UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Zaključna naloga

# Reprodukcija dejanskega okolja v virtualno resničnost s pomočjo para kamer ter Google Cardboard prikazovalnika

(Reproduction of actual environment in virtual reality with a camera pair and Google Cardboard viewer)

Ime in priimek: Tomaž Sabadin
Študijski program: Računalništvo in informatika
Mentor: doc. dr. Peter Rogelj

**Koper, september 2015**

# Ključna dokumentacijska informacija

Ime in PRIIMEK: Tomaž SABADIN

Naslov zaključne naloge: Reprodukcija dejanskega okolja v virtualno resničnost s pomočjo para kamer ter Google Cardboard prikazovalnika

Kraj: Koper

Leto: 2015

Število listov: 46 Število slik: 12 Število tabel: 3

Število prilog: 1 Število strani prilog: 1 Število referenc: 8

Mentor: doc. dr. Peter Rogelj

Ključne besede: programsko inženirstvo, Java, Android, OpenCV, virtualna resničnost

**Izvleček:**
Zaključna naloga predstavlja potek programskega procesa izgradnje računalniškega programa ter mobilne aplikacije za reprodukcijo dejanskega okolja v virtualni resničnosti. Namen zaključne naloge je predstaviti ter prikazati ključne faze, potrebne za izdelavo takega sistema. Zaključna naloga pokriva večino faz programskega procesa. Podrobneje smo obravnavali sledeče faze: študija izvedljivosti, analiza in definiranje zahtev, načrtovanje sistema, načrtovanje komponent, izvedba ter testiranje. Izpuščeni sta le fazi predaja ter obratovanje in vzdrževanje, saj naročnik ni izražen, ker gre za produkt zaključne naloge. Razvit sistem je namenjen reprodukciji dejanskega okolja v virtualno resničnost s pomočjo para kamer ter Google Cardboard prikazovalnika. Sistem je implementiran s pomočjo programskega jezika Java, z uporabo OpenCV knjižnice ter deluje na arhitekturnem modelu odjemalec - strežnik. Strežnik zajema, obdeluje in pošilja slike realnega okolja odjemalni napravi ter v našem primeru lahko deluje na večini operacijskih sistemov, ki podpirajo Javo. Odjemalec pa je prenosna naprava z operacijskim sistemom Android, ki poskrbi za prejem ter prikaz realnega okolja na osnovi od strežnika prejetih slik.

# Key words documentation

Name and SURNAME: Tomaž SABADIN

Title of final project paper: Reproduction of actual environment in virtual reality with a camera pair and Google Cardboard viewer

Place: Koper

Year: 2015

Number of pages: 46 Number of figures: 12 Number of tables: 3

Number of appendices: 1 Number of appendix pages: 1 Number of references: 8

Mentor: Assist. Prof. Peter Rogelj, PhD

Keywords: software engineering, Java, Android, OpenCV, virtual reality

**Abstract:**
The thesis presents the course of the software development process of building a computer program and a mobile application for the reproduction of the actual environment in virtual reality. The purpose of the thesis is to present and demonstrate the main stages of the software development process. These stages are: problem definition, feasibility study, analysis and definition of requirements, system design, implementation and testing. Stages, deployment, operation and maintenance are excluded, because this product is developed for thesis purposes and the customer is not known. Developed system is designed to reproduce the actual environment in virtual reality with a camera pair and the Google Cardboard viewer. The system is implemented with the Java programming language and with the help of OpenCV library. The system is based on the client - server model. The server captures, processes and transmits images of the actual environment to the client and can run on most operating systems that support Java. The client is a portable device with the Android operating system, which receives and displays the real environment based on the images received from the server.

# Zahvala

Rad bi se zahvalil mentorju za koristne napotke ter usmerjanje pri izboru in izdelavi zaključne naloge. Prav tako bi se rad zahvalil svojim najbližjim, ki so me pri študiju podpirali ter spodbujali.

# Kazalo vsebine

# Kazalo tabel

# Kazalo slik

# Kazalo prilog

# Slovarček

| | |
|---|---|
| *tj.* | to je |
| *itd.* | in tako dalje |
| *ipd.* | in podobno |
| *SDK* | paket za razvoj programske opreme (angl. software development kit) |
| *NFC* | komunikacija kratkega dosega (angl. near field communication) |
| *Java* | programski jezik |
| *Android* | operacijski sistem za pametne telefone in naprave |
| *IP* | internetni protokol (angl. internet protocol) |
| *IDE* | integrirano razvojno okolje (angl. integrated development environment) |
| *TCP* | povezavni protokol transportnega sloja (angl. transmission control protocol) |
| *Mbps* | mega bitov na sekundo (angl. megabits per second) |

# 1 Uvod

Virtualna resničnost je definirana kot simulacija nekega okolja, ki ga zgradimo s pomočjo programske opreme. To okolje nato predstavimo s pomočjo različnih prikazovalnikov ali pripomočkov uporabniku na tak način, da ga s svojimi čuti ter prepričanjem doživi kakor resničnost [1]. Poznamo veliko različnih pripomočkov za prikaz virtualne resničnosti, nekateri izmed teh so Oculus Rift, Microsoft HoloLens, Samsung Gear VR in mnogi drugi. Podjetje Google je razvilo svoj prikazovalnik, imenovan Google Cardboard (v nadaljevanju: Cardboard), ki je prikazan na sliki 1. Že z izborom imena Cardboard (slovensko: karton ali lepenka) je Google nakazal, da gre za prikazovalnik, ki je izdelan povsem iz kartona. S tem so dosegli relativno nizko ceno, saj želijo s takim pristopom povečati zanimanje za virtualno resničnost. Razvili so tudi pripadajoč SDK za delo s Cardboardom in tako še bolj poenostavili izdelavo aplikacij [2]. Cardboardovi sestavni deli so poleg kartonastega ogrodja še par leč, ki poskrbi za pravilen prikaz slik, magneti, ki jih lahko uporabimo za interakcijo v aplikacijah ter različne dodatne komponente, kot so npr. NFC značka, ježki ipd. Za prikaz slik nekega okolja potrebujemo Android pametni telefon, ki ga vstavimo pred leči in tako lahko začnemo svoje doživetje virtualne resničnosti.


Slika 1: Google Cardboard prikazovalnik. Vir: [8]

Ideja zaključne naloge je razviti prototip sistema za reprodukcijo dejanskega okolja v virtualno resničnost s pomočjo para spletnih kamer. Za doživetje virtualne resničnosti bomo uporabili že prej predstavljeni Cardboard. Naloga bo poleg samega razvoja sistema vsebovala tudi opis faz in postopkov programskega inženirstva na raz-

vitem sistemu. Predstavili bomo večino faz programskega inženirstva, in sicer definicijo problema, študijo izvedljivosti, analizo in definiranje zahtev, načrtovanje, izvedbo ter testiranje. Izpuščeni bosta le fazi predaja ter obratovanje in vzdrževanje sistema, saj gre za produkt zaključne naloge in tako nimamo končnega kupca.

Cilj zaključne naloge je, da preko razvoja sistema poglobimo svoje znanje programskega inženirstva. Ker pa zaključna naloga obsega tudi izdelavo prototipa, bomo s tem poglobili tudi svoje znanje programskega jezika Java ter se spoznali z operacijskim sistemom Android. Posledično bomo poglobili svoje znanje o računalniških omrežjih, saj bosta izdelani komponenti komunicirali prek omrežja ter se spoznali z OpenCV knjižnico, ki je namenjena računalniškemu vidu.

Zaključna naloga vsebuje osem poglavij, vključno z uvodom. V vsakem od naslednjih poglavij je predstavljena posamezna faza programskega procesa. V zaključku pa so navedene ideje za nadgradnjo sistema ter kratek povzetek zaključne naloge.

# 2 Definicija problema

Okolje virtualne resničnosti v katero vstopa uporabnik je v večini primerov zgrajeno s pomočjo programske opreme in predstavlja neko umetno zgrajeno oziroma izdelano okolje. S tem uporabnik doživi virtualno resničnost kot premik v neko nerealno okolje. Naša želja je ponuditi uporabniku virtualno resničnost, ki pa ne bi bila umetno zgrajena, vendar bi s pomočjo para kamer umestili realen svet v virtualno resničnost. S takim pristopom lahko uporabnikom prikažemo realen svet na oddaljenih lokacijah ter jim s tem omogočimo nepozabno doživetje teh lokacij kar iz domačega naslonjača.

Sistem še nima določenega končnega uporabnika, vendar menimo da bi lahko koristil širokemu krogu uporabnikov. Vzemimo za primer nekoga, ki si želi obiskati oddaljene kraje, vendar si teh krajev ne more ogledati, zaradi finančnih ali drugih ovir. Z našim sistemom bi lahko te kraje doživel, a ne le to, lahko bi se v njih kadarkoli vrnil.

Drugo prednost našega sistema vidimo v opazovanju dogodkov oziroma aktivnosti v oddaljenem okolju. Z našim sistemom se bomo lahko enostavno "udeležili" različnih koncertov, tekmovanj, prireditev ipd. Sistem bo pričaral občutek, da smo resnično med obiskovalci ter smo del samega dogodka. Prav tako ne bo meja kje je prizorišče prireditve. Lahko se bomo udeležili domačih kot tudi tujih dogodkov. S tem bomo tudi organizatorjem dogodkov omogočili večji "obisk" marsikaterega dogodka.

Naš namen pa je tudi približati ter predstaviti virtualno resničnost uporabnikom, saj velika večina ljudi ne pozna tega področja ter se bo z njim prvič srečala ravno z našim sistemom.

# 3 Študija izvedljivosti

S študijo izvedljivosti želimo zagotoviti, da je projekt tehnično izvedljiv, ekonomsko upravičen ter operativno izvedljiv, torej moramo preveriti, da ne kršimo zakonov, da ni aplikacija moralno sporna ali v nasprotju z določeno kulturo ljudi. Ne smemo pa pozabiti tudi na časovni rok za izvedbo celotnega projekta.

## 3.1 Tehnična izvedljivost

Predlagana rešitev je izdelava sistema, ki bo vseboval dve komponenti. Sistem se tako deli na strežniški del, ki bo vseboval zajem, obdelavo ter pošiljanje slik odjemalcem ter odjemalni del, ki bo poskrbel za prejem slik ter pravilen prikaz le-teh.

