices 14 May 2013
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Software Engineering
Instructor(s)	Peter Hjort, Senior Lecturer Markku Karhu, Head of Degree Programme

The purpose of this project was to design motion-based gestures for the Android platform and to collect them in a code library. These gestures were designed to be used in the same way as other Android layout-components so that software developers can easily implement them in their own applications.

The sensors used in this project were an accelerometer, gyroscope, geomagnetic field sensor and rotation vector sensor. These sensors monitored the orientation and movement of the device. The methods and listeners that recognized gestures were in one Java class.

Finally, custom attributes, such as choice for gesture and sensitivity for Shake, were created in attrs.xml so that users could call them in their layout file and give them unique ID tags. With these ID tags users can locate components in their application activities.

Detecting gestures was not flawless and there could be a better way to monitor device movements and orientation, but the most important thing was that this project provides a framework, or some sort of a guide, to create sensor-based gestures and implement these easy to use components for an application, and in terms of that, this the project was a success. The final code library contains Gesture.java and attrs.xml files.

Keywords	Android, Sensors, Gesture, Java, XML

Sisällys

Lyhenteet

API	*Application Programming Interface*. Ohjelmointirajapinta, joka sisältää kuvaukset luokkien määrittelylle ja niiden käyttäytymiselle.
Java	Sun Microsystemsin kehittämä olio-ohjelmointikieli ja ohjelmistoalusta.
MEMS	*Micro-Electro-Mechanical Systems*. Pienimmillään muutaman mikrometrin kokoon rakennettuja sähkömekaanisia järjestelmiä.
SDK	*Software Development Kit*. Kokoelma työkaluja ohjelmistoalustalle tehtävään sovelluskehitykseen.
XML	*Extensible Markup Language*. World Wide Web Consortiumin kehittämä tekstipohjainen merkintäkieli tiedon jäsenneltyyn esittämiseen.

1 Johdanto

Kosketusnäytöllisten älypuhelinten vallankumous alkoi Applen lanseerattua markkinoille iPhonen vuonna 2007 [1]. Toki puhelimia, joissa toimintoja ohjataan kosketuksella, oli kokeiltu markkinoilla jo muun muassa Nokian toimesta sen harvemmin tunnetuissa malleissa [2], mutta nämä eivät suurta menestystä osakseen saaneet.

Mikä iPhonen käyttöliittymästä teki ylivoimaisen muihin aiempiin yrityksiin, oli sen yksinkertaisuus ja helppokäyttöisyys. Esimerkiksi tutut numeronäppäimet, joilla puhelimella kirjoitetaan tai näppäillään numeroita, oli tehty ohjelmallisesti näkymään puhelimen näytöllä.

Uusia ominaisuuksia oli myös puhelimen näytön kääntäminen pystyasennosta vaakatasoon, mikä käänsi puhelimessa näkyvän kuvan tai internet-sivun orientaatiota ja joka oli sekin mullistava uutuus puhelinteknologiassa ja käyttöliittymäsuunnitellussa.

Ensimmäisessä iPhone-versiossa oli puhelimeen sisäänrakennettuina kiihtyvyysanturi, etäisyysanturi sekä ympäristön valoisuuden tunnistin, joita käyttämällä saatiin uudenlainen käyttäjäkokemus sekä samalla puhelimen toimintaa auttavia ominaisuuksia parannettua. Etäisyysanturi tunnisti, kun käyttäjä asetti puhelimen korvalleen, jolloin näyttö sammui akun säästämiseksi sekä puhumisen aikana tahattomat hipaisut näyttöön estyivät [1].

Kun muut puhelinvalmistajat huomasivat, että tällaiselle tekniikalle käyttöliittymäsuunnitelussa puhelimiin oli kysyntää, alkoivat he pian tuoda markkinoille omia mallejaan, joita ohjattiin tällä uudella tavalla sensoreita apuna käyttäen. Google osti ja jatkokehitti Androidia vastaukseksi iPhonen iOS:n käyttöjärjestelmälle ja samaan aikaan Nokia kehitti omia innovaatioitaan. Huolimatta eri valmistajien käyttöjärjestelmistä oli sensoriteknologia tullut älypuhelimiin jäädäkseen.

Jossain vaiheessa kesken tätä älypuhelinten vallankumousta mietittiin erilaisia keinoja käyttää puhelimen toimintoja muuten kuin perinteisellä tavalla, joten kokeiltiin eleitä, jotka suunniteltiin käyttämään hyväksi puhelinten kosketusnäyttötekniikkaa.

Näitä kuvioiden piirtämiseen näytölle perustuvia eleitä ei kuitenkaan suuri käyttäjäkunta osannut ottaa omakseen, eikä tällaisesta tavasta käyttää puhelinta ole koskaan, ainakaan vielä, tullut mikään trendi.

Vasta hiljattain on tullut markkinoille puhelimia, joissa tiettyjä puhelimen toimintoja ohjataan liike-elein, kuten esimerkiksi puhelimen musiikkisoittimen soittolistaa voi sekoittaa eleellä tai vaikkapa mykistää laite kääntällä se väärinpäin pöydälle.

Näitä eleitä tunnistetaan erilaisin sensorein, jotka havainnoivat laitteen liikettä ja asentoa. Ne voivat myös tarkastella ympäristön olosuhteita, kuten valoisuutta tai ilmanpainetta. Näitä samoja sensoreita käyttävät myös eri mobiilipelit hyväkseen, ja ehkä suurin sensoreita ohjelmissaan käyttävä ryhmä onkin pelinkehittäjät.

Sensoreiden toimintaa sekä niiden pohjalta luotuja liike-eleitä ei ole käsitelty muuta kuin harvakseltaan ja pintapuolisesti eri kirjallisuuslähteissä. Myöskään sovelluskehittäjille ei ole vielä tehty helppokäyttöistä tapaa käyttää eleitä omissa sovelluksissaan, joten aihepiiri on harvinaisuutensa vuoksi jännittävä, ja on mielenkiintoista tutkia sensoreiden käyttöä pintaa syvemmältä sekä oppia enemmän niiden käytöstä sovelluskehityksessä.

Työn tavoitteena on suunnitella muutama yleiskäyttöinen sensoripohjainen ele käyttäen mahdollisuuksien mukaan erityyppisiä sensoreita ja pyrkiä tekemään niistä mahdollisimman helppokäyttöiset muiden sovelluskehittäjien käytettäväksi omissa ohjelmissaan.

Tämän jälkeen kokonaisuus voidaan halutessa koota koodikirjastoksi niin, että sen voi laittaa yleisesti Android-yhteisön saataville internetiin.

2 Sovelluskehitys Androidille

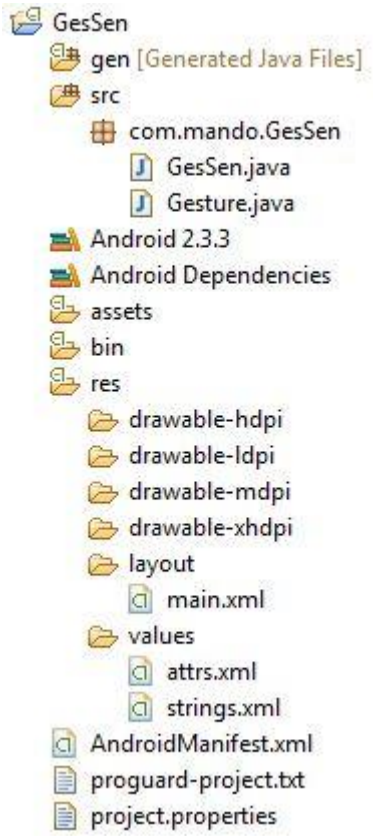
Android on alun perin Android Inc:n kehittämä ja Googlen ostama ja jatkokehittämä käyttöjärjestelmä ja ohjelmistoalusta mobiililaitteille [3]. Se pohjautuu Linuxin ytimeen, ohjelmistokirjastoihin sekä Java-pohjaiseen Dalvikin virtuaalikoneeseen, joka ajaa sovellusten ohjelmakoodia.

Ensimmäinen Android versio 1.0 näki päivänvalon 23. syyskuuta 2008, jonka jälkeen Google on julkaissut uusia käyttöjärjestelmäversioita 3–5 vuoden välein. Viimeisin vakaa versio on 4.2.2 *Jelly Bean* [3].

Androidin sovellukset kirjoitetaan Java-ohjelmointikielellä ja käännetään Androidin omilla sovelluskehitystyökaluilla. Nämä sovellukset ovat *apk*-päätteisiä paketteja, joissa on ohjelmakoodin lisäksi sovelluksessa käytettävät kuvat, animaatiot sekä XML-tyylitiedostot.

Sovellukset voidaan jakaa neljään eri sovelluskomponenttiin: aktiviteetteihin (*activities*), palveluihin (*services*), sisällöntarjoajiin (*content providers*) sekä lähetysvastaanottimiin (*broadcast receivers*), joista jokaisella komponentilla järjestelmässä on omanlaisensa elinkaari, joka määrittää miten komponentti luodaan ja lopetetaan järjestelmässä [4].

Sovellus voi koostua yksinkertaisimmillaan yhdestä sovelluskomponentista, manifestista (*AndroidManifest.xml*) sekä resursseista. Manifestissa ilmoitetaan käytettävistä sovelluskomponenteista, kuten aktiviteeteista, ja resursseihin sisällytetään sovelluksen tyylitiedostot XML-muodossa sekä mahdolliset kuvat ja animaatiot [4].



Kuva 1. Android-sovelluksen ohjelmistorakenne.

Kuvassa 1 on nähtävissä ohjelmistorakenne yksinkertaisesta sovelluksesta. Käytetty esimerkki on tämän insinöörityön testisovelluksesta. *GesSen.java* on aktiviteetti ja *Gesture.java* puolestaan on eleitä varten luotu java-luokka. Myös *attrs.xml* on tätä työtä varten luotu tiedosto, jota käsitellään luvussa 5.1.

Muut kuin edellä mainitut sovelluksen osat ovat sovelluskehitystyökalun automaattisesti luotuja tiedostoja, joita tarvitaan ohjelmakoodin kääntämisessä sovellukseksi.

On monia sovellustyökaluja, joilla luotu ohjelmakoodi saadaan paketoitua valmiiksi sovelluksesi. Tässä työssä käytettiin sovelluskehitysympäristönä Eclipseä, johon asennettiin ADT-lisäosa sekä Androidin SDK-työkalupaketti.

3 Sensorit

3.1 Liike-, ympäristö- ja paikkasensorit

Monissa Android-laitteissa on sisäänrakennettuina sensorit, jotka voivat mitata laitteen liikettä sekä asentoa ja vaikkapa tarkastella ympäristön olosuhteita. Nämä sensorit voidaan jakaa kolmeen ryhmään:

- Liikesensorit mittaavat laitteen kiihtyvyydestä sekä pyörähtämisestä aiheutuneita voimia.
- Ympäristösensorit tarkastelevat vallitsevan ympäristön olosuhteita, kuten ilman lämpötilaa ja painetta, valoisuutta sekä ilman kosteutta.
- Paikkasensorit tarkastelevat laitteen fyysistä asentoa.

Sensoreita voidaan käyttää Androidin sensorikehyksen kautta, jonka avulla saadaan muun muassa tietää, mitä sensoreita laitteessa on sekä saadaan yksityiskohtaista tietoa jokaisesta laitteen sensorista. Tämän kehyksen kautta saadaan myös käytettäväksi jokaisen sensorin antama käsittelemätön data sekä voidaan rekisteröidä sensoreiden tapahtumankuuntelijoita, jotka tarkkailevat sensorin muutoksia [5].

Vaikkakin eri laitteiden välillä voi olla eroja saatavilla olevista sensoreista, on hyvä huomioida myös Android alustan päivitetyt sensoriluokat. Kuten kuvasta 2 Android Developer-sivustolta voidaan huomata, on luokkia lisätty alustan kehityksen myötä sekä luovuttu toisista ja päivitetty parempiin [5].

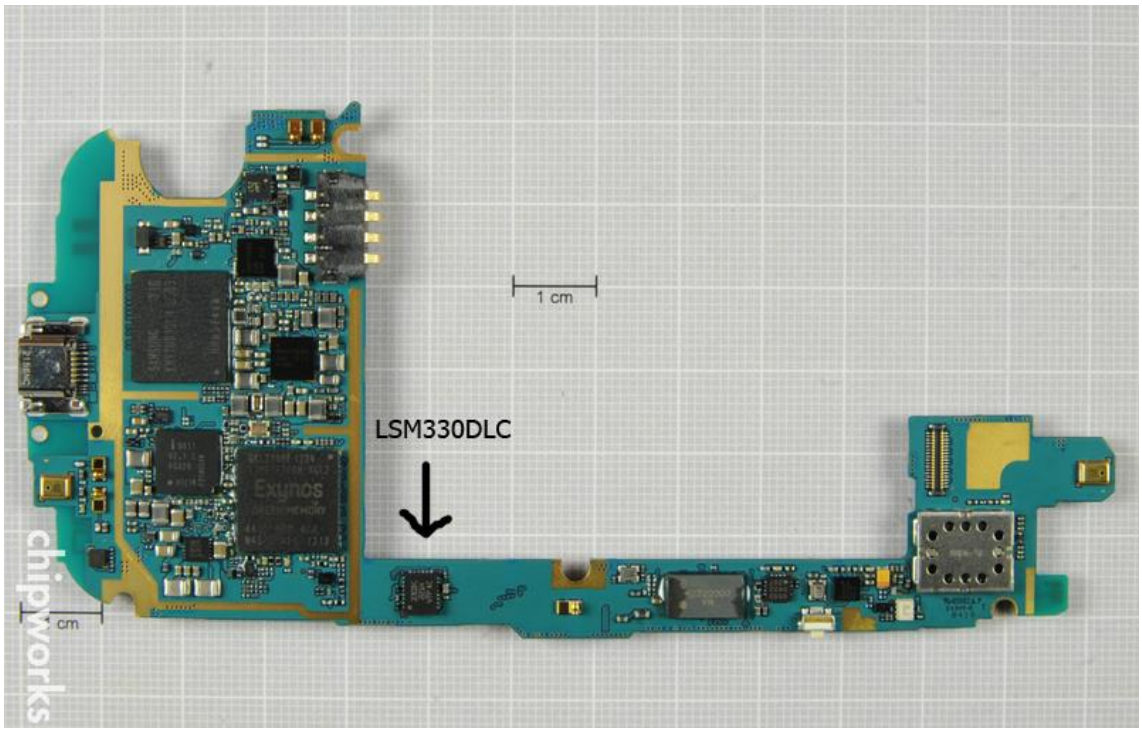
Sensor	Android 4.0 (API Level 14)	Android 2.3 (API Level 9)	Android 2.2 (API Level 8)	Android 1.5 (API Level 3)
TYPE_ACCELEROMETER	Yes	Yes	Yes	Yes
TYPE_AMBIENT_TEMPERATURE	Yes	n/a	n/a	n/a
TYPE_GRAVITY	Yes	Yes	n/a	n/a
TYPE_GYROSCOPE	Yes	Yes	n/a[1]	n/a[1]
TYPE_LIGHT	Yes	Yes	Yes	Yes
TYPE_LINEAR_ACCELERATION	Yes	Yes	n/a	n/a
TYPE_MAGNETIC_FIELD	Yes	Yes	Yes	Yes
TYPE_ORIENTATION	Yes[2]	Yes[2]	Yes[2]	Yes
TYPE_PRESSURE	Yes	Yes	n/a[1]	n/a[1]
TYPE_PROXIMITY	Yes	Yes	Yes	Yes
TYPE_RELATIVE_HUMIDITY	Yes	n/a	n/a	n/a
TYPE_ROTATION_VECTOR	Yes	Yes	n/a	n/a
TYPE_TEMPERATURE	Yes[2]	Yes	Yes	Yes

[1] This sensor type was added in Android 1.5 (API Level 3), but it was not available for use until Android 2.3 (API Level 9).

