Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Luka Vezzosi

Upravljanje pametnega telefona s pomočjo prostorskih gest

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: viš. pred. dr. Alenka Kavčič

Ljubljana 2013

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00418/2013
Datum: 09.04.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **LUKA VEZZOSI**

Naslov: **UPRAVLJANJE PAMETNEGA TELEFONA S POMOČJO PROSTORSKIH GEST**

USING SPATIAL GESTURES FOR SMART PHONE HANDLING

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Današnji pametni telefoni imajo vgrajenih veliko senzorjev, preko katerih lahko pridobivamo podatke iz okolja. Eden takih senzorjev je tudi pospeškometer, ki nam daje podate o pospeških pri premikih telefona v vseh treh smereh v prostoru.
S pomočjo senzorja za merjenje pospeškov realizirajte aplikacijo za platformo Android, ki bo omogočala zaznavanje s telefonom narejenih prostorskih gest uporabnika in se nanje odzvala z aktiviranjem ustrezne aplikacije. V okviru naloge poiščite najprimernejši način shranjevanja in prepoznavanja uporabnikovih gest. Aplikacija naj omogoča shranjevanje nove geste, prepoznavanje narejene geste ter povezovanje geste z aplikacijo, ki jo gesta sproži.

Mentor:
Alenka Kavčič
viš. pred. dr. Alenka Kavčič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Luka Vezzosi, z vpisno številko **63090292**, sem avtor diplomskega dela z naslovom:

Upravljanje pametnega telefona s pomočjo prostorskih gest

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Alenke Kavčič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 9. septembra 2013 Podpis avtorja:

Zahvaljujem se mentorici, viš. pred. dr. Alenki Kavčič, za pomoč pri mojem diplomskem delu.

Zahvaljujem se Nataši Kozelj, prof. slovenščine za strokovno lektoriranje diplomskega dela.

Kazalo

Povzetek

Diplomsko delo opisuje postopke za upravljanje pametnega telefona s pomočjo gest. Opisuje teoretično ozadje gest in senzorjev, kar predstavlja temelje za razumevanje tega diplomskega dela. Programerski del diplomskega dela smo z orodjem za razvoj programske opreme rešili v okolju operacijskega sistema Android. Orodje vsebuje ogrodje Sensor, ki je zadolženo za prikaz podatkov integriranih senzorjev, in predstavlja programersko središče. Določili smo lastne osnovne elemente gest ter napisali algoritem za njihovo prepoznavanje in ujemanje, shranjevanje podatkov pa smo realizirali s podatkovno bazo SQLite. Končni rezultat je aplikacija, ki omogoča uporabo gest za vhod v poljubno aplikacijo. V zaključku diplomskega dela je ocenjena uporabnost aplikacije, omenjene pa so tudi ideje za nadaljnje raziskovanje in izboljšanje tega področja.

Abstract

Diploma thesis describes the procedure for smartphone device management with motion gestures. It describes theoretical background of gestures and sensors which are the basics of understanding this paper. The programmatic part of the thesis was developed using software development kit in Android operating system. Sensor framework's task is to provide data from integrated sensors and represents the programmatic core. We defined our own basic motion gesture elements, wrote an algorithm for their recognition and matching and used SQLite database for data saving. The result is an application which enables the usage of motion gestures to open another application. The conclusion consists of usability rate of the application and ideas for further research and improvement of this thesis.

Poglavje 1

Uvod

Geste so v uporabi človeštva že tisočletja. Prvič so se pojavile že pri neandertalcih, saj je bil to dolgo časa edini način komunikacije. Iz primitivnih gest s telesom so se nato razvile geste z uporabo pripomočkov, kot so palice in kamni. Uporaba gest se je skozi zgodovino človeštva močno spreminjala in razvijala, saj je človek vedno našel način za izboljšave. Človeška narava nas žene k temu, da si vedno želimo olajšati vsakdanja opravila. Ravno zaradi tega dejstva in obdobja razcveta (informacijske) tehnologije geste vedno bolj pridobivajo na pomenu tudi v tej stroki.

Ker pa se ta veja tehnologije še naprej razvija in izboljšuje, so jo začeli uporabljati tudi v panogah, ki zahtevajo večjo pozornost. Pospeškometri pomagajo pri zaznavanju potresov in vulkanskih izbruhov, saj z njimi in preostalimi senzorji lahko izračunajo nihanje in napovedo potresno aktivnost v določenem območju. Zaznavanje in analiza gibanja uporabnika postaja vedno bolj pogost način komunikacije z računalnikom, saj gre za človeku bolj naraven pristop. Veliko se razvija in uporablja v biomehaniki, in sicer pri manjših operacijah, kjer je odprtina velika od 0.5 do 1.5 cm, in jo lahko kirurg spremlja samo v obliki povečave na zaslonu. Pri operacijah se uporablja analiza premikanja in na podlagi analize omogoča urjenje kirurgov in izboljševanje njihovih sposobnosti. V učni množici so zapisani najboljši ter najslabši gibi, ki se jih pri določeni operaciji uporablja, kirurg pa z izvaja-

njem gibov dobi povratno informacijo, kako dobro je nek gib izvedel in na katerem mestu je naredil največ napak [6]. Najbolj pa so geste uporabljene v zabavni tehnologiji. Srečamo jih v igralnih konzolah (Nintendo Wii, Xbox Kinect), tabličnih računalnikih (iPad, Google Nexus) ter pametnih telefonih (iPhone, Samsung Galaxy). Nintendo Wii je bil tako pionir pri zajemu gibanja s svojim kontrolerjem Wii Remote (slika 1.1), ki zaznava premike v treh dimenzijah prostora in predstavlja središče iger, ki zahtevajo geste premikanja. V kontroler sta vgrajena pospeškometer ter optični senzor, s katerimi je omogočeno prepoznavanje gest.

Slika 1.1: Kontroler Wii Remote

Naša diplomska naloga se osredotoča na uporabo gest na pametnih telefonih. Geste se lahko uporablja za izvajanje funkcij telefona glede na vnaprej predpisane geste. Primer je gesta, ko telefon obrnemo z ekranom navzdol in ga položimo na mizo, kar sproži prenehanje zvonjenja. Pri igrah na telefonu geste največkrat služijo kot orodje za upravljanje z nekim objektom, igranje igre pa nam je na ta način zelo olajšano, velikokrat pa tudi mnogo bolj naravno. Primer je nagnjenje telefona v levo ali desno pri dirkalnih igrah, saj s tem posnemamo resnični avtomobilski volan. Nekaj gest pa je definiranih že v samem operacijskem sistemu Android. Tu je kot primer vredno omeniti

zaznavanje orientacije naprave v horizontalni ter vertikalni smeri. Ko telefon spremeni orientacijo, se ustrezno spremeni tudi pogled.

