UNIVERZITET U NOVOM SADU **FAKULTET TEHNIČKIH NAUKA U NOVOM SADU**

**DIPLOMSKI RAD**

Prenosivi sistem za zaštitu od provale

Kandidat: Mentor:

Katarina Ninković prof. dr Vladimir Rajs

Novi Sad, 2020.god.

|  |  |
| --- | --- |
|  | UNIVERZITET U NOVOM SADU ⚫**FAKULTET TEHNIČKIH NAUKA**  21000 NOVI SAD, Trg Dositeja Obradovića 6 |
| **KLJUČNA DOKUMENTACIJSKA INFORMACIJA** |

|  |  |  |  |
| --- | --- | --- | --- |
| Redni broj, **RBR**: | |  | |
| Identifikacioni broj, **IBR**: | |  | |
| Тip dokumentacije, **TD**: | | Monografska publikacija | |
| Тip zapisa, **TZ**: | | Tekstualni štampani dokument | |
| Vrsta rada, **VR**: | | Diplomski rad | |
| Аutor, **АU**: | | Katarina Ninković | |
| Меntor, **МN**: | | Prof. dr Vladimir Rajs | |
| Naslov rada, **NR**: | | Prenosivi sistem za zaštitu od provale | |
| Јеzik publikacije, **JP**: | | Srpski | |
| Јеzik izvoda, **JI**: | | Srpski | |
| Zemlja publikovanja, **ZP**: | | Republika Srbija | |
| Uže geografsko područje, **UGP**: | | Autonomna Pokrajina Vojvodina | |
| Godina, **GО**: | | 2020. | |
| Izdavač, **IZ**: | | Autorski reprint | |
| Мesto i adresa, **МА**: | | 21000 Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6 | |
| Fizički opis rada, **FО**: (poglavlja/strana/citata/таbela/slika/grafika/priloga) | | 10 / 42 / 10 / 1 / 55 / 8 / 0 | |
| Naučna oblast, **NO**: | | Primenjena elektronika | |
| Naučna disciplina, **ND**: | | Računarske komunikacije, Primenjena elektronika | |
| Predmetna odrednica/Ključne reči, **PO**: | | Internet of things | |
| **UDK** | |  | |
| Čuva se, **ČU**: | | Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad | |
| Važna napomena, **VN** | |  | |
| Izvod, **IZ**: | | U projektu je implementiran sistem za detekciju provalnika koji primenom računarske komunikacije (*Ethernet* - *TCP/IP* protokola) obaveštava povezane klijentske uređaje o provali i prenosi (eng. “*streaming*”) snimak uživo. Prenosivi uređaj je *RaspberryPi* mini računar koji radi na *Linux* platformi i izvršava *Python* konzolnu aplikaciju koja rukovodi (eng. “*handles*”) komunikacijom sa klijentskim uređajima i očitavanjima sa senzora. Sa druge strane, klijent predstavlja platformno uniformnu aplikaciju koja je podržana kako na *Windows*, tako i na *Linux* operativnim sistemima. | |
| Datum prihvatanja teme, **DP**: | | Septembar 2020. | |
| Datum odbrane, **DO**: | |  | |
| Članovi komisije, **КО**: | Predsednik: |  |
|  | Član: |  | Потпис ментора |
|  | Mentor: | doc. dr Vladimir Rajs |  |

|  |  |
| --- | --- |
|  | UNIVERSITY OF NOVI SAD⚫**FACULTY OF TECHNICAL SCIENCES**  21000 NOVI SAD, Trg Dositeja Obradovića 6 |
| **KEY WORDS DOCUMENTATION** |

|  |  |  |  |
| --- | --- | --- | --- |
| Accession number, **ANO**: | |  | |
| Identification number, **INO**: | |  | |
| Document type, **DT**: | | Monographic publication | |
| Type of record, **TR**: | | Textual printed material | |
| Contents code, **CC**: | | Bachelor Thesis | |
| Author, **AU**: | | Katarina Ninković | |
| Mentor, **MN**: | | PhD Vladimir Rajs | |
| Title, **TI**: | | Portable system for housebreaking protection | |
| Language of text, **LT**: | | Serbian | |
| Language of abstract, **LA**: | | English | |
| Country of publication, **CP**: | | Republic of Serbia | |
| Locality of publication, **LP**: | | Autonomous Province of Vojvodina | |
| Publication year, **PY**: | | 2020. | |
| Publisher, **PB**: | | Author’s reprint | |
| Publication place, **PP**: | | Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad | |
| Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes) | | 10 / 42 / 10 / 1 / 55 / 8 / 0 | |
| Scientific field, **SF**: | | Applied Electronics | |
| Scientific discipline, **SD**: | | Computer Communications, Applied Electronics | |
| Subject/Key words, **S**/**KW**: | | Internet of things | |
| **UC** | |  | |
| Holding data, **HD**: | | Library of the Faculty of Technical Sciences, Novi Sad | |
| Note, **N**: | |  | |
| Abstract, **AB**: | | In this project has been implemented system for housebreaking detection which informs client devices using Computer communication (Ethernet - TCP/IP interface) about housebreaking and it streams live video. Portable device is *RaspberryPi*, it is amini computer which runs on *Linux* platform and it executes *Python* console application which handles communication with client devices and reading values from sesnors. From another side, client represents platform independent which is runnable both on *Windows* and *Linux* operative systems. | |
| Accepted by the Scientific Board on, **ASB**: | | September 2020. | |
| Defended on, **DE**: | |  | |
| Defended Board, **DB**: | President: | Associate  Definitions of associate  noun  a partner or colleague in business or at work.  he arranged for a close associate to take control of the institute  synonyms: partner, colleague, coworker, workmate, comrade, ally, affiliate, confederate, connection, contact, acquaintance, collaborator, crony, peeps  verb  connect (someone or something) with something else in one's mind.  I associated wealth with freedom  synonyms: link, connect, relate, identify, equate, bracket, set side by side  adjective  joined or connected with an organization or business.  an associate company  2 more definitions  See also  associate  Translations of associate  noun  помоћник  aide, deputy, helper, associate, aid, help  колега  colleague, fellow, associate, fellow worker  заменик  deputy, assistant, substitute, associate, lieutenant, locum tenens  ортак  partner, consort, fellow, associate, bedfellow, yokefellow  учесник  participant, partaker, entrant, participator, partner, privy  verb  везати  tie, bind, link, attach, moor, connect  асоцирати  associate  везивати  bind, connect, relate, associate, tie down  удружити  join, associate, pool, unite, affiliate  adjective  ванредан  associate, extramural, part-time |
|  | Member: |  | Menthor’s sign |
|  | Member, Mentor | PhD Vladimir Rajs |  |

|  |  |  |  |
| --- | --- | --- | --- |
| NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA IZDAJE SE ZADATAK ZA DIPLOMSKI RAD, SА SLEDEĆIM ELEMENTIMA: | | | |
| Student: | Katarina Ninković | Broj indeksa: | EE 32/2015 |
| Predmet: | Primenjena elektronika | | |
| Mentor: | Doc. dr Vladimir Rajs | | |

**TEMA DIPLOMSKOG RADA:**

|  |
| --- |
| Prenosivi sistem za zaštitu od provale |

**ТEKST ZADATKA:**

|  |
| --- |
| 1. Podići operativni sistem (*Raspbian*) na prenosivom uređaju, kao i instalirati neophodne programe i dodati biblioteke 2. Omogućiti komunikaciju između servera i klijenata i pripremiti teren za prenos podataka 3. Omogućiti očitavanje sa senzora detekcije (udaljenosti od objekta), te izveštavanje klijentskih uređaja o prisutnosti provalnika 4. Omogućiti vizualnu detekciju lica uz pomoć kamere priključene za prenosivi uređaj 5. Omogućiti prenos snimka prilikom provale ka svim klijentskim uređajima 6. Realizovati platformno-nezavisnu klijentsku konzolnu |

|  |  |
| --- | --- |
| Šef katedre: | Mentor rada: |
| dr Mirjana Damnjanović | dr Vladimir Rajs |

|  |
| --- |
| Primerak zа: - Studenta; - Mentora; - Katedru; - Studentsku službu fakulteta; |

Obrazac **Q4.09.11** - Izdanje 1

Sadržaj

[Lista skraćenica 3](#_Toc52830438)

[1 Uvod 4](#_Toc52830439)

[2 Blok šema sistema 5](#_Toc52830440)

[3 Komponente i uređaj na serverskoj strani 6](#_Toc52830441)

[3.1 Mašina 6](#_Toc52830442)

[3.2 Senzor udaljenosti 7](#_Toc52830443)

[3.3 Kamera 9](#_Toc52830444)

[4 Alat za detekciju čovekovog lica – OpenCV 10](#_Toc52830445)

[5 TCP/IP soket i mrežno programiranje 11](#_Toc52830446)