[2] This sensor is available, but it has been deprecated.

Kuva 2. Sensoreiden saatavuus eri Android alustaversioissa [5].

Tässä insinöörityössä käytettiin kiihtyvyysanturia (*TYPE_ACCELEROMETER*), gyroskooppia (*TYPE_GYROSCOPE*), kiertovektorisensoria (*TYPE_ROTATION_VECTOR*) sekä magnetometriä (*TYPE_MAGNETIC_FIELD*) ja testilaitteena Samsung Galaxy S3-puhelinta, jossa on STMicrocontrollerin valmistama MEMS-sensorimoduuli LSM330DLC (kuva 3). Tämä moduuli sisältää 3D-kiihtyvyysanturin ja 3D-gyroskoopin yhteen mikropiiriin asennettuna [6].

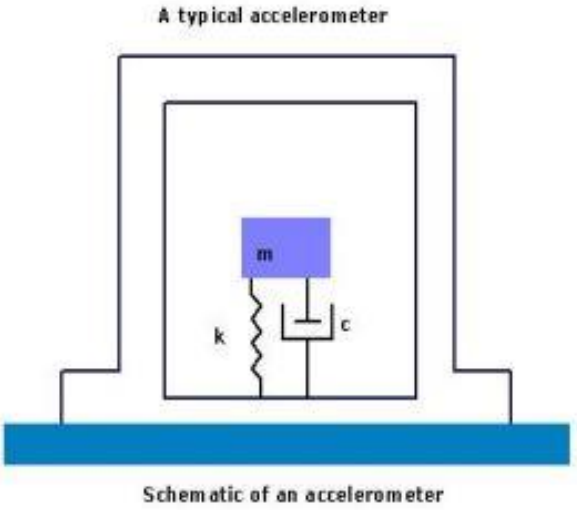


Kuva 3. Samsung Galaxy S3:n emolevy ja MEMS moduuli [6].

Kiihtyvyysanturi luokitellaan liikesensoreihin, joiden avulla tarkkaillaan laitteen kolmelle akselille saamaa kiihtyvyyttä m/s^2. Painovoiman vaikutusta ei poisteta arvoista, joten jos laite on esimerkiksi pöydällä paikoillaan, laitteella on kiihtyvyys 9.81 m/s^2. Laitteen taas ollessa vapaassa pudotuksessa kiihtyvyys on 0 m/s^2 [7].

Tämän moduulin kiihtyvyysanturin tarkan mittausalueen maksimiarvot saa valmistajan mukaan säädettyä neljälle eri kiihtyvyysalueelle. Lineaarisen kiihtyvyyden luotettavan lukematarkkuuden mittausalueet ovat ±*2 g*, ±4 *g*, ±8 *g* tai ±16 *g*, jossa merkintää *g* käytetään anturin kiihtyvyydestä. Anturin herkkyys on ilmoitettu sensorimoduulin teknisissä tiedoissa eri maksimitarkkuuden asetuksen perusteella 1, 2, 4 tai 12 m*g*/digitin herkkyydelle [8].

Koska jokaiselle kolmelle akselille mitataan kiihtyvyyttä, on kiihtyvyysantureita kolme. Kuvassa 4 on esitetty tyypillinen kiihtyvyysanturin rakenne, jossa massa *m* liikkuu jousen varassa, jolla on tietty jousivakio *k*. Tätä massaa vaimennetaan jollain tietyllä vaimennuskertoimella *c*. Näiden arvojen perusteella saadaan laskettua kiihtyvyyden suunta ja voimakkuus [9, luku 2].



Kuva 4. Tyypillisen kiihtyvyysanturin toimintaperiaate [9, luku 2].

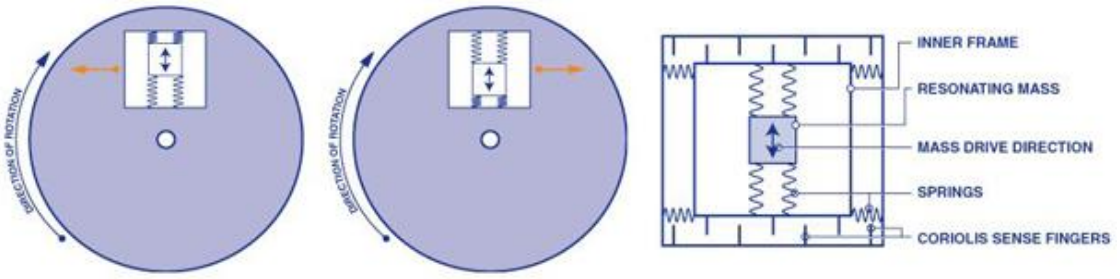
Gyroskooppi tarkkailee laitteen kiertymistä akseleiden suhteen ja se ilmaistaan radiaaneina sekunnissa. Kiertonopeus on positiivista kun akselia kierretään vastapäivään. Koordinaatisto on sama kuin kiihtyvyysanturissa (luku 3.2) [10].

MEMS-moduuleissa käytetään värähtelyrakenteisia gyroskooppeja, joiden rakenne sopii hyvin mobiililaitteille pienen koonsa puolesta [11].

Gyroskoopit toimivat käyttäen hyväksi koriolisvoiman periaatteita, joiden avulla pystytään tarkkailemaan kiihtyvyyden muutoksia. Koriolisvoima on selitetty Insinöörin fysiikka osa 1 kirjassa [12] seuraavasti:

> Maapallon pyörimisestä aiheutuu maapallon suhteen liikkuvaan kappaleeseen sen nopeuteen suoraan verrannollinen näennäisvoima, jota kutsutaan koriolisvoimaksi. Koriolisvoiman takia ammuttu luoti ei lennä suoraan, vaan kaartuu sivulle. Samasta syystä tuulen suunta kaartuu ja matalapainekeskusta lähestyttäessä ilmavirtaukset alkavat kiertää kyseistä keskusta.

Tietenkään laitteen gyroskooppi ei tarkkaile maapallon pyörimisestä johtuvia kiihtyvyyksiä vaan laitteen omia kulmakiihtyvyyksiä jokaiselle akselille. Gyroskoopin anturikehys on asennettu pyörivään alustaan, ja massa, johon nämä korioliskiihtyvyydet kohdistuu, liikkuu jousien varassa alustalla (kuva 5).



Kuva 5. Värähtelyrakenteisen gyroskoopin toimintaperiaate [11].

Samsung Galaxy S3:ssa gyroskoopin antaman kulmakiihtyvyyden tarkan mittausalueen rajat voivat olla joko ±250, ±500 tai ±2000 asteessa sekunnissa (dps) ja kulmakiihtyvyyden herkkyys taas 8,75, 17,50 tai 70 mdps/digit [8].

Lämpötilavaihtelut voivat aiheuttaa hyvin pientä poikkeamaa sensoreiden tarkkuudessa tarkkuusrajojen valinnasta riippuen [8].

Kiertovektorisensorin avulla saa selville laitteen asennon. Sensori, joka ei ole mikään fyysinen sensori, vaan käyttää laskuissaan kiihtyvyysanturia, gyroskooppia sekä magnetometriä, antaa jokaiselle laitteen akselille kiertovektorikomponentin. Tämä komponentti on yksikötön arvo [13]. Koordinaattijärjestelmä poikkeaa kiihtyvyysanturin ja gyroskoopin järjestelmästä (luku 3.2).

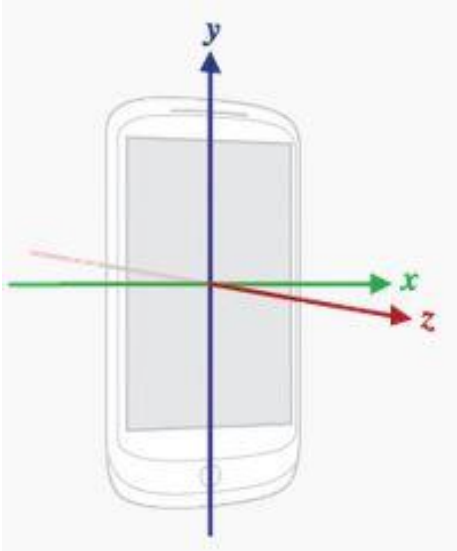
Akseleiden tavanomaisten vektorikomponenttien lisäksi kiertovektorisensorista saadaan haluttaessa neljäskin arvo, joka on skalaarikomponentti kiertovektorista [13].

Magnetometri tarkkailee maapallon magneettikentän muutoksia. Arvot annetaan mikroteslоina (μT) [14]. Tätä sensoria ei yleisesti tarvita yksinään mihinkään, mutta yhdessä kiihtyvyysanturin sekä kiertovektorisensorin kanssa saadaan kiertomatriisi. Tällä kiertomatriisilla saadaan selville laitteen asento ympäröivän maailman suhteen.

3.2 Sensoreiden koordinaattijärjestelmä

Sensoreiden koordinaattijärjestelmä käyttää kolmiakselista esitystapaa. Kun laite asetetaan pystyasentoon, X-akseli osoittaa oikealle, Y-akseli ylöspäin ja Z-akseli laitteen näytöstä kohtisuoraan poispäin (kuva 6). Tätä koordinaattijärjestelmää käytetään ylei-

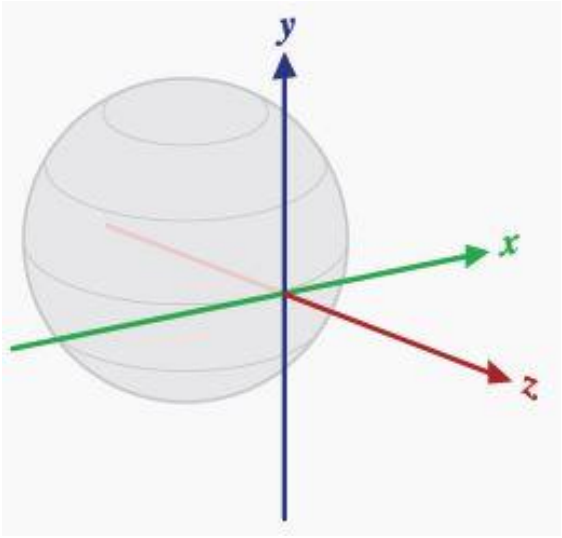
sesti liikesensoreissa, kuten kiihtyvyysanturissa, painovoimasensorissa, gyroskoopissa, lineaarisessa kiihtyvyysanturissa sekä magnetometrissä [15].



Kuva 6. Liikesensoreiden koordinaattijärjestelmä [5].

Koordinaattijärjestelmästä on tärkeää huomioida se, että kun laitteen näyttöä kiertää pystyasennosta (*portrait*) vaakatasoon (*landscape*), eivät akseleiden paikat vaihdu eikä koordinaattijärjestelmä muutu [15].

Kaikki sensorit eivät kuitenkaan käytä samaa esitystapaa kuin liikesensorit, vaan ilmaisevat laitteen asentoa suhteessa maapalloon (kuva 7). Tällaisia sensoreita ovat Orientation-sensori sekä kiertovektorisensori [15].



Kuva 7. Paikkasensoreiden koordinaattijärjestelmä [13].

Kuvan 7 koordinaattijärjestelmän mukaan Y-akseli osoittaa magneettiselle pohjoisnavalle ja Z-akseli kohtisuoraan taivaalle. X-akselin suunta lasketaan näiden kahden vektorin vektoritulona Y x Z [13].

3.3 Sensoreiden liittäminen sovellukseen

Jotta sensorit saadaan liitettyä sovellukseen, luodaan instanssi SensorManagerista. Tämän luokan metodeilla päästään käsiksi kaikkiin laitteen sensoreihin sekä voidaan esimerkiksi tarkistaa, mitä sensoreita laitteessa on [16].

SensorManagerin avulla kutsutaan haluttua sensoria, kuten esimerkkikoodi 1:n tapauksessa kiihtyvyysanturia (esimerkkikoodi 1), ja jos laitteesta sellainen löytyy, luodaan siitä Sensor-luokan ilmentymä.

```
mSensorManager = (SensorManager)getSystemService(SENSOR_SERVICE);
mAccelerometer =
        mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Esimerkkikoodi 1. SensorManager- ja Sensor-luokan instanssin luominen.

Kun haluttu sensori on löydetty ja sen instanssi luotu, pitää seuraavaksi asettaa tapahtumankuuntelija SensorManager-luokan metodin SensorManager.registerListener() avulla, jotta sensorin saa toimintaan. Esimerkkikoodissa 2 tähän metodiin laitetaan

parametreiksi SensorEventListener, haluttu sensori sekä se, millä nopeudella se lukee olosuhteiden muutoksia. Tässä esimerkin tapauksessa asetetaan SensorEventListener-parametri *this*-muodossa, koska kyseinen rajapinta on implementoitu sovelluksen luokkaan.