Za razvoj smo izbrali operacijski sistem Android, saj trenutno pokriva okoli 80-odstotni delež vsega trga in je najbolj prijazno okolje za razvoj nove aplikacije. Cilj diplomske naloge je izdelati aplikacijo, ki je namenjena uporabniku olajšati vhod v njegove največkrat uporabljene aplikacije. Klasičen vhod v aplikacijo na telefonu z operacijskim sistemom Android je preko bližnjic na začetnem ekranu ali pa preko menija, kjer imamo na voljo vse naložene aplikacije. Uporabnik z našo aplikacijo lahko hitro dostopa do aplikacije z uporabo geste, ki jo sam posname ter veže na poljubno aplikacijo. Drugo poglavje opisuje teoretično ozadje senzorjev, bolj podrobno pa je opisan tudi pospeškometer, saj smo ga uporabljali za razvoj te diplomske naloge. V tem poglavju se srečamo tudi s pojmom gesta, bolj podrobno pa so predstavljene še vrste gest. Sledi poglavje, ki opisuje, kako smo se lotili zaznavanja in prepoznavanja gest, na kakšne težave smo naleteli v razvoju, katerih nismo uspeli rešiti ter kakšne so končne zmožnosti zaznavanja in prepoznavanja gest. Četrto poglavje razlaga implementacijo algoritmov zaznavanja in prepoznavanja gest v aplikaciji. V sklopu tega poglavja je opisan razred Gesture, ki predstavlja objekt geste, razložen je način shranjevanja gest v aplikacijo, opisana pa je tudi aktivnost, ki pomaga pri upravljanju z gestami. Za zaključek so navedene še izboljšave, ki služijo kot orientacija za nadaljnje delo.

Poglavje 2

Teoretično ozadje

2.1 Opis pametnega telefona in operacijskega sistema Android

Pametni telefon je mobilni telefon z operacijskim sistemom in je na splošno bolj podoben računalniku, saj je precej bolj zmogljiv kot ostali telefoni in ponuja več funkcij za uporabo (pogovori, slikanje in snemanje z integrirano kamero, shranjevanje podatkov, predvajanje multimedije in podobno). Za razvoj diplomske naloge smo si izbrali operacijski sistem Android, ki je osnovan na Linuxu.

2.2 Opis senzorjev

Večina naprav, ki podpirajo operacijski sistem Android, ima vgrajene senzorje, ki pobirajo različne podatke iz okolja. Zajeti podatki imajo zelo visoko natančnost in zanesljivost, uporabimo pa jih lahko za nadaljnje procesiranje. Operacijski sistem podpira tri kategorije senzorjev [2].

Senzorji gibanja merijo krožne sile in sile pospeška v x, y ter z smeri. Ta sklop vsebuje pospeškometre, težnostni senzor, žiroskop in senzor rotacijskih vektorjev.

Senzorji okolja pobirajo podatke iz okolja, kot so pritisk, svetloba, temperatura zraka, vlažnost zraka in druge. Ta sklop vsebuje barometre, fotometre in termometre.

Pozicijski senzorji merijo fizično pozicijo naprave. Ta sklop vsebuje orientacijski senzor ter magnetometre.

Nekateri senzorji so osnovani na strojni, drugi na programski opremi. Senzorji, osnovani na strojni opremi, so vgrajeni v napravo in so praviloma precej različni (dražje naprave imajo posledično dražje in bolj natančne senzorje kot cenejše). Večina senzorjev je strojne narave, iz njih pa izhajajo senzorji, osnovani na programski opremi. Primer sta senzor za linearni pospešek in gravitacijski senzor, ki sta rezultat procesiranja integriranih senzorjev in se ne pojavljata v fizični obliki. Ker pa so senzorji te vrste odvisni od strojnih senzorjev, tudi tukaj velja pravilo dražjih naprav.

Večina aktualnih Android naprav vsebuje le pospeškometer in žiroskop, tiste dražje pa vsebujejo še preostale senzorje, ki niso veliko v uporabi povprečnega uporabnika. V nekaterih napravah se lahko pojavljajo tudi duplikati senzorjev, saj se s tem zagotovi večja pokritost območja, v katerem merimo. Primer je senzor gravitacijskega polja, ki je bolj občutljiv na majhne spremembe, in druga različica, ki je bolj občutljiva na večje spremembe.

Za prepoznavanje in zapisovanje naših gest smo se odločili za uporabo linearnega pospeškometra, saj za razliko od normalnega pospeškometra ne vsebuje težnostnega pospeška, ki za naše geste ni relevanten. V m/s^2 nam vrne pospešek v določeni smeri koordinatnega sistema.

2.3 Senzorji gibanja in ogrodje Sensor

Operacijski sistem Android nam ponuja naslednje senzorje gibanja.

1. Senzorji, osnovani na strojni opremi:
    - pospeškometer,

- žiroskop.

2. Senzorji, osnovani na programski ali strojni opremi:

   - težnostni senzor,
   - linearni pospeškometer,
   - senzor rotacijskih vektorjev.

Senzorji gibanja so zelo koristni zaradi prepoznavanja in klasificiranja gest, kot so na primer zamah, rotacija, pretres ali nagnjenje naprave. Senzorji so lahko v uporabi neposredno, na primer igranje dirkalne igre, kjer z nagnjenjem naprave usmerjamo volan, ali pa posredno, primer je kalkulacija pospeškov avtomobila, ko imamo napravo v avtu.

Do podatkov posameznih senzorjev dostopamo preko ogrodja *Sensor*, ki je del paketa *android.hardware*. Paket vsebuje naslednje razrede in vmesnike [1].

1. Razred *SensorManager*, s katerim ustvarimo instanco razreda za nadzor senzorjev. Razred ponuja različne metode za dostop in upravljanje s senzorji. Razred prav tako vsebuje konstante, ki se uporabljajo za poročanje natančnosti senzorjev in njihovo kalibriranje.

2. Razred *Sensor*, s katerim ustvarimo instanco specifičnega senzorja. Vsebuje tudi metode, ki uporabniku omogočajo nastavitev zmogljivosti izbranega senzorja.