[6 Klasifikatori i njihova upotreba 13](#_Toc52830447)

[6.1 Odabir HAAR-karakteristika 13](#_Toc52830448)

[6.2 Kreiranje integralne slike i *Adaboost* treniranje 14](#_Toc52830449)

[6.3 Kaskadno klasifikovanje 15](#_Toc52830450)

[7 Algoritam rada serverske aplikacije 18](#_Toc52830451)

[8 Algoritam rada klijentske aplikacije 22](#_Toc52830452)

[9 Instaliranje sistema 25](#_Toc52830453)

[10 Zaključak 26](#_Toc52830454)

[11 Literatura 27](#_Toc52830455)

# Lista skraćenica

**Spisak skraćenica na engleskom jeziku**

**TCP** Transmmision Control Protocol – Transmisioni kontrolni protokol

**HDMI** High Definition Multimedia Interface – Multimedijski interfejs visoke rezolucije

**USB** Universal Serial Bus – Univerzalna serijska magistrala

**LAN** Local Area Network – Lokalna računarska mreža

**VCC** Voltage Common Collector – Zajednički kolektor napona

**GND** Ground – Masa (Uzemljenje)

**CSI** Camera Serial Interface – Serijski intefejs kamere

**BSD** Berkeley Software Distribution – Distribucija softvera *Berkeley*

**API** Application Programming Interface – Interfejs za programiranje aplikacija

**IP** Internet Protocol – Internet protokol

**UDP** User Datagram Protocol – Korisnički protokol datagrama

**XML** Extensible Markup Language – Proširivi jezik za označavanje

**GPIO** General Purpose Input Output – Ulaz/izlaz opšte namene

**GUI** Graphical User Interface – Grafički korisnički interfejs

# 1 Uvod

Od davnina postoji strah kod ljudi za zaštitu svoje imovine. U današnje vreme sama bezbednost je dobila na porastu zbog tehnološkog napretka, gde svako domaćinstvo olako može da priušti neki oblik zaštitnog sistema, kao što su kamere, alarmi itd. Jedan od primera zaštitnog sistema prikazan je na slici 1.

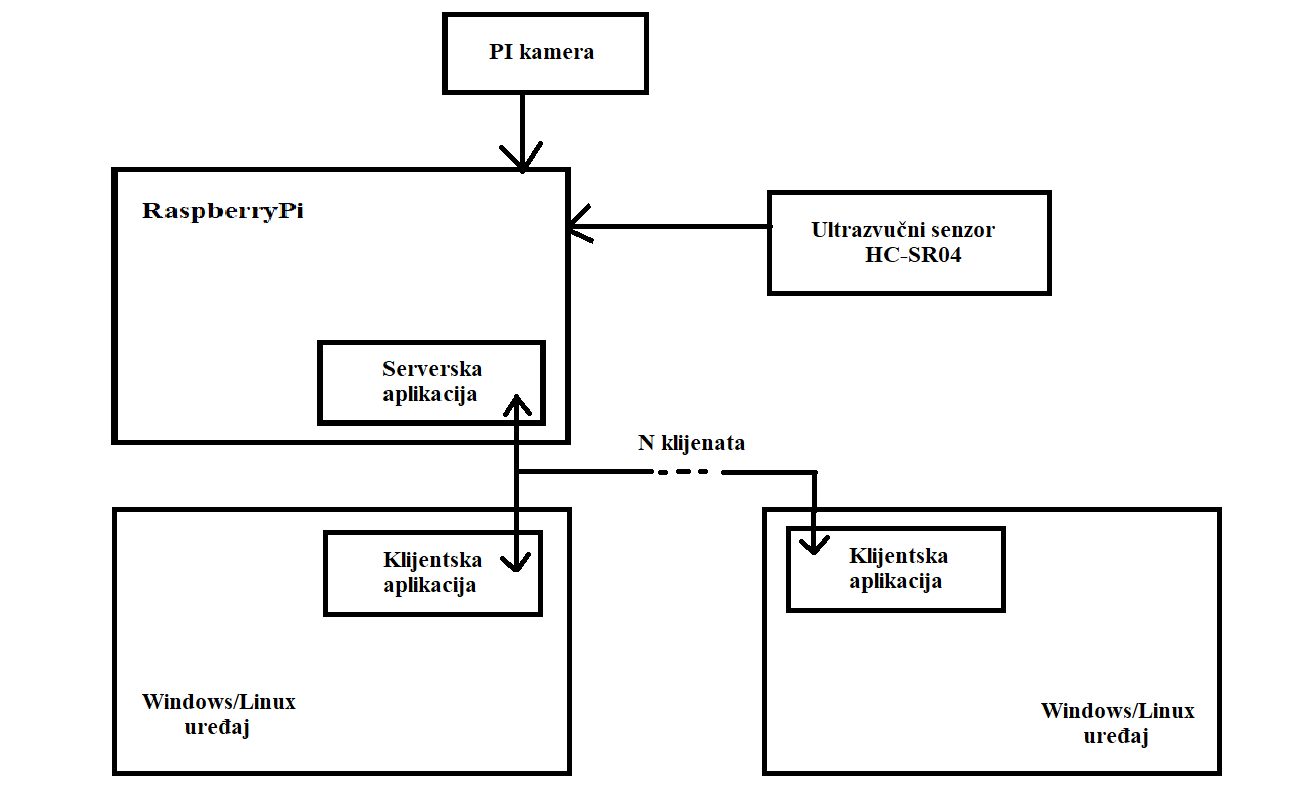
Tema ovog projekta vezana je za kreativan način implementiranja (instalacije) bezbednosnog uređaja. Glavnu ulogu u okviru ovog sistema igra prenosivi uređaj (*RaspberryPi*) koji uz pomoć senzora dolazi do zaključaka da li se u kući nalazi uljez, a potom izveštava korisničku (klijent) aplikaciju.



Slika . Bezbednosne kamere

# 2 Blok šema sistema

Kao što se može zaključiti iz uvoda sistem je podeljen u dve celine – serversku (nadzornu) i klijentsku (korisničku) stranu. Blok šema celokupnog sistema prikazana je na slici 2. Na serverskoj strani se nalaze *RaspberryPi* kao srce ove celine, zatim senzor udaljenosti (*HCSR04)* i kamera. Sa korisničke strane može se naći bilo kakav uređaj (računar) koji ima podržano izvršavanje *Python* aplikacije. Ove dve celine komuniciraju uz pomoć *TCP* soketa (eng. *Transmission Control Protocol socket),* koja uspešno funkcioniše na lokalu (lokalnoj mreži), informacije koje se šalju pakuju se u *Ethernet* paket (eng. *Frame*) i putuje ka odabranoj destinaciji. Komunikacija je bidirekciona, što znači da oba uređaja će biti kako prijemnik, tako i predajnik istovremeno.



Slika . Blok šema celokupnog sistema

# 3 Komponente i uređaj na serverskoj strani

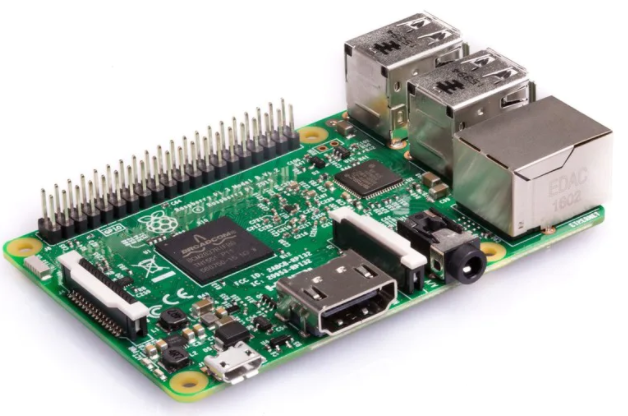
U ovom poglavlju su koncizno opisane hardverske zavisnosti serverskog bloka. Serverski uređaj zajedno čine mašina na kojoj se izvršava aplikacija, kao i senzori koji su povezani na nju.

## 3.1 Mašina

*RaspberryPi* je ekonomičan računar, veličine kreditne kartice. Na njega je moguće povezati monitor ili bilo koji drugi eksterni displej sa *HDMI (*eng. *High-Definition Multimedia Interface)* konektorom, kao i *USB* (eng. *Universal Serial Bus*) miš i tastaturu. Ovaj uređaj pokriva veliku većinu multimedijalnih mogućnosti kao što su “surfovanje” po internetu, reprodukcija video zapisa, obrada teksta i igranje nisko-grafičkih igara. Jedna od glavnih odlika ovog uređaja je prenosivost, mala veličina, pristupačna cena itd.