```
mSensorManager.registerListener(this, mAccelerometer, SensorManag-
er.SENSOR_DELAY_NORMAL);
```

Esimerkkikoodi 2. Sensorin tapahtumankuuntelijan rekisteröiminen.

Nopeuden voi itse määritellä millisekunteina, mutta SensorManager-luokassa on valmiina kolme oletusarvoa eri tarkoituksia varten [16]:

- SENSOR_DELAY_FASTEST, jolloin dataa luetaan mahdollisimman nopeasti
- SENSOR_DELAY_GAME, jolloin dataa luetaan peleille sopivalla nopeudella
- SENSOR_DELAY_NORMAL, jolloin nopeus on sopiva laitteen asennon tarkkailemiselle
- SENSOR_DELAY_UI, jolloin nopeus on käyttöliittymälle sopiva.

Kun tarvittavat sensorit on luotu ja niiden kuuntelijat on rekisteröity halutuilla parametreillä, aloitetaan niistä lukemaan tietoja. Tämä tapahtuu SensorEventListener-rajapinnan takaisinkutsumetodin OnSensorChangedin kautta, jota käsitellään tarkemmin luvussa 4.2.2.

3.4 Käytetyimpiä liike-eleitä

Liikepohjaisia eleitä ei ole juurikaan käytetty älypuhelimissa laitevalmistajien toimesta. Muutamia harvoja eleeksi luettavia toimintoja käytetään kiihtyvyysanturin avulla muun muassa Applen iPhone 4s:ssä, jossa musiikkikirjaston sekoittamisen voi tehdä laitetta ravistamalla [17]. Samsung Galaxy S3:ssa ravistusta taas käytetään internetsivun päivittämiseen.

Toinen yleisempi ele on puhelimen nostaminen pöydältä, jotta tapahtuu toiminto. Tälläisiä toimintoja on esimerkiksi HTC One X:ssä soittoäänen voimakkuuden laskeminen tai Samsung Galaxy S3:ssa soitetaan näytöllä olevalle kontaktille [18].

Muita käytössä olevia eleitä ovat laitteen kääntäminen näyttö ylös- tai alaspäin, laitteen yläreunan kevyt naputus sormilla sekä liikkeen tunnistaminen esimerkiksi, jos laite on laukussa tai taskussa [18;19].

4 Eleet

4.1 Suunnittelu

Uusia eleitä suunniteltaessa huomioitiin ensisijaisesti niiden käytön yksinkertaisuus. Haluttiin myös käyttää monipuolisesti eri sensoreita jokaisessa eleessä, koska eri valmistajien laitteissa ei aina ole kaikkia mahdollisia sensoreita saatavilla.

Yksi eleistä suunniteltiin niin, että kun laite asetetaan pöydälle tai muulle tasaiselle alustalle, tapahtuu takaisinkutsu. Mikä laitteen orientaatio alustalla on, ilmoitetaan takaisinkutsussa id-numerolla. Tätä elettä kutsuttiin nimellä Flipflop.

Koska tässä eleessä oli tarpeen tarkastella laitteen asentoa, suunniteltiin käytettäväksi kiertovektorisensoria. Asennon olisi pystynyt määrittämään myös Orientation-sensorilla, mutta kyseinen sensori on vanhentunut – viimeisin tuettu versio oli Android 2.2 –, joten tähän työhön sitä ei haluttu käyttää [20].

Toinen eleistä on Shake eli ravistus. Laitetta ravistetaan kaksi kertaa, jonka jälkeen tapahtuu takaisinkutsu.

Tätä elettä suunniteltaessa mietittiin tarkemmin sitä, pitääkö laitteen olla tietyssä asennossa, kun ravistus tapahtuu. Ajatuksesta luovuttiin, koska ele mahdollisti yksinkertaisen toteutuksen pelkkää kiihtyvyysanturia käyttäen, joten ele soveltuisi siten monelle eri laitteelle. Kuitenkin haluttiin, että ravistus tapahtuu tiettyyn suuntaan, joten kiihtyvyyttä tulisi tarkastella tietyn akselin suuntaisesti.

Kolmas ele Twotap suunniteltiin toimimaan niin, että kun laitetta pidetään kädessä ja heilautetaan kaksi kertaa laitteen yläreunaa alaspäin, niin että laite kiertyy kädessä tietyllä voimakkuudella, tapahtuu eleessä takaisinkutsu. Tässä eleessä voitaisiin käyttää gyroskooppia.

4.2 Gesture-luokka

4.2.1 Tapahtumankuuntelija

Koska eleitä haluttiin käyttää samalla tavalla kuin Androidissa muitakin komponentteja, kuten nappeja, luotiin Gesture-luokkaan rajapinta jokaiselle eleelle.

Ensimmäisessä versiossa oli vain yksi rajapinta, josta luotiin jokaiselle eleelle takaisinkutsumetodit, vaikka vain yhtä elettä käytettäisiin. Tämä muutettiin lopulliseen versioon niin, että jokaiselle eleelle saatiin oma rajapinta, jota kautta ohjelmaan saadaan takaisinkutsu-metodit. Näin saatiin yksinkertaistettua eleiden käyttöä ja niiden rekisteröimistä, eikä käyttämättömiä metodeja implementoida turhaan.

4.2.2 Tiedot sensoreilta

Jotta sensoreilta saataisiin tietoa, implementoitiin Gesture-luokkaan SensorEventListener-rajapinta. Tämän rajapinnan kautta päästään käsiksi kahteen takaisinkutsumetodiin OnAccuracyChange ja OnSensorChanged [21].

OnAccuracyChange-takaisinkutsussa on kaksi parametriä: metodia kutsuneen sensorin tyyppi sekä sensorin antaman lukeman tarkkuus [16].

Lukeman tarkkuus kertoo, kuinka luotettavasti sensori on pystynyt tarkkailemaan olosuhteiden muutoksista, jotka ilmoitetaan neljällä tavalla:

- SENSOR_STATUS_ACCURACY_HIGH, jolloin sensori ilmoittaa datan suurimmalla tarkkuudella

- SENSOR_STATUS_ACCURACY_MEDIUM, jolloin sensorin raportoiman datan tarkkuus on keskinkertainen ja sensorin kalibroiminen voi auttaa lukemien tarkkuutta

- SENSOR_STATUS_ACCURACY_LOW, jolloin sensorin antama data on tarkkuudeltaan huono ja kalibrointi on tarpeen

- SENSOR_STATUS_UNRELIABLE, jolloin sensorin antama data on lukukelvotonta, eikä siihen pidä luottaa.

Testaamisen aikana ei sensoreiden lukemissa arvoissa esiintynyt omituisuuksia, eikä näin ollen osattu kaivata OnAccuracyChanged-metodista saatavia yksityiskohtaisempia tietoja, joten tätä ei käytetty työssä lainkaan. Jälkikäteen olisi tälle voinut löytyä käyttöä toimintavarmuuden lisäämiseksi erinäisin datan tarkistuksin.

OnSensorChangedia kutsutaan aina, kun sensorin antama lukema muuttuu. Tämän metodin parametrinä on SensorEvent-luokan ilmentymä, josta saadaan tietää lukemaansa muuttaneen sensorin tyyppi, tapahtuman aikaleima nanosekunteina, lukeman tarkkuus sekä tietysti sensorin antamat arvot [22].

Arvot annetaan liukulukuja sisältävässä taulukossa, jonka koko ja arvojen tyyppi riippuu tapahtuman aiheuttaneesta sensorista. Esimerkiksi kiihtyvyysanturin tapauksessa taulukko on kolmisoluinen, jossa on kiihtyvyyden m/s^2-komponentit jokaiselle akselille.

OnSensorChanged-metodin sisälle ohjelmoitiin switch-case-rakenne, joka sensorin tyypin perusteella ohjattiin oikeaan haaraan. Sensorin tyypistä riippuen valmisteltiin saatu data eleiden metodeja varten (liite 1, sivu 21).

Kiihtyvyysanturin kutsuessa onSensorChanged-metodia laitettiin ensiksikin saatu data talteen kiertovektorisensoria varten, jonka jälkeen eliminoitiin maan painovoiman vaikutus tuloksista Shake- ja Twotap-eleitä varten, jotta jokaiselle akselille saatiin käyttökelpoiset lukemat. Lineaarisen kiihtyvyyden laskemiseen käytettiin ali- ja ylipäästösuodinta (esimerkkikoodi 3).

```
final float alpha = 0.8f;

// Isolate the force of gravity with the low-pass filter.
gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

// Remove the gravity contribution with the high-pass filter.
linear_acceleration[0] = event.values[0] - gravity[0];
linear_acceleration[1] = event.values[1] - gravity[1];
linear_acceleration[2] = event.values[2] - gravity[2];
```

Esimerkkikoodi 3. Yli- ja alipäästösuodin [7].

On huomauttamisen arvoista, että Android tarjoaa API versiosta 9 eteenpäin jo valmiiksi lineaarisen kiihtyvyysanturin, josta painovoiman vaikutus on jo poistettu [23]. Tässä anturissa on tosin aina poikkeamaa, joka pitää tuloksista poistaa esimerkiksi kalibroimalla laite ennen ohjelman käyttöä, eikä tämä käyttömalli istunut alkuperäiseen suunnitelmaan tehdä eleiden käytöstä yksinkertaista.

Tämän jälkeen jos Shake-ele on käytössä, metodissa on viimeisenä vielä ehtolauseke, joka kutsuu Shaken tunnistusmetodia parametreinään lineaarinen kiihtyvyys sekä sensorin antama aikaleima.

Kiertovektorisensori tarvitsee lukemat magnetometriltä sekä kiihtyvyysanturilta, jotta pystytään laskemaan kiertomatriisi. Tämä kiertomatriisi muuttaa jokaisen akselin suuntaisen vektorin laitteen koordinaattijärjestelmästä ympäröivän maailman koordinaatistoon, jonka jälkeen tiedetään, mihin suuntaan jokainen akseli osoittaa [24].

GetOrientation-metodin paluuarvona saadaan joukko muuttujia, kuten laitteen jokaisen akselin kiertymisen radiaaneina. *Azimuth* kuvaa kiertymistä Z-akselin, *Pitch* X-akselin ja *Roll* taas Y-akselin ympäri [25].

Tämän jälkeen kutsuttiin Flipflop-eleen tunnistusmetodia, jossa parametreinä oli lineaarinen kiihtyvyys, juuri laskettu laitteen asema sekä aikaleima.

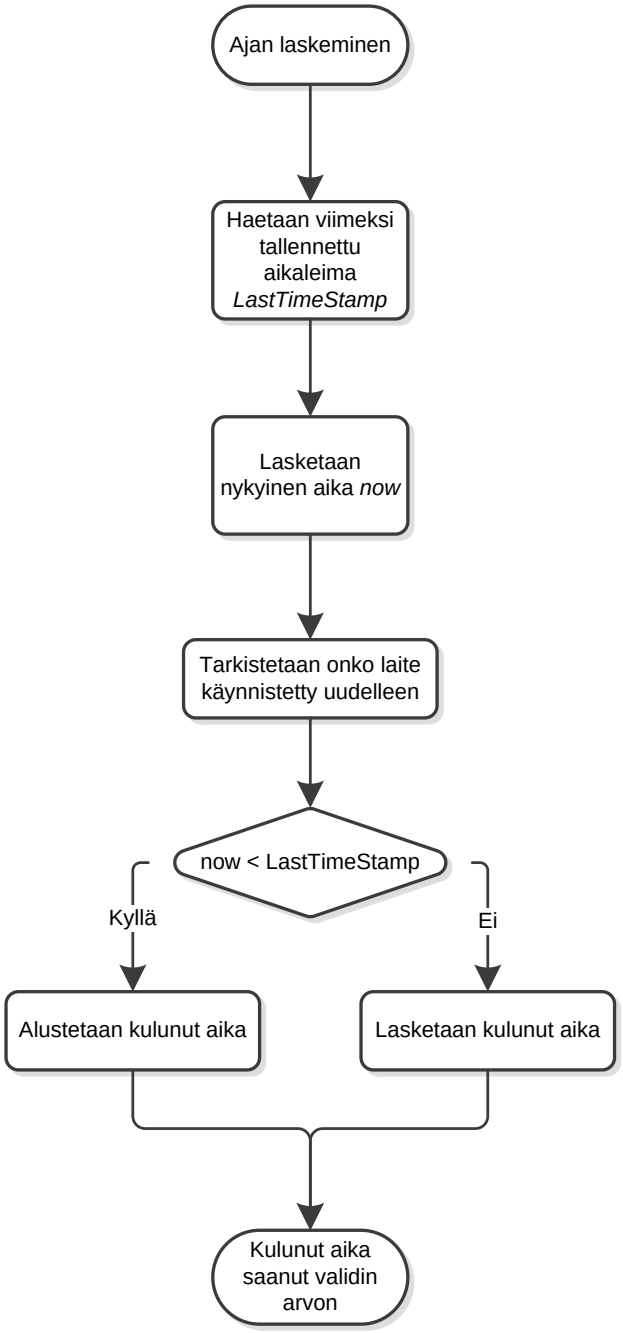
Gyroskoopin kutsuessa onSensorChanged-metodia suoritetaan aliohjelmakutsu Two-tap-eleen tunnistamista varten. Aliohjelmakutsuun annetaan parametreinä lineaarinen kiihtyvyys, gyroskoopin lukemat sekä aikaleima.

Magnetometrin lukemia ei tarvittu muuhun kuin kiertomatriisin laskemiseen.

4.2.3 Aikaleimojen hallinta ja laskeminen

Jokaisessa eleessä tarvittiin aikaleimoja, jotta pystyttiin laskemaan kulunut aika esimerkiksi ajan hetki siitä, kun viimeksi käytiin metodissa tai kun ele käynnistyi. Sensoreiden aikaleimat ovat nanosekunteja, jotka muutettiin testaamisen edetessä sekunneiksi käytännöllisyyden vuoksi.

Riippumatta siitä, minkä eleen metodia kutsuttiin, metodille annettiin sensorin antama aikaleima parametrinä. Tämä aikaleima vähennettiin tallennetusta aikaleimasta josta saatiin kulunut aika. Testaamisen aikana huomattiin, että jos laitteen käynnistää uudelleen, sensorin antama aikaleima on pienempi kuin tallennettu aika. Tämän vuoksi rakennettiin yksinkertainen ehtolauseke alustamaan aika, jos näin on käynyt (kaavio 1).

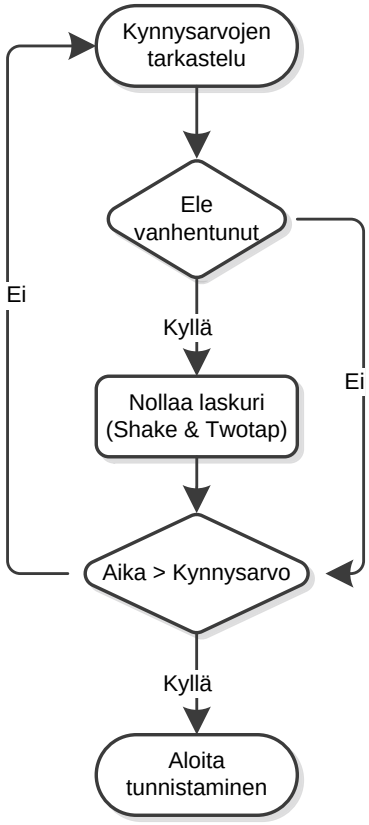


Kaavio 1. Vuokaavio aikaleimojen tarkistamisesta.

Sensoreiden kuuntelijoita rekisteröidessä annettiin tieto, kuinka nopealla tahdilla sensori lukee dataa. Ohjelmakoodissa ei kuitenkaan ole tarpeen käydä koko tunnistusprosessia läpi, jos joku tunnistamisen aloittamisen ehto on epätosi.

Tämän takia metodeissa rajoitettiin pääsyä tunnistamista varten ohjelmoituun ehtolausekkeeseen sovelluksen kuormituksen vähentämiseksi sekä asetettiin kynnysarvo sii-

hen, milloin elettä alettiin sen tapahtumisen jälkeen kuunnella uudelleen. Lisäksi Shake- ja Twotap-eleessä alustettiin laskuri jos eleen aloittamisesta oli kulunut liian pitkä aika (kaavio 2).



Kaavio 2. Vuokaavio eleiden tunnistamisen aloittamisesta.

Jos eleen tunnistamisen kynnysarvo ylitettiin, jatkettiin ohjelmassa seuraavaan lausekkeeseen, jossa itse tunnistaminen aloitettiin ja sensoreiden antamaa dataa tutkittiin.

4.2.4 Arvojen tallentaminen

Näytön kääntäminen aiheuttaa toiminnassa olevan aktiviteetin uudelleen käynnistymisen, joten eri muuttujien arvot alustuvat [26]. Tästä aiheutui ongelmia, koska ohjelman piti pitää lukua Shaken ravistuskerroista, Twotapin heilautuksista sekä laskea, kuinka kauan eleen takaisinkutsusta on kulunut. Tyypillisessä tapauksessa laitetta ehdittiin ravistaa kerran, jolloin laskuri kasvatti arvoaan mutta jonka jälkeen, kun ruudun näkymä luotiin uudelleen laitteen hetkellisen asennon muuttuessa, laskuri nollaantui.

Jos käyttäjän ei tarvitse käyttää ohjelmassaan aktiviteetteja tai hän suorittaa ohjelmaansa taustalla palvelun muodossa, tai muissa tapauksissa milloin näytön kääntymistä ei tarvitse huomioida, tätä ongelmaa ei esiinny.

Ratkaisu ongelmaan löydettiin Androidin omasta API:sta SharedPreferencesistä, joka tallentaa avain-arvopareja [27;28]. Tätä käyttäen ohjelmoitiin kaksi metodia: savePrefsValue sekä getSavedPrefsValue (liite 1, sivu 24).

SavePrefsValue-metodi ottaa parametrinä kaksi merkkijonoa, joista toinen on avain ja toinen arvo. GetSavedPrefsValue-metodi taas ottaa parametrinään avaimen, ja palauttaa sitä vastaavan arvon.

Metodeissa käytettiin pelkästään *String*-tyyppisiä muuttujia eli merkkijonoja, koska haluttiin tallentaa liukulukuja, kokonaislukuja sekä merkkijonoja yhtälailla. Liukulukuja ja kokonaislukuja talletettaessa ne muutettiin kyseisen luokan toString-metodilla sopivaan muotoon ennen kuin tallennusmetodia kutsuttiin. Arvoja haettaessa merkkijonot jäsenneltiin taas haluttuun muotoon.

4.2.5 Laitteen nopeuden laskeminen

Kiihtyvyysanturin antamaa dataa käytettiin poikkeuksetta jokaisessa eleessä. Joko haluttiin tietää, onko laite paikallaan, tai kuinka kovaa laitetta ravistetaan. Tätä varten laskettiin laitteen nopeus eli nopeusvektorin pituus.

Laitteen akseleiden kiihtyvyydet ovat skalaarikomponentteja avaruusvektorille ā. Joten kun tämän vektorin ā pituus lasketaan, eli otetaan itseisarvo, saadaan laitteelle nopeus (kaava 1). Mitä suurempi nopeus laitteella on, sitä suurempi on vektorin ā pituus.

$$|\bar{a}| = \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2} \quad (1)$$

4.2.6 Eleiden tunnistaminen

Flipflopin tunnistaminen

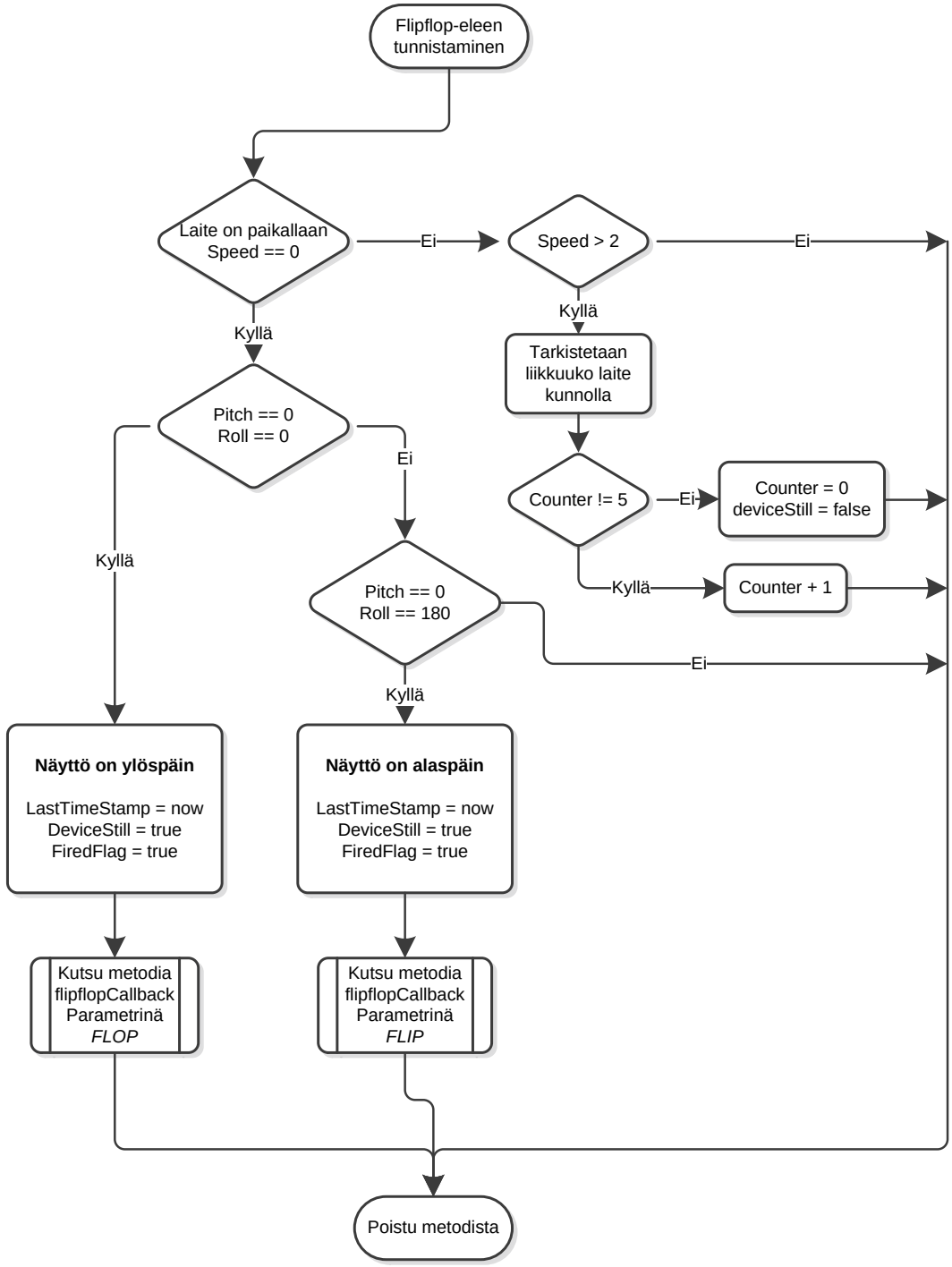
Eleen tunnistamisen suunnittelun apuna käytettiin Log-luokan debug-metodia, jonka avulla sovelluskehittäjä saa tietoa esimerkiksi, missä kohtaa ohjelmakoodia sovellus etenee. Tämän metodin parametriksi voidaan asettaa muuttujia, joiden arvot näkyvät sovelluskehittimessä, joka tässä tapauksessa oli Eclipse.

Muuttujat, joista haluttiin tietoa, olivat luonnollisestikin *Pitch* ja *Roll*. Kun laite asetettiin tasaiselle, liikkumattomalle alustalle siihen asentoon, jolla haluttiin eleen käynnistyvän, otettiin Eclipsestä nämä arvot talteen. Arvoja tutkimalla saatiin oikeat arvot eleen tunnistamisen aloittamiseksi.

Koska testaaminen osoitti, että kiertovektorisensori voi yhtäkkiä kääntää jonkun akselin kulman positiivisesta negatiiviseksi yhden lukukerran ajaksi, otettiin kulmista itseisarvot. Akselin kulman yhtäkkistä lukuvirhettä ei lähdetty sen tarkemmin selvittämään. Sensorit antavat dataa erittäin nopealla tahdilla, ja kiertovektorisensori saa tietoa myös magnetometriltä sekä kiihtyvyysanturilta, joten yksikin häiriö näissä voi aiheuttaa lukuvirheitä.

Koska *Pitch* ja *Roll* saatiin parametreinä OnSensorChanged-metodista, tarvittiin enää laskea laitteen nopeus, jotta saadaan tietää, onko laite paikoillaan (luku 4.2.5).

Kun tarvittavat arvot oli selvitetty ja laskettu, aloitettiin eleen tunnistaminen (kaavio 3). Tunnistamisen marginaaliksi asetettiin neljä astetta, koska käytännössä on hankala löytää niin tasaista paikkaa jotta, ele toimisi, eikä se ole edes tarkoituksenmukaistakaan.



Kaavio 3. Vuokaavio Flipflopin tunnistamisesta.

Shaken tunnistaminen

Laitteen nopeus laskettiin samalla tavalla kuin flipflop-eleessä (luku 4.2.5).

Oikean nopeuden arvon etsiminen aloitettiin kirjoittamalla ohjelmakoodiin debug-metodi, samalla tavoin kuin Flipflopissa. Nyt muuttujaksi, jonka arvoja tutkittiin, asetettiin laitteen nopeusvektori. Kun testisovellus käynnistettiin, pidettiin laitetta mahdollisimman liikkumattomana oletetussa eleen lähtöasennossa. Tämän jälkeen tehtiin tutkittava ele, Shake, jonka jälkeen testisovellus sammutettiin.

Kun muuttujien arvot otetaan talteen Eclipsessä, jokaiseen tapahtumaan liitetään aikaleima ja käyttäjän määrittelemä tägi, jonka avulla osataan seuloa oikea tieto näkyville (esimerkkikoodi 4). Tämän tiedon saa siirrettyä tavalliseksi tekstitiedostoksi, kuten myös tässä tapauksessa tehtiin.