3. Razred *SensorEvent*, s katerim ustvarimo objekt dogodka senzorja, ki ponuja informacijo o trenutnih podatkih senzorja. Objekt dogodka senzorja vsebuje surove podatke senzorja, tip senzorja, ki je ustvaril dogodek, točnost in konsistentnost podatkov ter časovno oznako za dogodek.

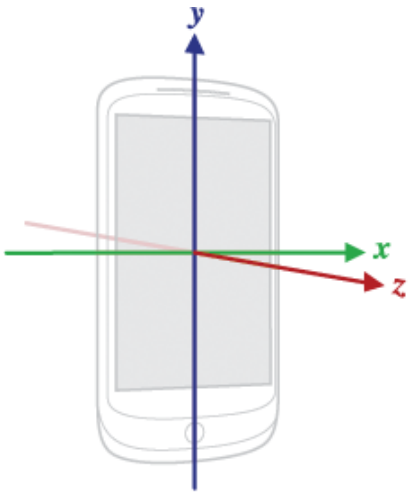
4. Vmesnik *SensorEventListener*, s katerim ustvarimo metodi, ki prejemata sporočila, ko se spremeni vrednost senzorja ali njegova točnost

(*onAccuracyChanged()* in *onSensorChanged()*). Ogrodje *Sensor* ponuja metode za identifikacijo senzorjev in nastavitev zmožnosti, kot so maksimalen domet, resolucija in poraba energije.

Vsi senzorji vračajo večdimenzijsko tabelo treh realnih vrednosti senzorja za vsak objekt razreda *SensorEvent*. Vrnjena vrednost pospeška (v m/s^2) se zapiše v tabelo *SensorEvent.values*, in sicer po vrsti vrednosti v smeri x, y in z osi.

2.4 Koordinatni sistem

Ogrodje Sensor uporablja kartezični koordinatni sistem, ki je prikazan na sliki 2.1. V pokončni (privzeti) postavitvi naprave os x kaže v desni smeri, os y navzgor, os z pa iz ekrana naprave proti nam.



Slika 2.1: Koordinatni sistem naprave

Koordinatni sistem se s spreminjanjem orientacije ekrana, horizontalno ali vertikalno, ne spreminja. Pri telefonih je privzeta orientacija vertikalna, pri tablicah pa horizontalna, kar je treba upoštevati pri dojemanju koordinatnega sistema.

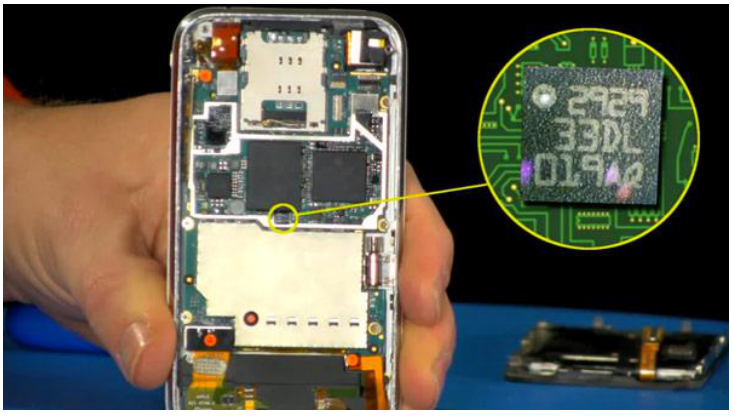
Koordinatni sistem uporabljajo pospeškometer, težnostni senzor, žiroskop, linearni pospeškometer in senzor geomagnetnega polja.

2.5 Opis pospeškometra

Pospeškometer je naprava, ki meri pospešek v določeni smeri. Pomembno je izpostaviti, da pospeškometer upošteva tudi gravitacijski pospešek, kar pomeni, da v mirujočem stanju kaže pospešek (g=9.81 m/s^2) v določeni smeri [7]. To obnašanje pospeškometra nas pripelje do dejstva, da senzor v prostem padu ali v vesolju kaže vrednost 0.

Konceptualno se pospeškometer obnaša kot masa na vzmeti. Ko na pospeškometer deluje sila pospeška, se masa premakne do točke, ko je vzmet sposobna maso pospešiti. Iz razlike med končno in začetno pozicijo mase nato izračunamo pospešek, ki ga izražamo v m/s^2.

V elektronskih napravah se nahajajo elektronski pospeškometri, ki delujejo na principu električnega toka.



Slika 2.2: Pospeškometer v pametnem telefonu iPhone

Pospeškometer, ki ga prikazuje slika 2.2, je razvit iz silicija in je sestavljen iz nepremičnega ohišja in seizmične mase. Ti enoti skupaj sestavljata diferencialni kondenzator. Ko se seizmična masa premakne oziroma zavibrira med dvema nepremičnima stebričkoma, pospešek zaznamo kot spremembo v električnem toku [4].

Meje zmožnosti zaznavanja sile pospeška takih pospeškometrov postavlja frekvenca resonance mase. Če hočemo doseči širšo operativnost pospeškometra, moramo povečati frekvenco resonance. To se naredi z lažjo seizmično maso (manjša kot je seizmična masa, manjša bo občutljivost senzorja) [5].

2.6 Geste

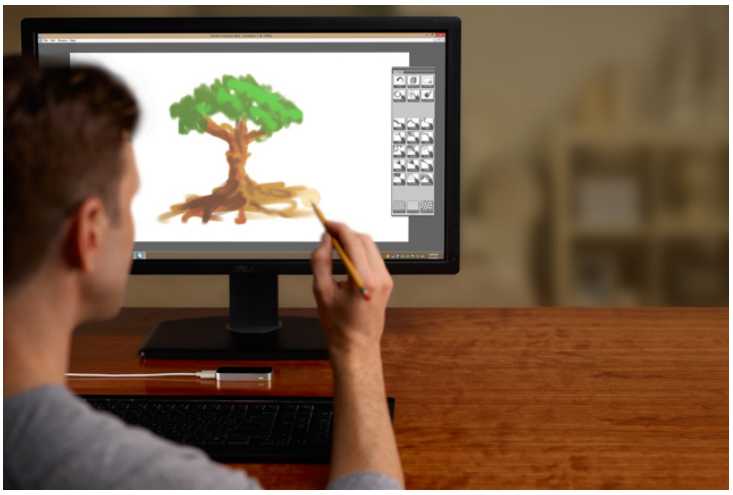
Gesta je način neverbalne komunikacije. V uporabi je že skozi celotno zgodovino človeštva, prvič pa se je začela uporabljati s preprostim mahanjem rok, ki pomeni vznemirjenost. Človeku najbolj domače so obrazne geste, ki jih delamo vsak dan, ne da bi se jih zavedali. S pomočjo gest smo razvili tudi znakovni jezik za gluhoneme, v uporabi pa je še v raznih prometnih signalizacijah, športih, glasbi, iz gest je sestavljena tudi igra Kamen, papir, škarje. Geste skratka opazimo na vsakem koraku in so postale zelo pomemben del našega življenja.

