Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Janez Jaka Cerar

Pametni telefon kot periferna naprava za krmiljenje v treh oseh

diplomsko delo
na univerzitetnem študiju

Mentor: doc. dr. Matija Marolt

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: 01973 / 2013
Datum: 5.11.2013

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

Kandidat: **JANEZ JAKA CERAR**

Naslov: **PAMETNI TELEFON KOT PERIFERNA NAPRAVA ZA KRMILJENJE V TREH OSEH**
USING SMARTPHONES AS 3D INPUT DEVICES

Vrsta naloge: DIPLOMSKO DELO UNIVERZITETNEGA ŠTUDIJA

Tematika naloge:

V diplomski raziščite zmožnosti uporabe pametnega telefona kot vnosne naprave za 3D manipulacijo s predmeti na računalniškem zaslonu. Opišite ozadje mobilnih platform in gonilniškega sklada Windows ter realizirajte ustrezne komponente, ki pametni telefon z operacijskim sistemom Android v okolju Windows predstavijo kot 3D vhodno napravo. Koncept preizkusite v kontekstu 3D programa za oblikovanje in evaluirajte njegovo uporabnost.

Mentor:

doc. dr. Matija Marolt

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:

prof. dr. Nikolaj Zimic

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Janez Jaka Cerar, z vpisno številko **24940584**, sem avtor diplomskega dela z naslovom:

Pametni telefon kot periferna naprava za krmiljenje v treh oseh

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno, pod mentorstvom doc. dr. Matije Marolta;
- 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 19. avgusta, 2013 Podpis avtorja:

Zahvaljujem se mentorju doc. dr. Matiji Maroltu za korektno obravnavo, družini za dolgoletno podporo in potrpežljivost, prijateljem in znancem za občasne besede vzpodbude, predvsem pa davkoplačevalcem Republike Slovenije za pretežno brezplačen študij.

Kazalo

Povzetek

V diplomski nalogi je opisan razvoj programske rešitve, ki omogoča uporabo pametnega telefona kot vnosne naprave za 3D krmiljenje objektov na računalniškem ekranu. Opisano je teoretsko ozadje s kratkimi povzetki konceptov in tehnologij ter uporabljenih razvojnih orodij. V nadaljevanju so predstavljene posamezne komponente rešitve, ki telefon na računalniku z operacijskim sistemom Microsoft Windows predstavi kot napravo HID. To vključuje uporabniško aplikacijo na telefonu Android za zaznavo lege in sklad gonilnikov HID. Predstavljena je uporaba gonilnika na primeru programa za 3D oblikovanje Trimble SketchUp, zaključujemo pa z diskusijo izdelane rešitve in možnosti za nadaljnje delo.

Ključne besede:

Android, HID, gonilnik, Bluetooth, 3D, mobilna naprava

Abstract

This graduate thesis describes the development of a software solution for the use of a smartphone as an input device for 3D manipulation of objects on a computer screen. We start with a description of the theoretical background with a brief overview of concepts, technologies and utilized development tools. What follows is a presentation of the individual components of the solution, which exposes the smartphone as a HID device on a computer running Microsoft Windows. This includes an Android user application for sensing orientation and a HID driver stack. Use of the driver is illustrated on the example of the Trimble Sketchup 3D modeling program, and we conclude with a discussion of the completed solution and possibilities for further work.

Keywords:

Android, HID, driver, Bluetooth, 3D, mobile device

Poglavje 1

Uvod

Računalniška miš je že dolga leta ena od dveh standardnih vnosnih naprav vsakega osebnega računalnika. V zadnjih letih jo na prenosnih računalnikih uspešno zamenjuje integrirana, na dotik občutljiva tablica (*touchpad*), ki pa v funkcionalnem smislu bistveno ne odstopa od te železne klasike. Dejstvo, da v mobilnem okolju vseprisotni na dotik občutljivi ekrani tukaj niso poželi praktično nikakršnega uspeha, priča o tem, kako dobro je pisana na kožo manipulacijam v dvodimenzionalnem prostoru tipične namizne aplikacije. Nekoliko manj to velja za nekatere specializirane aplikacije, kjer imamo opravka s 3D prostorom, kot na primer računalniške igrice ali programi za računalniško podprto oblikovanje. Iz tega razloga se je na tržišču z leti nabralo lepo število alternativnih perifernih vnosnih naprav - od joystickov in bolj specializiranih krmil v okolju računalniških iger, do raznoraznih 3D miši (npr. *SpaceController*, *3Dconnexion SpaceNavigator*, *SpacePilot Pro*) v aplikativni domeni 3D oblikovanja.

S povečevanjem računske moči in količine integriranih senzorjev v sodobnih »pametnih« prenosnih telefonskih aparatih, se odpira možnost inovativne uporabe v najrazličnejše namene. Cilj pričujočega diplomskega dela je tako, na strojni podlagi običajnega telefona z operacijskim sistemom Android, narediti napravo, ki bi v povezavi z osebnim računalnikom omogočala intuitivno pregledovanje modela v trodimenzionalnem prostoru. Naprava naj

bi se predstavila kot HID, kar bi omogočilo široko uporabnost v različnih aplikacijah znotraj platforme Microsoft Windows, konkreten primer uporabe pa se prikaže na programu za grafično modeliranje Trimble SketchUp. Glede na to, da gre za vertikalno povezavo uporabniškega in gonilniškega sloja na dveh napravah z različnima programskima platformama (Android in Microsoft Windows), predstavlja naloga v prvi vrsti problem integracije različnih konceptov in tehnologij v koherentno celoto.

Poglavje 2

Koncepti in tehnologije

2.1 Telefon kot računalnik

V zadnjih letih smo priča eksploziji računske moči v prenosnih telefonih, kar postavlja osnovno funkcijo zvočne komunikacijo v ozadje. Tako je sodoben mobilni telefonski aparat vse manj sredstvo za govorno komunikacijo in vse bolj ultra kompakten in prenosen osebni računalnik. V spletnih trgovinah (Apple App Store, Google Playstore) najdemo na sto tisoče bolj ali manj uporabnih aplikacij, mnoge od katerih bi še pred par leti prej povezovali z namiznimi računalniki, kot mobilnimi napravami.

2.2 Senzorji v mobilnih napravah

Z miniaturizacijo in masovno proizvodnjo je postala tehnološko in cenovno dostopna množica senzorjev, ki oblikujejo in bogatijo uporabniško izkušnjo. Tako si na primer ne znamo več predstavljati, da se ne bi naprava prilagodila, ko obrnemo zaslon v ležeč položaj. Paleta senzorjev, ki so na voljo, odpira možnosti za vrsto aplikacij, od relativno banalnih v stilu elektronskega kompasa ali vodne tehtnice, do inovativnih, med katerimi velja posebej izpostaviti trend obogatene resničnosti (*Augmented Reality*). V nadaljevanju je opisanih nekaj najpogostejših tipov vgrajenih senzorjev.

Ambient Light Sensor (ALS) *Senzor svetlobe v okolju* je uporaben pri določanju zadostne osvetlitve ekrana, kar pomaga pri energetski učinkovitosti mobilne naprave.

Proximity sensor *Senzor bližine* omogoča izklop zaslona ob telefoniranju, kar varčuje z energijo, hkrati pa prepreči neželeno interakcijo z zaslonom občutljivim na dotik.

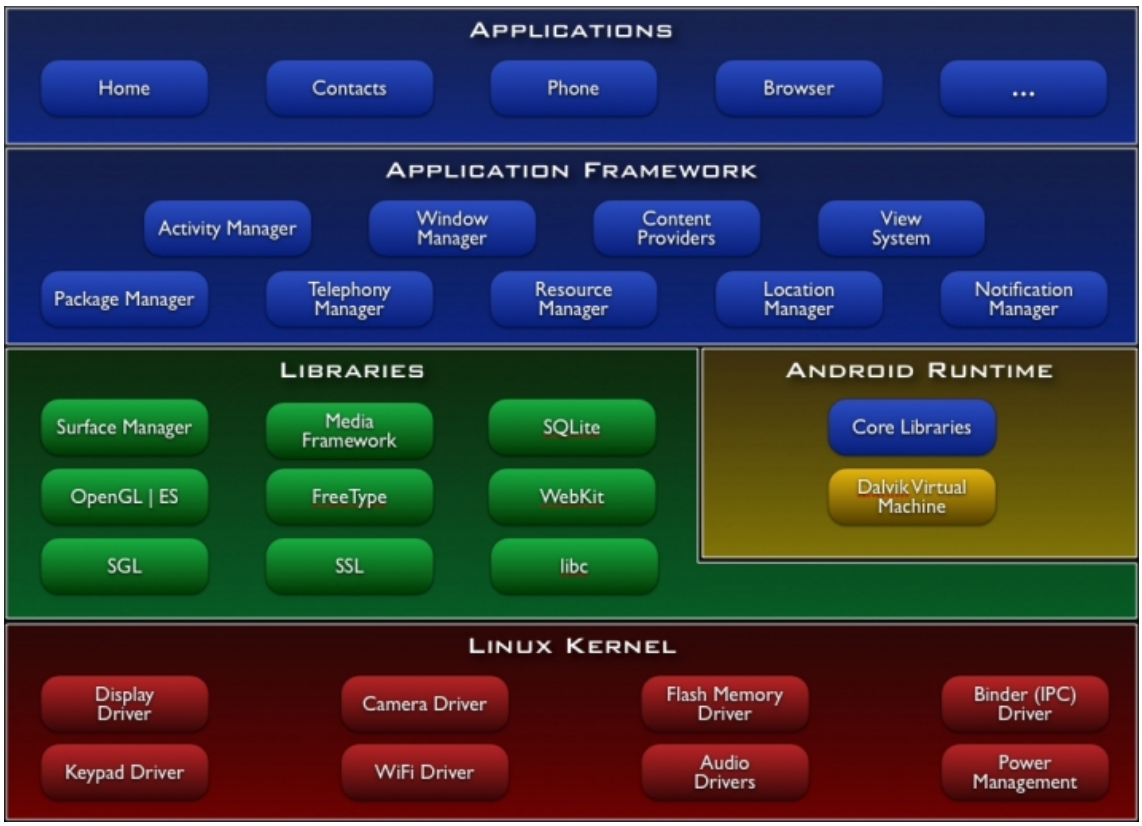
Global Positioning System (GPS) *Sistem satelitske navigacije*, na podlagi časovnega zamika potovanja signalov od štirih ali več satelitov k sprejemniku, omogoča določanje položaja naprave na nekaj metrov natančno.

Accelerometer / Gyroscope *Pospeškomer* in *žiroskop* omogočata zaznavanje linearnih premikov in rotacij v treh oseh.