```
02-25 17:39:25.965: D/shake_vektori(32382): speed 0.08981341388877193
02-25 17:39:26.035: D/shake_vektori(32382): speed 0.04580415518072529
02-25 17:39:26.100: D/shake_vektori(32382): speed 0.038804371130840024
02-25 17:39:26.165: D/shake_vektori(32382): speed 0.02068126152121437
02-25 17:39:26.235: D/shake_vektori(32382): speed 0.06304154620523619
```

Esimerkkikoodi 4. Ote muokkaamattomasta debug-datasta.

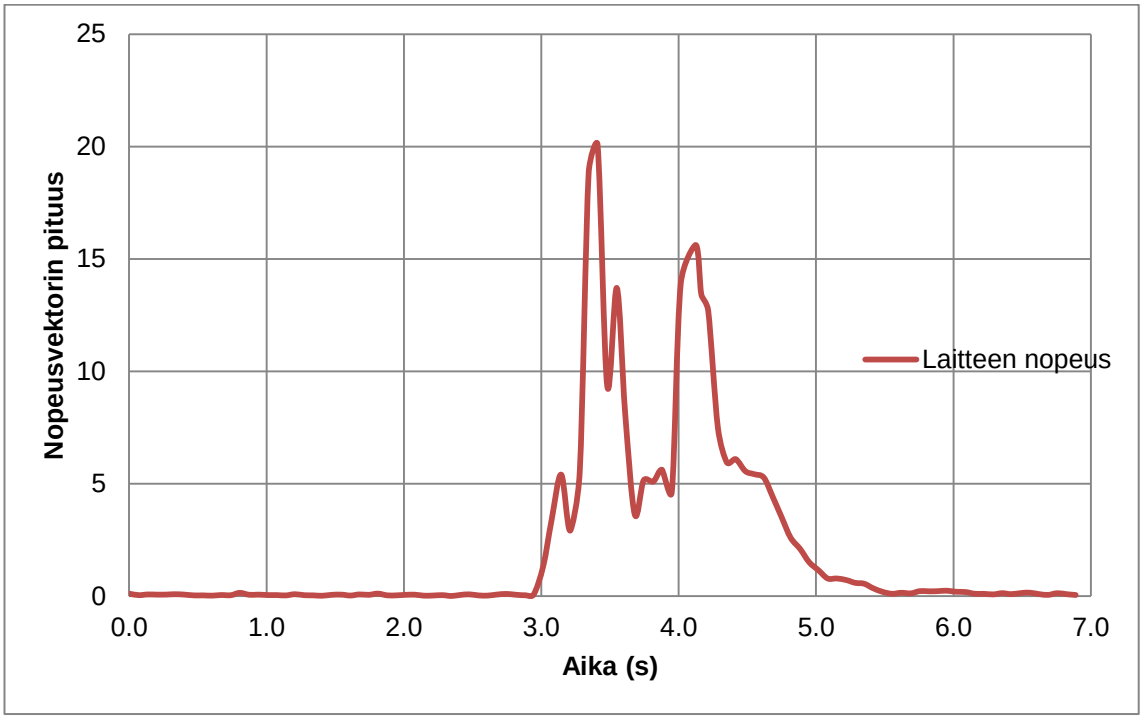
Debug-tiedot sellaisenaan eivät olleet käyttökelpoisia eikä niistä pystytty ilman käsittelyä piirtämään tunnistamista helpottavia käyrästöjä, joten tieto muokattiin Linuxin komentotulkin *cut*-sovelluksen avulla (esimerkkikoodi 5).