Strežniški del sistema bo implementiran s pomočjo programskega jezika Java. S takim izborom programskega jezika smo rešili problem uporabe različnih operacijskih sistemov za strežniški del, saj bo lahko program tekel na različnih operacijskih sistemih brez dodatnih prilagajanj. Prednost izbora programskega jezika Java je tudi v sami tehnični podkovanosti razvijalca ter številnih brezplačnih in odprtokodnih knjižnic. Za zajem ter obdelavo slik bomo potrebovali OpenCV knjižnico, ki pa je dostopna tudi za programski jezik Java. S tem smo torej zadostili tudi tej potrebi. Za prenos slik prek omrežja pa bodo poskrbeli vtiči (angl. sockets), ki jih prav tako podpira izbrani programski jezik.

Odjemalni del sistema bo prav tako izdelan s pomočjo programskega jezika Java, saj bomo aplikacijo uporabljali na prenosnih napravah z nameščenim operacijskim sistemom Android. Za prejem slik bomo uporabili že prej omenjene vtiče, prikaz slik pa bo omogočala prav temu namenjena komponenta, imenovana *ImageView*.

Glede same tehnične izvedljivosti je pričakovati največ zapletov pri obdelavi slik na strežniškem delu sistema. Temu delu bo namenjeno tudi največ časa pri samem načrtovanju ter izvedbi. Težave se bodo lahko pojavile tudi pri sami hitrosti pošiljanja slik iz strežniškega dela sistema odjemalcu, saj želimo doseči prikaz slik v realnem času.

## 3.2 Ekonomska upravičenost

Za realizacijo predlagane rešitve ne pričakujemo večjih stroškov, ker gre navsezadnje za projekt zaključne naloge. Potrebujemo le Cardboard prikazovalnik ter par enakih spletnih kamer. Za Cardboard prikazovalnik smo že omenili, da ima zelo nizko ceno, tako da je ta strošek zanemarljiv. Pričakujemo le nekoliko večji strošek pri nakupu spletnih kamer, vendar je lahko tudi ta strošek izvzet, če si kameri le izposodimo. Stroški nakupa Android naprave ter osebnega računalnika so prav tako izvzeti, saj pričakujemo, da razvijalec že ima obe od naštetih komponent.

## 3.3 Operativna izvedljivost

S predlagano rešitvijo ne kršimo nobenega od zakonov. Sistem ni zlonameren ter ni moralno sporen.

## 3.4 Časovna izvedljivost

Rok za oddajo zaključne naloge je konec avgusta 2015, torej smo časovno omejeni na približno štiri tedne. Trije tedni bodo namenjeni izdelavi sistema, en teden pa pisanju dokumentacije. Kot že prej omenjeno pričakujemo največ težav pri obdelavi slik, zato bo ta del časovno najbolj potraten.

## 3.5 Sklep

Študija izvedljivosti je zaključena ter je naša predlagana rešitev označena kot izvedljiva. S študijo izvedljivosti pa nismo le potrdili izvedbe sistema, vendar smo spoznali tudi možne zaplete pri sami izvedbi, kakšni bodo stroški celotnega projekta ter kakšni so časovni roki. Z izgradnjo sistema bomo dosegli cilj, ta pa je uspešno dokončati zaključno nalogo ter s tem dodiplomski študij.

# 4 Analiza in definiranje zahtev

Študiji izvedljivosti sledi faza analize in definiranja zahtev. S to fazo definiramo zahteve programskega produkta. To fazo izvaja sistemski analitik, v našem primeru je to avtor zaključne naloge kot edini razvijalec sistema. Sistemski analitik mora poskrbeti, da je specifikacija zahtev, tj. končni dokument te faze, konsistenten ter da zahteve niso v nasprotju z nobeno od drugih zahtev. S specifikacijo zahtev se torej seznanimo s potrebami uporabnika, razumemo problem, ki ga rešujemo ter uskladimo rešitev s končnim uporabnikom oziroma naročnikom. Kot smo že omenili pa gre v našem primeru za zaključno nalogo, zato naročnik kot tak ne obstaja. Pričakujemo pa, da bo našo aplikacijo uporabljalo veliko število ljudi, predvsem uporabnikov pametnih telefonov, tako lahko avtor zaključne naloge kar sam prevzame vlogo končnega naročnika. Nekaj zahtev je bilo podanih tudi s strani mentorja, tako da lahko tudi njega štejemo v skupino končnih uporabnikov.

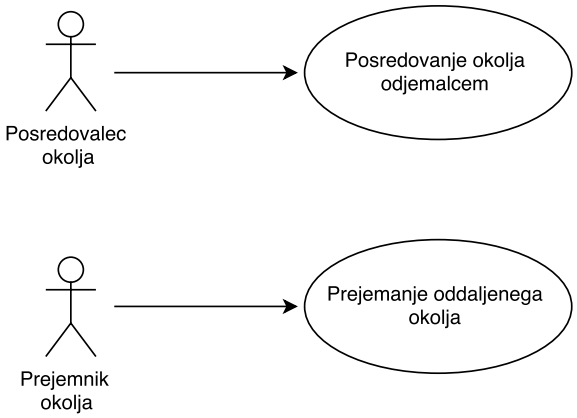
## 4.1 Namen sistema

Sistem bo služil za replikacijo dejanskega okolja v virtualno resničnost. Prednost takega sistema vidimo v uporabi pri 3D spletnih konferencah, pri nadzornih sistemih, pri uporabi v turistične namene itd. Vzemimo za primer uporabo za turistične namene. Uporabnik bo s takim sistemom lahko "odpotoval" v oddaljene kraje z minimalnimi stroški. V kolikor bo strežniški del že nameščen ter pripravljen za uporabo, bo uporabnik potreboval le Google Cardboard ter Android napravo. In že bo lahko užival na peščenih plažah ali na vrhovih najvišjih gora. Namen sistema je tudi približati uporabnikom virtualno resničnost ter jih seznaniti s tem področjem.

## 4.2 Primeri uporabe

Iz diagrama primerov uporabe (slika 2) je razvidno, da ima naš sistem dve glavni funkciji. Ti dve funkciji sta:

- Posredovanje okolja odjemalcem
- Prejem oddaljenega okolja




Slika 2: Diagram primera uporabe sistema

S slike primerov uporabe (slika 2) je razvidno, da imajo lahko uporabniki v našem sistemu dve različni vlogi. Glede na vloge pa imamo posledično tri vrste uporabnikov. Imamo uporabnike, ki bodo nudili le vpogled v svoje okolje (posredovalce okolja). Tukaj lahko vzamemo za primer ponudnika hotelskih storitev, ki želi gostom prikazati okolico hotela. Njegova naloga bo poskrbeti za posredovanje okolja odjemalcem, torej bo skrbel le za strežniški del sistema. Druga vrsta uporabnikov so samo odjemalci oddaljenega okolja (prejemniki okolja). Torej tukaj lahko vzamemo za primer nekoga, ki si želi ogledati okolico hotela prvega uporabnika. Tako se bo ta uporabnik povezal na strežniški del sistema prvega uporabnika ter si bo lahko ogledal okolje, ki ga bo posredoval prvi uporabnik. Tak uporabnik bo potreboval le Android napravo ter Google Cardboard za prejem oddaljenega okolja, torej potreboval bo odjemalni del sistema. Nato pa imamo še tretjo vrsto uporabnikov. To so uporabniki, ki bodo posredovali ter prejemali okolje, torej imajo obe vlogi, vlogo posredovalca ter prejemnika okolja. Gre za uporabnike, ki bodo uporabljali sistem za lastne namene npr. za nadzor lastne okolice. Poskrbeti bodo morali za posredovanje okolja odjemalnim napravam oziroma za strežniški del sistema ter za prejem oddaljenega okolja oziroma za odjemalni del sistema.

## 4.3 Posredovanje okolja odjemalcem

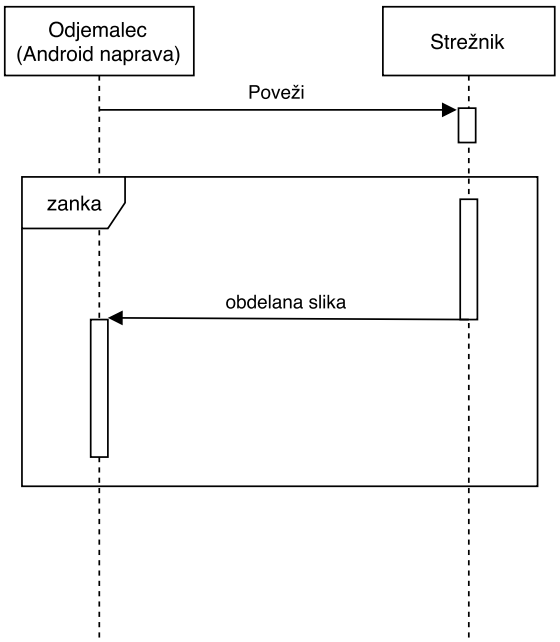
Posredovanje okolja odjemalcem oziroma oddajanje slik iz strežnika na odjemalca je najpomembnejša ter s tem tudi primarna funkcionalnost našega sistema. Ob zagonu strežniškega dela sistema se nam bo prikazala okenska aplikacija. Obenem se bo strežnik inicializiral. Bolj podrobno, inicializirali se bosta kameri, strežnik bo preveril njuno stanje ter izpisal stanje obeh kamer. Prav tako bo v zato namenjen prostor izpisal še svoj IP naslov ter domensko ime. Uporabnik mora poznati ter vpisati enega od obeh v odjemalno napravo, da bi lahko strežnik ter odjemalec uspešno vzpostavila povezavo. Po uspešno vzpostavljeni povezavi (postopek povezave med strežnikom in odjemalcem je opisan v naslednjem poglavju *Prejem oddaljenega okolja*) bo nato strežnik oddajal slike odjemalcu, vse dokler ne prekinemo ali zaustavimo pošiljanja na strežniku oziroma dokler odjemalec ne prekine povezave s strežnikom.