U projektu je korišćen *RaspberryPi 3 Model B*, čiji je fizički izgled prikazan na slici 3. Predstavlja treću generaciju ovog proizvoda. Koristi se za mnoge aplikacije i zamenjuje orginalni B+ model kao i *RaspberryPi 2 Model B*. Iako održava popularni izgled *single-board* ploče, ovaj model donosi procesor znatno boljih performansi, 10 puta brži od modela prve generacije. Pored toga, omogućene su i bežične *LAN* i *Bluetooth* veze.

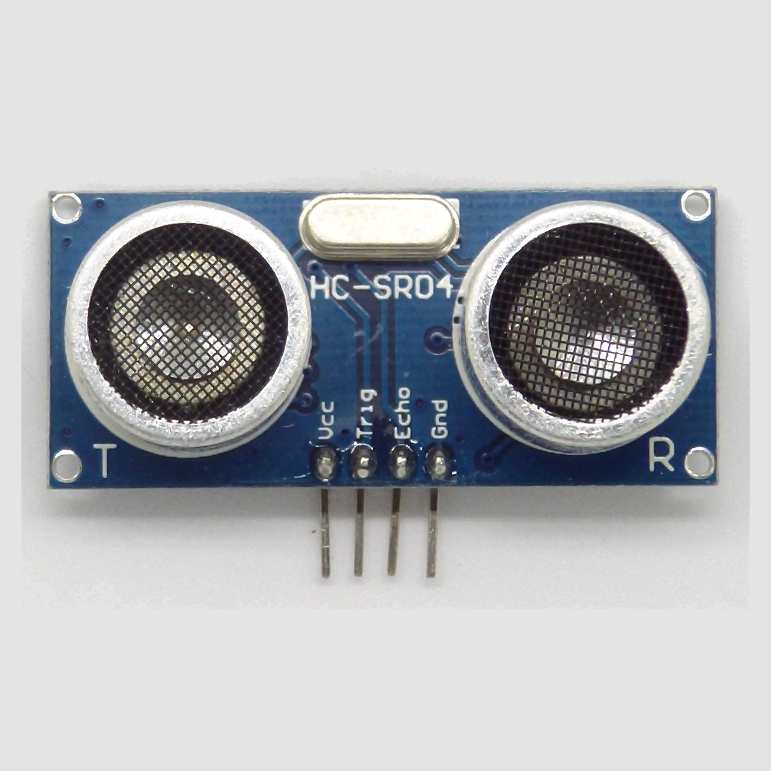
Zasnovan je na *Linux* platformi a poseduje i oficijalni *Raspbian* operativni sistem. Jedna od glavnih karakteristika mu je to što ima izvučene kontakte (pinove) na koje se mogu dovesti različiti senzori i aktuatori. U slučaju ovog projekta na te kontakte doveden je senzor udaljenosti, a putem *USB-a* ili posebnog ulaza je povezana kamera.



Slika . Fizički izgled uređaja RaspberryPi

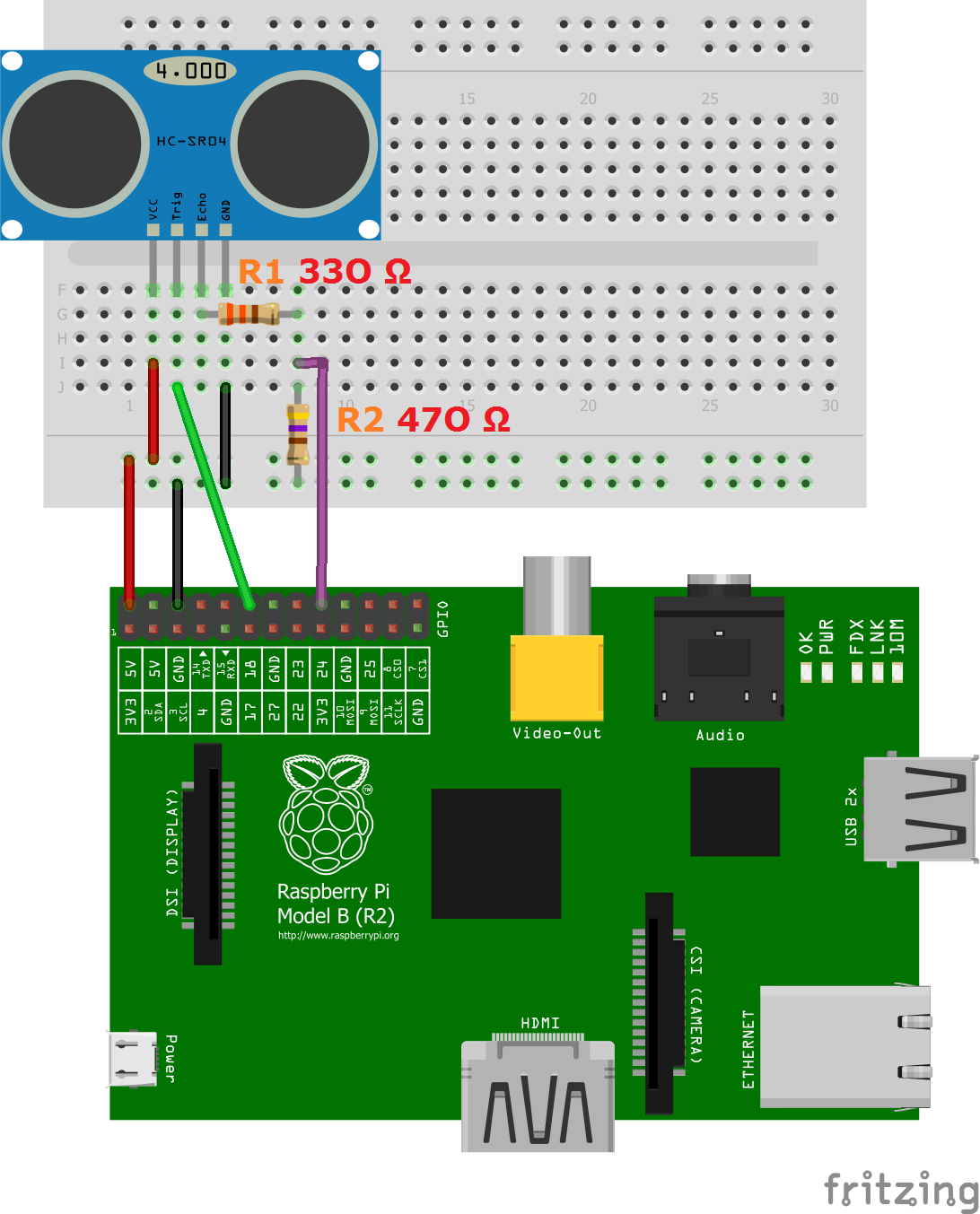
## 3.2 Senzor udaljenosti

U okviru ovog projekta ulogu senzora koji je zadužen za detekciju igra ultrazvučni senzor udaljenosti HC-SR04, čiji je fizički izgled prikazan na slici 4. U pitanju je ekonomični senzor beskontaktnog merenja i detektuje telo koje se nalazi naspram njega na udaljenostima u opsegu od 2-400 *cm*, sa tačnošću do 3 *mm*.



Slika . Fizički izgled senzora udaljenosti HC-SR04

Svaki HC-SR04 modul sadrži ultrazvučni predajnik, prijemnik i kontrolno kolo. Senzor poseduje 4 pina sa kojim se može upravljati senzorom, od toga 3 ulazna i jedan izlazni. Pored pinova zaduženih za napajnje (*VCC* i *GND*), senzor poseduje okidajući (eng. *Trigger*) pin kojim se pobuđuje tako što prima impuls od najmanje 10 *µs*. Pomoću odzivnog (eng. *Echo*) pina moguće je dobiti izmerenu udaljenost.

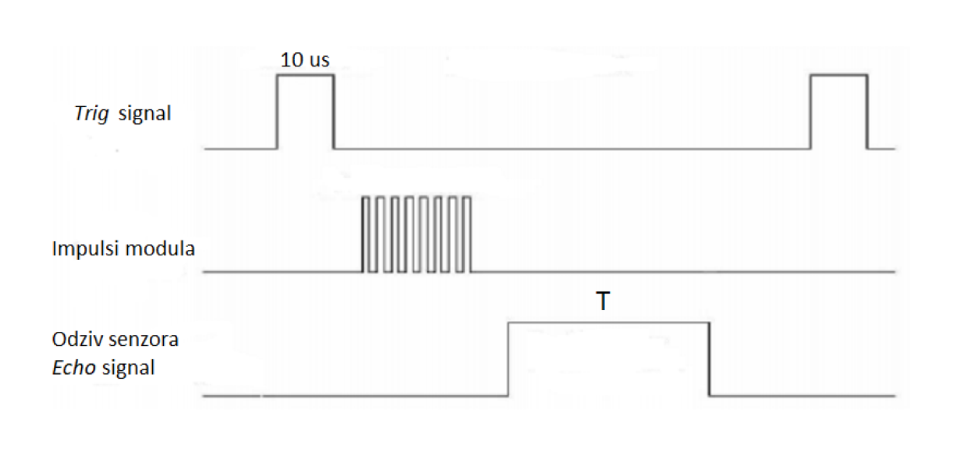


Slika . Način povezivanja ultrazvučnog senzora udaljenosti HC-SR04

Na slici 6 možemo videte princip rada senzora. Započinje tako što se senzor pobuđuje impulsom od 10 *µs* preko okidajućeg pina. Kao odgovor na to senzor emituje zvučni „rafal“ od 8 impulsa na 40 *kHz*. Ova povorka je interni signal koji predstavlja „ultrazvučni potpis“ (eng. *Ultrasonic signature*), jer ga prijemnik jednostavno identifikuje i razlikuje od potencijalnih ambijentalnih šumova. Kada se impulsi reflektuju nazad, odnosno senzor je detektovao prepreku, odzivni signal biće na visokoj vrednosti vremensku periodu koji je ekvivalentan „putovanju“ signala ka prepreci i nazad (1).