Ker pa nam geste močno olajšajo življenje in ker k temu stremi tudi sama ideja informacijske tehnologije, so se geste začele pojavljati tudi v tej veji tehnologije. Te so najbolj v uporabi v zabavni tehnologiji. Srečujemo jih v igrah, kot orodje za upravljanje v programih, nenazadnje jih srečamo tudi že integrirane v sam operacijski sistem. Vsak sodobni pametni telefon ali tablični računalnik trenutno deluje s pomočjo zaslona na dotik, na katerem uporabnik lahko izvaja večtočkovne geste. Z gestami se v operacijskem sistemu Android srečamo že takoj na začetku, saj ima operacijski sistem vgrajen varnostni sistem za odklep naprave. Ta varnostni sistem je sestavljen iz zaznavanja gest v dvodimenzionalnem prostoru zaslona na dotik. Uporabnik na poljuben način poveže do 9 točk, to pa operacijski sistem nato uporabi kot verifikacijo pri poskusu odklepanja naprave.

Za razumevanje našega diplomskega dela pa so najpomembnejše geste premikanja. Tudi ta vrsta gest je že integrirana v sam operacijski sistem Android. Primer je avtomatično spreminjanje orientacije ekrana iz horizontalne

v vertikalno postavitev, ki jo telefon izvede, ko se spremeni vrednost senzorjev. Druga omembe vredna funkcija je sprejem klica s premikom telefona k ušesu, ki nam olajša sprejemanje klicev, saj ob sprejemu klica ni potrebno klikniti na gumb. Podobno lahko s premikom telefona na neko podlago, z navzdol obrnjenim ekranom, klic enostavno in brez klikanja zavrnemo. Da bi se izognili čakanju pri brisanju besed v urejevalnikih lahko s hitrimi kretnjami levo in desno (pretres naprave) izbrišemo, kar smo v urejevalniku že imeli napisano, in lahko ponovno začnemo pisati od začetka.

Na področju gest premikanja je trenutno aktualna naprava Leap Motion (slika 2.3), ki olajšuje interakcijo z računalnikom.



Slika 2.3: Naprava za zaznavanje gest premikanja Leap Motion

Naprava vsebuje dve kameri in tri infrardeče diode LED, s katerimi zaznava premikanje prstov ali prstom podobnih objektov preko območja zaznavanja. Naštetih je nekaj gest, ki jih lahko uporabljamo z napravo:

- premiki z različnim številom prstov, kjer ima večje število prstov drugačno funkcijo,
- manipulacija virtualnih objektov s posnemanjem resničnih gest (primer na sliki 2.4),
- posnemanje klika miške s premikanjem prsta naprej,

- povečanje ali pomanjšanje slike s premikanjem dveh prstov skupaj oziroma narazen.

Slika 2.4: Premikanje objekta s pomočjo geste

Podobna naprava, ki se za zajemanje gest uporablja v zabavni tehnologiji, je Microsoftov Kinect, le da se ta uporablja za zaznavanje gest na področju celotnega telesa. Kinect je podobno kot v uvodu omenjeni kontroler Wii Remote orodje za interakcijo v igrah ali programih.

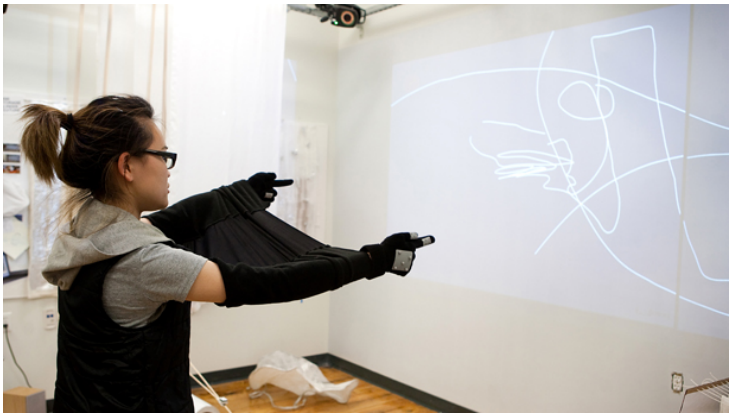
Geste premikanja so za marsikoga še vedno stvar prihodnosti. Dejstvo je, da je tehnologija za zaznavanje gest vedno boljša, naprave pa vedno manjše, zaradi teh dveh faktorjev pa je v uporabi potrošnikov naprav vedno več. Ideja, da bi lahko večino navideznih stvari upravljali povsem naravno, brez kontrolerjev v roki, je vedno bolj uresničljiva. Tako kot je bilo upravljanje računalnika z računalniško miško veliko bolj naravno kot samo s tipkovnico, se bo v prihodnosti računalništva pojavil tudi dan, ko bomo miško in tipkovnico zamenjali za napravo za zaznavanje gest.

Poglavje 3

Zaznavanje in zapis gest

3.1 Prostorske geste in njihov zajem

Prostorska gesta je način neverbalne komunikacije s pomočjo premikanja v prostoru. V računalništvu z izvedeno gesto sprožimo nek dogodek, ki ga gesta predstavlja. Na sliki 3.1 je prikazano risanje na računalniku s pomočjo gest, izvedenih z rokami. Izvedena gesta na računalniku sproži ustrezen dogodek (v primeru na sliki način izrisa).



Slika 3.1: Komunikacija z računalnikom s pomočjo gest

Za zajem prostorskih gest lahko uporabljamo različne, bolj kompleksne naprave, kot so kontroler Wii Remote, napravi Kinect ali Leap Motion, lahko

pa podatke zajamemo že s pomočjo preprostih senzorjev. Za razvoj našega diplomskega dela smo se odločili za uporabo senzorjev v pametnih telefonih, osredotočili pa smo se na linearni pospeškometer.