Strežniški sistem ne bo omogočal posredovanja okolja več odjemalcem hkrati. Če se bo želel povezati na strežniški del sistema še en odjemalec, medtem ko bomo že posredovali okolje drugemu odjemalcu, bomo novega odjemalca zavrnili.

## 4.4 Prejem oddaljenega okolja

Prejem oddaljenega okolja je prav tako ena od ključnih funkcionalnosti sistema. Po uspešni inicializaciji strežnika (postopek je opisan v poglavju *Posredovanje okolja odjemalcem*) mora uporabnik zagnati aplikacijo na odjemalcu. Aplikacija na odjemalni napravi se bo ob zagonu inicializirala ter prikazala vnosno polje za vnos IP naslova oziroma domenskega imena strežniškega dela sistema. Uporabnik mora vpisati IP naslov oziroma domensko ime strežnika v za to namenjeno vnosno polje ter vnos potrditi s klikom na gumb *OK*. Po vnosu podatkov s strani uporabnika se bo odjemalec povezal na strežniški del sistema. Tako se bo uspešno vzpostavila povezava med strežnikom in odjemalcem. Kot že navedeno v prejšnjem poglavju bo nato po uspešno vzpostavljeni povezavi strežnik oddajal slike odjemalcu, vse dokler ne prekinemo ali zaustavimo pošiljanja na strežniku oziroma dokler odjemalec ne prekine povezave s strežnikom.

Prikaz povezave odjemalca na strežnik je prikazana na sliki 3, ki prikazuje sekvenčni diagram.




Slika 3: Sekvenčni diagram - pošiljanje podatkov med odjemalcem ter strežnikom

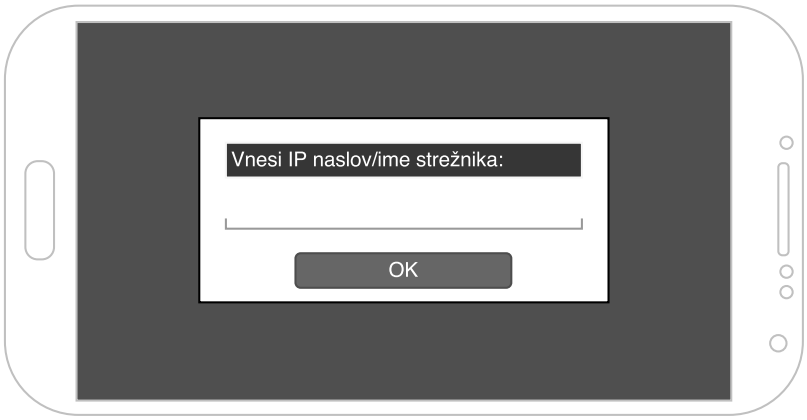
## 4.5 Uporabniški vmesnik

Definirati moramo kako in na kakšen način bo uporabnik uporabljal naš sistem. Na strežniškem delu sistema bomo uporabniku olajšali interakcijo s preprostim grafičnim vmesnikom. Grafični vmesnik bo vseboval štiri komponente. Uporabnik bo moral za povezavo odjemalne naprave s strežnikom poznati IP naslov oziroma domensko ime strežnika. Tako bo prvi element izpis IP naslova ter domenskega imena strežnika. Pod izpisanim IP naslovom in domenskim imenom bo grafični vmesnik imel dva gumba. Prvi gumb bo služil za začetek ter prekinitev oziroma konec posredovanja okolja odjemalni napravi. Drugi gumb pa bo služil za pregled stanja kamer, torej če sta kameri priklopljeni ter pripravljeni za uporabo. Med samim oddajanjem uporabnik ne bo mogel preveriti stanja kamer, tako bo ta gumb med oddajanjem onemogočen. Onemogočen bo prav tako prvi gumb, vse dokler se odjemalec ne bo povezal na strežnik. Priložena je tudi skica grafičnega vmesnika na sliki 4.

Na odjemalni napravi bomo prav tako potrebovali preprost grafični vmesnik. Ob zagonu aplikacije se nam bo pojavilo vnosno polje, v katerega bo uporabnik vnesel IP naslov ali domensko ime strežnika. S klikom na potrditveni gumb bo potrdil vnos podatkov ter se tako uspešno povezal na strežniški del sistema. Priložena je skica grafičnega vmesnika Android naprave na sliki 5. Ne želimo imeti veliko dodatnih funkcij na sami Android napravi, saj bo naprava nameščena pred parom leč Cardboard prikazovalnika. Zato je dostop do ekrana onemogočen in bi morali napravo jemati iz

prikazovalnika ter jo nato nazaj pravilno namestiti.


Slika 4: Skica grafičnega vmesnika




Slika 5: Skica Android aplikacije - vnosno polje

## 4.6 Zahteve strežniškega dela sistema

Na strežniški del sistema bosta priklopljeni kameri, ki bosta služili za zajem slik dejanskega okolja. Kameri predstavljata enega od glavnih komponent za pravilno delovanje sistema, zato moramo uporabniku omogočiti, da preveri stanje kamer. Strežniška aplikacija bo tako omogočala pregled stanja kamer. Uporabnik bo lahko s klikom na gumb *Preveri kameri* preveril stanje kamer, torej ali sta kameri priključeni ter pripravljeni za uporabo. V kolikor bomo zagnali strežniški sistem brez priklopljenih kamer, bo naš sistem javil napako in tako bo potrebno priklopiti kameri ter ponovno inicializirati

strežnik. Ob zagonu z le eno kamero bo sistem prav tako nepravilno deloval ter bo potrebno ponoviti postopek inicializacije strežnika, po tem ko bomo priklopili še drugo kamero.

Strežniški del sistema potrebuje tudi medmrežno povezavo. Zaželeno je, da ima medmrežna povezava dovolj visoko hitrost za nemoteno ter neprekinjeno oddajanje okolja odjemalnim napravam, saj želimo slike prikazovati v realnem času. Povprečna velikost stisnjene slike pri ločljivosti 1280 x 640 pikslov je okoli 30 KB. Če pošljemo 10 slik na sekundo je torej v eni sekundi potrebno prenesti 300 KB. Torej minimalna zahtevana prepustnost omrežja je vsaj 300 KB/s.

Strežniški del sistema bo po zajemu slik moral opraviti tudi obdelavo slik za pravilen prikaz na Android napravi. To bomo naredili že na strežniškem delu sistema iz preprostega razloga, saj ne želimo tega opravljati na odjemalnem delu sistema. Vemo, da so Android pametni telefoni postali vse bolj zmogljivi, vendar obstaja še velika večina naprav, ki nimajo velike zmogljivosti. Tako se bomo izognili preobremenjevanju odjemalnih naprav. Torej bo strežniški del sistema poskrbel za obdelavo slik, odjemalni del sistema pa bo le prejemal ter prikazoval slike.

## 4.7 Ostale zahteve

Uporabnik bo potreboval za pravilno delovanje sistema dve enaki spletni kameri, ki pa morata biti pravilno nameščeni, na pravilni razdalji ter morata biti enako nagnjeni. Poleg tega bo uporabnik moral imeti na svojem računalniku nameščeno Javo ter OpenCV knjižnico. Uporabljena bo verzija 2.4.9. OpenCV knjižnice. Prav tako bo potreboval Android napravo. Android naprava mora imeti nameščen operacijski sistem verzije vsaj 4.0.3 (oznaka verzije Android operacijskega sistema je tudi *Ice Cream Sandwich*). Zaželeno je, da je velikost ekrana primerna za Cardboardov prikazovalnik. V našem primeru bo uporabljena Android naprava z velikostjo diagonale zaslona pet palcev (angl. inch) oziroma 12,7 centimetra.

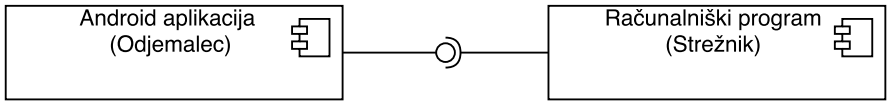
# 5 Načrtovanje sistema ter komponent

Fazi analiza in definiranje zahtev sledi faza načrtovanja sistema. Načrtovanje sistema je postopek, ko specifikacijo zahtev preslikamo v načrtovalski model sistema. Z načrtovanjem sistema želimo zagotoviti stabilno arhitekturo, saj nam le ta zagotavlja zanesljivost sistema in preprosto vzdrževanje. S tem pa tudi enostavno dodajanje novih funkcionalnosti z le majhnimi spremembami arhitekture [3]. Fazi načrtovanja sistema sledi faza načrtovanja komponent. V našem primeru bomo obe fazi načrtovanja združili v enotno poglavje.

Naš sistem je zgrajen na arhitekturnem modelu odjemalec - strežnik. Iz tega razloga lahko sam sistem razdelimo na dve večji komponenti, ti dve komponenti pa bomo kasneje po potrebi razdelili na še manjše podkomponente. Glavni komponenti našega sistema sta:

- Strežnik virtualnega okolja (računalniški program):
  *služil bo za zajem, obdelavo ter pošiljanje slik odjemalcu virtualnega okolja*
- Odjemalec virtualnega okolja (Android aplikacija):
  *služil bo za prejem ter prikaz okolja*

Komponenti in njuna povezava je prikazana s komponentnim diagramom sistema (slika 6).




Slika 6: Komponentni diagram sistema

Komponenti bosta komunicirali prek TCP/IP protokola ter s pomočjo vtičev. Strežniški del sistema bo ob samem zagonu ter inicializaciji začel poslušati na določenih vratih ter bo počakal na povezavo s strani odjemalca. Po uspešni vzpostavitvi povezave bomo lahko začeli s komunikacijo (posredovanjem okolja oddaljeni napravi). Komunikacijo

lahko prekinemo s pomočjo grafičnega vmesnika na strežniku ali z izhodom iz aplikacije na odjemalni napravi.

## 5.1 Strežnik virtualnega okolja

Glavne naloge strežnika lahko ločimo na posamezne dele, in to so:

- Komunikacija z odjemalcem
- Zajem slik
- Obdelava slik

Vseboval bo tudi preprost grafični vmesnik, ki ga nismo šteli med osnovne naloge sistema.