```
cut -d ' ' -f 5 speed.txt > speed_edited.txt
cut -d ' ' -f 2 speed.txt > speed_timestamp.txt
```

Esimerkkikoodi 5. Cut-sovelluksen komentoja debug-datan muokkaamiseen.

Esimerkkikoodissa 5 otetaan talteen *speed.txt*-tiedostosta viides tyhjällä merkillä erotettu tekstinpätkä, jossa tässä tapauksessa on nopeusvektorin arvo. Nämä arvot tallennetaan uudeksi tiedostoksi *speed_edited.txt*. Tämä tiedosto sisältää pelkästään nopeusvektorin arvot samassa järjestyksessä kuin alkuperäisessä tiedostossa. Tapahtumien aikaleimat otetaan talteen samalla tavalla ja tallennetaan omaksi tiedostokseen.

Näistä tuloksista piirrettiin käyrästö Microsoft Officen Excel-ohjelman avulla, jotta tulosten analysointi olisi selkeämpää (kuvio 1).



Kuvio 1. Nopeusvektorin pituuden muutos Shake-eleessä.

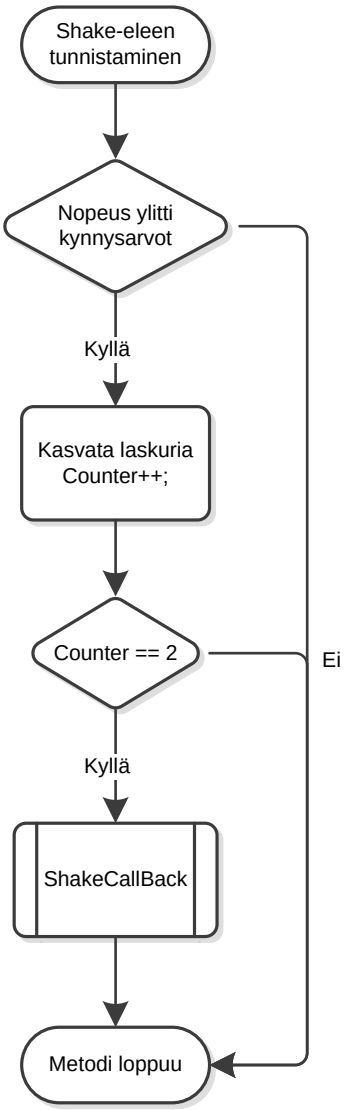
Kuviossa 1 punainen viiva esittää laitteen nopeutta tietyllä ajanhetkellä. Kuviosta nähdään selvästi nopeuden muutos kahtena piikkinä käyrästössä kyseisen eleen kahden ravistuskerran aikana. Tästä pääteltiin keskimääräinen nopeus liikkeen aikana sekä se, kuinka kauan ravistuskertojen välillä olisi syytä olla aikaa.

Koska debug-datan sekaan otettiin talteen arvoja jokaisesta akselista sekä myös muista muuttujista tunnistamisprosessin ymmärtämiseksi, pääteltiin eleessä tarvittava X-akselin arvo tästä materiaalista, joten näistä arvojen muutoksia ei enää nähty tarpeelliseksi piirtää käyrästöä.

Testaamiskertojen lisääntyessä huomattiin, että oikean, tai paremminkin hyvän, nopeuden arvon valitseminen oli hankalaa, joten Gesture-luokan muodostimeen lisättiin mahdollisuus valita ravistuksen voimakkuus elettä luodessa, jos käyttäjä niin haluaa.

Jos käyttäjän ei tarvitse säätää herkkyyttä, käytetään valmiiksi määriteltyä oletusarvoa. X-akselin kiihtyvyyden arvo lasketaan jakamalla nopeus kolmella, jotta lukema on suhteessa mahdolliseen käyttäjän antamaan arvoon.

Sen jälkeen kun nopeus, X-akselin kiihtyvyys sekä muut arvot saatiin laskettua ja asetettua, siirryttiin ohjelmassa tunnistamaan Shake-elettä (kaavio 4).



Kaavio 4. Vuokaavio Shaken tunnistamisesta.

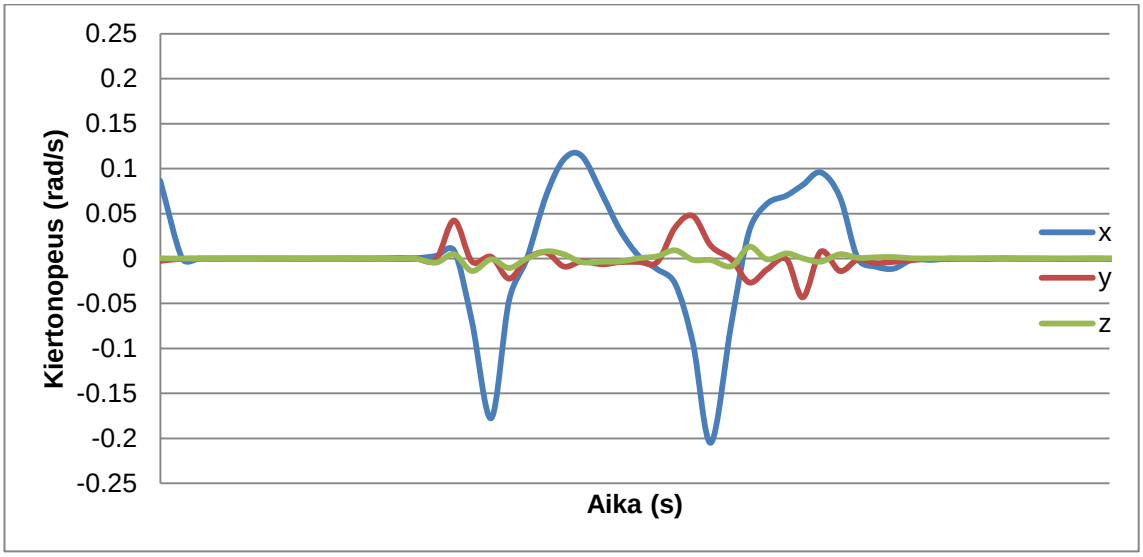
Jos käyttäjä suorittaa eleen ja se tunnistetaan, kutsutaan ohjelmassa metodia, joka luo tapahtumankuuntelijan ja joka kutsuu tämän rajapinnan takaisinkutsumetodia parametrinään aikaleima, jolloin ele tapahtui.

Twotapin tunnistaminen

Twotap-metodiin saavuttaessa tehtiin tavalliset aikaleimojen tarkastukset sekä laskettiin laitteen nopeusvektori, kuten muissakin eleissä.

Gyroskoopin akseleiden kulmakiihtyvyydet normalisoitiin, jotta kohinaa ei esiintyisi ja jotta saataisiin joka akselille kiertovektorit. Tämä tehtiin Android Developer sivustolta löytyvän esimerkin avulla [10]. Loppujen lopuksi näistä arvoista ei tarvinnut tutkia muuta kuin X-akselin kiertovektorin lukemia.

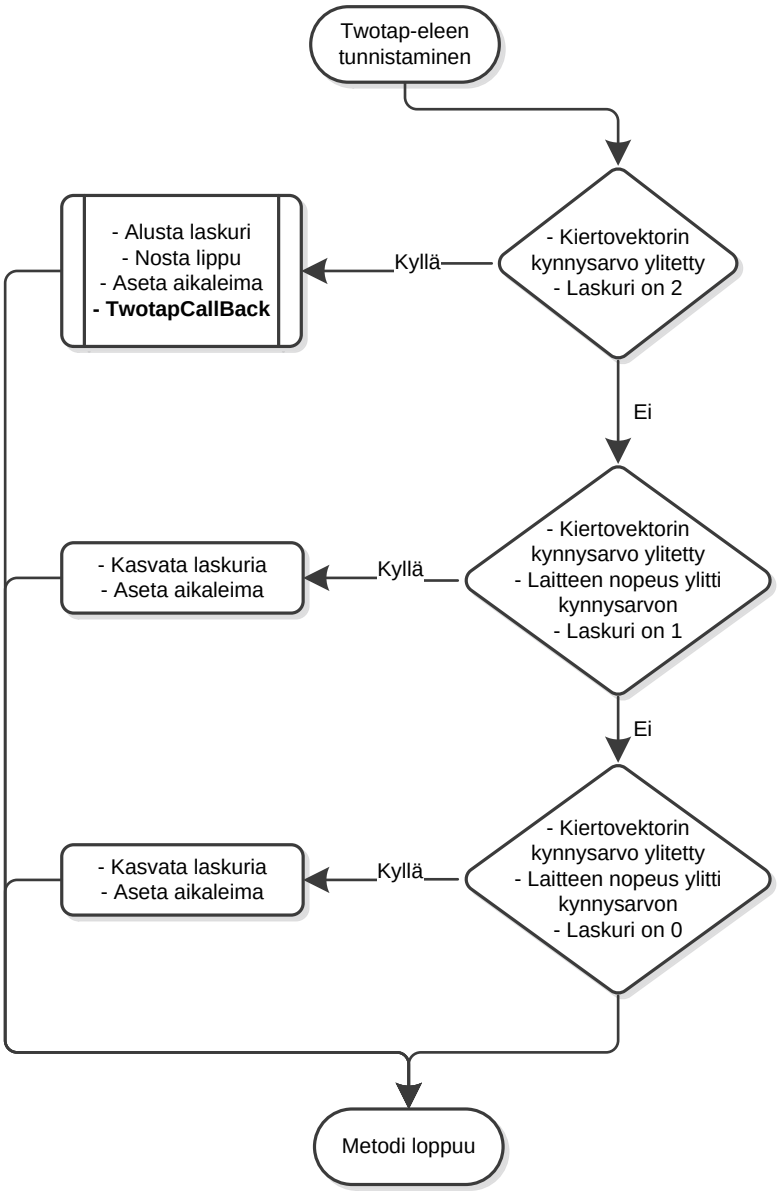
Eleen tunnistusta suunniteltaessa otettiin kuitenkin jokaisen akselin kiertovektoreiden lukemat talteen samalla tavalla kuin Shaken kohdalla ja myös näistä tehtiin käyrästö (kuvio 2).



Kuvio 2. Laitteen akseleiden kiertovektoreiden arvot Twotap-eleessä

Kuviossa 2 sininen viiva kuvaa X-akselin, punainen Y-akselin ja vihreä Z-akselin kiertymisen nopeutta. Kuviossa huomataan selvästi, että kun eleessä suoritetaan ensimmäinen heilautus, X-akselin arvo pienenee, ja takaisin käyttäjää päin suunnatulla heilautuksella arvo nousee. Tämän jälkeen tehdään vielä toinen heilutus, jossa arvot muuttuvat samalla tavalla kuin ensimmäisessä.

Kun oikeat rajat nopeudelle oli löydetty, aloitettiin ohjelmassa eleen tunnistus (kaavio 5), ja jos ele tunnistettiin, suoritettiin takaisinkutsu.



Kaavio 5. Vuokaavio Twotapin tunnistamisesta.

5 Eleiden luominen ja käyttö

5.1 Eleet näkymän komponentteina

Kun eleet saatiin toimimaan ja ilmentymien luominen onnistui aktiviteetin sisällä perinteiseen Java-tyyliin (esimerkkikoodi 6), oli ratkaistava miten näitä eleitä voidaan käyttää nappien (*Button*) tapaan niin, että ne voidaan esitellä näkymän komponentteina näkymän asettelun (*layout*) XML-tiedostossa.

```
Gesture mFlipflop = new Gesture(Context, Gesture.Flipflop);
```

Esimerkkikoodi 6. Eleiden luominen aktiviteetissa ilman XML:ää.

Buttonit ovat näkymäluokan View-jäseniä, jotka ensin esitellään näkymän asettelussa, ja jotka haetaan aktiviteetin ohjelmakoodissa etsimällä niille asetettu tunniste, joten tässä pyrittiin samankaltaiseen toimintamalliin. Esimerkkikoodi 7 havainnollistaa haluttua lopputulosta eleen luomisesta.

```
Gesture mFlipflop = (Gesture)findViewById(R.id.flipflop);
```

Esimerkkikoodi 7. Eleen luominen näkymän komponenttina.

Jotta eleet saataisiin luotua edellä mainitulla tavalla, tarvitsi luoda kustomoidut attribuutit eletyypin valinnalle sekä Shaken herkkyyden asettamiselle, jonka jälkeen kutsua näitä valintoja Gesture-luokan muodostimessa sekä viimeiseksi esitellä ne layout-tiedostossa, esimerkiksi *main.xml*:ssä [29].

Ensimmäiseksi luotiin *attrs.xml*-tiedosto, jossa esiteltiin halutut valinnat (liite 2). Tiedosto piti tallentaa resursseihin *values*-kansioon, jotta kääntäjä löytää kustomoidut (*styleable*) attribuutit ja osaa ilmoittaa ne projektin *R.java*-tiedostossa.

Tämän jälkeen Gesture-luokka kävi läpi hienoisen muutoksen. Koska eleet haluttiin näkymän komponenteiksi, Gesture-luokka periytettiin View-luokasta. Tämän jälkeen muodostimeen asetettiin parametreiksi AttributeSet ja Context (esimerkkikoodi 8).

```
public class Gesture extends View implements SensorEventListener
{
  public Gesture(Context context, AttributeSet attrs)
  {
    super(context, attrs);
            TypedArray a = context.obtainStyledAttributes(attrs,
            R.styleable.Gesture);
  }
}
```

Esimerkkikoodi 8. Gesture-luokan esittely ja muodostin.

Esimerkkikoodissa 4 Context-luokka on rajapinta, josta saadaan globaalia tietoa Android-järjestelmästä sekä myös kutsuja aktiviteettien, palveluiden tai lähetyksien käynnistämisestä [30]. AttributeSet-rajapinnan kautta saadaan obtainStyledAttributes-metodin avulla jäsenneltyä XML-tunnisteiset attribuutit [31].

Paluuarvona obtainStyledAttributes-metodista saadaan TypedArray. Tämä on järjestelmän resurssien yhteisessä käytössä oleva objekteja sisältävä yksirivinen taulukko, jossa kustomoidut attribuutit ovat ja joka arvojen käsittelemisen jälkeen pitää lopuksi kierrättää (*recycled*) [32].

Kun layout-tiedostoon asetetut parametrit oli saatu talteen ja koska Gesture-luokka periytettiin View-luokasta, tarvittiin enää asettaa komponentille korkeus ja leveys eli rajat, miten ele näkyy näytöllä.

Koska kääntäjä vaatii dimension asettamisen layout-tiedostossa, vaikka komponenttia ei tarvitse ”nähdä” samoin kuin esimerkiksi napin määrittämisessä, asetettiin näkymän korkeus ja leveys nollaksi ohjelmallisesti käyttämällä View-luokan metodia onMeasure. Näin saadaan varmistuttua siitä, että ele on taustalla, eikä ohjelma luo näyttöön tyhjiä kuvioita. Käyttäjän on vain asetettava nämä parametrit tiedostoon niiden arvoista välittämättä.

Nyt kun ele luodaan layout-tiedostossa, pitää ensinnäkin näkymän esittelyssä ilmoittaa paketti, josta Android-järjestelmän omien ohjelmakirjastojen resurssien ulkopuoliset kustomoidut attribuutit löytyvät. Komponenttia luotaessa käytetään koko paketin nimeä (esimerkkikoodi 9).

```
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
xmlns:custom="http://schemas.android.com/apk/res/com.mando.GesSen"
<com.mando.GesSen.Gesture
                android:id="@+id/shake"
                custom:gestureChoice="shake"
                custom:shakeResponse="15" />
</LinearLayout>
```

Esimerkkikoodi 9. Shake-eleen luominen layout-tiedostossa (liite 3).

Paketin nimen jälkeen esitellään eleen komponentti, jolle asetetaan id-numero, jotta se pystytään löytämään sovelluksen aktiviteetissa, sekä eleen tyyppi. Tässä tapauksessa eleeksi on valittu Shake, joten lopuksi voidaan valita Shaken voimakkuus.

5.2 Tapahtumakuuntelijoiden rekisteröiminen

Tapahtumakuuntelijoiden rekisteröinnin voi tehdä ainakin kahdella tapaa. Ensimmäinen tapa on lisätä suoraan aktiviteetin luokan esittelyyn haluttu kuuntelija, jonka jälkeen se rekisteröidään ja implementoidaan eleen metodi (esimerkkikoodi 10).

```
public class GesSen extends Activity implements OnFlipflopListener {

   Gesture mFlipflop = (Gesture)findViewById(R.id.flipflop);
   mFlipflop.registerFlipflopListener(this);
}
@Override
public void onFlipflop(int id, long timestamp) {
       // Takaisunkutsumetodi
}
```

Esimerkkikoodi 10. Tapahtumankuuntelijan rekisteröinti sekä erillinen takaisinkutsumetodi.

Tällä tavalla luoduissa takaisinkutsumetodeissa kuormitetaan koko luokkaa, ja kun samaa elettä tuskin luodaan enempää kuin yksi, on tämä ehkä turhan järeä keino kuuntelijoiden rekisteröimisessä.

Toisessa tavassa ei tarvitse luokan esittelyyn lisätä mitään, vaan eleen kuuntelijaa rekisteröidessä luodaan samalla uusi tapahtumankuuntelija, jonka sisään luodaan takaisinkutsumetodi (esimerkkikoodi 11).

```
public class GesSen extends Activity {

   mShake = (Gesture)findViewById(R.id.shake);

   mShake.registerShakeListener(new OnShakeListener() {
      @Override
      public void onShake(long timestamp) {
         // Takaisinkutsumetodi
      });
}
```

Esimerkkikoodi 11. Tapahtumankuuntelijan luominen rekisteröitäessä.

Esimerkkikoodi 11:n keino rekisteröidä tapahtumankuuntelija on näppärä tapa käynnistää eleen kuuntelu sovelluksessa esimerkiksi jonkun tapahtuman, kuten napin painalluksen seurauksena ilman että koko aktiviteetti-luokka kuormittuu turhaan; voihan olla ettei elettä tarvitse sovelluskomponentin elinkaaren aikana kuunnella kertaakaan.

5.3 Testisovellus

Testisovellukseksi tehtiin hyvin yksinkertainen aktiviteetti (liite 4), johon lisättiin jokaiselle eleelle ToggleButton, josta voidaan asettaa sekä lopettaa eleen tapahtumankuuntelija. Sovelluksessa otettiin huomioon myös se, että jos tapahtumankuuntelija jää päälle, lopetetaan eleiden kuuntelu joka tapauksessa aktiviteetin onPause-metodissa (esimerkkikoodi 12).