3.2 Predstavitev gest v računalniku

Geste v računalniku predstavimo s pomočjo poljubnih elementov. Za boljšo natančnost so v uporabi numerične, za večjo zanesljivost zaznavanja pa opisne vrednosti. Za slednje smo se odločili zaradi dejstva, da bi bilo shranjevanje in zaznavanje gest s surovimi (numeričnimi) vrednostmi senzorjev preveč potratno in nezanesljivo. Z opisnimi vrednostmi imamo tako zagotovljeno stoodstotno prepoznavanje gest.

Geste smo nato ločili na enostavne in kompleksne. Med enostavne sodi šest osnovnih gest:

X - premik v pozitivni smeri osi x,

X- - premik v negativni smeri osi x,

Y - premik v pozitivni smeri osi y,

Y- - premik v negativni smeri osi y,

Z - premik v pozitivni smeri osi z,

Z- - premik v negativni smeri osi z.

Kompleksne geste so sestavljene iz poljubnega števila enostavnih gest, kot rezultat pa lahko dobimo neomejeno število možnih kombinacij osnovnih elementov. Opisanih je nekaj primerov gest:

X,Y,X-,Y- - uporabnik nariše navpičen kvadrat,

X-,Y,X - uporabnik nariše črko C,

Y,Z,X-,Z- - uporabnik nariše vodoravni kvadrat.

Geste, sestavljene iz osnovnih elementov, so potem zapisane v objekt razreda *Gesture*, ki predstavlja zapis naše geste in ima naslednje štiri atribute:

id - zaporedna identifikacijska številka geste,

gestureName - ime geste, ki si jo uporabnik izbere sam,

pname - ime paketa aplikacije, ki jo vežemo na gesto,

gestureComponents - z opisnimi vrednostmi zapisano zaporedje osnovnih elementov geste.

Razred vsebuje prazen konstruktor in konstruktor z vsemi atributi. Za vsak atribut smo ustvarili tudi metode za nastavljanje in pridobivanje atributov, preko katerih atribute izbrane geste lahko posodabljamo in shranjujemo.

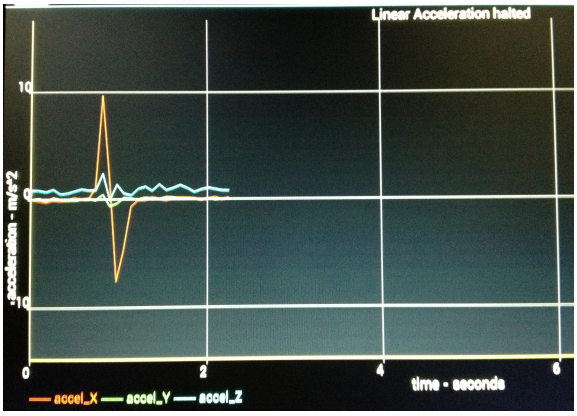
Pri zajemu podatkov smo naleteli na težavo s podatki v smeri osi z, prihaja namreč do odstopanj. Ko je naprava v vodoravnem položaju, vrednost pospeška v smeri x skače od 0,7 do 0,9 m/s^2, ko pa je v navpičnem položaju, pa pade na normalno vrednost (je podobna preostalima dvema). Težava je lahko v slabi natančnosti senzorja, saj bi linearni pospeškometer moral prikazovati vrednosti brez gravitacijskega pospeška, tu pa je očitno nek ostanek gravitacijskega pospeška v smeri z, saj pride do izraza samo v vodoravnem položaju, ko je os z usmerjena proti zemlji. Težavo smo rešili tako, da pri zajemu podatkov vrednosti v smeri z, ko je naprava v vodoravnem položaju, odbijemo neko povprečno vrednost 0,8 m/s^2.

3.3 Zaznavanje gest

Za zaznavanje gest smo napisali algoritem, ki najprej preveri, v katero smer deluje največji pospešek. Ko najde smer, preveri, če je sila pospeška višja od poljubnega praga zaznavanja. Prag uporabnik lahko samodejno nastavi, določa pa intenzivnost zaznavanja gest. Če je prag nižji, to pomeni zaznavanje gest že pri nižjih vrednostih pospeškov, če je prag višji pa obratno.

S testiranjem smo prišli do rezultatov, da pospešek pri bolj intenzivnih gibih skoči čez 4 m/s^2, pri manj intenzivnih gibih pa se giblje okoli 2 m/s^2. Če gre pospešek čez določen prag, smo zaznali enega izmed osnovnih elementov geste. Dokler je pospešek manjši od praga, algoritem predpostavlja, da geste ne izvajamo. S tem se izognemo raznim napakam, ki so rezultat neželjenega zaznavanjanja. Manjši gibi, ki jih nevede izvedemo med gesto, bi bili z napačno nastavljenim pragom oziroma brez praga tudi obravnavani, s tem pa bi izgubili na funkcionalnosti. Prag torej občutno izboljša robustnost zaznavanja.

Pri zaznavanju gest smo naleteli na težavo senzorjev, ki nam je ni uspelo rešiti in smo jo zato obvozili oziroma drugače izkoristili. Ko izvedemo gib v desno stran, sila pospeška najprej kaže v desno, ko pa se ustavimo, sila skoči v nasprotno smer. To je povsem naravni odziv senzorjev, saj pomeni stabilizacijo sile oziroma mehanizma senzorja, zato do teh odstopanj tudi pride. Problem je nastal pri zaznavanju gest, saj nam je ob izvedenem gibu v eno smer na koncu zapisal tudi gib v nasprotno smer. Težava je vizualno predstavljena na sliki 3.2, kjer smo izvedli gib v desno, nato pa se na mestu ustavili. Težavo smo rešili tako, da smo to obnašanje senzorjev enostavno ignorirali in zapisali vse elemente, ki se ob določenem gibu pojavijo. Stabilizacija senzorja je ves čas prisotna in se pojavlja ob vsakem gibu, tako da jo brez skrbi lahko izpustimo.



Slika 3.2: Problem stabilizacije senzorja

Poglavje 4

Implementacija gest v mobilni aplikaciji

4.1 Priprava delovnega okolja

Za razvijanje diplomskega dela smo potrebovali ustrezno programersko okolje, za katerega smo izbrali Android SDK (orodje za razvoj programske opreme). Android SDK nam v javanskem okolju omogoča razvoj aplikacij na operacijskem sistemu Android. Vsebuje razhroščevalnik, knjižnice, emulator QEMU, dokumentacijo in posamezne primere.