### 5.1.1 Komunikacija z odjemalcem

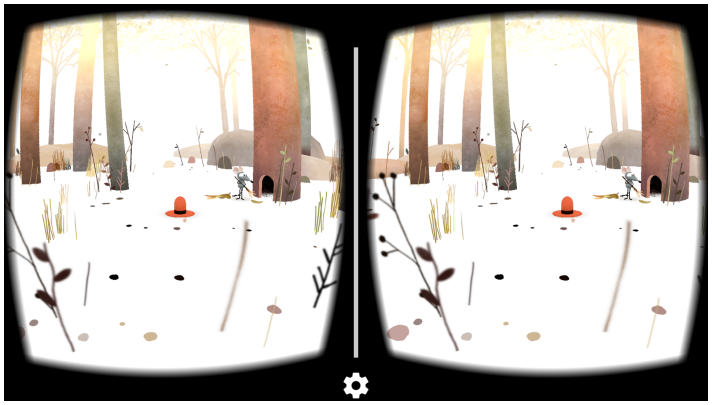
Strežnik bo z odjemalcem komuniciral preko TCP/IP protokola. Strežnik bo na vratih 8080 vzpostavil vtič in čakal na povezavo odjemalca. Ob kliku uporabnika na gumb *Začni oddajanje* bo strežnik začel s pošiljanjem slik na odjemalno napravo po uspešno vzpostavljeni komunikacijski poti. Slike moramo predhodno pripraviti za pravilno pošiljanje prek vtiča. Slike zajete iz spletnih kamer so na strežniškem delu sistema v surovi obliki (*raw*). Prav tako jih obdelujemo v dotičnem formatu. Za prenos preko vtičev jih je potrebno najprej spremeniti v *jpg* format. Izbrali smo *jpg* format, ker je pretvorba hitra. Hitra je iz razloga, saj gre za izgubno stiskanje (angl. lossy compression) digitalnih fotografij. Iz *jpg* formata pa jih spremenimo v polje bajtov (angl. byte array). V takem formatu jih pošljemo prek vtiča ter jih na odjemalni napravi brez težav prejmemo. Prejem na odjemalni napravi bomo obravnavali v nadaljevanju poglavja, pri opisu odjemalca virtualnega okolja. Uporabnik sam ne bo mogel nadzorovati hitrosti pošiljanja slik. To nalogo smo prepustili vtičem samim.

### 5.1.2 Zajem slik

Zajem slik bo realiziran s pomočjo knjižnice OpenCV. Uporabljena bo verzija 2.4.9. Knjižico bomo podrobneje predstavili v nadaljevanju zaključne naloge. Strežnik bo po uspešni vzpostavitvi povezave z odjemalcem začel z zajemom slik iz spletnih kamer. Obe sliki bo moral zajeti istočasno. Nato bosta sliki šli v obdelavo, da jih bomo lahko prikazali na Android napravi v pravilni obliki.

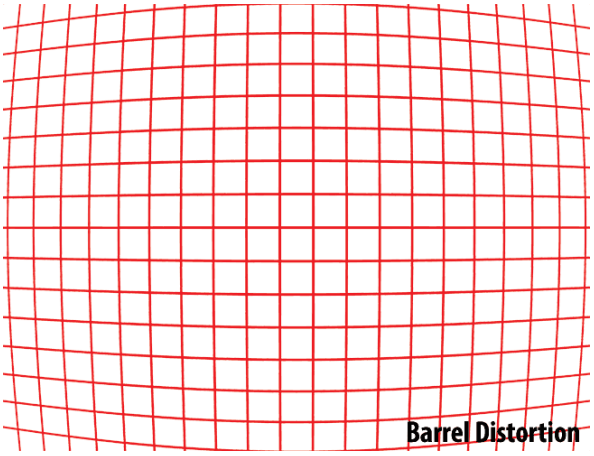
### 5.1.3 Obdelava slik

Po uspešni vzpostavitvi povezave z odjemalcem in zajemom slik pride na vrsto obdelava slik. Cardboard potrebuje za pravilno prikazovanje slik "sodčkasto distorzijo" oziroma angleško "barrel distortion". Na sliki 7 je prikazana zaslonska slika Cardboardove demonstracijske aplikacije "Windy Day". Na sliki 7 lahko vidimo, kako je treba obdelati slike za pravilen prikaz.



Slika 7: Prikaz Cardboard aplikacije

Sodčkasta distorzija (angl. barrel distortion) je oblika radialne distorzije. Prepoznamo jo po tem, da so črte ob robovih slike ukrivljene navznoter in spominjajo na obliko soda (angl. barrel). Od tod tudi ime sodčkasta distorzija oziroma barrel distortion. Sodčkasta distorzija je posledica lastnosti leče (objektiva) ter jo srečamo predvsem pri širokokotnih objektivih. Posledica temu je slika takšne oblike, kot jo prikazuje slika 8.




Slika 8: Sodčkasta distorzija. Vir: [4]

Na sliki 8 je lepo razvidno, da bolj kot se oddaljujemo od sredine slike, bolj so črte

zaobljene. Na sredini pa delujejo črte ravne. Torej to pomeni, da je predmet ali objekt, ki ima ravne stranice, na sliki prikazan z zaobljenimi stranicami.

Za izvedbo sodčkaste distorzije je potrebno izračunati premike pikslov po x in y osi. Splošna enačba za izračun premikov po x osi je:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{5.1}$$

ter po y osi:

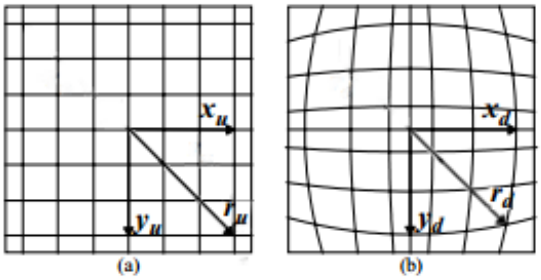
$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{5.2}$$

Pri čemer sta $x_{corrected}$ in $y_{corrected}$ obdelani koordinati točke izhodne/obdelane slike, točki $x$ in $y$ pa pripadajoči koordinati točke neobdelane slike. Koeficienti $k_1, k_2, k_3$ so koeficienti radialne distorzije [7]. $r$ predstavlja razdaljo od optičnega centra slike (običajno središča slike).

Pri našem sistemu bomo uporabili izpeljano enačbo za premike po x in y osi, ki je nižjega reda. Enačba se glasi:

$$r_u = r_d(1 + kr_d^2) \tag{5.3}$$

Pri čemer sta $r_u$ ter $r_d$ razdalji od središča slike pri obdelani sliki ter pri zajeti sliki. $k$ je koeficient distorzije, ki je za vsako lečo drugačen [6]. Za lažje razumevanje je priložena slika 9, ki prikazuje na sliki (a) nespremenjeno sliko ter na sliki (b) obdelano sliko.



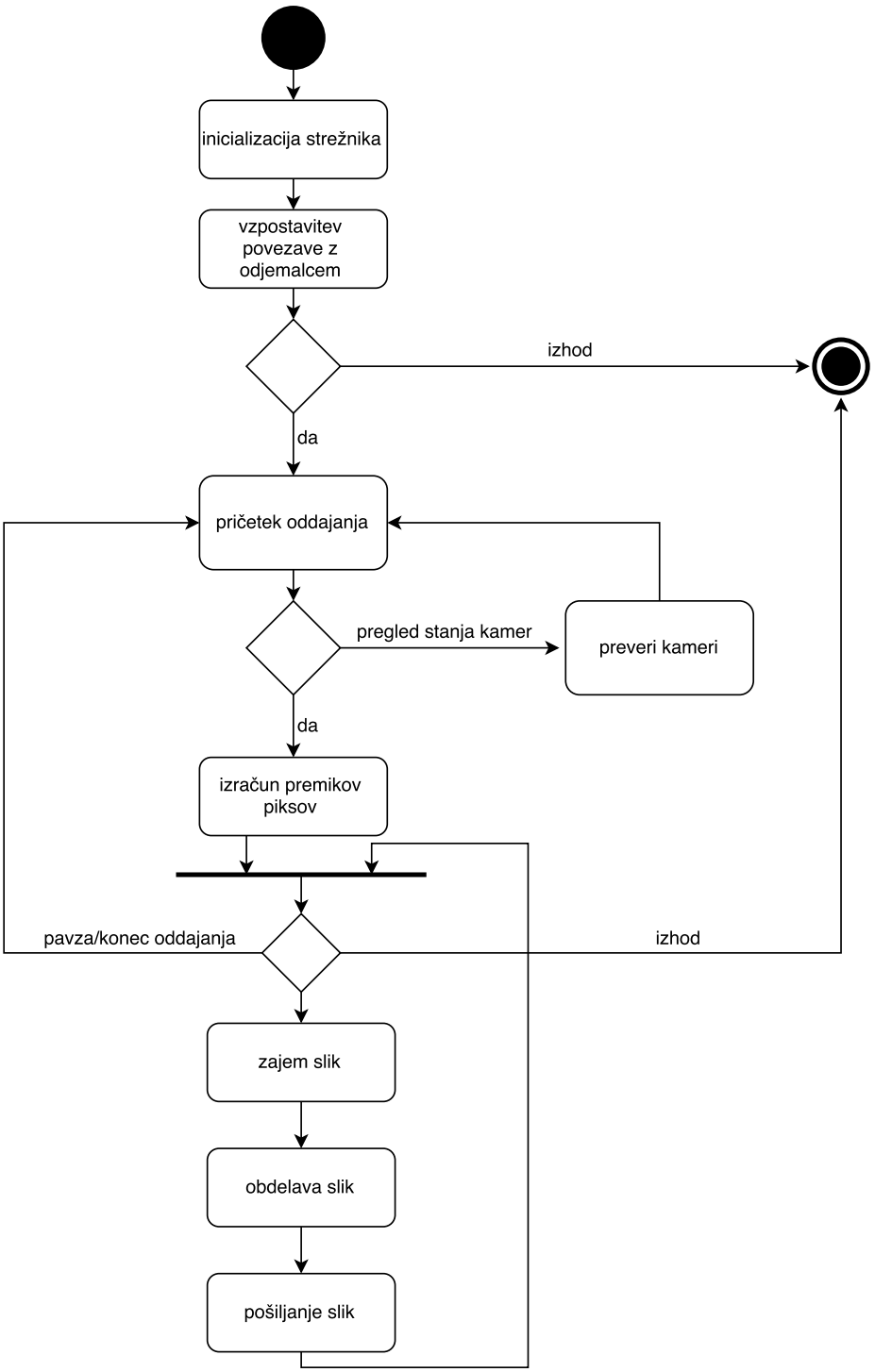

Slika 9: Prikaz sodčkaste distorzije

Točne koeficiente za naš sistem bomo definirali kasneje, v fazi izvedbe sistema.