```
public void onPause() {
   super.onPause();

        mFlipflop.unregisterFlipflopListener();
        mShake.unregisterShakeListener();
        mTwotap.unregisterTwotapListener();
}
```

Esimerkkikoodi 12. Eleiden tapahtumakuuntelijoiden lopettaminen.

Tulevat käyttäjät tuskin haluavat lopettaa eleiden tunnistamista, kun sovellus suljetaan, mutta testauksessa tämä oli tarkoituksenmukaista, ja sovelluskehittäjät päättävät itse, miten he näitä haluavat käyttää.

Kuvassa 8 näkyy tilanne, jossa puhelin on juuri tunnistanut Shaken ja josta näytetään tieto Toastina. Toast on Androidin pienimuotoisiin ilmoituksiin varten luotu näkymä [33].

Kuva 8. Näytönkaappaus testisovelluksesta.

Testaamisen aikana myös kokeiltiin, voiko samoja eleitä olla toiminnassa samaan aikaan, kuten esimerkiksi jos käyttäjä luo vaikka kaksi Shake-elettä, mutta tämä osottautui mahdottomaksi. Kaksi täysin samanlaista elettä jotka luotiin samalla tavalla, eivät toimineet. Ainoastaan se ele, jonka tapahtumankuuntelija rekisteröitiin ensimmäiseksi, antoi takaisinkutsun.

6 Yhteenveto

Kun ensi alkuun työtä suunniteltaessa käytettiin aikaa siihen, minkälaisia eleitä kehitetään, tuli tästä kehitystyöstä vähemmän merkitsevää työn loppua kohti mentäessä. Eleitä pystyy aina kehittämään ja suunnittelemaan erilaisia, mutta tärkeimpänä – mikä vasta työn edetessä tuli huomattua – oli valmistaa eräänlainen kehys tai pohja, miten näitä eleitä voidaan käyttää. Tärkeää oli myös se, että aloittelevat sovelluskehittäjät

pystyvät viemään nämä eleet omiin ohjelmiinsa helposti. Tässä onnistuttiin vähintäänkin hyvin.

Eleiden suunnittelu niin, että ne olisivat yksinkertaisia ja helppoja käyttää, oli haastavaa, eikä mitään mullistavaa keksittykään. Suunnitelluista eleistä ainakin Shakea, sekä erilaisia variaatioita Flipflopista on jossain määrin käytetty eri puhelinvalmistajien toimesta, mutta Twotapin tapaista elettä ei ole vielä tullut vastaan.

Sensoreiden käyttäminen ja niistä ymmärrettävien arvojen saaminen oli omanlaisensa haaste. Nämä antavat erittäin tarkkoja lukemia ja nopeasti, ja jos sensorijärjestelmää ei käytetä maltilla, on tuloksena koko laitteen vasteajan halvaantuminen. Nyt suunnitellut eleet käyttivät yhteensä neljää sensoria, ja vaikka sensoreiden vasteaika oli kohtuulliseksi asetettu, ei suurempia ongelmia esiintynyt. Toki se, että sensoreiden antamaa dataa osattiin tulkita ja jatkokäsitellä oikein, oli välillä hankalaa, mutta tässä lopulta onnistuttiin hyvin.

Gesture-luokan eleiden tunnistamisessa voi löytyä vielä joitain omituisuuksia ja tunnistusprosessia pystyttäisiin varmasti vielä hienosäätämään, mutta kun suurimmat ongelmat, kuten arvojen ja aikaleimojen tallentaminen toimii halutulla tavalla, voidaan yleisesti ottaen kuitenkin olla tyytyväisiä. Mahdolliset jatkokehitykset luultavasti osuvat juuri tähän osa-alueeseen.

Ylipäätään eleiden tunnistamista sekä niiden tarkkuutta pohtiessa tulee mieleen tässä työssä kehitetty ele Flipflop. Tässä eleessähän tutkittiin laitteen asentoa, jota hienosäädettiin kiihtyvyysanturin lukemien avulla. Eleen toimintavarmuudesta voitaisiin päätellä niin, että tulevaisuudessa eleiden suunnittelussa olisi syytä miettiä sitä, miten laitteen sensorit pelaamaan yhteen, eikä suunnitella elettä vain yhden sensorin antaman datan perusteella.

Koodikirjasto käsittää nyt Gesture-luokan ja *attrs.xml*-tiedoston, jotka ovat valmiina Android-yhteisön käytettäväksi, kunhan ne ladataan internetiin. *Attrs.xml*-tiedostoa ei ole välttämätöntä lisätä pakettiin, kunhan mainitaan, mitä kyseisessä tiedostossa kuuluu olla.

Koska aihetta oli käsitelty hyvin harvakseltaan eleiden sekä sensoreiden käytön osalta, oli lähes ainoa paikka, mistä ajantasaista tietoa saa, Androidin omat sovelluskehittäjille tarkoitetut internet-sivut. Koska tekniikka sensoreiden osalta kehittyy huimaa vauhtia, oli materiaalin etsiminen tätä työtä varten hankalaa.

Insinöörityö oli haastava ja mielenkiintoinen, ja siitä tuli opittua paljon. Uskon, että tulevaisuudessa entistä enemmän laitteita ohjataan liike-elein, joten tämä työ antaa mielestäni hyvän pohjan tulevaisuudelle ohjelmistoinsinöörinä.

Lähteet

1 Apple Reinvents the Phone with iPhone. 9.1.2007. Verkkodokumentti. Apple. <http://www.apple.com/pr/library/2007/01/09Apple-Reinvents-the-Phone-with-iPhone.html>. Luettu 23.4.2013.

2 Device details Nokia 7710. 2013. Verkkodokumentti. Nokia. <http://www.developer.nokia.com/Devices/Device_specifications/7710/>. Luettu 23.4.2013.

3 A Brief History of Android. 2013. Verkkodokumentti. Visually. <http://visual.ly/brief-history-android>. Luettu 16.4.2013.

4 Application Fundamentals. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/components/fundamentals.html>. Luettu 26.3.2013.

5 Sensors Overview. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_overview.html>. Luettu 10.4.2013.

6 Inside the Samsung Galaxy SIII. 2013. Verkkodokumentti. Chipworks. <http://www.chipworks.com/blog/recentteardowns/2012/06/01/inside-the-samsung-galaxy-siii/>. Luettu 18.4.2013.

7 Using the Accelerometer. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-accel>. Luettu 11.3.2013.

8 LSM330DLC Datasheet. 2013. Verkkodokumentti. STM. <http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00037200.pdf>. Luettu 18.4.2013.

9 Design simulation and fabrication of micromachined acceleration sensor. Siva Prasad, M S Y. 18.8.2011. Verkkodokumentti. <http://shodhganga.inflibnet.ac.in/handle/10603/2272>. Luettu 22.4.2013.

10 Using the Gyroscope. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-gyro>. Luettu 20.3.2013.

11 John Geen ja David Krakauer. 2003. Analog Devices, Inc.. <http://www.analog.com/library/analogDialogue/archives/37-03/gyro.html>. Luettu 22.4.2013.

12 Mikko Hautala ja Hannu Peltonen. Insinöörin (AMK) Fysiikka Osa 1, 6. painos. 2002. Lahden Teho-opetus Oy. Luettu 22.4.2013.

13 Using the Rotation Vector Sensor. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-rotate>. Luettu 10.4.2013.

14 Using the Geomagnetic Field Sensor. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_position.html#sensors-pos-mag>. Luettu 16.4.2013.

15 Sensor Coordinate System. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_overview.html#sensors-coords>. Luettu 10.4.2013.

16 SensorManager. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/hardware/SensorManager.html>. Luettu 23.4.2013.

17 Saarikivi, Antti. 2013. Järjestelmäasiantuntija, Dna Oy, Helsinki. IRC-keskustelu 10.4.2013.

18 Liike-eleet. 2013. Verkkodokumentti. HTC. <http://www.htc.com/fi/support/howto.aspx?p_id=610&id=339637&p_name=htc-one>. Luettu 10.4.2013.

19 How to use Motion gestures on the Galaxy 3. 2013. Verkkodokumentti. Android Central. <http://www.androidcentral.com/how-use-motion-gestures-galaxy-s3>. Luettu 10.4.2013.

20 Using the Orientation Sensor. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_position.html#sensors-pos-orient>. Luettu 25.2.2013.

21 SensorEventListener. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/hardware/SensorEventListener.html>. Luettu 21.3.2013.

22 SensorEvent. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/hardware/SensorEvent.html>. Luettu 7.3.2013.

23 Using the Linear Accelerometer. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/guide/topics/sensors/sensors_motion.html#sensors-motion-linear>. Luettu 21.3.2013.

24 SensorManager Public Methods: getRotationMatrix. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/hardware/SensorManager.html#getRotationMatrix(float[], float[], float[], float[])>. Luettu 25.2.2013.

25 SensorManager Public Methods: getOrientation. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/hardware/SensorManager.html#getOrientation(float[], float[])>. Luettu 25.2.2013.

26 Recreating an Activity. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/training/basics/activity-lifecycle/recreating.html>. Luettu 8.3.2013.

27 SharedPreferences. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/content/SharedPreferences.html>. Luettu 8.3.2013.

28 Saving Key-Value Sets. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/training/basics/data-storage/shared-preferences.html>. Luettu 8.3.2013.

29 Define Custom Attributes. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/training/custom-views/create-view.html#customattr>. Luettu 25.3.2013.

30 Context Class Overview. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/content/Context.html>. Luettu 26.3.2013.

31 AttributeSet Class Overview. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/util/AttributeSet.html>. Luettu 26.3.2013.

32 TypedArray Class Overview. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/content/res/TypedArray.html>. Luettu 26.3.2013.

33 Toast. 2013. Verkkodokumentti. Android Developers. <http://developer.android.com/reference/android/widget/Toast.html>. Luettu 15.4.2013.

Gesture.java

```
/*********************************************************************
*********
 *
 * Opinnäytetyö / Thesis - Gesture.java
 * @author Mikko Soininvaara / 1002599
 *
 *
 *  FlipFlop:
 *
 *     - Put device at the table or any flat surface screen facing up
(Flip)
 *    or screen facing down (Flop)
 *
 *  Shake:
 *
 *     - When device is upward (upper edge pointing to the sky),
 *    Shake device two times along x-axis
 *
 *
 *  TwoTap:
 *
 *     -Rotate device two times around the x-axis at medium speed
 *
 *                  y
 *                      ^
 *                      |
 *                      |
 *                      |
 *                      ----------> x
 *             /
 *               /
 *              /
 *             z
 *

*********************************************************************
*********/


package com.mando.GesSen;

import android.content.Context;
import android.content.SharedPreferences;
import android.content.res.TypedArray;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.AttributeSet;
import android.util.Log;
import android.view.View;

public class Gesture extends View implements SensorEventListener {

        private SensorManager mSensorManager;
```

```
        private Sensor mRotationVectorSensor;
        private Sensor mAccelerationSensor;
        private Sensor mMagneticFieldSensor;
        private Sensor mGyroscopeSensor;

        private SharedPreferences settings;

        private float[] mRotationMatrix = new float[9];
        private float[] mInclinationMatrix = new float[9];
        private float[] mGravity = new float[3];
        private float[] mGeomagnetic = new float[3];
        private float[] mOrientation = new float[3];

        // Create a constant to convert nanoseconds to seconds.
        private static final float NS2S = 1.0f / 1000000000.0f;

        // low-pass filter
        float[] gravity = new float[3];
    float[] linear_acceleration = new float[3];
    float[] filteredGravity = new float[3];

        // Gesture identifiers
        public static final int FLIPFLOP = 0;
        public static final int SHAKE = 1;
        public static final int TWOTAP = 2;

        // handler for listeners
        private OnFlipflopListener mFlipflopHandler;
        private OnShakeListener mShakeHandler;
        private OnTwotapListener mTwotapHandler;

        private int gestureChoice = -1;

        private static final String GESTURE_SETTINGS = "settings";


        // ---- FLIPFLOP ----
        private final String FLIPFLOP_SAVE = "flipflop_save";
        private final String FLIPFLOP_COUNTER = "flipflop_counter";
        private final String FLIPFLOP_FLAG = "flipflop_flag";

        private final float FLIPFLOP_REQ_LIMIT = 4; // Gesture recog-
nizion limits (degrees)
        private final float FLIPFLOP_TRESHOLD = 0.3f; // seconds

        public static final int FLIP = 0;
        public static final int FLOP = 1;

        private final int PITCH = 1;
        private final int ROLL = 2;

        private final float TIME_BETWEEN_GESTURES = 5; // seconds


        // ---- SHAKE ----
        private final String SHAKE_SAVE = "shake_save";
        private final String SHAKE_COUNTER = "shake_counter";
        private final String SHAKE_FLAG = "shake_flag";
```

```
	private final double SHAKE_TRESHOLD = 0.2; // seconds
	private final double SHAKE_EXPIRED = 0.9; // seconds
	private int tresholdSpeed; // vector lenght
	private int tresholdXvectorSpeed;
	public final int SHAKE_DEFAULT_SPEED = 9;
	public static final int SHAKE_HIGH_SPEED = 14;
	public static final int SHAKE_LOW_SPEED = 4;


	// ----TWOTAP ----
	private final String TWOTAP_SAVE = "twotap_save";
	private final String TWOTAP_COUNTER = "twotap_counter";
	private final String TWOTAP_FLAG = "twotap_flag";

	private static final int X_AXIS = 0;
	private static final int Y_AXIS = 1;
	private static final int Z_AXIS = 2;

	private final float TWOTAP_TRESHOLD = 0.25f; // seconds
	private final float TWOTAP_EXPIRED = 0.7f; // seconds
	private final float TWOTAP_SPEED_POS = 0.6f; // vector lenght
	private final float TWOTAP_SPEED_NEG = -0.3f; // vector lenght
	private final int TWOTAP_OMEGA_SPEED = 3;

	private final float[] deltaRotationVector = new float[4];
	private double time_stamp;
```

```
/*************************************************************************
***************
 * INTERFACES
 *
 * OnFlipflopListener
 * OnShakeListener
 * OnTwotapListener
 *
*************************************************************************
*************
 */
	public interface OnFlipflopListener {
		void onFlipflop(int id, long timestamp);
	}

	public interface OnShakeListener {
		void onShake(long timestamp);
	}

	public interface OnTwotapListener {
		void onTwotap(long timestamp);
	}


/*************************************************************************
***************
```

```
 * GESTURE CONSTRUCTOR
 *
 * @param: Context context
 * @param: AttributeSet attrs
 * @throws Exception
 *
**************************************************************************
*************
 */

       public Gesture(Context context, AttributeSet attrs) {
              super(context, attrs);

              int shakeResponse = 0;

              // get styled attributes and values
              TypedArray a = context.obtainStyledAttributes(attrs,
R.styleable.Gesture);


              try {


                     // Gesture choice from xml
                     gestureChoice =
a.getInt(R.styleable.Gesture_gestureChoice, -1);

                     // Shake sensitivity from xml
                     shakeResponse =
a.getInt(R.styleable.Gesture_gestureChoice, -1);


              } finally {

                     // garbage control
                     a.recycle();
              }


                     // FLIPFLOP 0
                     // SHAKE       1 (Shake sensitivity 1 - 20)
                     // TWOTAP      2
                     if(gestureChoice >= 0 && gestureChoice < 3) {
                      Log.d("registerMyGestureListener", "Get an in-
stance of the SensorManager");

                      // Get an instance of the SensorManager
                      mSensorManager = (SensorManager
)context.getSystemService(Context.SENSOR_SERVICE);
                      this.setGestureChoice(gestureChoice);


       this.setSettings(context.getSharedPreferences(GESTURE_SETTINGS,
0));


                              // Eliminate screen rotation "bug" in
flipflop
```

```
       this.savePrefsValue(Boolean.toString(true), FLIPFLOP_FLAG);
                              Log.d("flipflop_method", "flipflop_flag "
+ this.getSavedPrefsValue("flipflop_flag"));


                              // Get needed sensor for shake
                              if(gestureChoice == SHAKE) {

                                     // check if proper value for sen-
sitivity
                                     if(shakeResponse > 0 && shakeRe-
sponse <= 20){

                                            tresholdSpeed = shakeRe-
sponse;
                                            tresholdXvectorSpeed =
shakeResponse / 3;

                                     } else {

                                            tresholdSpeed =
SHAKE_DEFAULT_SPEED;
                                            tresholdXvectorSpeed =
SHAKE_DEFAULT_SPEED / 3;
                                     }

                                     mAccelerationSensor =
this.findAccelerationSensor(mSensorManager);
                                     mMagneticFieldSensor =
this.findMagneticFieldSensor(mSensorManager);
                                     mRotationVectorSensor =
this.findRotationVectorSensor(mSensorManager);


                              // Get needed sensor for flipflop &
twotap
                              } else if(gestureChoice == FLIPFLOP ||
gestureChoice == TWOTAP) {

                                     mAccelerationSensor =
this.findAccelerationSensor(mSensorManager);
                                     mMagneticFieldSensor =
this.findMagneticFieldSensor(mSensorManager);
                                     mRotationVectorSensor =
this.findRotationVectorSensor(mSensorManager);