Trenutno je na trgu pametnih telefonov z operacijskem sistemom Android največ naprav z različicami API 4.1 Jelly Bean ter 2.3 Gingerbread. Zaradi večje pokritosti področja naprav za našo aplikacijo smo izbrali kot minimalno različico API 7 (Android 2.2 - Froyo) in kot ciljno različico API 17 (Android 4.2 - Jelly Bean). Različica API je knjižnica, ki vsebuje specifične spremenljivke, razrede, rutine in strukture. Pove nam, kako nekatere komponente programske opreme delujejo z drugimi. Nižja kot bo minimalna različica API, bolj bo aplikacija pokrivala področje naprav.

4.2 Implementacija zaznavanja in prepoznavanja gest

Za uspešno zaznavanje in prepoznavanje gest smo najprej morali pridobiti podatke iz senzorja linearnega pospeškometra. Podatke smo pridobili tako, da smo najprej ustvarili objekt razreda *SensorManager*, s katerim smo na objekt razreda *Sensor* vezali ustrezen senzor. Nato smo preverili, če je prišlo do spremembe vrednosti v katerem izmed senzorjev. To smo naredili z metodo *onSensorChanged(SensorEvent event)*. Tip senzorja, kjer je do spremembe prišlo, smo dobili z metodo *event.sensor.getType()*. Ker je zaznavanje gest odvisno od pravilnega senzorja, smo za zadnji korak pridobivanja podatkov morali tip senzorja enačiti z linearnim pospeškometrom, nato pa smo dobili vrednosti senzorja [1].

Naslednji korak je bila implementacija zaznavanja in prepoznavanja gest. Za oba dela smo uporabili algoritem, ki je opisan v Poglavju 3. Osnovni elementi geste, ki jih algoritem izračuna, se sproti zapisujejo v začasen znakovni niz, ta pa se pri prepoznavanju gest ohranja eno sekundo po zadnjem zaznanem elementu. Na podlagi pridobljenih rezultatov pri zapisu ali prepoznavanju smo nato izvedli ustrezne funkcije. Pri zapisu gest, se gesta dokončno shrani v podatkovno bazo in je takoj na voljo za uporabo pri prepoznavanju. Pri prepoznavanju se znakovni niz najprej primerja z vsemi atributi *gestureComponents* objekta razreda *Gesture*, ki so shranjeni v podatkovni bazi. Če pride do ujemanja, se zažene aplikacija, ki je vezana na izvedeno gesto.

4.3 Aktivnosti za izbiro aplikacije in shranjevanje gest

Prvi korak k izgradnji aplikacije je bil razred *AppChooserActivity*, ki nam omogoča izbiro aplikacije, ki jo vežemo na gesto. Vse informacije o aplikacijah, ki so nameščene na napravi, smo pridobili s pomočjo objekta razreda *PackageManager*. Ker pa je vseh aplikacij na telefonu preveč in s stališča

uporabniškega vmesnika to predstavlja problem za uporabnika, smo prikaz aplikacij razdelili na dva dela, sistemske (so del operacijskega sistema) in nesistemske (uporabnik jih naloži sam) aplikacije.

Ob izbiri aplikacije je potrebno preveriti, če je aplikacija že vezana na kakšno gesto, saj s tem onemogočimo pojavitev duplikatov. V primeru, da je aplikacija že v uporabi pri kateri izmed gest v podatkovni bazi, na ekran z objektom razreda *Toast* izpišemo kratko sporočilo v obliki obvestila, da aplikacijo neka gesta že uporablja. V primeru, da na izbrano aplikacijo še ni vezana nobena gesta, se ime paketa izbrane aplikacije shrani v atribut *pname* objekta *Gesture*. Za lažjo navigacijo med shranjenimi gestami smo v aplikacijo dodali tudi možnost za lastno določanje imena gest s pomočjo okna *AlertDialog*.

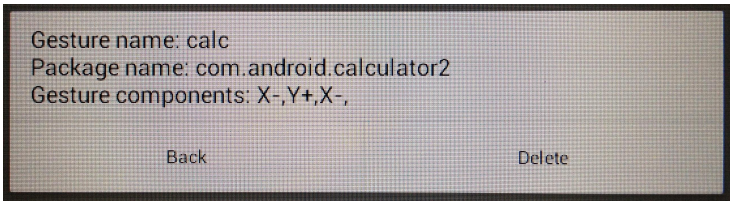
Shranjevanje gest nadzoruje razred *GestureRecorderActivity*. Na podlagi algoritma za zaznavanje gest, ki je opisan v Poglavju 3, uporabnik izvede poljubno gesto, algoritem pa sproti določa, kdaj je osnovni element geste prepoznan in za kateri element gre. Elementi geste se sproti shranjujejo v znakovni niz, ob končanju geste pa se celoten niz shrani v atribut *gestureComponents* objekta *Gesture*.

Aktivnost se po končanem shranjevanju vsake geste posodobi, saj s tem aplikaciji omogočimo zaznavanje pravkar shranjene geste.

4.4 Aktivnost za nadzor shranjenih gest

Za dostop do shranjenih gest smo ustvarili razred *ManagerActivity*, ki služi kot nadzornik za shranjene geste. V seznamu izpišemo vse shranjene geste, ob kliku na katerokoli izmed njih pa se nam odpre okno z informacijami o gesti in možnostjo njenega izbrisa. Ob izbrisu je zaradi posodobljenega zaznavanja pomembno aktivnost osvežiti, saj v primeru, da tega ne naredimo, izbrisano gesto aplikacija še vedno zaznava.

Na sliki 4.1 je prikazano okno *AlertDialog*, ki uporabniku izpiše ime aplikacije, ime paketa aplikacije in osnovne elemente geste.



Slika 4.1: Pogovorno okno z informacijami o gesti

4.5 Shranjevanje gest v podatkovno bazo

Ker večina aplikacij za normalno delovanje potrebuje shranjevanje podatkov, nam operacijski sistem Android ponuja več opcij za shranjevanje podatkov na napravo. Najbolj uporabljene opcije so:

- shranjevanje parov ključ-vrednost v datoteko SharedPreferences,
- shranjevanje v datoteko v datotečnem sistemu Android,
- uporaba podatkovne baze.

Za potrebe naše diplomske naloge je bila najbolj primerna opcija uporabe podatkovne baze SQLite, saj je majhna in hitra odprtokodna, relacijska podatkovna baza, kar pa je pri razvoju naše aplikacije zelo pomembno, saj je vse odvisno od hitrega prepoznavanja gest.