Ambient Magnetic Field Sensor *Senzor magnetnega polja* se najpogosteje uporablja v funkciji digitalnega kompasa.

2.3 Platforma Android

V tem trenutku obstajata na trgu pametnih mobilnih naprav dva operacijska sistema, ki skupaj dosegata preko 90 % tržnega deleža: Android z 79 % in iOS s 13 %, merjeno v prodanih napravah[1] (podatek je iz drugega četrtletja leta 2013). Številčno premoč Androida gre pripisati predvsem odprtosti platforme, tako v smislu dostopnosti izvorne kode, kot v samem licenciranju. Medtem, ko je uporaba iOS dovoljena izključno na napravah proizvajalca Apple, pa licenca Androida dovoljuje praktično vsakemu proizvajalcu, da poljubno prilagodi in uporabi ta operacijski sistem na svojih napravah. Zato ni nobeno presenečenje, da smo priča pravi eksploziji različnih naprav, ki jih poganja Android, od telefonov in tabličnih računalnikov, pa do televizorjev, igralnih konzol, digitalnih fotoaparatov. Raziskave[2] so pokazale, da v juliju 2013 obstaja najmanj 11.868 različnih modelov naprav, na katerih je naložen operacijski sistem Android.



Slika 2.1: Prikaz strukture operacijskega sistema Android

Kot vidimo na sliki 2.1, operacijski sistem Android temelji na okrnjenem Linuxovem jedru, na katerem tečejo sistemske knjižnice, napisane v jeziku C/C++. Višji nivo predstavlja v Javi napisano aplikacijsko ogrodje (*Application Framework*), sledijo pa prav tako v Javi napisane aplikacije. Javanska koda ne teče v Javanskem virtualnem stroju (*Java Virtual Machine* - JVM), pač pa v Dalviškem virtualnem stroju (*Dalvik Virtual Machine* - DVM), zato je pred instalacijo prevedena iz Javine strojne kode (*byte code*) v kodo *Dalvik Executable*. Prednost DVM pred JVM je predvsem v nižji porabi pomnilnika in učinkoviti konkurenčnosti, kar je še kako pomembno vsled dejstva, da vsaka aplikacija teče v svoji instanci DVM.

2.4 Human Interface Device

Da bi se izognili podvajanju logike v gonilnikih, USB specifikacija [3] predvideva grupiraje naprav s podobnimi lastnostmi v tako imenovane razrede naprav (*device class*). Tako imajo podobne naprave, namesto vsaka svojega, le en skupen programski gonilnik, imenovan razredni gonilnik (*class driver*).

Eden od njih je razred HID, kar je kratica za *Human Interface Device* oziroma **naprava za komunikacijo s človekom**. Naprave, ki sodijo vanj, morajo biti sposobne na standardiziran način opisati svoje zmogljivosti razrednemu gonilniku. Čeprav USB-HID specifikacije [4] [5] objavlja *USB Implementers Forum, Inc.* (neprofitna korporacija v lasti skupine podjetij, ki so razvila tehnologijo USB), pa ta ni lastna samo implementaciji preko USB, pač pa v enaki meri drži tudi na ostalih fizičnih transportnih slojih (Bluetooth, I^2C...). Omenjene lastnosti ustvarjajo pogoje za enostaven priklop pestre vrste naprav, za kar tipično ni potrebno nalaganje po meri narejenih gonilnikov.

Opozoriti velja na dejstvo, da kljub nazivu, interakcija s človekom ni absoluten predpogoj za uvrščanje v razred HID. Tako sicer tipični predstavniki, kot so miši in tipkovnice, v celoti ustrezajo opisu, vendar med napravami HID najdemo tudi bolj eksotične primerke, kot so na primer programatorji integriranih vezij, termometri itd.

Kot rečeno, naprava sama opiše svoje lastnosti s tako imenovanimi deskriptorji, ki so statične podatkovne strukture, podrobneje opisane v nadaljevanju.

2.4.1 Deskriptor HID

Deskriptor HID (*HID descriptor*) na kratko opisuje samo napravo:

- katero verzijo specifikacije HID podpira naprava;
- morebitna država lokalizacije (pomemben podatek pri tipkovnicah na primer);

- število in tip podrejenih deskriptorjev (najmanj en deskriptor poročila).

2.4.2 Deskriptor poročila

Poročilo HID (*HID Report*) je dejanski podatkovni paket, ki omogoča izmenjavo podatkov med napravo HID in programskim odjemalcem. Obstajajo trije tipi poročil:

vhodno poročilo (*input report*) v smeri od naprave k aplikaciji, običajno ob spremembi stanja naprave npr. premiku miške;

izhodno poročilo (*output report*) v smeri od aplikacije k HID napravi, npr. za nastavitev stanja LED indikatorjev na tipkovnici;

lastnostno poročilo (*feature report*) za branje in pisanje, običajno podatkov povezanih s konfiguracijskimi nastavitvami.

Deskriptor poročila (*Report descriptor*) torej opisuje strukturo poročil, ki jih nudi naprava. Funkcionalnosti naprave so združene v eno ali več **krovnih zbirk** (*Top level collection*), ki na podlagi svojega tipa (tipkovnica, miš, senzor, prikazovalnik...) omogočajo programskemu porabniku identifikacijo zbirk, ki bi ga utegnile zanimati. V nomenklaturi specifikacije HID [4] se krovni zbirki reče tudi aplikacijska zbirka (*Application collection*). Poleg aplikacijskih, obstajajo še logične (*Logical collection*) in fizične zbirke (*Physical collection*). Slednje služijo za združevanje podatkov, ki se tičejo iste fizične entitete (npr. X in Y koordinati lokacije miši), medtem ko prva služi za grupiranje polj, ki skupaj tvorijo logično celoto (npr. podatkovni medpomnilnik in števec količine podatkov).

Posamezno funkcionalnost naprave se opisuje s tako imenovano **uporabo HID** (*HID Usage*), vsebinsko sorodne uporabe pa so združene v **strani uporabe HID** (*HID Usage Page*). Seznam vseh najdemo v tabelah uporab HID (*HID Usage Tables*)[5], ki tako predstavljajo neke vrste slovar možnih funkcionalnosti naprav HID. Poleg **identifikatorja uporabe HID** (*HID Usage ID*), katerega skupaj tvorita stran uporabe in uporaba sama, deskriptor HID

```
Usage Page (Generic Desktop),           ;Use the Generic Desktop Usage Page
Usage (Mouse),
    Collection (Application),           ;Start Mouse collection
    Usage (Pointer),
    Collection (Physical),              ;Start Pointer collection
      Usage Page (Buttons)
      Usage Minimum (1),
      Usage Maximum (3),
      Logical Minimum (0),
      Logical Maximum (1),              ;Fields return values from 0 to 1
      Report Count (3),
      Report Size (1),                  ;Create three 1 bit fields (button 1, 2, & 3)
      Input (Data, Variable, Absolute), ;Add fields to the input report.
      Report Count (1),
      Report Size (5),                  ;Create 5 bit constant field
      Input (Constant),                 ;Add field to the input report
      Usage Page (Generic Desktop),
      Usage (X),
      Usage (Y),
      Logical Minimum (-127),
      Logical Maximum (127),            ;Fields return values from -127 to 127
      Report Size (8),
      Report Count (2),                 ;Create two 8 bit fields (X & Y position)
      Input (Data, Variable, Relative), ;Add fields to the input report
    End Collection,                     ;Close Pointer collection
End Collection                          ;Close Mouse collection
```

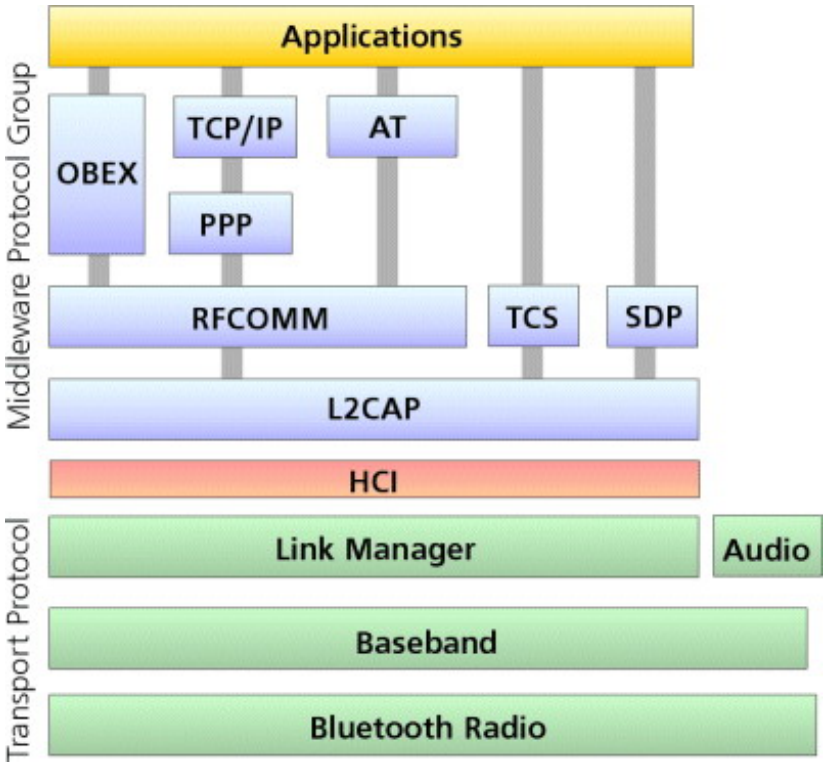
Slika 2.2: Primer deskriptorja poročila HID za generično miško

tvorijo še podatki o predstavitvi opazovane količine: dolžina zapisa, število enakih zapisov, logičen minimum in maksimum zaloge vrednosti, podatek ali gre za vhodno, izhodno ali lastnostno poročilo, ter vrsta zastavic, od katerih je najpomembnejša ta, da gre za absolutno ali relativno količino.

Na sliki 2.2 je prikazan primer deskriptorja poročila za računalniško miško.

2.5 Bluetooth

Bluetooth je standard za brezžično povezovanje naprav na kratkih razdaljah. Nastal je kot brezžična alternativa protokolu RS-232 in relativno hitro nadomestil na infrardeči tehnologiji temelječ protokol IrDA, katerega slabost je bila predvsem zahteva po direktni optični vidnosti obeh naprav. Deluje po načelu preskakovanja frekvenc na razpršenem spektru med 2,4 in 2,5 GHz (*frequency hopping spread spectrum*). Omogoča varno komunikacijo med



Slika 2.3: Shematski prikaz protokolskega sklada Bluetooth

različnimi napravami, kot so telefoni, računalniki, tiskalniki, GPS aparati, digitalni fotoaparati itd. Primeren je predvsem za aplikacije, kjer ni zahtev po visoki pasovni širini, slednjim je bolj na kožo pisan standard Wifi.