(1)

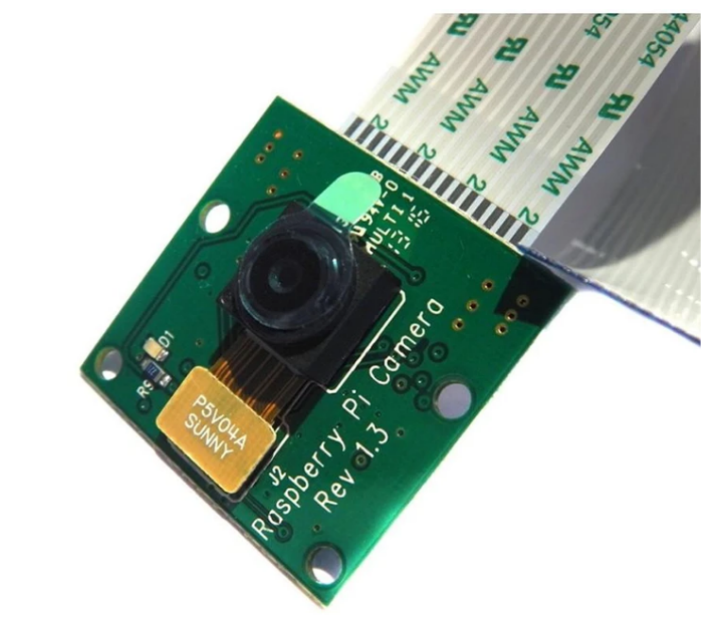
Gde je *D* – Udaljenost (eng. *Distance*), *Vs* – Brzina zvuka (eng. *Velocity of Sound*), *T* – Proteklo vreme (eng. *Time*)



Slika . Signali za upravljanje i očitavanje podataka sa ultrazvučnog senzora

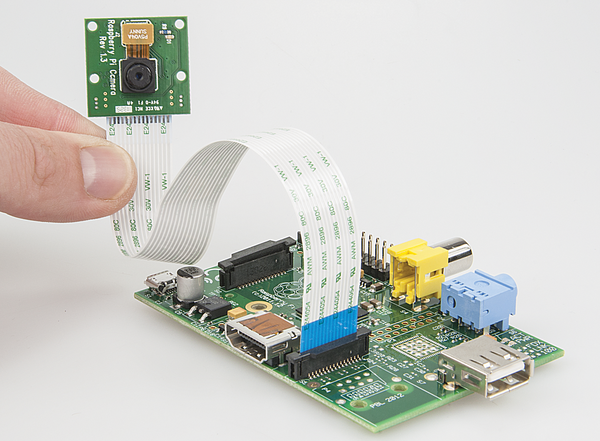
## 3.3 Kamera

Ulogu kamere u slučaju ovog uređaja igra *RaspberryPi Camera Module*. Na slici 7 prikazan je njen fizički izgled. U pitanju je specijalno dizajnirani dodatak za *RaspberryPi* od 5 megapiksela, koji sadrži sočiva sa fiksnim fokusom. Senzor pruža statičke slike od 2592x1944 *pix*, a takođe podržava video zapise 1080p30, 720p60 i 640x480p60/90.



Slika . Fizički izgled RaspberryPi kamere

Kamera se priključuje na *RaspberryPi* pomoću jedne od malih utičnica na gornjoj površini ploče i koristi namenski *CSI* interfejs, dizajniran posebno za povezivanje sa kamerama. Način povezivanja kamere prikazan je na slici 8.



Slika . Način povezivanja kamere sa prenosivim uređajem

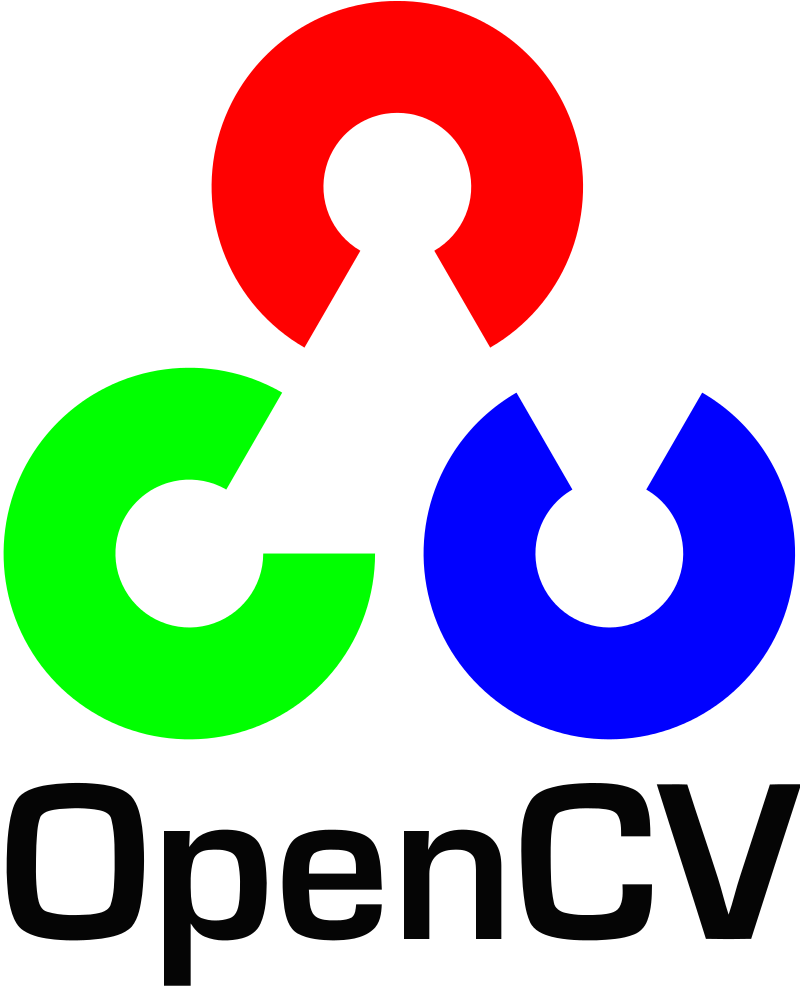
# 4 Alat za detekciju čovekovog lica – OpenCV

*OpenCV* (*Open Source Computer Vision Library*) je softverska biblioteka otvorenog koda zadužena za računarsku viziju i mašinsko učenje. Logo date biblioteke prikazan je na slici 9. *OpenCV* je napravljen da omogući zajedničku infrastrukturu za aplikacije na kojima se izvršava neka vrsta računarske vizije i da ubrza primenu mašinske percepcije u komercijalnim proizvodima. Pošto je *BSD*-licenciran proizvod, *OpenCV* čini poslovanje olakšanim, a kod lako modifikovanim i primenjivim.

Biblioteka poseduje više od 2500 optimizovanih algoritama, što uključuje sveobuhvatan skup kako klasičnih tako i najsavremenijih algoritama za računarsku viziju i mašinsko učenje. Ovi algoritmi mogu biti korišćeni za detekciju i prepoznavanje lica, identifikaciju objekata, klasifikaciju čovekovih pokreta u video zapisima, praćenje pokreta kamere, ka i kretanje objekata, ekstrakciju 3D modela od objekata itd. *OpenCV* poseduje zajednicu od 47 hiljada korisnika i procenjeni broj preuzimanja biblioteke na neverovatnih 18 miliona. Biblioteka se intenzivno koristi u kompanijama, istraživačkim grupama i od strane državnih organa.

Biblioteka je prvobitno pisana na *C++* programskom jeziku od strane *Intel*-ovog istraživačkog centra u Nižnjem Novgorodu, u Rusiji. Naknadno je napravljena adaptacija za *Python* i *Java* programske jezike, ali i za *MATLAB* interfejs. Podržana je na svim poznatim platformama - *Linux*, *Windows*, *Android* i *MacOS*.