Razred *DatabaseHandler*, ki upravlja s podatkovno bazo, smo razširili z razredom *SQLiteOpenHelper* (razred, ki ureja kreiranje podatkovne baze in upravljanje verzije), nato pa smo prekrili dve njegovi metodi:

- *onCreate()* - tu smo napisali stavke za ustvarjanje naše tabele, metoda pa se pokliče, ko se ustvari podatkovna baza,
- *onUpgrade()* - ta metoda se pokliče, ko se baza nadgradi.

Ustvarili smo tudi vse operacije za upravljanje naše podatkovne baze (za ustvarjanje, branje, posodabljanje in brisanje elementov). Te operacije so:

- *public void addGesture(Gesture gesture)* {},

- *public Gesture getGesture(int id)* {},
- *public List<Gesture> getAllGestures()* {},
- *public int getGesturesCount()* {},
- *public int updateGesture(Gesture gesture)* {},
- *public void deleteGesture(Gesture gesture)* {}.

Glede na atribute, iz katerih je sestavljen naš objekt *Gesture*, smo si začrtali našo tabelo SQL, ki je prikazana v tabeli 4.1.

TABLE NAME: GESTURES

Field	Type	Key
id	INT	PRI
name	TEXT	
package_name	TEXT	
gesture	TEXT	

Tabela 4.1: Tabela atributov objekta Gesture za zapis v podatkovno bazo SQLite

V vseh aktivnostih naše aplikacije dostopamo do podatkovne baze z objektom razreda *DatabaseHandler*, s katerim kličemo zgoraj navedene metode za upravljanje in ustvarjanje podatkovne baze.

4.6 Zagon aplikacije

Za prepoznavanje geste uporabljamo identično kodo kot pri zapisu gest. Osnovni elementi geste se zapisujejo v začasen niz toliko časa, dokler v roku ene sekunde zagotovo zaznamo nov osnovni element. Ko z gesto zaključimo, se string z osnovnimi elementi geste, ki smo jo ravnokar izvedli, primerja z vsako gesto v naši podatkovni bazi. Če pride do ujemanja, se nato zažene aplikacija, ki je vezana na izbrano gesto.

Aplikacijo zaženemo s pomočjo objekta razreda *Intent*. Objekt razreda *Intent* je podatkovna struktura, ki vsebuje abstrakten opis akcije, ki bo izvedena. V našem primeru je ta akcija *startActivity(i)* (i je objekt razreda *Intent*), ki zažene novo aplikacijo. Preko upravljalca objektu razreda *Intent* pred zagonom aplikacije dodamo še atribut *pname* objekta razreda *Gesture*, saj s pomočjo tega podatka ve, katero aplikacijo mora zagnati.

4.7 Problem stalnega preverjanja ujemanja gest

Med preverjanjem ujemanja gest smo prišli do problema stalnega preverjanja. Aplikacija je ob izvajanju gest neprestano preverjala ujemanje gest, tudi ko smo neko aplikacijo že imeli odprto. Rezultat je bilo odpiranje novih aplikacij vsakič, ko je prišlo do ujemanja. To je s stališča uporabnosti izjemno slabo, saj aplikacija izgubi na funkcionalnosti. Problem smo rešili s preverjanjem vidnosti aplikacij. Ko aplikacijo zaženemo, se ta spremeni v proces, ki je lahko v štirih stanjih [3].

Aktivnost v ospredju je aktivnost, ki je trenutno vidna in je v uporabi. Je aktivnost z največjo prioriteto, operacijski sistem pa jo bo uničil le v primeru, da aktivnost prekorači porabo kapacitete pomnilnika naprave.

Vidna aktivnost je aktivnost, ki je uporabniku vidna, a ni v ospredju (primer je aktivnost, ki leži pod oknom AlertDialog). Je izjemno pomembna in ne bo uničena, razen če je to nujno za ohranitev aktivnosti v ospredju.

Aktivnost v ozadju je aktivnost, ki je uporabnik ne vidi in je trenutno na premoru. Nima kritičnega pomena, zato jo operacijski sistem lahko uniči, da dodeli prostor pomnilnika aktivnosti v ospredju. Če se proces aktivnosti v ozadju uniči, se bo ob ponovnem premiku uničene aktivnosti v ospredje poklicala metoda onCreate(Bundle) z onSaveInstanceState(Bundle), kar aktivnost postavi v prej shranjeno stanje.

Prazen proces ne poganja nobenih aktivnosti ali aplikacijskih komponent. Ko primanjkuje pomnilniškega prostora, operacijski sistem najprej uniči prazne procese.

Za pravico dostopa do informacij delujočih aplikacij smo najprej morali dodati vrstico datoteki *Manifest.xml*, ki naši aplikaciji dovoljuje dostop do informacij delujočih aplikacij.

```
<uses-permission android:name="android.permission.GET\_TASKS"/>
```

Nato smo ustvarili metodo, ki preverja, katera je trenutno vidna aplikacija, in nam izpiše ime njenega paketa. Preverimo, če je ime paketa trenutno vidne aplikacije enako kot *com.android.launcher* (domači zaslon operacijskega sistema) ali *com.example.gesturerecorder* (naša aplikacija). Če je enak enemu izmed teh, aplikaciji dovolimo preverjanje ujemanja gest. Brez te metode bi aplikacija ves čas preverjala ujemanje gest, kar pa bi pomenilo neželjeno odpiranje aplikacij, ko smo v eni izmed odprtih aplikacij že bili.

Poglavje 5

Sklepne ugotovitve

5.1 Delovanje in ocena uporabnosti aplikacije

Algoritem za zaznavanje in prepoznavanje bolj okornih in lažjih gest deluje dobro, pri kompleksnejših gestah, kjer bi bilo potrebno upoštevati vrednosti senzorjev v več smereh, pa je prepoznavanje nezanesljivo. S takim algoritmom bi lahko igrali igro s funkcijami, kot so premik levo, premik desno, skok, počep, ne bi bilo pa mogoče virtualno risati v poljubni smeri. Omogočeno bi bilo samo v smeri x ali y oziroma tudi z, če je govora o prostorskem in ne površinskem risanju. Ocena uporabnosti aplikacije bi bila odvisna predvsem od zahtev uporabnikov. Če bi uporabniki ciljali na zagon aplikacij samo z lažjimi gestami, potem bi bila aplikacija dokaj uporabna. Ker pa je dejstvo, da si uporabnik s to aplikacijo želi predvsem olajšati uporabo pametnega telefona, bi verjetno večina uporabnikov uporabljala našo aplikacijo samo za zagon tistih aplikacij, ki jih najbolj pogosto uporabljajo in za te bi uporabili najbolj preproste geste.