### 5.1.4 Grafični vmesnik

Strežnik bo vseboval tudi preprost grafični vmesnik (slika 4) za poenostavljeno interakcijo z uporabnikom. Na vrhu bo izpisan IP naslov ter domensko ime strežnika. Uporabnik mora vpisati enega od teh v odjemalno napravo za uspešno vzpostavitev povezave. Po uspešni povezavi z odjemalcem, bo s klikom na gumb *Začni oddajanje* možno začeti oddajanje slike iz strežnika na odjemalca. Gumb bo med oddajanjem okolja služil za prekinitev oddajanja, besedilo se bo tako spremenilo v *Končaj oddajanje*. S klikom na gumb *Stanje kamer* pa posodobimo besedilo o stanju kamer. Gumb bo mogoče klikniti le med tem, ko ne bomo oddajali slike na odjemalno napravo. Če bosta kameri priključeni ter pripravljeni za uporabo bomo izpisali besedo *OK*, če pa bo prišlo do napake, bomo izpisali besedno zvezo *NOT OK*. Grafični vmesnik bo implementiran s pomočjo Swing knjižnice.

### 5.1.5 Diagram aktivnosti strežnika




Slika 10: Diagram aktivnosti strežniškega dela sistema

**Potek izvajanja programa:**

1. Uporabnik zažene aplikacijo, ob tem se strežnik inicializira
2. Strežnik počaka na vzpostavitev povezave s strani odjemalne naprave
   - 2.1. Alternativni potek: uporabnik zapusti aplikacijo
3. Uporabnik začne oddajanje s klikom na gumb *Začni oddajanje*
   - 3.1. Alternativni potek: uporabnik klikne na gumb *Pregled stanja kamer*
     - 3.1.1. Aplikacija izpiše stanje kamer
     - 3.1.2. Aplikacija se vrne v stanje 3.
4. Aplikacija izračuna funkciji za premik pikslov pri obdelani sliki
   - 4.1. Alternativa: uporabnik zaustavi/konča oddajanje
     - 4.1.1. Aplikacija se vrne v stanje 3.
   - 4.2. Alternativa: uporabnik zapusti aplikacijo
5. Aplikacija zajame sliko
6. Aplikacija obdela sliko
7. Aplikacija pošlje sliko odjemalcu
8. Vrnemo se na stik po koraku 4.

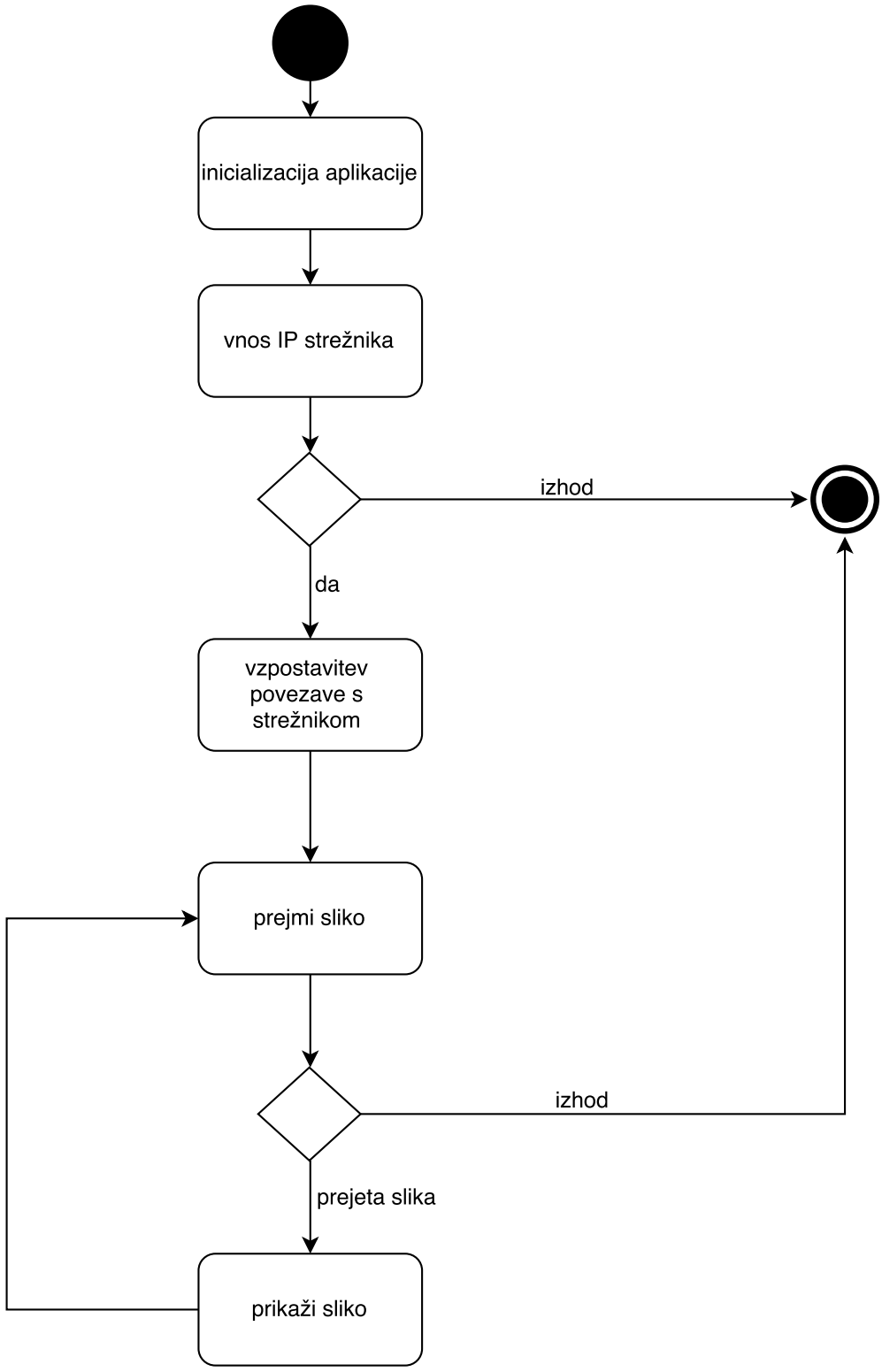
## 5.2 Odjemalec virtualnega okolja

Druga komponenta našega sistema je Android aplikacija, ki bo služila kot odjemalni del. Naloga te komponente bo vzpostavitev povezave s strežniškim delom sistema, prejem slik iz strežnika ter prikaz le-teh. Kot že omenjeno v poglavju *Analiza in definiranje zahtev* smo pri specifikaciji zahtev dodelili odjemalcu karseda malo nalog iz preprostega razloga, saj nimajo vse Android naprave velike procesorske moči in ne želimo obremenjevati sistema z nepotrebnimi opravili. Tako strežnik opravi vse potrebne korake za pravilno prikazovanje slik (obdelava, združevanje). Naloga Android naprave je le ta, da dobljeno sliko izriše na ekran naprave. Za prikaz slike bomo uporabili element *ImageView* (tj. element za prikazovanje slik v Android aplikacijah), ki se bo raztezal čez celoten zaslon aplikacije. Aplikacija bo potrebovala tudi dodatne omejitve, ekran ne bo smel iti v spanje, saj bi s tem prekinili prejem slik ter aplikacija mora biti pognana v ležečem položaju. Prav tako mora aplikacija delovati v celozaslonskem načinu, torej odstraniti moramo opravilno vrstico Android operacijskega sistema.

Naša aplikacija potrebuje dostop do internetne povezave za uspešno povezavo s strežnikom in prejem slik. Zato moramo v *AndroidManifest.xml*[1] datoteko dodati naslednjo pravico *android.permission.INTERNET* za dostop do internetne povezave. V isti datoteki definiramo tudi, da se bo aplikacija zagnala v celozaslonskem načinu ter ležečem položaju.

[1]Manifest datoteka je datoteka, ki jo vsebuje vsaka od Android aplikacij. V tej datoteki so shranjene vse informacije o aplikaciji, ki jih sistem potrebuje še pred zagonom naše aplikacije.

### 5.2.1 Diagram aktivnosti odjemalca




Slika 11: Diagram aktivnosti odjemalnega dela sistema

**Potek izvajanja programa:**

1. Uporabnik zažene aplikacijo, ob tem se aplikacija inicializira
2. Uporabnik vnese IP naslov/domensko ime strežnika
   2.1. Alternativni potek: uporabnik zapusti aplikacijo
3. Aplikacija vzpostavi povezavo s strežnikom
4. Aplikacija prejme sliko
   3.1. Alternativni potek: uporabnik zapusti aplikacijo
5. Aplikacija prikaže sliko
6. Vrnemo se v stanje 4.

# 6 Izvedba

Fazama načrtovanja sistema ter načrtovanja komponent sledi faza izvedbe. Gre za eno najtežjih faz razvoja programskega produkta ter je hkrati eno od najbolj kritičnih. V tej fazi lahko odkrijemo veliko napak, ki smo jih pri prejšnjih fazah storili, spregledali ali enostavno pozabili nanje. Vendar temu ne bi smelo biti tako, saj linearni programski proces ne dovoljuje vračanja v predhodne faze razvoja. Veliko projektov lahko zaradi slabe študije izvedljivosti, slabega definiranja zahtev ter slabega načrtovanja propade. Pojavijo se lahko številni zapleti, npr. projektu smo dodelili premalo časa za samo implementacijo. To pa lahko vodi do nezadovoljstva pri končnem naročniku ali do izdelave produkta, ki ne zadovoljuje vseh potreb naročnika.