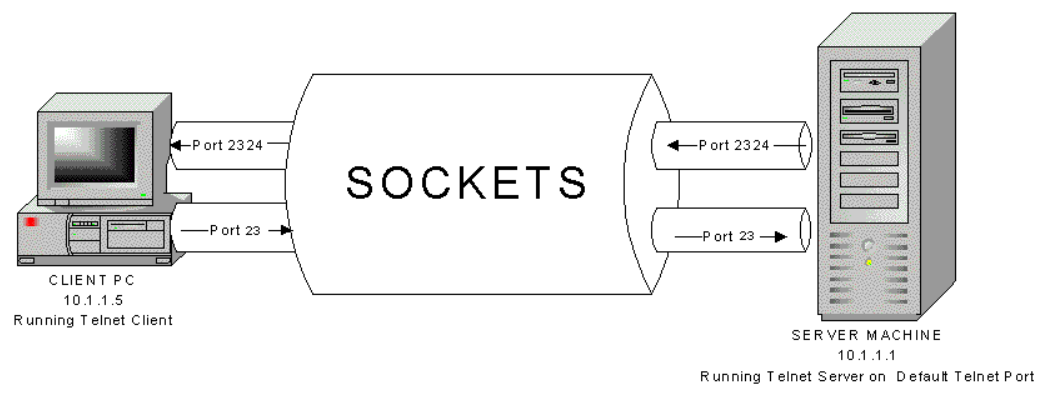
U okviru ovog projekta ova biblioteka je korišćena za povlačenje slika sa drajvera kamere, te za njihovu obradu radi detekcije čovekovog lika i za njihovu ekranizaciju, ali i za kreiranje video zapisa.



Slika . Logo biblioteke OpenCV

# 5 TCP/IP soket i mrežno programiranje

U oblasti računarskih komunikacija termin soket (eng. *Socket*) ili mrežni soket je krajnja tačka dvosmernog međuprocesnog komunikacionog toka preko računarske mreže bazirane na internet protokolu, kao što je *Ethernet*. Mrežni soketi predstavljaju aplikacioni programski interfejs (*API*) koji je obično podržan od strane operativnog sistema. Oni grade mehanizam za dostavljanje paketa podataka (frejmova) odgovarajućem aplikacionom procesu ili niti, na osnovu kombinacija lokalnih i udaljenih *IP* adresa i portova. Operativni sistem preslikava svaki soket u proces ili nit aplikacije koja komunicira sa udaljenim računarom. Soket adresa je spoj *IP* adrese (lokacije računara) i porta (koji se preslikava u proces ili nit) u jedinstveni identitet. Na slici 10 prikazan je primer soketa kao medijuma između klijenta i servera.



Slika . Ilustrativni primer soketa kao medijuma između klijenta i servera

Internet soket je određen jedinstvenom kombinacijom:

* Protkola (*TCP*, *UDP* ili sirovi *IP*). Zbog ovoga *TCP* port jedne vrednosti ne predstavlja isti soket kao *UDP* port iste te vrednosti.
* Lokalne adrese soketa (lokalna *IP* adresa i broj porta)
* Udaljene adrese soketa. Ova adresa odnosi se samo na uspostavljanje *TCP* soketa. Ovo je potrebno jer *TCP* server može da opslužuje više klijenata u isto vreme. Server pravi po jedan soket za svakog klijenta, dok soketi dele istu lokalnu adresu soketa.

Operativni sistem prosleđuje dolazne *IP* pakete odgovarajućem aplikacionom ili servisnom procesu nakon što iz zaglavlja *IP* i transportnog protokola izvuče podatak o adresi soketa. Unutar aplikacije koja je napravila soket, soket se referiše jedinstvenim celim brojem koji se naziva identifikator soketa ili broj soketa.

U okviru ovog projekta korišćen je *TCP* soket zato što:

1. Pouzdan je – paketi ispušteni u mreži se detektuju na osnovu potvrde (eng. *Acknowledgment*) i pošiljalac ih ponovo prenosi ukoliko isporuka nije uspela.
2. Ima isporuku podataka po redosledu – aplikacija koja dobija podatke ih čita onim redosledom kojim ju je napisao pošiljalac.

Zbog ovih osobina *TCP* *Ethernet* interfejsa i komuniciranja između soketa protok podataka može da izgubi na brzini. U situaciji gde je brzina prenosa od presudnog značaja, a ne sam kvalitet, odnosno podudarnost poslatog paketa, kao i pouzdanost da li će poruka uopšte stići, više se koristi *UDP* protokol. Kao primer primene *TCP*-a može se uzeti slanje elektronske pošte (eng. *e-mail*), gde je od krunskog značaja da sadržaj bude baš onakav kakvog ga je pošiljalac napisao, kao i da zagarantovano stigne. Sa druge strane, primer *UDP*-a bio bi video razgovor.

# Kreiranje instance serverskog soketa pomoću konstruktora bez parametara

serversock = socket.socket()

# Postavljanje postavki soketa

# Prvi parametar SOL\_SOCKET naglašava da je u pitanju nezavisan protokol

# Drugi parametar SO\_REUSEADDR omogućava da se soket poveže na port koji

# neki drugi soket koristi na silu

# Treći parametar je vrednost koja se prosleđuje specifičnoj soket opciji

serversock.setsockopt(socket.SOL\_SOCKET, socket.SO\_REUSEADDR, 1)

# Veži se na prosleđenu IP adresu i port kao server

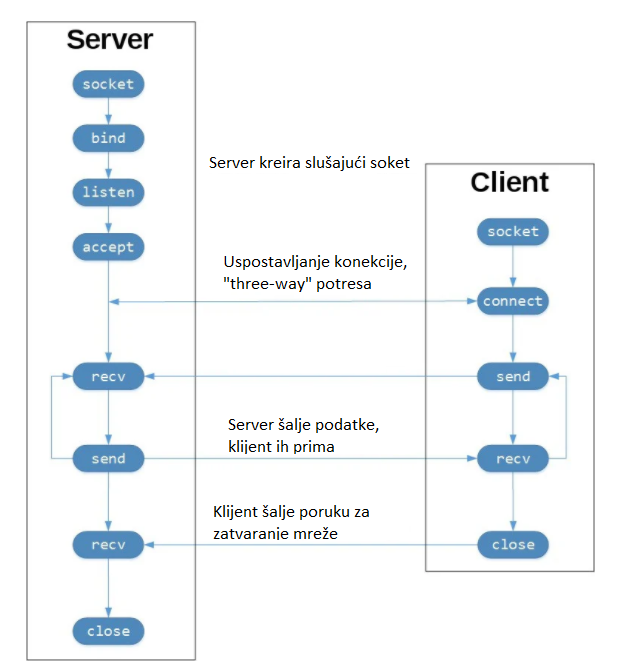
serversock.bind((host,port))

# Započni slušanje zahteva klijenata za konekciju

serversock.listen()

*Deo koda u okviru kojeg se izvršava inicijalizacija serverskog soketa*

*Python* soket modul pruža interfejs za *Berkeley Sockets API*. Da bi se soket mogao koristiti neophodno je pri soket instanciranju (instanciranje konstruktora sa parametrima) proslediti dva parametra ili ih naknadno proslediti uz pomoć postavljačke (eng. *Set*) metode *setsockopt* (slučaj instanciranja praznog konstruktora). U pitanju su statičke konstante iz soket klase čija je uloga da definišu način primene instance. Potom je neophodno proslediti serversku *IP* adresu i komunikacioni port. Ukoliko je adresa “živa”, tj. aktivna i port slobodan aplikacija nastavlja dalje bez izuzetka (eng. *Exception*). U slučaju da soket igra ulogu servera onda se uređeni par (adresa, port) prosleđuju na *bind* metodu, dok za klijente je neophodno izvršiti povezivanje (*connect* metoda). Nakon toga razmena podataka je moguća tako što soket podatke šalje preko *send* metode, a prijem preko *recv* metode. Princip funkcionisanja komunikacije između soketa prikazan je na slici 11.



Slika . Princip funkcionisanja komunikacije između soketa

# 6 Klasifikatori i njihova upotreba

Kaskadni klasifikatori, odnosno kaskada pojačanih klasifikatora koji rade sa *HAAR*-sličnim karakteristikama, je specifični slučaj mašinskog učenja preko ansambl metode, koji se zove pojačavanje (eng. *Boosting*). Obično se oslanja na *Adaboost* klasifikatore (i druge modele kao što su *Real Adaboost*, *Gentle Adaboost* itd.).