Deluje po sistemu *master/slave*, vsaka glavna naprava je lahko hkrati povezana z do sedmimi podrejenimi. Za vzpostavitev povezave je potrebno izvesti postopek sparjenja, ki z ali brez uporabnikovega posredovanja (odvisno od sposobnosti naprav) potrdi zvezo med napravama.

2.5.1 Protokoli

Bluetooth omogoča dve vrsti povezav: sinhrone za časovno kritične aplikacije, kot so prenos govora (*Synchronous Connection-Oriented link* - SCO) in asinhrone za paketne prenose podatkov (*Asynchronous Connection-Less link*

- ACL). Na sliki 2.3 vidimo shemo protokolskega sklada Bluetooth.

Logical Link Control and Adaption Protocol - L2CAP je vmesni člen med višjimi protokolskimi sloji in povezavo ACL. Skrbi za souporabo med višjimi protokoli, deljenje (in na drugi strani ponovno sestavljanje) podatkov na 64-kbitne pakete, skupinsko upravljanje (*Group Management*) - enosmerne prenose skupini drugih naprav in upravljanje s kvaliteto storitve (*Quality of Service Management*).

Radio Frequency Comunication - RFCOMM je protokol, ki na podlagi L2CAP skrbi za emulacijo serijskih vrat. Predstavlja preprost, zanesljiv podatkovni tok, z dobro podporo na večini operacijskih sistemov in kratko krivuljo učenja zaradi podobnosti protokolu TCP.

Service Discovery Protocol (SDP) omogoča objavljanje in zaznavo storitev, ki jih ponuja posamezna naprava.

2.5.2 Storitve in profili

Storitve so opisane s tako imenovanimi profili. Vsak profil ima svoj UUID (*Universally Unique IDentifier* - univerzalno edinstven identifikator), pri čemer so standardni profili Bluetooth opisani s 16-bitnim UUID, ki dopolnjuje osnovni 128-bitni Bluetooth UUID (`00000000-0000-1000-8000-00805F9B34FB`). Uporabniške storitve se identificirajo s poljubnim 128-bitnim UUID.

Profil SPP (*Serial Port Profile*) na podlagi protokola RFCOMM ponuja emulacijo serijskih vrat. Če naprava podpira profil SPP, jo lahko povežemo naprimer z računalnikom, na katerem se vzpostavijo virtualna serijska vrata, ki jih lahko uporabljamo povsem enako, kot običajna (ožičena) serijska vrata.

Profil HID (*Human Interface Device Profile*) nudi storitve protokola *Human Interface Device*, ki je sicer definiran za USB naprave. V resnici gre za preprosto ovojnico okoli USB implementacije in omogoča

standardizirano podporo za vnosne naprave, kot so miške, tipkovnice in druge vnosne naprave.

2.6 Gonilniški sklad v okolju Microsoft Windows

Gonilnik je računalniški program, ki omogoča komunikacijo med priključeno strojno opremo in operacijskim sistemom. Kot tak je specifičen za vsako kombinacijo naprave in operacijskega sistema. Microsoft za svoje operacijske sisteme objavlja dokumentacijo za razvoj gonilnikov v MSDN Library[6].

2.6.1 WDM

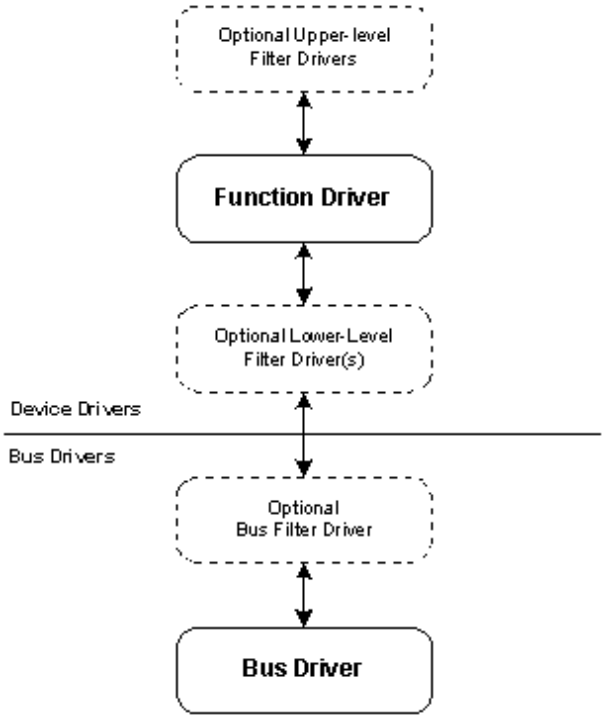
Windows Driver Model (WDM) je ogrodje za gonilnike za operacijske sisteme Microsoft Windows, uvedeno z različico Windows 98.

WDM temelji na večslojnem konceptu, katerega shematski prikaz vidimo na sliki 2.4. Kot gradniki slojev nastopajo različni tipi gonilnikov WDM:

Gonilnik vodila (*bus driver*) služi podpori za vhodno izhodno vodilo. Za vsa običajna vodila je gonilnike napisal že Microsoft (PCI, SCSI, USB itd), po potrebi pa lahko proizvajalec razvije tudi svoj gonilnik vodila.

Funkcijski gonilnik (*function driver*) je glavni gonilnik naprave, brez katerega (razen v izjemnih primerih) njena uporaba ni mogoča. Glavne naloge funkcijskega gonilnika so nadziranje pisanja in branja podatkov z naprave in upravljanje politike porabe moči naprave.

Filtrirni gonilnik (*filter driver*) je opcijska komponenta, ki omogoča razvijalcem spreminjanje obnašanja ali drugo dodano vrednost na poljubnem nivoju gonilniškega sklada. Filtrirni gonilnik prestreže podatke z nižjega nivoja in jih obdelane posreduje naprej na višji nivo. Če bi želeli denimo vgraditi nelinearne pospeške v premike miške, bi ustrezen filtrirni gonilnik namestili med gonilnik vodila in funkcijski gonilnik.



Slika 2.4: Shematski prikaz večslojne arhitekture WDM

Komunikacija med različnimi nivoji poteka s pomočjo vhodno/izhodnih paketov zahtev (*I/O request packet* - IRP).

Kot odgovor na nepotrebno podvajanje kode v množici gonilnikov za podobne naprave se je za funkcijske gonilnike uvedla arhitektura *port/miniport*. Gonilnike tipa *port*, v katerih je implementirana funkcionalnost, ki je skupna celemu razredu naprav, razvije Microsoft, gonilnike *miniport*, kjer je zajeta samo funkcionalnost specifična za ciljno napravo, pa proizvajalci strojne opreme. Tak par torej predstavlja kompleten funkcijski gonilnik, s to prednostjo, da je razvoj za proizvajalca naprave precej enostavnejši, hkrati pa se zmanjša možnost programskih hroščev, ker je dobršen del občutljive kode že razvit in temeljito preizkušen.

WDM torej uvaja nekaj konceptov, zaradi katerih predstavlja bistveno izboljšavo v primerjavi s svojim predhodnikom (VxD):

- eno razvojno okolje za vse podprte operacijske sisteme;
- enotna izvorna koda za vse podprte operacijske sisteme;
- Microsoft napiše gonilnike vodil (*bus drivers*);
- spodbujanje *port/miniport modela* gonilnikov - za kar največje število razredov naprav Microsoft prispeva *port* del gonilnikov, kar proizvajalcem naprav močno poenostavi razvoj gonilnikov.

2.6.2 WDF (Windows Driver Foundation)

Kljub vrsti prednosti, ima WDM nekaj pomanjkljivosti, ki otežujejo razvoj in slabšajo stabilnost sistema:

- Interakcije z upravljanjem porabe moči in sistemom *Plug and Play* so kompleksne za implementacijo. Če je to prepuščeno proizvajalcem naprav, zaradi programskih hroščev v gonilnikih neizbežno prihaja do situacij, kjer pravilen prehod v ali iz hibernacije ni možen.
- Težavna implementacija prekinitev vhodno/izhodnih operacij.

- Velika količina podporne kode, ki ima več skupnega z internalijami operacijskega sistema, kot pa s samo poslovno logiko gonilnika.
- Ni podpore za gonilnike, ki bi tekli v uporabniškem načinu.

Zaradi naštetega je WDM od različice Windows 2000 dalje nadgrajen z WDF (*Windows Driver Foundation*). Gre za orodja, ki predstavljajo objektno orientiran model za razvoj gonilnikov na platformi MS Windows. Za razliko od WDM je tukaj v ospredju princip konceptualne razširljivosti. To pomeni, da za večino funkcij gonilnika obstaja neko smiselno privzeto obnašanje, ki ga je možno po potrebi spremeniti oziroma razširiti, sicer pa ostane implementacija skrita. Tako je izvorna koda osvobojena večine balasta iz naslova kopiraj&prilepi, hkrati pa še vedno omogoča kontrolo nad vsemi strukturami WDM, če je to potrebno.

Glavna gradnika ogrodja WDF sta KMDF in UMDF.

KMDF

Kernel Mode Driver Foundation (KMDF) predstavlja infrastrukturo za razvoj gonilnikov v načinu jedra. Nanjo lahko gledamo tudi kot funkcionalno okostje WDM gonilnika, ki v skladu s paradigmami WDF na objektno orientiran način skozi nastavitve in metode povratnih klicev (*callback methods*) ponuja celotno funkcionalnost WDM. KMDF API (*Application Programming Interface*) je napisan v programskem jeziku C.

UMDF

User Mode Driver Foundation (UMDF) je ogrodje za razvoj gonilnikov, ki tečejo v uporabniškem načinu.

Dejstvo, da teče v uporabniškem načinu, ima celo vrsto pozitivnih posledic:

- **Enostavnejše gonilniško okolje** - ni potrebno skrbeti za celo vrsto nižjenivojskih podrobnosti, ki so prisotne v jedrnem načinu.