V nadaljevanju bomo predstavili uporabljeno programsko opremo, potek same implementacije ter izpostavili bomo izvedbo glavnih delov našega sistema.

## 6.1 Uporabljena programska orodja

Faze izvedbe našega sistema smo se lotili z izborom primernih razvijalskih okolij, ki nam bodo čim bolj poenostavile samo implementacijo našega sistema. Izbrati je bilo potrebno tudi programski jezik, ki ga bomo uporabili. Izbran je bil programski jezik Java, saj so Android aplikacije napisane s pomočjo tega programskega jezika. Posledično smo se odločili, da bomo tudi strežniški del sistema implementirali s pomočjo istega programskega jezika. Za delo s slikami pa smo izbrali OpenCV knjižnico. Torej pripravili ter namestili smo si naslednja okolja, orodja ter knjižnice:

- JDK - Java Development Kit
- Android SDK
- Eclipse
- Android studio
- OpenCV knjižnica

### 6.1.1 Java

Programski jezik Java je objektno orientiran programski jezik. Ta programski jezik smo izbrali, zaradi dobre in obsežne dokumentacije, dobrih performans, prenosljivosti programov iz enega operacijskega sistema na drugi brez nepotrebnih sprememb v sami kodi ter zaradi podpore velikemu številu knjižnic. Za nas je bistvenega pomena, da Java podpira knjižnico OpenCV. Ta pogoj je izpolnjen, tako da tukaj ne bo prišlo do večjih zapletov. Java pa vsebuje tudi knjižnice za izdelavo grafičnih vmesnikov, tako bomo lahko razvili preprost grafični vmesnik za poenostavljeno interakcijo z uporabnikom. Dobra stran takega izbora je tudi ta, da je razvijalec dobro tehnično podkovan in ima precej znanja z uporabo dotičnega programskega jezika. Jave pa ne bomo uporabili samo na strežniškem delu sistema, vendar tudi na odjemalnem delu. S tem bomo rešili tudi težavo implementacije vtičev do katerih lahko pride, če poskušamo komunicirati prek vtičev s programi, napisanimi v različnih programskih jezikih.

Za začetek razvoja smo morali najprej namestiti Javo oziroma Java Development Kit[1]. Po uspešni namestitvi ter pripravi sistemskih spremenljivk smo lahko nadaljevali fazo izvedbe z namestitvijo Eclipse razvojnega okolja.

### 6.1.2 Eclipse

Eclipse[2] je integrirano razvojno okolje (IDE) napisano v Javi. Eclipse podpira številne programske jezike, med temi tudi Javo. Okolje vsebuje vse potrebno za razvoj programske opreme. Vsebuje urejevalnik programske kode, ki nam omogoča preprosto pisanje kode, saj nas opozarja na napake, omogoča samodejno vključevanje dodatnih knjižnic, samodejno zaključuje programske ukaze ter nam predlaga potrebne popravke. Prav tako vsebuje prevajalnik programske kode ter najpomembneje razhroščevalnik, ki nam omogoča pregled programske kode med izvajanjem, vrstico po vrstico. Omogoča tudi enostavno uvažanje knjižnic. Sami smo morali uvoziti OpenCV knjižnico ter jo dodati našemu projektu. Okolje Eclipse je ta postopek zelo poenostavil in le z nekaj kliki smo že imeli pripravljeno okolje za začetek izvedbe strežniškega dela sistema.

### 6.1.3 Android Studio

Android Studio[3] je integrirano razvojno okolje, ki ga je razvilo podjetje Google. Namenjeno je razvoju Android aplikacij. Prav tako kot Eclipse tudi Android Studio vsebuje vse potrebno za razvoj Android aplikacij. Okolje nam omogoča preprost razvoj, saj nam pripravi vse potrebne datoteke za začetek razvoja. Zelo priročen je tudi iz razloga,

[1] http://www.oracle.com/technetwork/java/javase/downloads/index.html
[2] https://eclipse.org/home/index.php
[3] https://developer.android.com/sdk/index.html

saj nam omogoča takojšnjo namestitev aplikacije na priključeno napravo. Tako lahko aplikacijo enostavno zaženemo na napravi ter spremljamo izvajanje našega programa, saj Android Studio omogoča izpis oziroma prikaz zapisov (angl. logs) aplikacije. Izpisani pa niso le zapisi, ki jim sami določimo, kdaj in kje se bodo izpisali, vendar nam javlja tudi vse napake, ki se zgodijo med izvajanjem aplikacije. Android Studio omogoča tudi grafično izdelavo aplikacij, tako lahko takoj vidimo, kam bomo postavili določene gradnike. Omogoča tudi predogled zaslona na številnih različnih napravah. Prednost je seveda v tem, saj lahko na enem mestu preverimo kako bo naša aplikacija prikazana na različnih napravah oziroma na različno velikih zaslonih.

### 6.1.4 Android SDK

Med samo namestitvijo razvojnega okolja Android Studio smo uspešno namestili tudi Android SDK. Namestili smo verzijo 22.0.1. SDK vključuje potrebne knjižnice za razvoj Android aplikacij. Za nas pomemben je tudi razhroščevalnik, ki nam omogoča preprosto razhroščevanje ter spremljanje poteka naše aplikacije. Razhroščevalnik je prav tako vsebovan v Android SDK-ju. SDK pa vsebuje tudi vodiče, primere implementacij ter emulator.

### 6.1.5 OpenCV

Knjižnica OpenCV[4] je odprtokodna knjižnica napisana v C/C++ programskem jeziku. Namenjena je računalniškemu vidu ter področjem strojnega učenja. Podpira številne programske jezike. Uporabimo jo lahko v povezavi s programskim jezikom C, C++, Python ter, za nas najbolj pomembno, Javo. OpenCV je zelo priljubljen med razvijalci programske opreme. Vsebuje več kot 2500 algoritmov, je preprosta za uporabo in omogoča hiter razvoj aplikacij. OpenCV ima tudi obsežno ter podrobno napisano dokumentacijo [5]. Sami bomo knjižnico OpenCV potrebovali pri zajemu slik iz kamere ter pri sami inicializaciji kamer. Potrebovali jo bomo tudi pri obdelavi slik ter pri pripravi slik za pošiljanje prek vtiča (združevanje slik ter sprememba formata).

## 6.2 Potek izvedbe

Po uspešni pripravi potrebnih okolij (Eclipse ter Android Studio) smo začeli z implementacijo. Izvedbo smo razdelili na manjše enote, tako smo lahko sistemu počasi dodajali funkcionalnosti, vse dokler nismo prišli do zaključene celote. Začeli smo z zajemom slik na strežniku. Po uspešni implementaciji zajema slik je bilo potrebno

[4]http://opencv.org/downloads.html

implementirati še združevanje slik, ki ga bomo kasneje potrebovali. To smo naredili za potrebe testiranja vtiča, ki smo ga razvili v naslednji fazi. Kot že rečeno smo nadaljevali izvedbo sistema z implementacijo vtiča za komunikacijo med strežnikom ter odjemalcem. Najprej je strežnik pošiljal sliki odjemalcu, ta pa jih je shranil na svoj zunanji pomnilnik. Tako smo preverili, da je vtič pravilno implementiran ter da imamo pravilno nastavljeno ter vzpostavljeno komunikacijo. Po komunikaciji smo implementacijo nadaljevali na odjemalni napravi. Pripravili smo potrebne omejitve, kot so preprečitev, da gre zaslon v spanje ter da se aplikacija zažene v ležečem položaju. Nato smo pripravili še *ImageView* za potrebe prikaza slike. Preuredili smo še vtič tako, da sedaj ne potrebujemo več shranjevanja slike na zunanji pomnilnik, ampak da sliko hranimo v notranjem pomnilniku in jo takoj prikažemo. Ker za pošiljanje slik uporabljamo *byte array*[5] moramo sliko pretvoriti v *Bitmap*, da jo lahko prikažemo na *ImageView* gradniku. Do tukaj smo imeli pripravljeno zajem slik, pošiljanje slik ter prikaz na Android napravi. Manjkala nam je le obdelava slik ter grafična vmesnika.

Sledila je implementacija obdelave slik, ki smo jo podrobneje obravnavali v podpoglavju *Obdelava slik* poglavja *Načrtovanje sistema in komponent*. Sedaj je naša naloga aplicirati tako distorzijo na naših slikah. Uporabili bi lahko že prej omenjeni Googlov SDK za delo s Cardboardom na Android napravah. Ampak mi ne želimo delati obdelave slik na odjemalni napravi, vendar na strežniku, zato smo to idejo opustili. Odločili smo se, da bomo slikam distorzijo vnašali s pomočjo OpenCV knjižnice. Uporabiti bo potrebno *remap()* funkcijo. Funkcija služi premiku pikslov iz ene lokacije na drugo po funkciji enačbe, ki ji jo podamo sami[6]. Funkcija *remap()* za svoje delovanje potrebuje sledeče parametre[7]:

- **src** - zajeta slika iz kamere
- **dst** - obdelana slika
- **map_x** - polje, ki določa nove koordinate x za vsak slikovni element izvorne slike
- **map_y** - polje, ki določa nove koordinate y za vsak slikovni element izvorne slike
- **interpolation** - metoda interpolacije

Vendar preden poženemo *remap()* funkcijo moramo pripraviti map_x ter map_y sliki. Za pripravo slik potrebujemo koeficient distorzije. Koeficient za x in y distorzijo smo morali poiskati sami. Poskusili smo koeficiente pridobiti z uporabo demonstracijske aplikacije Google Cardboard-a in razreda *Distortion*. Razred vsebuje metodo

[5]polje bajtov
[6]http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html#remap
[7]http://docs.opencv.org/java/