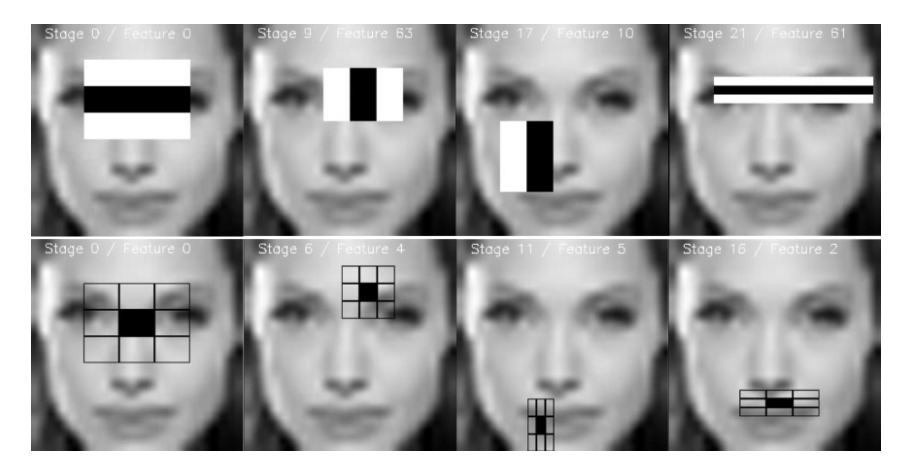
Kaskadni klasifikatori se treniraju na nekoliko hiljada uzoraka slika koje sadrže objekat koji je potrebno detektovati. Takvi uzorci nazivaju se pozitivi. Takođe, pri trening procesu mora biti sadržano znatno više uzoraka slika na kojima se ciljani objekat ne nalazi, a ti uzorci se nazivaju negativi.

U slučaju detekcije lica, da bi klasifikator uspešno izvršio detekciju neophodno je da koristi alogirtam koji se zove *Viola-Jones* objektno-detekcioni okvir (eng. *Framework*), koji uključuje sledeće korake neophodne za detekciju lica uživo:

1. Odabir HAAR-karakteristika, izvedenih iz HAAR talasa
2. Kreiranje integralne slike
3. *Adaboost* treniranje
4. Kaskadno klasifikovanje

## 6.1 Odabir HAAR-karakteristika

Svako ljudsko lice ima zajedničke osobine, kao što su tamno (zasenčeno) područje oka u poređenju sa gornjim delovima obraza (izloženi svetlosti), svetao region na nosu u poređenju sa očima itd. Ove karakteristike nazivaju se HAAR karakteristikama (eng. *Haar Features*).

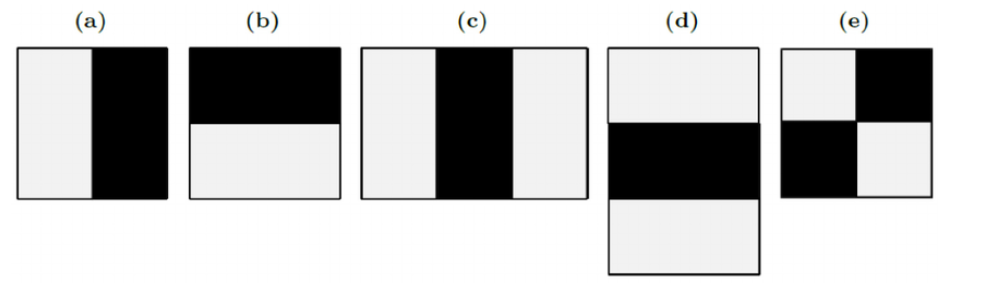


Slika . HAAR karakteristike koje klasifikator traži tokom treniranja

Na slici 12 možemo videte proces izdvajanja karakteristika. Prva karakteristika meri razliku u intezitetu između regiona očiju i regiona gornjeg dela obraza. Vrednost karakteristike se računa razlikom sume piksela u crnoj oblasti i sume piksela u beloj oblasti (2).

(2)

Na slici 13 je prikazano nekoliko vrsta pravougaonika koji se mogu primeniti za izdvajanje karaketristika.



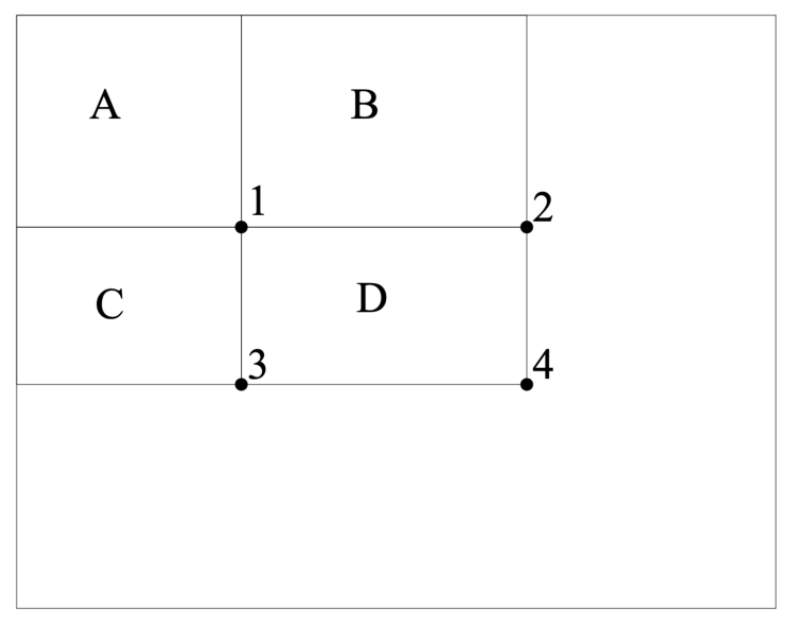
Slika . Tipovi HAAR pravougaonika

1. Karakteristika 2 pravougaonika je razlika između zbira piksela unutar dva pravougaona područja i koristi se za otkrivanje ivica (a, b)
2. Karakteristika 3 pravougaonika izračunava zbir unutar dva spoljna pravouganika oduzeta od zbira u središnjem, najčešće se koristi za otkrivanje lica (c, d)
3. Karakteristika 4 pravougaonika izračunava razliku između dijagonalnih parova (e)

## 6.2 Kreiranje integralne slike i *Adaboost* treniranje

Kada su karakteristike odabrane, primenjujemo ih na skup slika za treniranje koristeći *Adaboost* klasifikaciju.

Računanje karakteristika pravougaonika u konvolucijskom kernel stilu može biti veoma dugo, pa je kao rešenje predložen posredni prikaz slike, odnosno integralna slika. Uloga integralne slike je da omogući bilo koji pravougaoni zbir na jednostavan način koristeći samo četiri vrednosti. Ukoliko bismo želeli da odredimo karakteristike pravougaonika u datom pikselu, integralna slika bi bila suma piksela iznad i levo od datog piksela. Tako na primer, želimo da izračunamo sumu piksela u regionu D. Kao što se može videti sa slike 14 pored regiona D imamo i regione A, B i C.



Slika . Ilustrativni primer dobijanja integralne slike

* Vrednost integralne slike u tački 1 je zbir piksela u pravougaoniku A
* Vrednost u tački 2 je zbir regiona A i B
* Vrednost u tački 3 je zbir regiona A i C
* Vrednost u tački 4 je zbir regiona A, B, C i D

Na osnovu datih podataka suma piksela u regionu D se može jednostavno izračunati:

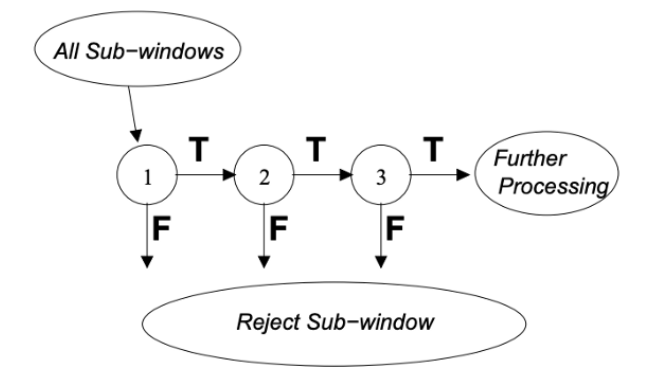
Tokom jednog prolaza izračunali smo vrednost unutar pravougaonika koristeći samo četiri reference niza.

## 6.3 Kaskadno klasifikovanje

Kaskadni klasifikatori se koriste za odbacivanje podprozora koji ne sadrže lica dok se identifikuju regioni koji to čine. Kako bi što pravilnije identifikovali lice želimo da minimiziramo stopu lažnih negativa. Niz klasifikatora primenjuje se na svaki podprozor i predstavljaju jednostavno stablo odlučivanja:

* Ako je prvi klasifikator pozitivan prelazimo na drugi
* Ako je drugi klasifikator pozitivan prelazimo na treći itd.

Bilo koji negativan rezultat u nekom trenutku dovodi do odbacivanja podprozora koji potencijalno sadrži lice. Početni klasifikator eliminiše većinu negativnih primera po niskim računskim naporima, dok naredni eliminišu dodatne negativne primere ali zahtevaju više računskih napora. Princip klasifikacije podprozora prikazan je na slici 15.