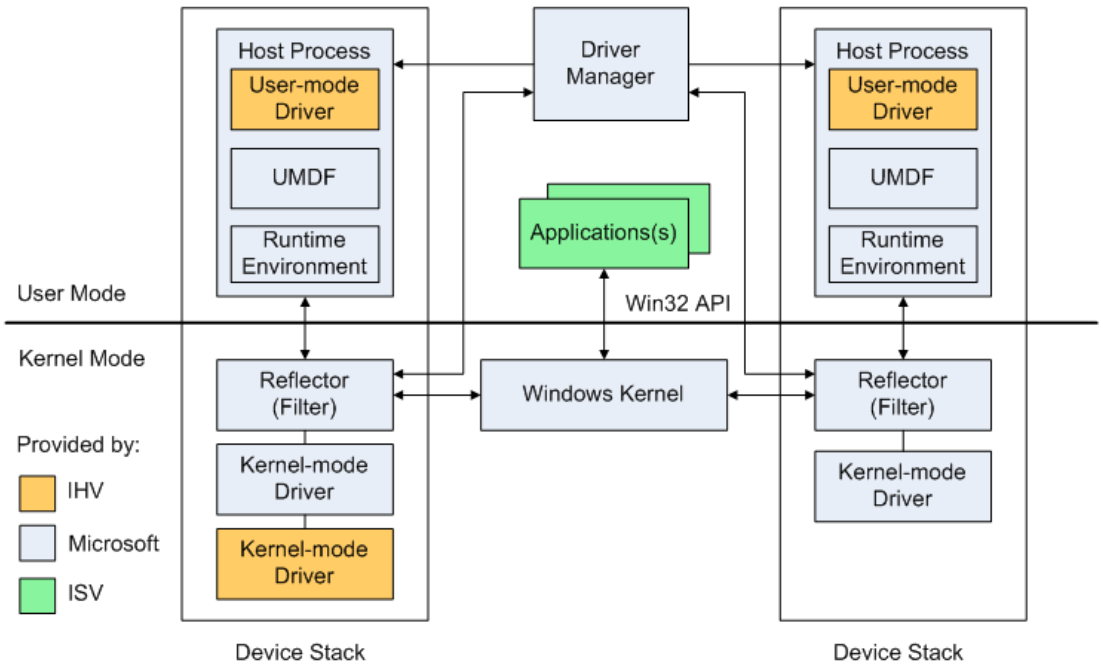
- Namesto klicev jedrnih funkcij je na voljo aplikacijskemu programerju dobro znani **API Win32**.

- Za razvoj je na voljo programski jezik **C++**, ki v primerjavi z jezikom C prinaša nekatere prednosti višjenivojskih jezikov.

- **Večja stabilnost sistema** - v uporabniškem načinu ni možno prepisati pomnilniških lokacij drugega uporabniškega procesa ali jedrnega procesa. Prav tako je morebitno "sesuvanje" omejeno na sam uporabniški proces, zato ni potrebe po ponovnem nalaganju celega sistema (tako imenovani BSOD - *Blue Screen Of Death*, oziroma moder ekran smrti)

Iz naštetih razlogov je priporočljiva uporaba povsod, kjer ni zahteve po rokovanju s prekinitvami, neposrednemu dostopu do pomnilnika (DMA), ali dostopu do drugih resursov jedra, kot npr. ne-ostranjen pomnilnik (mimo upravljalca virtualnega pomnilnika).

Na sliki 2.5 je shematski prikaz arhitekture UMDF. Kot vidimo, gre za neke vrste emulacijo, saj je v vsakem primeru na dnu sklada jedrni gonilnik, znotraj jedrnega dela pa še filtrirni gonilnik (reflektor), ki skrbi za povezavo s preostankom gonilniškega sklada, ki teče v uporabniškem načinu. Na levi je primer s po meri narejenim uporabniškim in jedrnim gonilnikom, seveda pa lahko naredimo rešitev tudi brez slednjega (desno).



Slika 2.5: Arhitektura gonilniškega modela UMDF

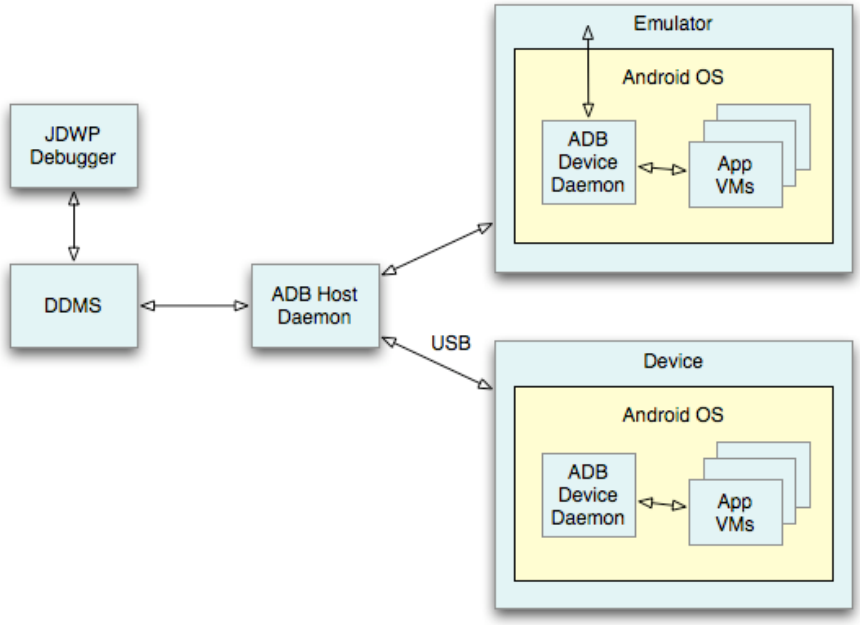
Poglavje 3

Razvojna orodja

Tako, kot je široka paleta tehnologij, potrebnih za reševanje obravnavanega problema, je pester tudi nabor uporabljenih razvojnih orodij. V nadaljevanju sledi kratek opis najpomembnejših.

3.1 Eclipse ADT

Aplikacije za platformo Android so običajno napisane v Javi. Za Javo je na voljo cela vrsta integriranih razvojnih okolij (*Integrated Development Environment* - IDE), tudi brezplačnih, med katerimi je zaradi Googlove podpore za razvoj Androidnih aplikacij posebej primeren **Eclipse**[9]. Omenjena podpora se kaže v obliki vtičnika (*plugin*) **Android Developer Tools (ADT)**[10], ki v grafično okolje integrira dobršen del orodij, ki so sicer na voljo kot orodja ukazne vrstice (SDK Tools) v okviru kompleta programskih orodij za razvijanje programske opreme **Android SDK**[11]. Najhitrejša pot do kompletnega razvojnega okolja za Android je preko paketa **ADT Bundle**[8], ki vsebuje tako Eclipse, vtičnik ADT, kot tudi Android SDK.



Slika 3.1: Razhroščevanje naprave Android preko ADB

3.2 Android Debug Bridge

Težko si predstavljamo razvoj netrivialnega programa brez možnosti razhroščevanja. Za večino aplikacij, ki jih razvijamo za okolje Android, je v ta namen najbolj primerna uporaba emulatorja, ki teče na istem računalniku, kot razvojno okolje. V konkretnem primeru pa je bistvo aplikacije interakcija z vgrajenimi senzorji, česar emulator žal ne omogoča. K sreči obstaja možnost razhroščevanja neposredno na ciljni napravi, ki jo z računalnikom povežemo z USB kablom. Protokol, ki to omogoča, se imenuje **Android Debug Bridge**, shematski prikaz delovanja vidimo na sliki 3.1. V Android SDK najdemo orodje ukazne vrstice **adb**, s katerim lahko izvajamo različne ukaze. Najzanimivješa med njimi sta ukaza za instaliranje aplikacij in odpiranje ukazne lupine (*command shell*). Slednje nam omogoča udoben dostop do praktično vseh resursov na priključeni napravi (npr. z orodjem ukazne vrstice **sdptool** lahko izpišemo SDP zapise protokola Bluetooth).

3.3 Visual Studio 2012

Visual Studio je Microsoftovo integrirano razvojno okolje za razvoj programov za operacijske sisteme Microsoft Windows. Omogoča razvoj v jezikih VB.NET, F#, C# in C/C++. Slednjega smo uporabili za razvoj gonilniškega dela rešitve. Del Visual Studia, ki omogoča razvoj aplikacij v jezikih C/C++ se imenuje Visual C++. Čeprav obstaja brezplačna različica Express, pa WDK (opisan v nadaljevanju) zahteva uporabo ene od plačljivih različic (Professional, Premium ali Ultimate).

3.4 WDK

Za razvoj gonilnikov je potrebna uporaba specializiranih knjižnic, orodij, dokumentacije, ki niso del običajnih okolij, namenjenih razvoju namiznih aplikacij. V ta namen Microsoft izdaja **Windows Driver Kit - WDK**[12] (nekoč znan pod imenom Driver Development Kit - DDK). Trenutno je aktualna verzija 8, ki jo je možno popolnoma integrirati v Visual Studio 2012. Poleg ostalega, so v WDK vključeni številni programski primeri, ki lahko predstavljajo dobro osnovo za razvoj našega gonilnika.

3.5 HID Descriptor Tool

Pri razvoju gonilnika HID neizbežno pride trenutek, ko je potrebno napisati deskriptor poročila HID. Pri tem se sicer lahko opiramo na dokumentacijo, vendar pa je končni rezultat zapisan v, za človeka, precej zapleteni binarni obliki. Zato nam je v veliko pomoč orodje **HID Descriptor Tool**[13], ki iz človeku razumljive predstavitve podatke pretvori v binarni zapis.

3.6 Razhroščevanje v načinu jedra

Ker statusa in izhoda gonilnika nikjer ne vidimo, je, dokler ne vzpostavimo delujoče povezave s programskim odjemalcem, iterativno približevanje temu cilju precej težavna reč. Pomagamo si lahko s programom **WinDbg** (vključen v WDK), ki je večnamenski razhroščevalec (*debugger*) za okolje Microsoft Windows. V načinu jedra lahko razhroščujemo samo z ločenega računalnika, kar je po svoje razumljivo. V načinu jedra teče namreč vse od osveževanja zaslonske slike dalje - ob prekinitvi izvajanja bi ne postorili več kaj dosti. Z virtualizacijo odpadejo zahteve po dveh fizičnih računalnikih in povezavo s kablom, tako da je smiselno za gostovanje gonilnika v razvoju uporabiti enega od programov za virtualizacijo. Sami smo uporabili **VMWare Player**[15].

Za lažjo vzpostavitev in hitrejše delovanje povezave lahko uporabimo VirtualKD[14], ki vgradi podporo v jedro operacijskega sistema virtualnega računalnika.

Za razhroščevanje z Visual Studiem bomo potrebovali Windows 8, v vsakem primeru pa lahko pregledujemo izpise v WinDbg (če na ustreznem mestu v kodi gonilnika s klicem metode `DbgPrint` ali `OutputDebugString` pošiljamo sporočila).