                                     // Only twotap needs gyro
                                     if (gestureChoice == TWOTAP) {

                                            mGyroscopeSensor =
this.findGyroscopeSensor(mSensorManager);
                                     }

                              }

                // Unknown gesture choice
                       } else {
```

```
                                throw new UnsupportedOperationExcep-
tion("Unknown gesture choice. " +
                                        " Use Gesture.FLIPFLOP," +
                                        " Gesture.SHAKE or" +
                                        " Gesture.TWOTAP");
                        }



        }

        @Override
        protected void onMeasure (int widthMeasureSpec, int height-
MeasureSpec) {
                super.onMeasure(widthMeasureSpec, heightMeasureSpec);

                // set view size 0 regardless xml-values
                // (note: xml have to contain layout_height & lay-
out_width parameters)
                this.setMeasuredDimension(0, 0);

        }


/*****************************************************************
***************
 * getSensors
 *
 * @param SensorManager mSensorManager
 * @return Sensor
 *
*****************************************************************
*************
 */

        // ACCELERATION SENSOR
        private Sensor findAccelerationSensor(SensorManager mSensorMan-
ager) {

                if (mSensorManag-
er.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
                        // Success!
                        return mSensorManag-
er.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

                } else {

                        throw new UnsupportedOperationExcep-
tion("Acceleration sensor not found!");
                }

        }
        // MAGNETIC FIELD SENSOR
        private Sensor findMagneticFieldSensor(SensorManager mSensor-
Manager) {
```

```
                if (mSensorManag-
er.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
                        // Success!
                        return mSensorManag-
er.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

                } else {

                        throw new UnsupportedOperationExcep-
tion("MagneticFieldSensor not found!");
                }

        }
        // ROTATION VECTOR SENSOR
        private Sensor findRotationVectorSensor(SensorManager mSensor-
Manager) {

                if (mSensorManag-
er.getDefaultSensor(Sensor.TYPE_ACCELEROMETER) != null){
                        // Success!
                        return mSensorManag-
er.getDefaultSensor(Sensor.TYPE_ROTATION_VECTOR);

                } else {

                        throw new UnsupportedOperationExcep-
tion("RotationVectorSensor not found!");
                }

        }
        // GYROSCOPE SENSOR
        private Sensor findGyroscopeSensor(SensorManager mSensorManag-
er) {

                if (mSensorManag-
er.getDefaultSensor(Sensor.TYPE_GYROSCOPE) != null){

                        // Success!
                        return mSensorManag-
er.getDefaultSensor(Sensor.TYPE_GYROSCOPE);

                } else {

                        throw new UnsupportedOperationExcep-
tion("GyroscopeSensor not found!");
                }

        }


/**********************************************************************
***************
 * registerFlipflopListener
 *
 * @param OnFlipflopListener listener
 *
**********************************************************************
************
```

```
 */
	public void registerFlipflopListener(OnFlipflopListener listen-
er) {

		Log.d("registerFlipflopListener", "listener: " + listen-
er.toString());


		this.mFlipflopHandler = listener;


		// Register sensor listeners
	Log.d("registerMyGestureListener", "Register sensor listen-
ers");

	try {
			mSensorManager.registerListener(this, mRotation-
VectorSensor, SensorManager.SENSOR_DELAY_UI);
		} catch (Exception e) {
			Log.d("registerMyGestureListener", "e: " + e);
			e.printStackTrace();
		}

	try {
			mSensorManager.registerListener(this, mAccelera-
tionSensor, SensorManager.SENSOR_DELAY_UI);
		} catch (Exception e) {
			Log.d("registerMyGestureListener", "e: " + e);
			e.printStackTrace();
		}

	try {
			mSensorManager.registerListener(this, mMag-
neticFieldSensor, SensorManager.SENSOR_DELAY_UI);
		} catch (Exception e) {
			Log.d("registerMyGestureListener", "e: " + e);
			e.printStackTrace();
		}


	}
/**********************************************************************
***************
 * unregisterFlipflopListener
 *
 * @param
 *
**********************************************************************
*************
 */
	public void unregisterFlipflopListener() {

		// Unregister sensor listeners
	 mSensorManager.unregisterListener(this, mRotationVectorSen-
sor);
	 mSensorManager.unregisterListener(this, mAccelerationSensor);
	 mSensorManager.unregisterListener(this, mMagneticFieldSensor);

	}
```

```
/**********************************************************************
***************
 * registerShakeListener
 *
 * @param OnShakeListener listener
 *
**********************************************************************
************
 */
	public void registerShakeListener(OnShakeListener listener) {


		this.mShakeHandler = listener;


		// Register sensor listeners
	 try {
				mSensorManager.registerListener(this, mAccelera-
tionSensor, SensorManager.SENSOR_DELAY_UI);
		} catch (Exception e) {

			e.printStackTrace();
		}

	}
/**********************************************************************
***************
 * unregisterShakeListener
 *
 * @param
 *
**********************************************************************
************
 */
	public void unregisterShakeListener() {

		// Unregister sensor listeners
	 mSensorManager.unregisterListener(this, mAccelerationSensor);


	}
/**********************************************************************
***************
 * registerTwotapListener
 *
 * @param OnFlipflopListener listener
 *
**********************************************************************
************
 */
	public void registerTwotapListener(OnTwotapListener listener) {



		this.mTwotapHandler = listener;

		// Register sensor listeners
```

```
        try {
                        mSensorManager.registerListener(this, mAccelera-
tionSensor, SensorManager.SENSOR_DELAY_UI);
                } catch (Exception e) {

                        e.printStackTrace();
                }

        try {
                        mSensorManager.registerListener(this, mGyro-
scopeSensor, SensorManager.SENSOR_DELAY_UI);
                } catch (Exception e) {

                        e.printStackTrace();
                }



        }
/*********************************************************************
***************
 * unregisterTwotapListener
 *
 * @param
 *
*********************************************************************
*************
 */
        public void unregisterTwotapListener() {

                // Unregister sensor listeners
         mSensorManager.unregisterListener(this, mAccelerationSensor);
         mSensorManager.unregisterListener(this, mGyroscopeSensor);

        }

/*********************************************************************
***************
 * FlipFlop-method
 *
 * @param float[] mOrientation
 * @param long timestamp
 *
*********************************************************************
*************
 */
        private void flipFlop(float[] mAcceleration, float[] mOrienta-
tion, long mTimestamp) {


                // Tags for preferences
                final String DEVICE_STILL = "flipflop_device_still";


                // Get saved timestamp
                String lastStr = this.getSavedPrefsValue(FLIPFLOP_SAVE);
                float lastTimestamp = Float.parseFloat(lastStr);

                // get gesture fired flag
```

```
                String flagStr = this.getSavedPrefsValue(FLIPFLOP_FLAG);
                boolean firedFlag = Boolean.parseBoolean(flagStr);


                // device not moving flag
                String stillStr = this.getSavedPrefsValue(DEVICE_STILL);
                boolean deviceStill = Boolean.parseBoolean(stillStr);


                // device still counter for error detection
                String counterStr =
this.getSavedPrefsValue(FLIPFLOP_COUNTER);
                int counter = Integer.parseInt(counterStr);



                // Get current pitch and roll
                float[] orientation = mOrientation;
                double currentPitch =
Math.toDegrees(orientation[PITCH]); // pitch
                double currentRoll = Math.toDegrees(orientation[ROLL]);
// roll

                // Get absolute values of pitch&roll for convienence
                currentPitch = Math.abs(currentPitch);
                currentRoll = Math.abs(currentRoll);


                // get acceleration vectors
                float x_vector = mAcceleration[0];
                float y_vector = mAcceleration[1];
                float z_vector = mAcceleration[2];


                // speedvector
                double speed = Math.sqrt((x_vector * x_vector) +
                                                                (y_vector *
y_vector) +
                                                                (z_vector *
z_vector));


                // resolve time between events
                float now = mTimestamp * NS2S;

                float elapsed;

                // if time is negative (device rebooted) reset time
                if(now < lastTimestamp) {

                        elapsed = 0;
                        this.savePrefsValue(Float.toString(now),
FLIPFLOP_SAVE);

                } else {

                        elapsed = (now - lastTimestamp);
                }
```

```
                // treshold between gestures
                if(elapsed > TIME_BETWEEN_GESTURES) {

                        firedFlag = false;
                        this.savePrefsValue(Boolean.toString(firedFlag),
FLIPFLOP_FLAG);


                }



                if (!firedFlag) {

                        if (elapsed > FLIPFLOP_TRESHOLD) {
                                Log.d("flipflop_method",
"FLIPFLOP_TRESHOLD. speed: " + speed);


                                // device is still
                                if(speed < 0.13 && speed != 0) {


                                        // device face down
                                        if (currentPitch >= 0 && cur-
rentPitch < FLIPFLOP_REQ_LIMIT
                                                && currentRoll <= 180
                                                && currentRoll > (180 -
FLIPFLOP_REQ_LIMIT)) {


                                                if(!deviceStill && elapsed
> 1.0) {

                                                        // device is face
down

                                                        // set time now

        this.savePrefsValue(Float.toString(now), FLIPFLOP_SAVE);

                                                        // raise device
stillflag
                                                        deviceStill = true;

        this.savePrefsValue(Boolean.toString(deviceStill), DE-
VICE_STILL);

                                                        // raise fired flag
                                                        firedFlag = true;

        this.savePrefsValue(Boolean.toString(firedFlag),
FLIPFLOP_FLAG);

                                                        // callback

        this.flipFlopCallBack(FLIP);
```

```
                                        }


                                    // device face up
                                    } else if (currentPitch >= 0 &&
currentPitch < FLIPFLOP_REQ_LIMIT
                                                && currentRoll >= 0
&& currentRoll < FLIPFLOP_REQ_LIMIT) {


                                            if(!deviceStill && elapsed
> 1.0) {

                                                // device is face
up

                                                // set time now

        this.savePrefsValue(Float.toString(now), FLIPFLOP_SAVE);

                                                // raise device
stillflag
                                                deviceStill = true;

        this.savePrefsValue(Boolean.toString(deviceStill), DE-
VICE_STILL);

                                                // raise fired flag
                                                firedFlag = true;

        this.savePrefsValue(Boolean.toString(firedFlag),
FLIPFLOP_FLAG);

                                                // callback

        this.flipFlopCallBack(FLOP);

                                            }

                                    }

                                // device moving
                                } else if(speed > 2 || speed == 0.0) {


                                    // device must be moving properly
for device still flag lowered
                                    // Increase counter
                                    if(counter != 5) {


                                            counter++;

        this.savePrefsValue(Integer.toString(counter),
FLIPFLOP_COUNTER);


                                    // lower flag and reset counter
```

```
                                        } else {

                                                counter = 0;

        this.savePrefsValue(Integer.toString(counter),
FLIPFLOP_COUNTER);

                                                deviceStill = false;

        this.savePrefsValue(Boolean.toString(deviceStill), DE-
VICE_STILL);

                                        }

                                    }

                                }

                        }

        }

        /*
*************************************************************************
*********
         * flipFlopCallBack
         *
         * @param int gestureId
         *
*************************************************************************
*********
         */
        private void flipFlopCallBack(int gestureId) {

                long time = System.currentTimeMillis();

                OnFlipflopListener listener = this.mFlipflopHandler;

                int id = gestureId;

                listener.onFlipflop(id, time);

        }
/*************************************************************************
***************
 * Shake-method
 *
 * @param float[] mAcceleration
 * @param long timestamp
 *
*************************************************************************
*************
 */
        private void shake(float[] mAcceleration, float mTimestamp) {
```

```
                // get counter
                String counterStr =
this.getSavedPrefsValue(SHAKE_COUNTER);
                int counter = Integer.parseInt(counterStr);


                // get gesture fired flag
                String flagStr = this.getSavedPrefsValue(SHAKE_FLAG);
                boolean firedFlag = Boolean.parseBoolean(flagStr);


                // Get saved timestamp
                String lastStr = this.getSavedPrefsValue(SHAKE_SAVE);
                float lastTimestamp = Float.parseFloat(lastStr);


                // get acceleration vectors
                float x_vector = mAcceleration[0];
                float y_vector = mAcceleration[1];
                float z_vector = mAcceleration[2];


                // speedvector
                double speed = Math.sqrt((x_vector * x_vector) +
                                                                (y_vector *
y_vector) +
                                                                (z_vector *
z_vector));


                // Current time
                float now = mTimestamp * NS2S;


                // Device rebooted, reset elapsed time
                float elapsed;

                if(now < lastTimestamp) {

                        elapsed = 0;
                        this.savePrefsValue(Float.toString(now),
SHAKE_SAVE);

                } else {

                        elapsed = (now - lastTimestamp);
                }


                // Lower flag when time exceeds time limit between ges-
tures
                if (elapsed > TIME_BETWEEN_GESTURES) {

                        firedFlag = false;
                        this.savePrefsValue(Boolean.toString(firedFlag),
SHAKE_FLAG);

                }
```

```
                // if too long when shake happened reset counter
                if( elapsed > SHAKE_EXPIRED ) {

                        counter = 0;
                        this.savePrefsValue("0", SHAKE_COUNTER);


                }

                // SHAKE
                if (!firedFlag) {
                        if (elapsed > SHAKE_TRESHOLD) {

                                // detect shake
                                if (speed > tresholdSpeed && x_vector >
tresholdXvectorSpeed) {

                                        // first shake, increase counter

        this.savePrefsValue(Float.toString(now), SHAKE_SAVE);

                                        counter++;

        this.savePrefsValue(Integer.toString(counter), SHAKE_COUNTER);

                                        // shake callback. reset counter
                                        if (counter == 2) {


        this.savePrefsValue(Float.toString(now), SHAKE_SAVE);
                                                this.savePrefsValue("0",
SHAKE_COUNTER);

        this.savePrefsValue(Boolean.toString(true), SHAKE_FLAG);
                                                this.shakeCallBack();

                                        }
                                }
                        }
                }

        }

        /*
*********************************************************************
*********
         * shakeCallBack
         *
         * @param
         *
*********************************************************************
*********
         */
        private void shakeCallBack() {

                long time = System.currentTimeMillis();

                OnShakeListener listener = this.mShakeHandler;
```

```
                listener.onShake(time);


        }
/*************************************************************************
****************
 * TwoTap-method
 *
 * @param float[] mOrientation
 * @param float[] mAcceleration
 * @param long timestamp
 *
*************************************************************************
*************
 */
        private void twotap(float[] mAcceleration, float[] mRotation,
float mTimestamp) {


                // error margin for normalization
                final int EPSILON = 1;

                // get counter
                String counterStr =
this.getSavedPrefsValue(TWOTAP_COUNTER);
                int counter = Integer.parseInt(counterStr);


                // get gesture fired flag
                String flagStr = this.getSavedPrefsValue(TWOTAP_FLAG);
                boolean firedFlag = Boolean.parseBoolean(flagStr);


                // get last timestamp
                String lastStr = this.getSavedPrefsValue(TWOTAP_SAVE);
                float lastTimestamp = Float.parseFloat(lastStr);


                // get acceleration vectors
                float x_vector = mAcceleration[0];
                float y_vector = mAcceleration[1];
                float z_vector = mAcceleration[2];


                // speedvector
                double speed = Math.sqrt((x_vector * x_vector) +
                                                                (y_vector *
y_vector) +
                                                                (z_vector *
z_vector));


                // get time between events
                float now = mTimestamp * NS2S;


                // Device rebooted, reset elapsed time
                float elapsed;
```

```
                if(now < lastTimestamp) {

                        elapsed = 0;
                        this.savePrefsValue(Float.toString(now),
TWOTAP_SAVE);