Slika . Ilustrativni primer klasifikacije preko stabla

U okviru ovog projekta korišćen je gotovi *OpenCV* kaskadni klasifikator za prepoznavanje čovekovog lika. Klasifikatori su zapakovani u *XML* formatu, te se lako mogu prenositi i deskriptovati na različitim okruženjima. U okviru aplikacije se koriste preko *OpenCV* instance objekta *CascadeClassifier* tako što se prosleđuje putanja fajla. Nakon toga, proces detekcije je takav da se objektu prosleđuje slika zabeležena preko kamere, te ukoliko je detektovano jedno ili više lica vraća se niz regiona od interesa, odnosno regiona u kojem se detektovani objekat (čovekov lik) nalazi, u suprotnom povratni niz će biti prazan (*None* u *Python* programskom jeziku).

# Uvoz OpenCV klasa i funkcija

import cv2

# Instanciranje klasifikatora koji je zadužen za detekciju lica

# prosleđuje mu se relativna putanja (u ovom slučaju je fajl u root putanji aplikacije

# ili apsolutna putanja u sistemu uključujući i naziv fajla sa ekstenzijom

# Prosleđena putanja pripada OpenCV kaskadnom klasifikatoru za detekciju lica

face\_cascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

# WebCam handler

# Zbog postojanja mogućnosti da serverska aplikacija

# bez rada ultrasonic senzora može raditi i na drugim platformama

# Windows instanca ove klase zahteva parameter više te se taj problem otklanja ifom

if platform == "linux" or platform == "linux2":

    cap = cv2.VideoCapture(0)

elif platform == "win32":

    cap = cv2.VideoCapture(0, cv2.CAP\_DSHOW)

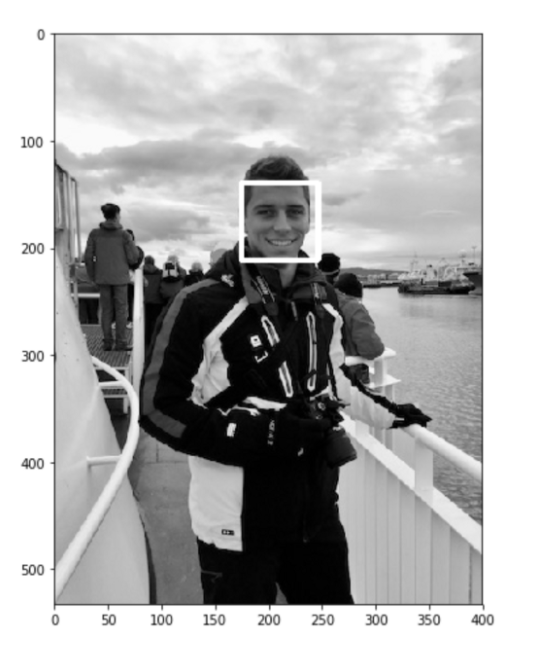
# Podešavanje širine i visine frejma preuzetog sa kamere

cap.set(3, 320)

cap.set(4, 240)

*Deo koda u okviru kojeg se izvršava inicijalizacija kaskadnog klasifikatora*

*OpenCV* klasifikator koji se koristi je poprilično brz, što znači da u većini slučajeva neće praviti dodatna kašnjenja ili ta kašnjenja neće biti od velikog značaja. Ova osobina ne mora da bude generalna za ostale klasifikatore, te je neophodno biti oprezan pri treniranju sopstvenog klasifikatora. Sam proces treniranje može da traje dugo, a rezultati ne moraju da budu na očekivanom nivou. Detekcija čovekovog lika uz pomoć kaskadnog klasifikatora prikazana je na slici 16.

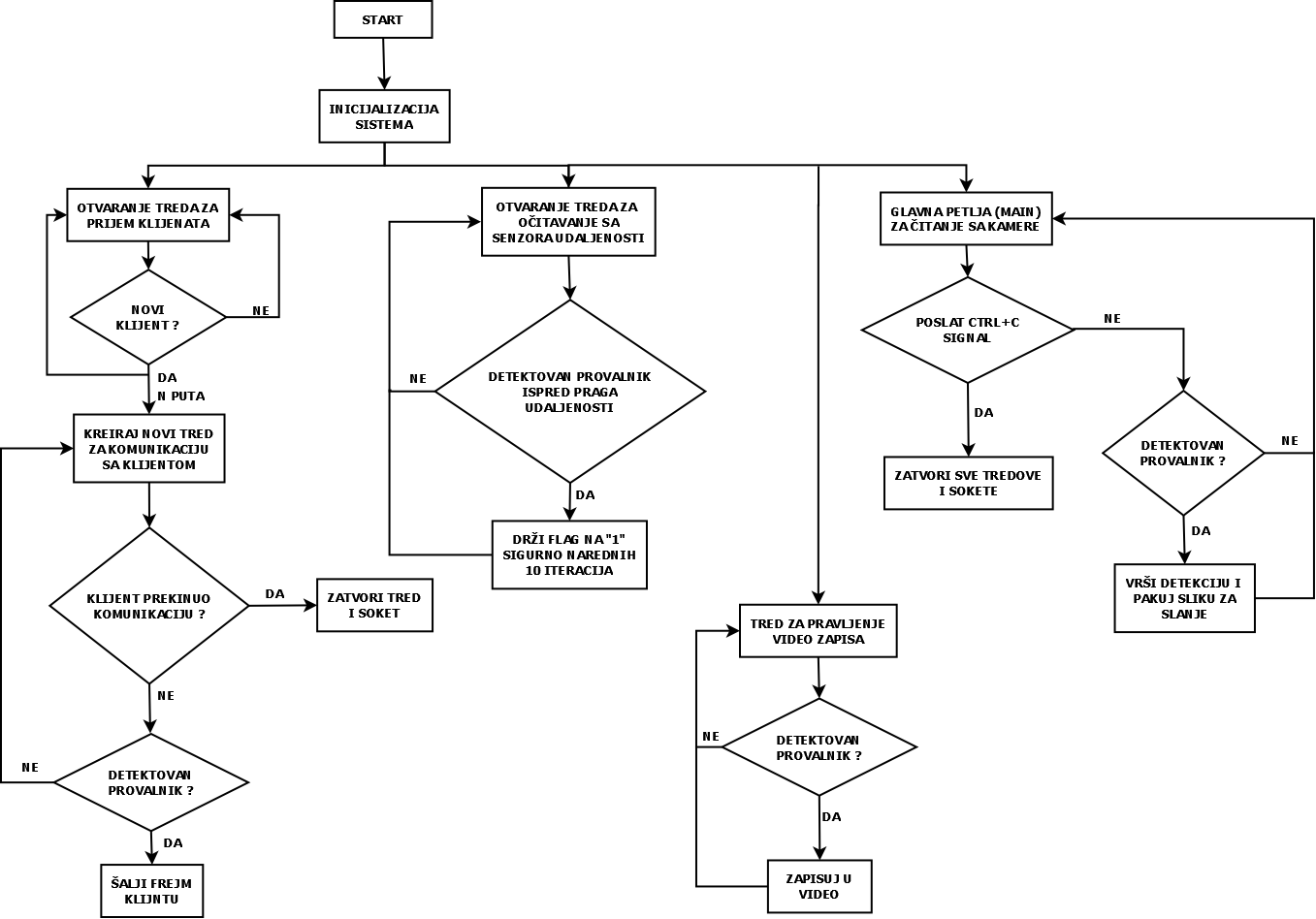


Slika . Primer detekcije čovekovog lika uz pomoć kaskadnog klasifikatora

# 7 Algoritam rada serverske aplikacije

U ovom poglavlju opisan je algoritam rada serverske aplikacije, čiji je blok dijagram prikazan na slici 17. Serverska aplikacija se izvršava u *RaspberryPi* uređaju. Zadatak serverske aplikacije je da radi očitavanja sa kamere i senzora udaljenosti, u paraleli sa tim da zaprima nove klijente, te aktivinim klijentima (klijentima koji su trenutno povezani) šalje slike (frejmove) sa kamere ukoliko je senzor udaljenosti detektovao prepreku, takođe izvršava i detekciju lica na slikama u tom slučaju. Pored toga, serverska aplikacija prilikom svake nove iteracije prenošenja (eng. *Streaming*) slika sa kamere kreira novi video zapis i skladišti ga u uređaju. Početna tačka (eng. *Start*) predstavlja inicijalizaciju neophodnih zavisnosti u koje spada:

* *OpenCV* instance zadužene za rukovođenje kamerom povezanom sa uređajem
* Postavljanje ulaznih i izlaznih uređaja (*GPIO* biblioteke *RaspberryPi*-a)
* Utičnice (eng. *Socket*) zadužene za komuniciranje putem *TCP* protokola



Slika . Blok dijagram algoritma rada serverske aplikacije

Naime, ukoliko nije prosleđen argument, odnosno *IP* adresa koja je dodeljena uređaju iz lokalne mreže, prosleđuje se uobičajena (eng. *Default*) vrednost koju aplikacija izvlači iz *Socket* metode *gethostname*. Ukoliko je soket dobio adresu i port na kojima može da otpočne komunikaciju sa potencijalnim klijentima nastupa paralelno izvršavanje nekoliko beskonačnih petlji primenom principa više niti (eng. *Multithreading*).