*getCoefficients()*, ki nam vrne koeficiente distorzije. Tako smo s pomočjo te metode dobili dva koeficienta in sicer 0.144 ter 0.156. SDK Google Carboard uporablja naslednji enačbi za izračun distorzije:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4) \tag{6.1}$$

ter

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4) \tag{6.2}$$

Mi pa bomo v našem primeru uporabili enačbo nižjega reda. Tako moramo zanemariti drugi koeficient. Enačbo (5.3), ki smo jo spoznali v podpoglavju *Obdelava slik* poglavja *Načrtovanje sistema in komponent*, smo v našem primeru implementirali na naslednji način za x os:

$$xcenter + (x - xcenter) * (1 + k * r^2) \tag{6.3}$$

ter tako za y os:

$$ycenter + (y - ycenter) * (1 + k * r^2) \tag{6.4}$$

Za izračun $r^2$ smo uporabili naslednjo enačbo:

$$r^2 = (x - xcenter) * (x - xcenter) + (y - ycenter) * (y - ycenter) \tag{6.5}$$

Nato smo morali pretvoriti enote distorzijskih koeficientov, ki so navedeni v Google Cardboard SDK-ju, da bodo ustrezali našim izračunom. Sami računamo radij po številu pikslov, medtem ko Cardboard-ov SDK računa radij v tangencialnih kotnih enotah. Da bi torej koeficient distorzije ustrezal, je bilo potrebno enačbo za $r^2$ deliti še z naslednjo enačbo za x os:

$$(xcenter - visinaSlike) * (xcenter - visinaSlike) \tag{6.6}$$

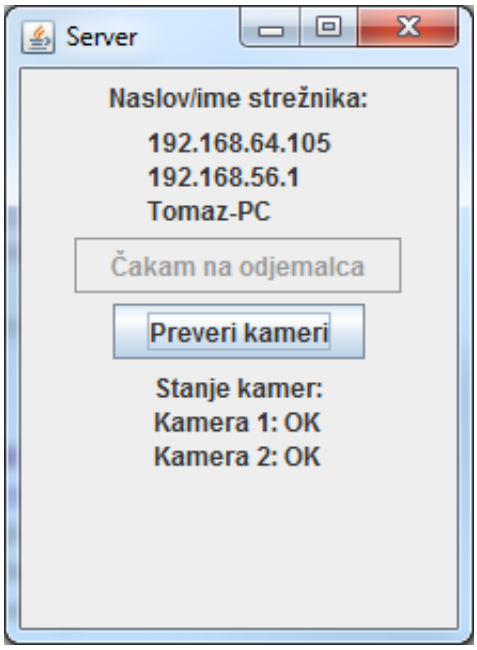
ter tako za y os:

$$(ycenter - sirinaSlike) * (ycenter - sirinaSlike) \tag{6.7}$$

Pri čemer je *xcenter* sredina slike po x osi ter *ycenter* sredina slike po y osi. *visinaSlike* predstavlja celotno višino slike, *sirinaSlike* pa predstavlja celotno širino slike. Višina ter širina sta podani v pikslih. Tako smo v vsakem ciklu delili $r^2$ s pripadajočo enačbo za x in y os. Sedaj ko smo dobili koeficient distorzije in je le-ta ustrezal, smo lahko pripravili obe sliki za premike pikslov po x in y osi. Izbrati smo moramo le še primerno interpolacijo. Izbrali smo linearno interpolacijo glede na dejstvo, da

je v članku [6] uporabljena ravno ta interpolacija. Interpolacijo potrebujemo iz tega razloga, ker nove izračunane vrednosti x in y niso celoštevilske številke in tako novo nastali piksel leži med piksli v originalni sliki. Z interpolacijo rešimo to težavo [6].

Sledila je še implementacija grafičnih vmesnikov. Za implementacijo grafičnega vmesnika na strežniku smo uporabili knjižnico Java Swing, ki vsebuje vse potrebne gradnike. Najprej smo določili razredu, da deduje lastnosti JFrame (tj. okno) razreda. Temu oknu smo dodali dve polji za besedilo oziroma gradnika *JLabel* ter dva gumba oziroma gradnika *JButton* za enostavno interakcijo. Eden bo služil za začetek ter konec oddajanja slik, drugi pa je namenjen preverjanju stanja kamer. Grafični vmesnik je prikazan na sliki 12. Na odjemalnem delu sistema smo prav tako pripravili vnosno polje za vnos IP naslova oziroma domenskega imena strežnika. Uporabili smo pojavno okno z vnosnim poljem ter potrditvenim gumbom.




Slika 12: Grafični vmesnik

Z vsemi naštetimi koraki smo prišli do zaključka implementacije našega sistema.

# 7 Testiranje

Po fazi izvedbe sledi faza testiranja. S to fazo želimo zagotoviti, da naš izdelani produkt pravilno deluje ter z validacijo preverimo, ali produkt ustreza uporabnikovim potrebam, to pa ugotovimo s preverjanjem skladnosti s specifikacijo zahtev. Testiranje je proces, s katerim želimo poiskati napake pred predajo končnemu uporabniku. Med samim testiranjem želimo porušiti pravilno delovanje programa ter s tem najti še neodkrite napake. Test je uspešen, kadar je najdena napaka, ki je bila prej neodkrita [3]. Testiranje lahko opravi nekdo, ki je razvijal produkt, ali neodvisni preizkuševalec, ki sistema ne pozna. V našem primeru bo testiranje opravil nekdo iz razvojne skupine in sicer bo to opravil razvijalec sam.

## 7.1 Strukturno testiranje

Strukturno testiranje (angl. white-box testing) je testiranje, ki temelji na poznavanju notranje strukture programa [3]. S strukturnim testiranjem želimo zagotoviti, da vsak od pogojev deluje pravilno, da smo preizkusili vse mogoče vejitve programa in da smo izvedli vsak del kode vsaj enkrat.

Pri našem sistemu je število vrstic programske kode majhno ter program ne vsebuje veliko vejitev. Vsaka iteracija programa gre skozi enak del kode. Preveriti je bilo potrebno le, da so pogoji pravilno napisani oziroma da niso zanke neskončne. Testiranje smo izvedli s pomočjo razvijalskega okolja Eclipse in možnostjo razhroščevanja programske kode. Tako smo lahko ob samem izvajanju programa pregledali, ali je vsak od pogojev pravilno obravnavan. Enak postopek smo ponovili na odjemalni napravi, le da smo si tam pomagali s pomočjo razvojnega okolja Android Studio ter s pomočjo *logcat*[1] orodja. S testiranjem smo ugotovili pravilno delovanje sistema ter smo tako lahko nadaljevali na naslednji korak testiranja.

[1] Orodje za izpis zapisev (angl. logs) aplikacije

## 7.2 Vedenjsko testiranje

Vedenjsko testiranje (angl. black-box testing) je testiranje, ki temelji na zunanjih lastnostih sistema [3]. S takim testiranjem želimo zagotoviti, da naš sistem deluje pravilno, torej da glede na vnesene vhodne podatke dobimo pričakovane izhodne podatke.

Naš sistem je sestavljen iz dveh komponent, ki sta medsebojno odvisni in tako je testiranje nemogoče opraviti le na strežniškem delu ali samo na odjemalnem delu. Iz tega razloga bomo poskušali opraviti testiranje na obeh delih hkrati.

### 7.2.1 Načrt testiranja

Testiranje bomo začeli na strežniškem delu sistema. Najprej bomo testirali grafični vmesnik. Preverili bomo delovanje obeh gumbov, torej gumb za začetek oddajanja ter gumb za stanje kamer. Ker ne moremo začeti oddajanja brez odjemalne naprave, bomo nato zagnali aplikacijo na odjemalni napravi ter tako testirali vnosno polje za IP naslov oziroma domensko ime strežnika. Z odjemalno napravo se bomo uspešno povezali na strežniški del sistema. Sedaj bomo lahko preverili delovanje gumba za oddajanje slik ter gumba za preverjanje stanja kamer. Po testiranju grafičnega vmesnika na strežniškem delu sistema (posledično bomo testirali tudi grafični vmesnik na odjemalni napravi, in sicer vnosno polje ter vzpostavitev povezave) bomo preverili, da so zajete slike pravilno obdelane in s tem tudi pravilno prikazane na odjemalni napravi. Nato bomo preverili še delovanje gumba za zaključek oddajanja. V tabeli 1 se nahajajo specifikacije testa, torej našteti so elementi testiranja ter pričakovani rezultati. V naslednjem poglavju bomo obravnavali rezultate testiranja ter pregledali, ali je kje prišlo do odstopanj med pričakovanimi ter dejanskimi rezultati. V kolikor bo do odstopanj prišlo, bomo poskušali identificirati zakaj prihaja do napačnih rezultatov.

Tabela 1: Tabela elementov testiranja ter pričakovani rezultati

| **Element testiranja** | **Pričakovan rezultat** |
|---|---|
| Vnosno polje odjemalne naprave | Vzpostavljena povezava s strežnikom |
| Gumb za začetek oddajanja okolja | Začetek oddajanja okolja na odjemalno napravo |
| Obdelava slik | Pravilno obdelane slike |
| Vzpostavitev povezave | Vzpostavljena povezava odjemalca s strežnikom |
| Pošiljanje slik | Pravilen prenos slik od strežnika do odjemalca |
| Prikaz slik | Pravilen prikaz slik na odjemalni napravi |
| Gumb za konec oddajanja okolja | Zaključek oddajanja okolja na odjemalno napravo |
| Gumb za preverjanja stanja kamer | Izpis stanja kamer |

### 7.2.2 Odkrite napake ter popravki

Testiranje smo opravili v celoti ter opazili nekatera odstopanja od pričakovanih rezultatov. Spodaj je naštetih nekaj primerov napak, ki smo jih identificirali in so se pojavile med samim testiranjem.

Prva napaka se pojavi, ko je strežnik v stanju oddajanja slik in odjemalna naprava prekine povezavo. V tem primeru lahko prihaja do različnih izjem (angl. Exception). Večkrat smo naleteli na *SocketException* (sl. izjema vtiča). Izjema se pojavi zaradi tega, ker se povezava prekine, medtem ko strežnik že pošilja podatke odjemalni napravi. Prav tako lahko prihaja do izjeme knjižnice OpenCV, če odjemalna naprava prekine povezavo med zajemom slik ali obdelavo slik. Napaka še ni bila odpravljena. Predvideno je, da se ob prekinitvi povezave odjemalca vrnemo v začetno stanje, kjer čakamo na ponovno povezavo odjemalca (lahko je to isti ali drug odjemalec).