Poglavje 4

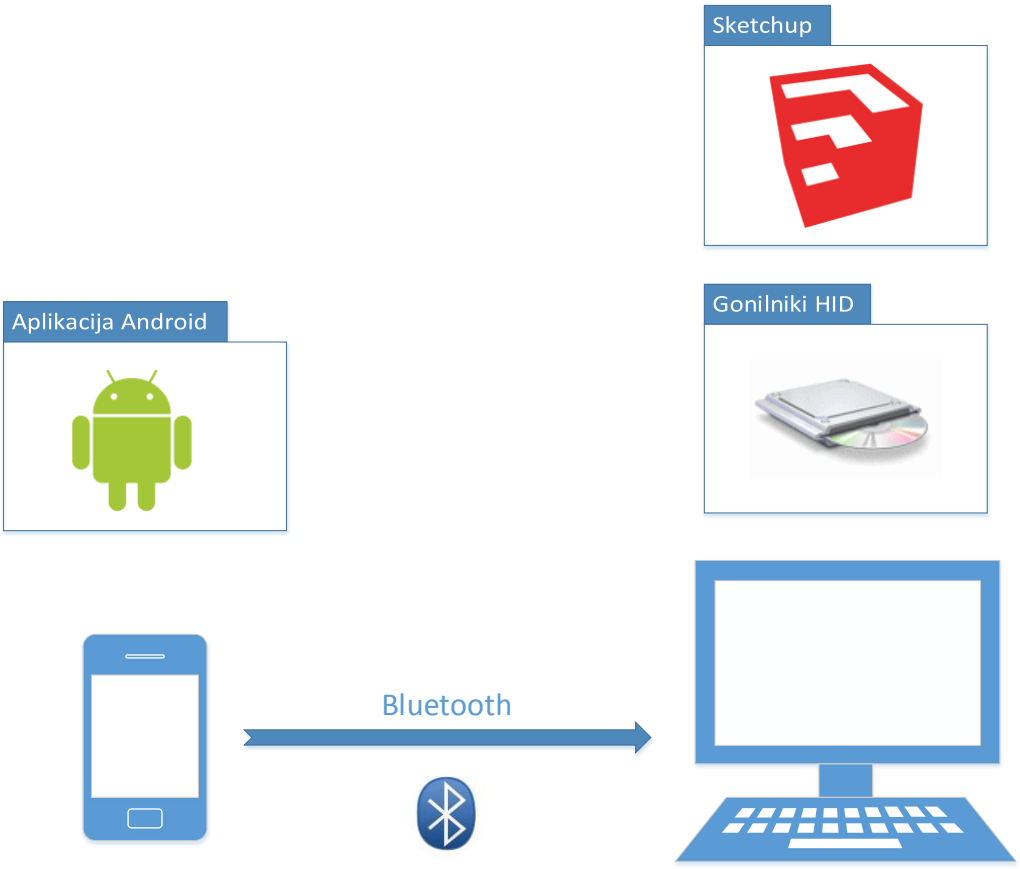
Implementacija

4.1 Arhitektura

Za uvod v opis implementacije na hitro opišimo glavne gradnike, prikazane na sliki 4.1. Vhodno napravo predstavlja mobilni telefon z operacijskim sistemom Android, na katerem teče aplikacija, ki zajema podatke vgrajenih senzorjev o svoji legi. Z osebnim računalnikom je vzpostavljena brezžična povezava Bluetooth, preko katere posreduje zajete podatke gonilniku HID. Uporabniška aplikacija, katero krmilimo, na koncu zahteva od gonilnika aktualne podatke in ustrezno prilagodi prikaz na ekranu.

Osnovna ideja koncepta HID je, da ponuja tako ogrodje, da je lahko vsa logika, ki jo potrebujemo za priklop poljubne naprave, vgrajena v napravo samo. Tudi v našem primeru bi bilo to uresničljivo, če bi operacijski sistem Android nudil podporo za Bluetooth profil HID. Ker temu ni tako, bi pot do take rešitve vodila skozi modifikacije Androidovega sklada Bluetooth BlueZ, torej namaščanje prilagojenega operacijskega sistema. To seveda ne bi bilo prav racionalno, zato je bilo potrebno poiskati alternativen pristop. Ta se kaže v obliki gonilnika, ki komunicira z ne-HID napravo, na drugi strani pa implementira vmesnik HID.

Za razvoj smo uporabili telefon Samsung Galaxy Ace z nameščeno različico Androida CyanogenMod 7 (Android 2.3.7).



Slika 4.1: Shematski prikaz rešitve

4.2 Aplikacija Android

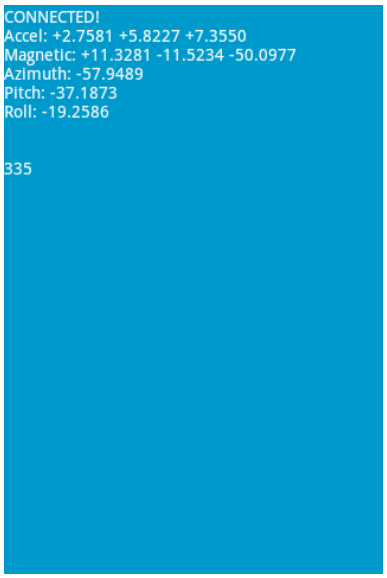
Naloga aplikacije na telefonu Android je zajem podatkov senzorjev, pretvorba surovih podatkov v logične količine in komunikacija z računalnikom preko povezave Bluetooth. Posledično je ta precej preprosta v smislu uporabniškega vmesnika in interakcije z uporabnikom.

4.2.1 Zajem podatkov

Vizualna komponenta se v okolju Android imenuje aktivnost (*activity*). Naša aplikacija ima eno samo, ki je uporabljena za kontrolni izpis trenutnih senzoričnih podatkov. Izbrali smo celozaslonsko (*fullscreen*) aktivnost, zaslonski posnetek je prikazan na sliki 4.2. Od vrha proti dnu si sledijo podatki o:

- stanju povezave,
- nizu vrednosti pospeškomera,
- nizu vrednosti magnetnega senzorja,
- izračunanem kotu rotacije okoli osi z,
- izračunanem kotu rotacije okoli osi x,
- izračunanem kotu rotacije okoli osi y,
- števcu prenesenih stanj.

Iz razlogov, ki jih podrobno opisujemo v razdelku o gonilniku HID, smo za predstavitev lege vnosne naprave izbrali Eulerjeve kote. Za to smo uporabili podatke iz dveh senzorjev - senzorja magnetnega polja in pospeškomera. V ta namen smo v aktivnosti implementirali interface `SensorEventListener`. Tako se ob inicializaciji prijavimo za spremljanje sprememb na omenjenih senzorjih:



Slika 4.2: Zaslonska slika aplikacije Android

```
sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);

sensorManager.registerListener(this,
          sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER),
          SensorManager.SENSOR_DELAY_GAME);

sensorManager.registerListener(this,
          sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD),
          SensorManager.SENSOR_DELAY_GAME);
```

`SensorManager.SENSOR_DELAY_GAME` je konstanta, ki določa frekvenco osveževanja. Gre sicer zgolj za neobvezno navodilo SensorManagerju in tudi sicer ni frekvenca nikjer v dokumentaciji količinsko opredeljena. Tipično pa na trenutno aktualnih napravah povzroči osveževanje na okoli 40 ms. Ob vsaki spremembi se nato pokliče metoda `onSensorChanged`, v kateri, glede na tip senzorja v parametrih dogodka, shranimo en ali drugi podatek. Ko

imamo oba podatka, preostane le še izračun matrike vrtenja in Eulerjevih kotov. Oboje je takorekoč trivialno, saj Android API v obliki razreda SensorManager ponuja že izdelani metodi:

```
SensorManager.getRotationMatrix(R, I, accels, mags);
SensorManager.getOrientation(R, values);
```

Na sliki 4.3 vidimo koordinatni sistem, uporabljen v Androidovi metodi `SensorManager.getOrientation()`. Vse rotacije v tem kontekstu so desnosučne, kar pomeni, da se vrednosti povečujejo v nasprotni smeri urinega kazalca. Za pravilno tolmačenje Eulerjevih kotov je pomemben vrstni red transformacij, ki opisujejo položaj: najprej rotacija okoli osi Z, nato rotacija okoli osi X, nazadnje pa še rotacija okoli osi Y.

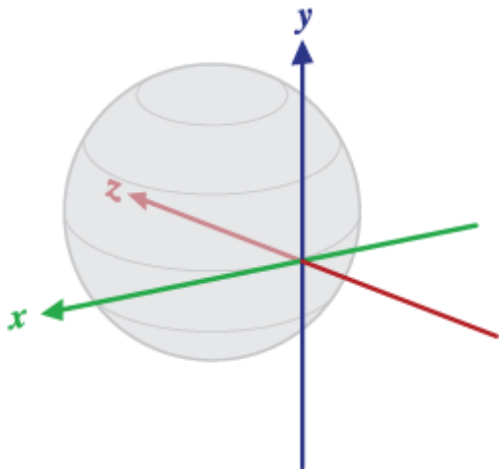
Že ob prvih poskusih je postalo očitno, da bo potrebno glajenje senzoričnih podatkov. Zaradi obremenjenosti okolja z elektromagnetnim sevanjem (zlasti v bližini elektronskih naprav, kot je računalnik) je posebej izhod senzorja magnetnega polja izredno podvržen šumu. Tako smo že v zgodnjih fazah razvoja implementirali enostaven nizkoprepustni filter surovih senzorskih podatkov. Gre za preprost filter z neskončnim impulznim odzivom, ki upošteva trenutno in novo vrednost s faktorjem dušenja 0,15 (za vsako izračunano vrednost upoštevamo le 15 % razlike med trenutno in novo vrednostjo). To je relativno zadovoljivo delovalo za izhod pospeškomera, medtem ko je senzor magnetnega polja še vedno povzročal nihanje izračunanega kota za 5-10 stopinj v mirovanju. Čeprav se je kazala izrazita potreba po dodatnem filtriranju le v izračunu azimuta, smo ocenili, da bi bil neobhoden padec odzivnosti bolj moteč, če bi bil omejen na eno samo os. Zato smo se odločili za filtriranje vseh izhodnih vrednosti. Vsak od treh Eulerjevih kotov je tako izračunan iz drsečega povprečja zadnjih n vrednosti. Pri izbiri števila n smo iskali kompromis med odzivnostjo in zglajenostjo, pri čemer ocenjujemo, da je pri tipični aplikaciji, ki bi utegnila uporabljati našo napravo, uporabniku bolj pomembna slednja lastnost. Skozi preizkušanje različnih vrednosti smo, na podlagi subjektivnega ocenjevanja, na koncu iz-

brali število 40. Za izračun drsečega povprečja je torej uporabljeno 40 zadnjih vrednosti, kar ustreza časovnemu oknu približno sekunde in pol. Glede na lastnosti kotov (350=-10), ni bilo mogoče uporabiti enostavne aritmetične sredine, pač pa smo vsak kot predstavili kot normiran vektor v dvodimenzionalnem prostoru, izračunali aritmetično sredino posameznih komponent in rezultat pretvorili nazaj v kot.