5.2 Izboljšave

Tako kot vsaka stvar, ki je v razvoju, ima tudi rezultat našega diplomskega dela prostor za izboljšave. Seznam vsebuje tri najpomembnejše izboljšave, s

katerimi bi izboljšali aplikacijo in prepoznavanje gest.

1. **Boljše prepoznavanje gest**. Gesto bi lahko razdelili na več osnovnih elementov, ki bi povečali spekter zaznavanja, kar bi posledično pomenilo še bolj robustno prepoznavanje in večji nabor gest. Lahko bi uvedli še osnovne elemente gest v kombinacijah vseh treh smeri. Primer so smeri XY, -XY, -Y-Z, ZXY in druge kombinacije. Smotrno bi bilo razmisliti tudi o uvedbi drugih senzorjev in njihovem skupnem delovanju. Žiroskop bi prepoznavanju tako dodal še tri osnovne elemente. To so Yaw (rotacija okoli osi y), Pitch (rotacija okoli osi z) in Roll (rotacija okoli osi x).

2. **Boljša klasifikacija gest**. Aplikacija bi ob namestitvi že vsebovala klasifikacijske razrede za vsak osnovni element geste. Ob izvedbi geste bi se ta klasificirala v razred, kateremu najbolj ustreza. Klasifikacijski razredi bi bili zgrajeni s pomočjo testnih množic, ki bi jih pridobili na snemanju gest, ki jih izvajajo različni testni subjekti (ljudje) [8]. Pri snemanju gest bi vsak izvedel določeno gesto drugače, kar bi bolj definiralo polje zaznavanja te geste. Pri izvajanju geste hitro pride do majhnih odstopanj, kar je problematično za zaznavanje, če je le to omejeno na pravila in ne na podlagi nekih že prej izvedenih gest. S pomočjo klasifikacije gest bi izničili ta problem in izboljšali sposobnost prepoznavanja.

3. **Boljši uporabniški vmesnik**. Uporabniku bi olajšali iskanje željene aplikacije tako, da bi v aktivnost dodali orodno vrstico *ActionBar*, ki ima opcijo integriranja funkcij. Tja bi dodali vnosno polje, kamor bi uporabnik vpisal ime željene aplikacije, ta pa bi nato skočila na vrh seznama za lažjo dostopnost. Druga izboljšava bi bila tudi aktivnost, ki grafično prikazuje osnovne elemente geste, medtem ko jo izvajamo. Namen te aktivnosti bi bil seznanjenje z elementi geste in uvajanje. Delovala bi kot pomočnik, saj bi ob neki gesti takoj dobili povratno informacijo, kaj smo dejansko naredili.

5.3 Zaključek

Področje gest na pametnih telefonih je še precej neraziskano. V uporabi so večinoma geste, ki so že vnaprej shranjene in jih prepoznavajo glede na klasifikacijo v testne množice, za katere ponavadi porabijo tudi do 50 ur in veliko ljudi, saj vsak posameznik gesto, ki jo snemajo, izvede nekoliko drugače. Z našo diplomsko nalogo smo določili lastne osnovne elemente geste ter uvedli algoritem za hitro shranjevanje in prepoznavanje gest. Aplikacija, ki smo jo razvili, bo tako lahko temelj nadaljnjemu raziskovanju, nekega dne pa mogoče celo osnova ali pa vsaj del nečesa, kar bo prišlo v vsakdanjo uporabo pametnih telefonov.

Literatura

[1] S. Komatineni, D. MacLean, S. Hashimi, *Pro Android 3*, New York, 2011

[2] "Sensors Overview", Sept. 5, 2013. Dostopno na: http://developer.android.com/guide/topics/sensors/sensors_overview.html

[3] "Activity", Sept. 5, 2013. Dostopno na: http://developer.android.com/reference/android/app/Activity.html

[4] J. Fingas, "Engineer Guy shows how a phone accelerometer works, knows what's up and sideways (video)", *Engadget*, Maj 22, 2012, Sept. 5, 2013. Dostopno na: http://www.engadget.com/2012/05/22/the-engineer-guy-shows-how-a-smartphone-accelerometer-works/

[5] "Piezoelectric Principle", Sept. 5, 2013. Dostopno na: http://www.mmf.de/piezoelectric_principle.htm

[6] F. Cavallo, G. Megali, S. Sinigaglia, O. Tonet, P. Dario, "Medicine Meets Virtual Reality 14: Accelerating Change in Healthcare: Next Medical Toolkit", *A Biomechanical Analysis of Surgeon's Gesture in a Laparoscopic Virtual Scenario*, IOS Press, Jan. 3, 2006

[7] P. Keir, J. Payne, J. Elgoyhen, M. Horner, M. Naef, P. Anderson, "Gesture-recognition with Non-referenced Tracking". Dostopno na: http://research.navisto.ch/pdf/3dui2006-3motion.pdf

[8] D. Ashbrook, T. Starner, "MAGIC: A Motion Gesture Design Tool", Sept. 5, 2013. Dostopno na: https://wiki.cc.gatech.edu/ccg/_media/people/dan/pap1375-ashbrook.pdf

[9] K. Graft, "Wii Remote Used for Robotic Research", *Edge*, Dec. 9, 2008 Sept. 5, 2013. Dostopno na: http://www.edge-online.com/news/wii-remote-used-robotic-research/

[10] D. Etherington, "Leap Motion Controller Tech To Be Embedded In, And Bundled With, Future HP Devices", *TechCrunch*, Apr. 16, 2013, Sept. 5, 2013. Dostopno na: http://techcrunch.com/2013/04/16/leap-motion-controller-tech-to-be-embedded-in-and-bundled-with-future-hp-devices/

[11] "How to move a DIV with your fingers... and the Leap Motion", *Leo Celis*, Sept. 5, 2013. Dostopno na: http://www.leocelis.com/how-to-move-a-div-with-your-fingers-and-the-leap-motion/

[12] N. Mayers, "Top 14 Technology Trends of 2012 that Changed Face of Technology", *Geeks Zine*, Sept. 5, 2013. Dostopno na: http://geekszine.com/top-14-technology-trends-of-2012-that-changed-face-of-technology-1819.html