Druga napaka se lahko pripeti, če začnemo oddajanje brez priklopa kamer na strežniško napravo, saj metoda *read()*, ki zajema slike iz kamere, ne more zajeti slike. Napaka je enostavno rešljiva, treba je le dodati pogoj za preverjanje ali sta kameri pripravljeni za uporabo pred samim zajemom slike. V kolikor kameri nista pripravljeni, je potrebno javiti napako uporabniku ter ga opozoriti na priklop kamer.

Opazili smo, da je potrebno za vzpostavitev povezave z novim odjemalcem ponovno inicializirati oziroma zagnati strežnik, ko končamo oddajanje predhodnemu odjemalcu. Potrebno bi bilo spremeniti programsko kodo na tak način, da bi lahko po končanem oddajanju predhodnemu odjemalcu sprejeli naslednjega odjemalca. Tukaj se je pokazala pomanjkljivost v fazi analize in definiranja zahtev.

Pojavile pa se niso le napake na strežniškem delu sistema, temveč so bile napake oziroma pomanjkljivost prisotne tudi na odjemalnem delu sistema. Opazili smo, da je nit, ki je bila zadolžena za prejemanje slik aktivna tudi po zaustavitvi aplikacije. Metoda *stop()*, ki poskrbi za to, da se nit zaustavi, je *deprecated* tj. zastarela, in ni pravilno zaustavila niti in je le ta še vedno tekla v ozadju, vse dokler nismo aplikacije dokončno "ubili" z namensko aplikacijo. Tako smo morali dodati poseben ukaz, ki pravilno zaustavi celotno aplikacijo.

## 7.3 Testiranje prepustnosti sistema

Naše testiranje ni bilo obsežno, saj smo vsako komponento testirali že med samo fazo izvedbe sistema. Tako smo se med fazo testiranja osredotočili tudi na testiranje časovne zahtevnosti vseh delov kode, da bi ugotovili možne zavore ter bi poskušali optimizirati kodo in s tem pohitrili naš sistem.

Začeli smo s testiranjem strežniškega dela sistema. Sumili smo, da je za počasno

prikazovanje slik na Android napravi kriva povezava. Tako smo testirali število slik, ki jih pošljemo od strežnika do odjemalca. Strežnik ter odjemalna naprava sta bila povezana na brezžično omrežje prek istega usmerjevalnika. Obe od naprav sta uporabljali standard IEEE 802.11n[2] in sta bili oddaljeni od usmerjevalnika približno 6 metrov. Hitrost na strežniškem delu sistema je bila po podatkih mrežne kartice približno 72Mbps na odjemalnem delu pa približno 52Mbps. Točne hitrosti povezave ne moremo podati, saj hitrost ni konstantna in je odvisna od števila povezanih naprav, vrste in obsega drugega prometa ipd. Zadevo smo testirali na večjih časovnih intervalih, recimo 10 sekund in izračunali povprečno število poslanih slik. Izračuni so pokazali, da povprečno pošljemo okoli 15 slik na sekundo, saj za eno sliko porabimo v povprečju 65,36 ms. Preverili smo tudi odjemalno napravo in na njej smo zaznali približno enako časovno zahtevnost za prejem slik. Vendar številka 15 slik na sekundo ni povsem točna, saj smo testirali le čas potreben za pošiljanje prek vtiča, ne pa celoten čas cikla, skupaj z zajemom slik, obdelavo ter pošiljanjem. Realno pošiljamo nekje okoli 10 slik na sekundo, če prištejemo še čas, potreben za zajem ter obdelavo slik. Tabela 2 prikazuje časovne zahtevnosti celotnega cikla ter posameznih operacij na strežniškem delu sistema. Tabela 3 pa prikazuje časovne zahtevnosti prejema ter prikaza slik na odjemalnem delu sistema.

Tabela 2: Tabela časovnih zahtevnosti posameznih operacij cikla strežnika

| | **1. meritev** | **2. meritev** | **3. meritev** | **Povprečje** |
|---|---|---|---|---|
| Celoten cikel | 99,54 ms | 98,38 ms | 100,34 ms | 99,42 ms |
| Zajem slik | $< 1$ ms | $< 1$ ms | $< 1$ ms | $< 1$ ms |
| Obdelava slik | 33,5 ms | 33,43 ms | 33,16 ms | 33,37 ms |
| Pošiljanje slik | 65,34 ms | 64,38 ms | 66,35 ms | 65,36 ms |

Tabela 3: Tabela časovnih zahtevnosti prejema ter prikaza slik odjemalca

| | **1. meritev** | **2. meritev** | **3. meritev** | **Povprečje** |
|---|---|---|---|---|
| Prejem in prikaz slik | 99,01 ms | 97,24 ms | 101,63 ms | 99,29 ms |

Testirali smo tudi druge dele kode kot npr. zajem slik, izdelava map za preslikavo x in y pikslov, pretvorba formata slike ipd. Le pri izračunu vrednosti potrebnih za premike pikslov po x in y osi smo zaznali večji čas izvajanja, okoli 80 ms, vendar te vrednosti računamo le enkrat in to še preden začnemo s pošiljanjem. Torej tu ne more prihajati do velikih zaostajanj, saj čeprav zahteva to več časa, se to izvede le pri inicializaciji in ne omejuje hitrosti delovanja (tj. število zajema in obdelave slik).

[2]Standard IEEE 802.11 je standard za brezžično lokalno omrežje

Nobeden od drugih delov kode ni pokazal velike časovne zahtevnosti, tako da smo prišli do zaključka, da je največja zavora našega sistema samo pošiljanje slik oziroma povezava med napravama.

Pri nadaljnjem razvoju bi bilo smotrno premisliti o spremembah v sami implementaciji sistema, da bi poskusili povečati prepustnost sistema. Lahko bi poskusili strežniški del sistema implementirati z uporabo drugega programskega jezika, recimo z uporabo programskega jezika C++. Poskusili bi lahko tudi s spremembo načina povezave med odjemalcem ter strežnikom. Nenazadnje pa bi lahko zmanjšali ločljivost slik.

# 8 Zaključek

V zaključni nalogi smo opisali razvoj sistema za reprodukcijo dejanskega okolja v virtualno resničnost z uporabo para spletnih kamer. Opisali smo celoten potek programskega procesa od študije izvedljivosti pa vse do testiranja. Fazi predaja ter obratovanje in vzdrževanje smo izpustili, saj je bil sistem razvit za namene odkrivanja novih možnosti pri uporabi obstoječe tehnologije ter za potrebe zaključne naloge. Med samo izdelavo zaključne naloge smo spoznali, da so vse faze programskega procesa pomembne. Že pri prvih fazah programskega procesa lahko opazimo, da je za kakovosten izdelek potrebno dobro sodelovanje s končnim uporabnikom, saj le ta pozna potrebe, ki jih potrebuje uporabnik. Vendar moramo tudi mi znati ločiti med potrebami in željami, saj lahko hitro pride do nasprotij pri zahtevah in s tem nemoč realiziranja takega produkta.

Načrti za nadaljnje delo so odprava omenjenih napak v poglavju o testiranju. Strežniški del sistema bi lahko razširili z možnostjo oddajanja slik več odjemalnim napravam hkrati in bi tako dobili nekakšen *multicast*[1] sistem. Če pa se malo oddaljimo od samega programskega produkta, je potrebno poskrbeti tudi za pravilno namestitev kamer, da bodo vedno nameščene vzporedno ter na pravilni razdalji. Tako si želimo izdelati tudi nekakšno stojalo za naši kameri, da ne bo potrebna "ročna" kalibracija kamer.

Razširitve samega sistema vidimo v združitvi naše rešitve z drugim sistemom, ki bi omogočal premike kamer. S tem bi pridobili možnost, da bi si lahko ogledovali celoten prostor z obračanjem glave. S tem bi še bolj poudarili doživetje virtualne resničnosti in bi se resnično počutili, da smo na neki oddaljeni lokaciji.

Zaključno nalogo zaključujemo z željo, da bi naš sistem preizkusilo čim večje število uporabnikov ter bi jih naš sistem ne le navdušil, temveč tudi pritegnil k spoznavanju novega področja, in sicer k spoznavanju virtualne resničnosti.

[1] angleški izraz za oddajanje sporočil več prejemnikom

# 9 Literatura in viri

[1] *What is virtual reality?*, http://whatis.techtarget.com/definition/virtual-reality. (Datum ogleda: 10. 8. 2015.) *(Citirano na strani 1.)*

[2] *Google Cardboard*, https://en.wikipedia.org/wiki/Google_Cardboard. (Datum ogleda: 10. 8. 2015.) *(Citirano na strani 1.)*

[3] P. Rogelj, *Skripta za predmet programsko inženirstvo*, http://www.famnit.upr.si/sl/resources/files/knjiznica/studijsko-gradivo/programskoinzenirstvoskripta.pdf. (Datum ogleda: 11. 8. 2015.) *(Citirano na straneh 12, 28 in 29.)*

[4] N. Mansurov, *What is Distortion?*, https://photographylife.com/what-is-distortion. (Datum ogleda: 13. 8. 2015.) *(Citirano na straneh VIII in 14.)*

[5] *About OpenCV*, http://opencv.org/about.html. (Datum ogleda: 10. 8. 2015.) *(Citirano na strani 24.)*

[6] K.T. Gribbon, C.T. Johnston, and D.G. Bailey, *A Real-time FPGA Implementation of a Barrel Distortion Correction Algorithm with Bilinear Interpolation*, http://sprg.massey.ac.nz/pdfs/2003_IVCNZ_408.pdf. (Datum ogleda: 13. 8. 2015.) *(Citirano na straneh 15 in 27.)*

[7] *Camera calibration with OpenCV*, http://docs.opencv.org/doc/tutorials/calib3d/camera_calibration/camera_calibration.html . (Datum ogleda: 18. 8. 2015.) *(Citirano na strani 15.)*

[8] *Google Cardboard*, https://developers.google.com/cardboard/images/one-cardboard.png. (Datum ogleda: 18. 8. 2015.) *(Citirano na straneh VIII in 1.)*

# Priloge

# A Izvorna koda

Priložena je zgoščenka z vso izvorno kodo sistema.