                } else {

                        elapsed = (now - lastTimestamp);
                }


                // Get deltarotationvectors and speedvector
                // (sample code from Android-developers)
                if (time_stamp != 0) {

                        final float dT = now;
                        // Axis of the rotation sample, not normalized
yet.
                        float axisX = mRotation[X_AXIS]; // rad/s
                        float axisY = mRotation[Y_AXIS]; // rad/s
                        float axisZ = mRotation[Z_AXIS]; // rad/s

                        // Calculate the angular speed of the sample

                        float omegaMagnitude = (float) Math.sqrt(axisX *
axisX + axisY

                           * axisY + axisZ * axisZ);

                        // Normalize the rotation vector if it's big
enough to get the axis
                        // (that is, EPSILON should represent your maxi-
mum allowable margin of error)
                        if (omegaMagnitude > EPSILON) {
                                axisX /= omegaMagnitude;
                                axisY /= omegaMagnitude;
                                axisZ /= omegaMagnitude;
                        }

                        // Integrate around this axis with the angular
speed by the timestep
                        // in order to get a delta rotation from this
sample over the timestep
                        // We will convert this axis-angle representa-
tion of the delta rotation
                        // into a quaternion before turning it into the
rotation matrix.
                        double thetaOverTwo = omegaMagnitude * dT /
2.0f;
                        float sinThetaOverTwo = (float)
Math.sin(thetaOverTwo);
                        float cosThetaOverTwo = (float)
Math.cos(thetaOverTwo);

                        deltaRotationVector[X_AXIS] = sinThetaOverTwo *
axisX;
                        deltaRotationVector[Y_AXIS] = sinThetaOverTwo *
axisY;
```

```
                        deltaRotationVector[Z_AXIS] = sinThetaOverTwo *
axisZ;
                        deltaRotationVector[3] = cosThetaOverTwo; //
seconds


                }
                time_stamp = now;


                // Lower flag when time exceeds time limit between ges-
tures
                if(elapsed > TIME_BETWEEN_GESTURES) {

                        firedFlag = false;
                        this.savePrefsValue(Boolean.toString(firedFlag),
TWOTAP_FLAG);
                }


                // if too long when twotap happened reset counter
                if(elapsed > TWOTAP_EXPIRED) {

                        counter = 0;
                        this.savePrefsValue(Integer.toString(counter),
TWOTAP_COUNTER);


                }


                // TWOTAP
                if (!firedFlag) {


                        if (elapsed > TWOTAP_TRESHOLD) {

                                // detect twotap gesture


                                // Final motion, step 3
                                if (deltaRotationVector[X_AXIS] >
TWOTAP_SPEED_POS
                                 && counter == 2) {

                                        // raise gesture fired flag
                                        firedFlag = true;

        this.savePrefsValue(Boolean.toString(firedFlag), TWOTAP_FLAG);


                                        // Set timestamp

        this.savePrefsValue(Float.toString(now), TWOTAP_SAVE);

                                        // reset counter
                                        counter = 0;

        this.savePrefsValue(Integer.toString(counter), TWOTAP_COUNTER);
```

```
                                        // callback
                                        this.twotapCallBack();

                                    // step 2
                                    } else if (deltaRotationVector[X_AXIS] <
TWOTAP_SPEED_NEG
                                                && counter == 1
                                                && speed >
TWOTAP_OMEGA_SPEED) {


                                        // Set timestamp

        this.savePrefsValue(Float.toString(now), TWOTAP_SAVE);


                                        // increase counter
                                        counter++;

        this.savePrefsValue(Integer.toString(counter), TWOTAP_COUNTER);


                                    // step 1
                                    } else if(deltaRotationVector[X_AXIS] >
TWOTAP_SPEED_POS
                                        && counter == 0
                                        && speed > TWOTAP_OMEGA_SPEED) {



                                    // Set timestamp
                                    this.savePrefsValue(Float.toString(now),
TWOTAP_SAVE);


                                    // increase counter
                                    counter++;

        this.savePrefsValue(Integer.toString(counter), TWOTAP_COUNTER);


                                    }
                            }
                    }


        }
        /*
*********************************************************************
*********
         * twotapCallBack
         *
         * @param int gestureId
         *
*********************************************************************
*********
         */
```

```
        private void twotapCallBack() {

                long time = System.currentTimeMillis();

                OnTwotapListener listener = this.mTwotapHandler;


                listener.onTwotap(time);

        }


        @Override
        public void onAccuracyChanged(Sensor arg0, int arg1) {


        }
/*************************************************************************
***************
 * onSensorChanged
 *
 *
 *
*************************************************************************
*************
 */
        @Override
        public void onSensorChanged(SensorEvent event) {

                // we received a sensor event. it is a good practice to
check
         // that we received the proper event
        switch (event.sensor.getType()) {


                // ACCELEROMETER
                case Sensor.TYPE_ACCELEROMETER: {

                        mGravity = event.values;       // for rotation
vector sensor


                        // Remove gravity value for gestures


                        final float alpha = 0.8f;

                                // Isolate the force of gravity with the
low-pass filter.
                                gravity[0] = alpha * gravity[0] + (1 -
alpha) * event.values[0];
                                gravity[1] = alpha * gravity[1] + (1 -
alpha) * event.values[1];
                                gravity[2] = alpha * gravity[2] + (1 -
alpha) * event.values[2];

                                // Remove the gravity contribution with
the high-pass filter.
```

```
                                linear_acceleration[0] = event.values[0]
- gravity[0];
                                linear_acceleration[1] = event.values[1]
- gravity[1];
                                linear_acceleration[2] = event.values[2]
- gravity[2];


                        if(this.getGestureChoice() == SHAKE) {

                                // call shake method
                                this.shake(linear_acceleration,
event.timestamp);
                        }

                        break;
                }

                // MAGNETIC FIELD
                case Sensor.TYPE_MAGNETIC_FIELD: {

                        mGeomagnetic = event.values;
                        break;
                }

                // ROTATION VECTOR
                case Sensor.TYPE_ROTATION_VECTOR: {


                     // Get rotationMatrix
                        SensorManager.getRotationMatrix(mRotationMatrix,
mInclinationMatrix, mGravity, mGeomagnetic);


                        // Get device orientation
                        SensorManager.getOrientation(mRotationMatrix,
mOrientation);


                        if(this.getGestureChoice() == FLIPFLOP) {

                                // call flipflop method
                                this.flipFlop(linear_acceleration, mOri-
entation, event.timestamp);

                        }
                        break;
                }

                // GYROSCOPE
                case Sensor.TYPE_GYROSCOPE: {


                        if(this.getGestureChoice() == TWOTAP) {

                          // call twotap method
                          this.twotap(linear_acceleration, event.values,
event.timestamp);
```

```
                        }
                        break;
                }

                default:
                        break;
                }

        }
/*********************************************************************
***************
 * SETTERS AND GETTERS:
 *
 *
 *
 * ListenerHandler
 * LastTimestamp
 * TimeGestureFired
 *
*********************************************************************
************
 */



        /*
         * set and get gesture choice
         */
        private void setGestureChoice(int choice) {

                this.gestureChoice = choice;
        }
        private int getGestureChoice() {

                return this.gestureChoice;
        }


        /**
         * @return the settings
         */
        private SharedPreferences getSettings() {
                return settings;
        }

        /**
         * @param settings the settings to set
         */
        private void setSettings(SharedPreferences settings) {
                this.settings = settings;
        }


/*********************************************************************
***************
 *  Helper functions
 *
 *  savePrefsValue - Save valuen in Androids preferences
```

```
 *  @param String value
 *  @param String tag
 *
**********************************************************************
*************
 */

        // save value
        private void savePrefsValue(String value, String tag) {


                SharedPreferences settings = this.getSettings();
                SharedPreferences.Editor editor = settings.edit();

                try {

                        editor.putString(tag, value);
                        editor.commit();

                } catch (Exception e) {

                        Log.d("savevalue_set", "Exception " + e);

                }
        }
        // get saved value
        private String getSavedPrefsValue(String tag) {

                SharedPreferences settings = this.getSettings();

                try {

                        String str = settings.getString(tag, "0");
                        return str;

                } catch (Exception e) {

                        Log.d("savevalue_get", "Exception " + e);
                        return "0";
                }

        }

}
```

Attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>

<!--

        Opinnäytetyö / Thesis - Attrs.xml
        @author Mikko Soininvaara / 1002599

-->

<resources>

    <declare-styleable name="Gesture">

       <attr name="shakeResponse" format="integer" />
       <attr name="gestureChoice" format="enum" >
           <enum name="flipflop" value="0" />
           <enum name="shake" value="1" />
           <enum name="twotap" value="2" />
       </attr>

   </declare-styleable>

</resources>
```

Main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:custom="http://schemas.android.com/apk/res/com.mando.GesSen"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical"
    android:background="@color/bg" >


   <TextView
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:textSize="30sp"
       android:textColor="@color/topic_text_color"
       android:text="Test app for Gesture" />

    <TextView
       android:layout_width="wrap_content"
       android:layout_height="30sp" />

<!-- FLIPFLOP -->
       <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:custom="http://schemas.android.com/apk/res/com.mando.GesSen"
           android:layout_width="wrap_content"
           android:layout_height="wrap_content"
           android:orientation="horizontal"
           android:layout_gravity="right"
           android:paddingBottom="15sp" >

           <TextView
               android:id="@+id/tv_flipflop"
               android:layout_width="fill_parent"
               android:layout_height="wrap_content"
               android:paddingRight="10sp"
               android:textSize="20sp"
               android:text="FlipFlop gesture"
               android:textColor="@color/reg_text_color" />

            <com.mando.GesSen.Gesture
               android:layout_width="100sp"
               android:layout_height="100sp"
               android:id="@+id/flipflop"
               custom:gestureChoice="flipflop" />

           <ToggleButton
               android:id="@+id/togglebutton_f"
                      android:layout_width="wrap_content"
                      android:layout_height="wrap_content"
                      android:textOn="on"
                  android:textOff="off" />
       </LinearLayout>
```

```
<!-- SHAKE -->
       <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:custom="http://schemas.android.com/apk/res/com.mando.GesSen"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:layout_gravity="right"
            android:paddingBottom="15sp" >

            <TextView
                android:id="@+id/tv_shake"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:paddingRight="10sp"
                android:textSize="20sp"
                android:text="Shake gesture"
                android:textColor="@color/reg_text_color" />

             <com.mando.GesSen.Gesture
                android:id="@+id/shake"
                android:layout_width="100sp"
                android:layout_height="100sp"
                custom:gestureChoice="shake"
                custom:shakeResponse="15" />

            <ToggleButton
                android:id="@+id/togglebutton_s"
                       android:layout_width="wrap_content"
                       android:layout_height="wrap_content"
                       android:textOn="on"
                   android:textOff="off" />
       </LinearLayout>

<!-- TWOTAP -->
       <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"

xmlns:custom="http://schemas.android.com/apk/res/com.mando.GesSen"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:layout_gravity="right"
            android:paddingBottom="15sp" >

            <TextView
                android:id="@+id/tv_twotap"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:paddingRight="10sp"
                android:textSize="20sp"
                android:text="Twotap gesture"
                android:textColor="@color/reg_text_color" />

             <com.mando.GesSen.Gesture
                android:id="@+id/twotap"
                android:layout_width="100sp"
```

```
                android:layout_height="100sp"
                custom:gestureChoice="twotap" />

            <ToggleButton
                android:id="@+id/togglebutton_t"
                       android:layout_width="wrap_content"
                       android:layout_height="wrap_content"
                       android:textOn="on"
                   android:textOff="off" />
        </LinearLayout>

</LinearLayout>
```

GesSen.java

```
/*********************************************************************
*********
 *
 *  Opinnäytetyö / Thesis - GesSen.java
 *  @author Mikko Soininvaara / 1002599
 *
 *  Test application for Gestures
 *

*********************************************************************
*********/

package com.mando.GesSen;

import com.mando.GesSen.Gesture.OnFlipflopListener;
import com.mando.GesSen.Gesture.OnShakeListener;
import com.mando.GesSen.Gesture.OnTwotapListener;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.util.Log;
import android.widget.CompoundButton;
import android.widget.Toast;
import android.widget.ToggleButton;

public class GesSen extends Activity  {


       private Gesture mFlipflop;
       private Gesture mShake;
       private Gesture mTwotap;

       private ToggleButton btn_flipflop;
       private ToggleButton btn_shake;
       private ToggleButton btn_twotap;


       private Context context;



 /*
  * ********
  * ONCREATE
  * ********
  */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Log.d("LOG", "onCreate");

        context = this.getApplicationContext();
```

```
        // find gestures
        mFlipflop = (Gesture)findViewById(R.id.flipflop);
        mShake = (Gesture)findViewById(R.id.shake);
        mTwotap = (Gesture)findViewById(R.id.twotap);

        // find buttons
        btn_flipflop = (ToggleButton) findViewBy-
Id(R.id.togglebutton_f);
        btn_shake = (ToggleButton) findViewById(R.id.togglebutton_s);
        btn_twotap = (ToggleButton) findViewById(R.id.togglebutton_t);



    }
/*
 * ********
 * ONSTART
 * ********
 */
    @Override
    public void onStart() {
        super.onStart();
        Log.d("LOG", "onStart");


       btn_flipflop.setOnCheckedChangeListener(new CompoundBut-
ton.OnCheckedChangeListener() {
           public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
               if (isChecked) {


                    mFlipflop.registerFlipflopListener(new On-
FlipflopListener() {

                                            @Override
                                            public void onFlipflop(int
id, long timestamp) {
                                                // TODO Auto-
generated method stub
                                                Toast toast1 =
Toast.makeText(context, "device is face down", Toast.LENGTH_SHORT);
                                                Toast toast2 =
Toast.makeText(context, "device is face up", Toast.LENGTH_SHORT);

                                                switch (id) {

                                                case Gesture.FLIP:


       toast1.show();
                                                    break;

                                                case Gesture.FLOP:


       toast2.show();
                                                    break;
```

```
                                                default:
                                                break;
                                            }
                                            }


                        });



                } else {
                    // The toggle is disabled
                       mFlipflop.unregisterFlipflopListener();
                }
            }
        });



        btn_shake.setOnCheckedChangeListener(new CompoundBut-
ton.OnCheckedChangeListener() {
            public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
                if (isChecked) {


                        mShake.registerShakeListener(new OnShakeListen-
er() {

                                                @Override
                                                public void onShake(long
timestamp) {
                                                        // TODO Auto-
generated method stub
                                                        Toast toast_ =
Toast.makeText(context, "shake", Toast.LENGTH_SHORT);
                                                        toast_.show();
                                                }



                        });



                } else {
                    // The toggle is disabled
                       mShake.unregisterShakeListener();
                }
            }
        });



        btn_twotap.setOnCheckedChangeListener(new CompoundBut-
ton.OnCheckedChangeListener() {
```

```
            public void onCheckedChanged(CompoundButton buttonView,
boolean isChecked) {
                if (isChecked) {


                        mTwotap.registerTwotapListener(new OnTwotapLis-
tener() {

                                                    @Override
                                                    public void onTwotap(long
timestamp) {
                                                            // TODO Auto-
generated method stub
                                                            Toast toast_ =
Toast.makeText(context, "twotap", Toast.LENGTH_SHORT);
                                                            toast_.show();
                                                    }


                        });



                } else {
                    // The toggle is disabled
                       mTwotap.unregisterFlipflopListener();
                }
            }
        });



    }

/*
 * ********
 * ONPAUSE
 * ********
 */
    @Override
    public void onPause() {
        super.onPause();
        Log.d("LOG", "onPause");

        mFlipflop.unregisterFlipflopListener(); // IMPORTANT
        mShake.unregisterShakeListener();
        mTwotap.unregisterTwotapListener();



    }
}
```