Pri testiranju pa smo naleteli še na en nepredviden problem - za naraven vtis vpliva gibanja naprave na položaj opazovanega objekta je ključna usmerjenost ekrana računalnika na točno določen azimut. V nasprotnem primeru nagib telefona povzroči nagib objekta na ekranu v napačno smer. Gre torej za problem usklajenosti koordinatnega sistema na zaslonu in globalnega koordinatnega sistema. Ker je običajen položaj ekrana s krajšo stranico poravnano z vektorjem sile težnosti, je problem omejen na rotacijo okoli Z osi. Kot rešitev smo uvedli kalibracijo orientiranosti zaslona / telefona ob zagonu aplikacije na telefonu. Takrat predpostavimo, da je telefon usmerjen naravnost proti ekranu. Odčitamo zaznan azimut in ga kot korekcijsko konstanto upoštevamo pri vseh nadaljnjih vrednostih. Za enak učinek med delovanjem aplikacije smo isto funkcijo povezali z dodatnim gumbom, ki se prikaže ob dotiku ekrana. Tako lahko med uporabo rekalibriramo napravo, če se spremeni magnetno okolje (npr. se odmaknemo od računalnika).

4.2.2 Komunikacija

Za komunikacijo z računalnikom skrbi ločena nit, poteka pa preko protokola RFCOMM. Telefon deluje v tem kontekstu kot strežnik. S spodnjim delčkom kode hkrati opravimo dve pomembni nalogi: odpremo strežniško vtičnico (*server socket*), kar pripravi vse potrebno za sprejem vhodne povezave, hkrati pa v SDP tabelo vpišemo podatek o novi storitvi. Slednje (na podlagi UUID) omogoča računalniku zaznavo storitve, ki jo ponuja naprava.



Slika 4.3: Koordinatni sistem metode SensorManager.getOrientation

```
BluetoothAdapter bluetoothAdapter;
BluetoothServerSocket serverSocket;

UUID MY_UUID = UUID.fromString("47017456-04f4-4d26-a847-19aad1550592");
serverSocket = bluetoothAdapter.listenUsingRfcommWithServiceRecord("HID", MY_UUID);
```

Sledi čakanje na vhodno povezavo:

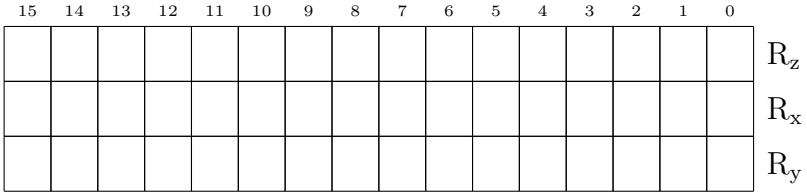
```
BluetoothSocket socket;

socket = serverSocket.accept();
```

Gre za tako imenovan *blocking* klic - nit stoji dokler ne pride do vzpostavitve povezave ali napake.

Po vzpostavitvi povezave operiramo s standardnimi vhodno/izhodnimi tokovi (*streams*).

Eulerjeve kote posredujemo naprej z resolucijo ene stotinke stopinje. Predstavljen je kot celoštevilski podatek, med 0 in 35999, za kar potrebujemo 16 bitov. Vsak prenos torej vsebuje 6 zlogov:



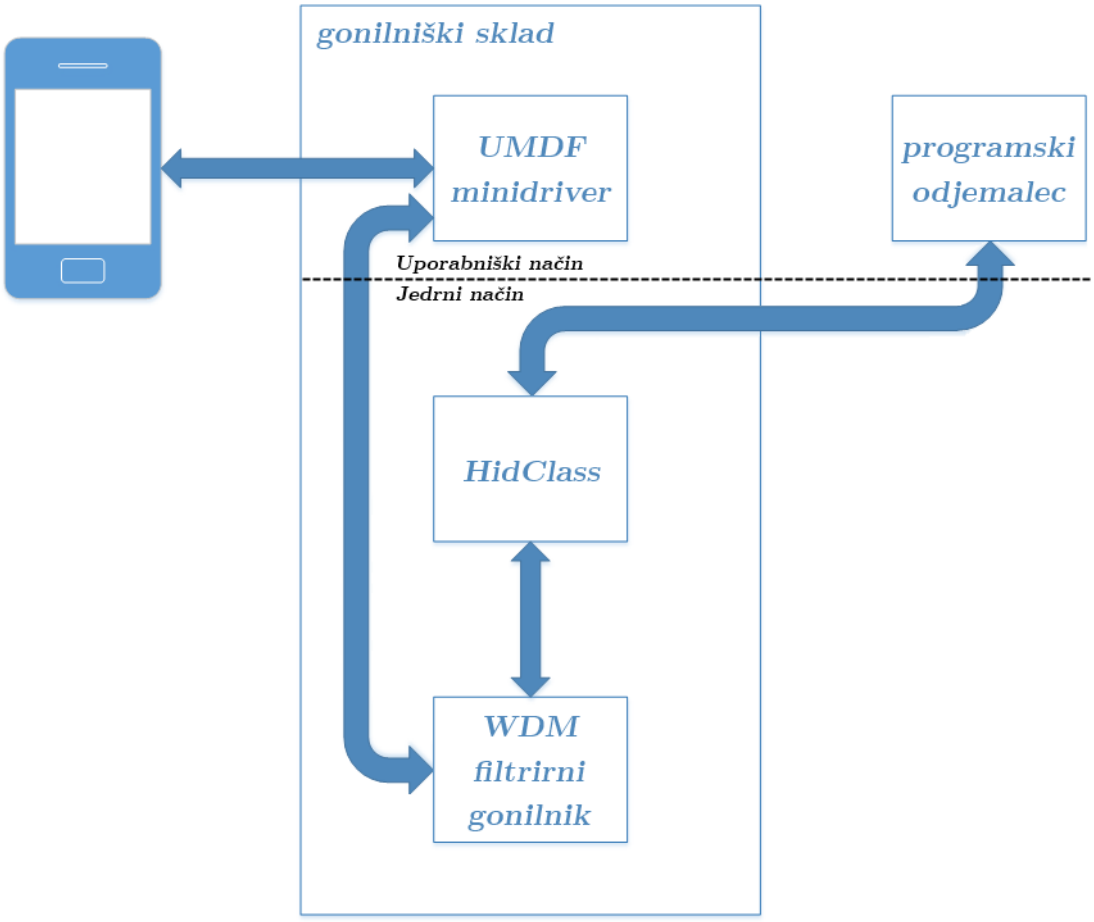
4.3 Gonilnik HID

Kot rečeno, implementacija standardne naprave HID v okolju Android se je izkazala za problematično, zato smo zadevo realizirali v obliki ne-HID naprave in virtualnega gonilnika HID. Iz razlogov, opisanih v razdelku 2.6.2, smo se odločili za implementacijo v UMDF. Gonilniki so napisani v jeziku C++, za osnovo pa smo vzeli vzorčni primer iz WDK, z nazivom *User-mode HID mini-driver sample*[19].

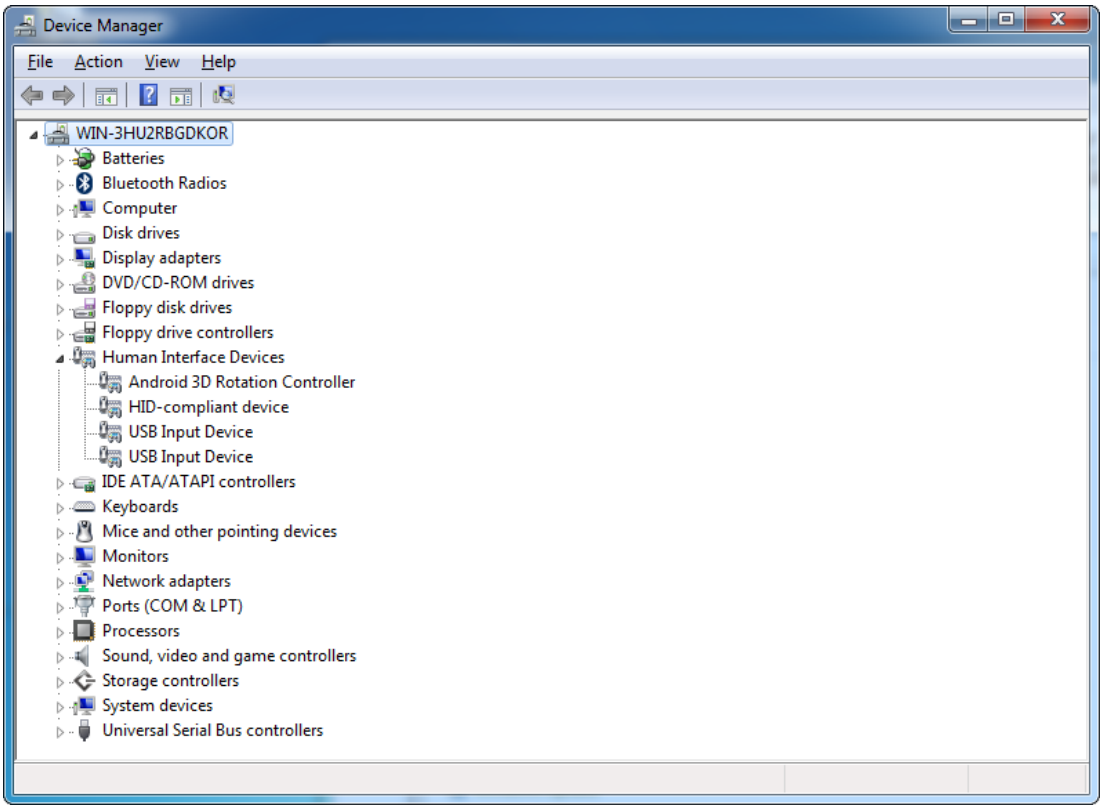
Shema gonilniškega sklada je prikazana na sliki 4.4. Ogrodje UMDF in sistemski razredni gonilnik HidClass (ki je gonilnik tipa port) imata nekompatibilne rutine za V/I razpošiljanje (*I/O dispatch routines*), zato je potrebna vmesna komponenta, ki se registrira kot minidriver, nato pa posreduje V/I podatke gonilniku UMDF. Ta vmesna komponenta je enostaven filtrirni gonilnik WDM. V neki standardni konfiguraciji bi bil filtrirni gonilnik vmesnik med razrednim gonilnikom in transportom. Na opisan način je glavnina logike realizirana v uporabniškem načinu, vključno s komunikacijo z napravo preko protokola Bluetooth. Programski odjemalec je uporabniška aplikacija, ki je porabnik podatkov, ki jih posreduje gonilnik, v našem primeru aplikacija SketchUp oziroma v ta namen razvita razširitev.

Za instalacijo gonilnikov smo uporabili orodje `devcon`, ki je del WDK. Instalacija poteka preko datoteke INF, ki vsebuje navodila glede kopiranja datotek, vnosov v register, registracije storitev (*service*) itd. Na sliki 4.5 vidimo zaslonsko sliko, ki prikazuje instaliran gonilnik v upravljalcu naprav.

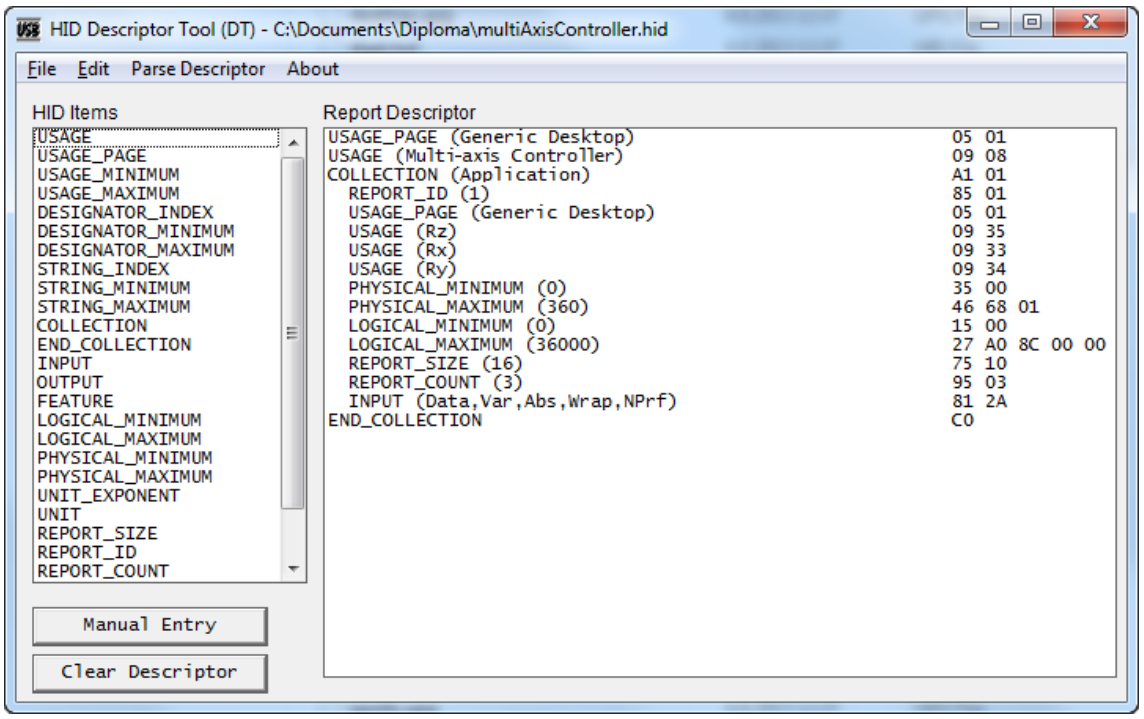
Ključni metodi gonilnika sta `GetReportDescriptor` in `GetInputReport`. Prva na zahtevo poroča o zmogljivostih naprave v obliki deskriptorja poročila



Slika 4.4: Struktura gonilniškega sklada



Slika 4.5: Zaslonska slika upravljalca naprav s prikazanim gonilnikom HID



Slika 4.6: Deskriptor poročila

HID. Na sliki 4.6 vidimo vsebino deskriptorja za našo aplikacijo. Rz, Rx in Ry predstavljajo kot zasuka okoli ustrezne osi in tvorijo sistem tako imenovanih Eulerjevih kotov. Za predstavitev lege bi bila sicer bolj primerna kar matrika vrtenja ali kvaternioni, vendar teh ne najdemo v tabelah uporab HID.

Na drugi strani je `GetInputReport` metoda, ki opravlja glavno nalogo našega gonilnika, torej poročanje zaznanih vrednosti. Seveda je oblika pogojena z deklaracijami v deskriptorju poročila:

```
typedef struct _HIDMINI_INPUT_REPORT {

    UCHAR ReportId;

    UINT16 DataZ;
    UINT16 DataX;
    UINT16 DataY;

} HIDMINI_INPUT_REPORT, *PHIDMINI_INPUT_REPORT;
```

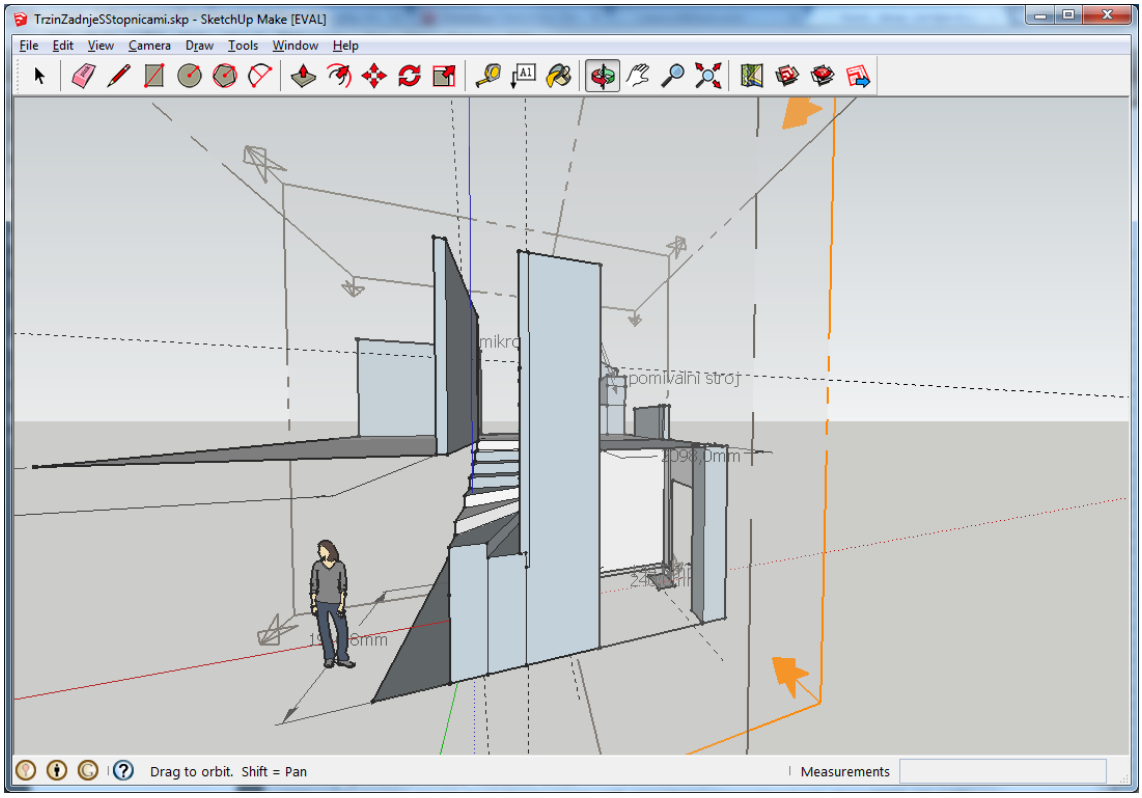
4.4 Integracija v program Trimble Sketchup

Trimble Sketchup[16] je program za 3D modeliranje. Uporaben je v širokem naboru aplikativnih domen, kot naprimer arhitekturno, strojno in industrijsko oblikovanje. Zaslonski posnetek je prikazan na sliki 4.7

SketchUp omogoča programske razširitve preko skript v objektno orientiranem skriptnem jeziku Ruby. V ta namen je v SketchUp integriran tolmač (v različici 13 je to Ruby 1.8.6 - 287), pri čemer je na mestu opozorilo, da vsebuje zgolj podmnožico standardnih Ruby knjižnic. Za interakcijo z modelom in elementi aplikacije SketchUp je na voljo SketchUp Ruby API[17]. Pri razvoju in razhroščevanju nam je v veliko pomoč konzola Ruby, ki jo lahko odpremo iz menija in omogoča interaktivno delo s tolmačem.

Ker Ruby ne omogoča direktnega klicanja sistemskih funkcij, ki so potrebne za komunikacijo z gonilnikom, je bilo potrebno vključiti še vmesni sloj. K sreči Ruby omogoča vključevanje razširitev, napisanih v programskem jeziku C[18].

Za pisanje razširitev v Cju moramo instalirati polno različico okolja Ruby in DevKit, ki vsebuje prevajalnik mingw32. Rezultat prevajanja je datoteka .so, ki predstavlja deljeni objekt (*shared object*), ki se ga dinamično vključi z ustrezno direktivo v Rubyu:



Slika 4.7: Zaslonska slika aplikacije Trimble Sketchup

```
require "hidaccess/hidaccess"
```

Skripto in datoteko .so vključimo v SketchUp tako, da ju enostavno prenesemo v podmapo Plugins naše instalacije SketchUpa. V našem primeru se modul imenuje hidaccess, nahaja pa se v mapi hidaccess.

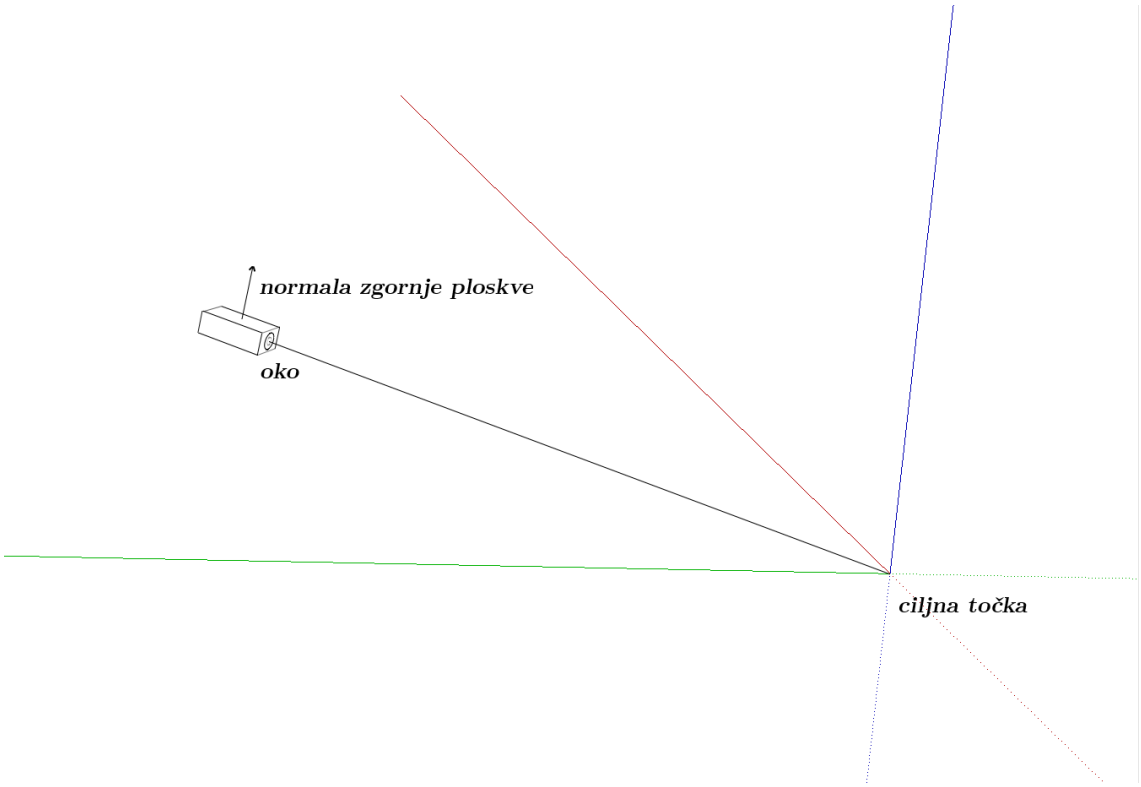
Ob nalaganju razširitve se najprej zažene globalna funkcija z nazivom Init_[ime razširitve]. V njej s funkcijama `rb_define_module` in `rb_define_singleton_method`, ki sta del Ruby C API, deklariramo modul in funkcije, ki naj bodo na voljo okolju Ruby:

```
void Init_hidaccess()
{
    VALUE mHIDAccess = rb_define_module("Hidaccess");
    rb_define_singleton_method(mHIDAccess, "hid_connect", hid_connect, 0);
    rb_define_singleton_method(mHIDAccess, "hid_report", hid_report, 0);
    rb_define_singleton_method(mHIDAccess, "get_x", get_x, 0);
    rb_define_singleton_method(mHIDAccess, "get_y", get_y, 0);
    rb_define_singleton_method(mHIDAccess, "get_z", get_z, 0);
}
```

Naša razširitev ponuja modul s petimi metodami:

- `hid_connect` se poveže z gonilnikom;
- `hid_report` zahteva trenutne vrednosti od gonilnika;
- `get_x` vrača rotacijo okoli osi x;
- `get_y` vrača rotacijo okoli osi y;
- `get_z` vrača rotacijo okoli osi z.

Pogled na 3D model je v programu SketchUp določen s kamero. To sestavljajo tri komponente: točka očesa, ciljna točka in normalni vektor zgornje ploskve, ki določa nagib. Grafično predstavitev modela kamere vidimo na sliki 4.8.



Slika 4.8: Model kamere v programu Trimble Sketchup

Za navidezno vrtenje modela moramo torej vrteti kamero okoli njega. Zaradi poenostavitve implementacije smo sprejeli predpostavko, da je opazovani objekt postavljen v bližino koordinatnega izhodišča, posledično tudi kamero vrtimo okoli koordinatnega izhodišča.

Če so R_z R_y in R_x rotacije okoli pripadajočih osi, so transformacije, ki določajo položaj kamere, določene s sledečimi matrikami:

$$M_z = \begin{bmatrix} cos(R_z) & sin(R_z) & 0 \\ -sin(R_z) & cos(R_z) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$M_y = \begin{bmatrix} cos(-R_y) & 0 & -sin(-R_y) \\ 0 & 1 & 0 \\ sin(-R_y) & 0 & cos(-R_y) \end{bmatrix}$$

$$M_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(R_x) & sin(R_x) \\ 0 & -sin(R_x) & cos(R_x) \end{bmatrix}$$

Za razliko od Androidovega, se v SketchUpovem koordinatnem sistemu vrednosti na y osi povečujejo od leve proti desni, tako da je pri rotaciji okoli te osi potrebno spremeniti predznak.

Za popoln opis položaja kamere je potrebno še definirati normalni vektor zgornje ploskve, ki pa je odvisen od rotacij okoli osi y in x:

$$N_y = \begin{bmatrix} cos(-R_(y)) & 0 & sin(R_y) \\ 0 & 1 & 0 \\ -sin(R_y) & 0 & cos(-R_y) \end{bmatrix}$$

$$N_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(R_x) & -sin(R_x) \\ 0 & sin(R_x) & cos(R_x) \end{bmatrix}$$

Da bi omogočili delovanje tako imenovanega *zooma* v SketchUpu, se pred vsakim osveževanjem pogleda izračuna oddaljenost od koordinatnega izhodišča

in končni položaj kamere ustrezno normira, da se po transformaciji ohrani enaka oddaljenost.

Poglavje 5

Sklepne ugotovitve

V pričujočem delu je predstavljen razvoj programske rešitve za uporabo pametnega telefona, kot vnosne naprave za krmiljenje objektov v trodimenzionalnem prostoru. Razvili smo aplikacijo, ki teče na telefonu Android in posreduje zaznano lego telefona preko Bluetooth povezave računalniku. Na njem smo za operacijski sistem Microsoft Windows razvili gonilnik HID, ki na standardiziran način ponuja podatke programskim odjemalcem (aplikacijam). Za prikaz uporabe smo na koncu preko Ruby skripte in razširitev v jeziku C povezali gonilnik s paketom za 3D oblikovanje Trimble SketchUp.

Pri razvoju smo se, predvsem v gonilniškem delu, srečevali s pomanjkanjem neformalnih spletnih resursov in uporabniške skupnosti, s čimer si sodobni programerji izdatno pomagamo pri delu z bolj prozaičnimi tehnologijami.

Poleg implementacije pričakovanih zahtev smo rešili tudi nekaj nepredvidenih težav: šumnost senzoričnih podatkov ter neusklajenost usmerjenosti naprave in ekrana.

Nasprotno pa se na nekaterih področjih, kjer smo pričakovali težave, te niso pojavile. Tako recimo nismo imeli nobenih težav s kardansko zaporo, kar pripisujemo dovolj veliki resoluciji in glajenju.

Možnosti za nadaljnje delo vidimo v dveh smereh: prilagoditev za množično uporabo in izboljšave v smislu praktične uporabnosti.

Če bi želeli pripraviti rešitev, primerno za množično uporabo, ostaja nekaj izzivov v smeri poenostavitve instalacije na različnih konfiguracijah. Kot opisano, trenutno postopek instalacije temelji na programu `devcon`, ki je del WDK. Ker nam licenca ne dovoljuje nadaljnje distribucije, bi bilo potrebno funkcionalnosti tega orodja realizirati v lastnem instalacijskem programu. Glede na to, da je na voljo izvorna koda orodja `devcon`, ki ne nosi teh licenčnih omejitev, to ne predstavlja pretežkega problema. Prav tako bi bil potreben prevod gonilnikov za preostale operacijske sisteme in ustrezne razširitve v INF datoteki. Trenutno je podprt 32-bitni operacijski sistem Windows 7.

Glede uporabnosti je naše mnenje, da rešitev lepo demonstrira delovanje in razkriva potencial, ima pa nekaj pomanjkljivosti za resnejšo uporabo. Tako je trenutno rešitev primerna za prezentacije ali ogledovanje 3D modela, manj pa za oblikovalsko delo. Če telefon namreč odložimo na ravno površino, se bo opazovani objekt vrnil v izhodiščni položaj, vsaj kar se tiče osi X in Y. Tozadevno bi bil v pomoč gumb za vklop/izklop rotacije, tako da bi lahko objekt zamrznili v določenem položaju.

Literatura

[1] "Wikipedia, Mobile operating system" Dostopno na:
http://en.wikipedia.org/wiki/Mobile_operating_system

[2] "Android Fragmentation Visualized (July 2013)" Dostopno na:
http://opensignal.com/reports/fragmentation-2013/

[3] "Universal Serial Bus Revision 2.0 specification" Dostopno na:
http://www.usb.org/developers/docs/usb_20_070113.zip

[4] "USB Device Class Definition for Human Interface Devices (USB HID Specification), Version 1.11" Dostopno na:
http://www.usb.org/developers/devclass_docs/HID1_11.pdf

[5] "USB HID Usage Tables, Version 1.12" Dostopno na:
http://www.usb.org/developers/devclass_docs/Hut1_12v2.pdf

[6] "MSDN Library, Windows Driver Development" Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/hardware/ff557573(v=vs.85).aspx

[7] "Introduction to the WDF User Mode Driver Framework", Microsoft, 2006 Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/hardware/gg463303.aspx

[8] "ADT Bundle" Dostopno na:
http://developer.android.com/sdk/installing/bundle.html

[9] "Eclipse" Dostopno na:
http://eclipse.org/

[10] "Android Developer Tools" Dostopno na:
http://developer.android.com/tools/help/adt.html

[11] "Android SDK" Dostopno na:
http://developer.android.com/sdk/index.html

[12] "Windows Driver Kit (WDK)" Dostopno na:
http://msdn.microsoft.com/en-us/library/windows/hardware/gg487428

[13] "HID Descriptor Tool" Dostopno na:
http://www.usb.org/developers/hidpage/dt2_4.zip

[14] "VirtualKD" Dostopno na:
http://virtualkd.sysprogs.org/

[15] "VmWare Player" Dostopno na:
http://www.vmware.com/products/player/

[16] "Sketchup" Dostopno na:
http://www.sketchup.com/

[17] "Sketchup Ruby API" Dostopno na:
http://www.sketchup.com/intl/en/developer/docs/index.php

[18] "Programming Ruby - The Pragmatic Programmerś Guide, Extending Ruby" Dostopno na:
http://www.ruby-doc.org/docs/ProgrammingRuby/html/ext_ruby.html

[19] "User-mode HID mini-driver sample" Dostopno na:
http://code.msdn.microsoft.com/windowshardware/WudfVhidmini-Sample-b304f83a

Slike