# Uvoz GPIO biblioteke kojom se upravlja pinovima RaspberryPi uređaja

import RPi.GPIO as GPIO

# Podešavanje GPIO moda enumeracije pinova (BOARD / BCM)

GPIO.setmode(GPIO.BCM)

# Postavljanje GPIO pinova

GPIO\_TRIGGER = 18

GPIO\_ECHO = 24

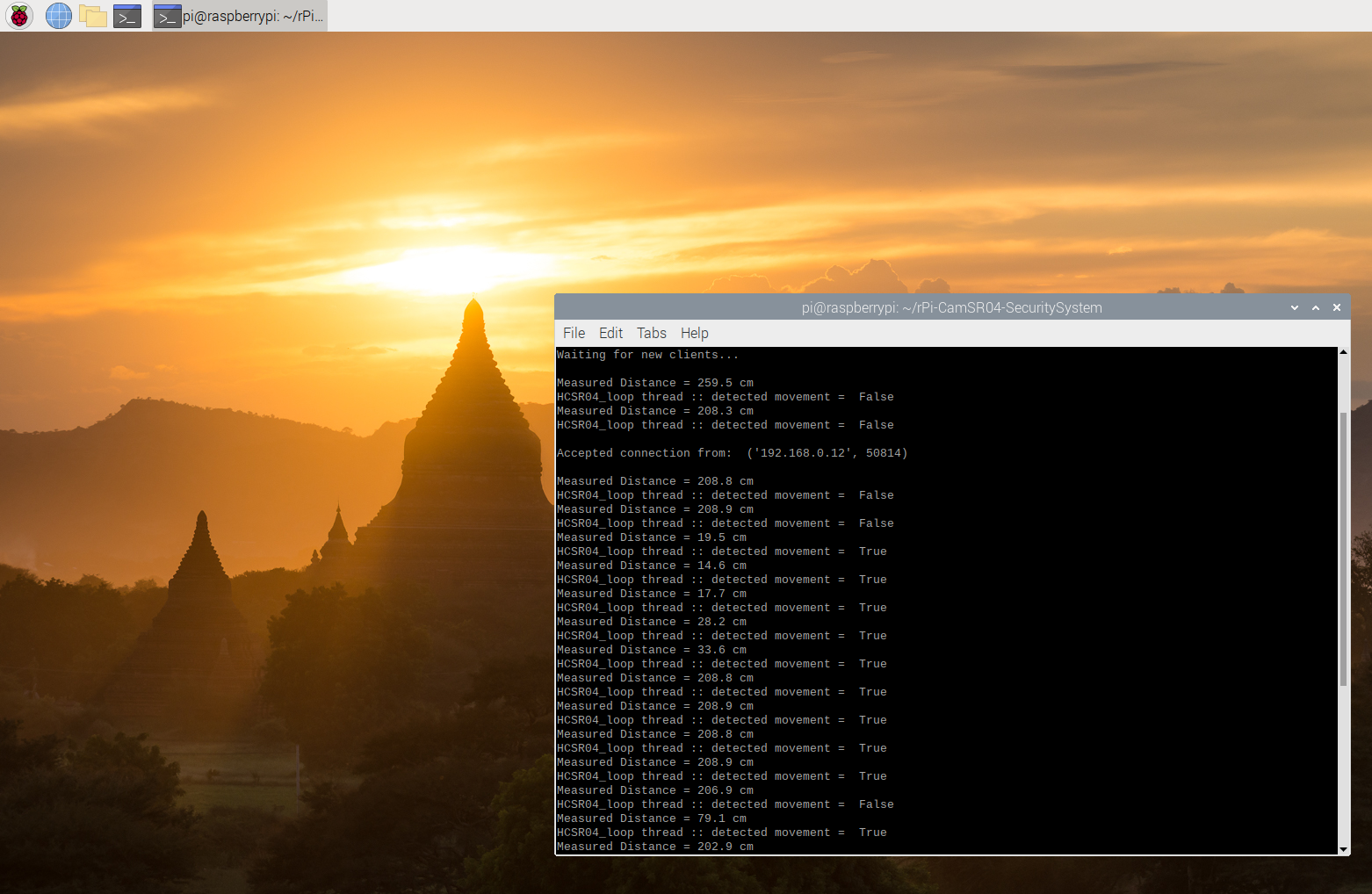
# Postavljanje direkcije pinova (ulazni / izlazni)

GPIO.setup(GPIO\_TRIGGER, GPIO.OUT)

GPIO.setup(GPIO\_ECHO, GPIO.IN)

*Deo koda u okviru kojeg se izvršava inicijalizacija ulaznih i izlaznih pinova RaspberryPi uređaja*

U okviru jednog treda izvršava se očitavanje sa ultrasoničnog senzora, gde se šalje neophodan okidajući signal, radi proračun i ispisivanje izmerene udaljenosti prepreke od senzora u periodičnoj vremenskoj iteraciji od 500 *ms*. Ukoliko očitana vrednost sa senzora prelazi ispod postavljenog praga (80 *cm*) postavlja se vrednost globalne promenljive (eng. *Flag*) na logičku visoku vrednost (‘1’ – eng. *True*) i kao takav *flag* zagarantovano ostaje u nekoliko iteracija. *Printscreen* ispisa detekcije udaljenosti i prijema novog klijenta prikazan je na slici 18.



Slika . Printscreen ispisa detekcije udaljenosti i prijema novog klijenta

Ovaj *flag* aktivira rad senzora kamere u glavnom tredu i izvršavanje detekcije čovekovog lika, te pakovanje slike u binarni niz koji se šalje ka aktivnim klijentima, ukoliko ih u datom trenutku ima. Pre svega, radi uštede procesorskog vremena i boljih performansi aplikacije, ova petlja je pasivna sve do trenutka detekcije provalnika uz pomoć senzora udaljenosti iz gore navedene petlje.

while True:

    try:

# Metoda read je primer Python funkcije koja vraća više promenljivih

# Promenljiva ret je Bool tipa i odgovara na pitanje da li je prosleđen frejm

# Promenljiva frame predstavlja sliku preuzetu sa kamere

        ret, frame = cap.read()

# Za slučaj da nije prosleđena slika izađi iz petlje

        if(ret == False):

            break

# Promenljiva detected je globalna promenljiva Bool tipa koja se na True

# vrednost postavlja ukoliko je HCSR04 detektovao pokret

        if detected == True:

# Konvertuj boju preuzete slike u sivu i prosledi gray promenljivoj

            gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# Izvrši detekciju lica za prosleđenu sliku i

# vrati detektovane regione u promenljivu faces

            faces = face\_cascade.detectMultiScale(gray,

                scaleFactor=1.3,

                minNeighbors=5)

# Iteriraj kroz sve detektovane regione, a njegove parametre

# mi dodeli lokalnim promenljivama x, y, w i h

# x, y – koordinate nulte tačka (piksela) - gornji levi ugao

# w, h – širina i visina regiona od interesa

            for(x, y, w, h) in faces:

# Nacrtaj crveni pravougaonik od jedne do druge tačke

# unete boje (crvena po BGR sistemu) i unete širine

                cv2.rectangle(frame, (x,y), (x+w, y+h), (0, 0, 255), 2)

# Dozvoli tredovima zaduženim za slanje paketa i upisivanje video frejma

# da iskoriste ovaj frejm

            sndMsg = True

writeVideo = True

# Enkodovanje paketa u sliku jpg format i pakovanje u binarni niz

            result, framePacked = cv2.imencode('.jpg', frame, encode\_param)

            data = pickle.dumps(framePacked, 0)

# U slučaju pritiska tastera Ctrl + C na tastaturi izađi iz glavne petlje

    except KeyboardInterrupt:

        break

*Deo koda u okviru kojeg se izvršava očitavanje sa senzora kamere i detekcija lica*

Pored ovog treda, ovaj *flag* pobuđuje slanje zapakovane slike na klijentskim tredovima čiji je broj ekvivalentan broju otvorenih klijentskih soketa. Kao i glavni tred, ovi tredovi su takođe pasivni sve do trenutka detekcije (postavljanja promenljive detekcije na logičku visoku vrednost). Pasivnost klijentskih tredova omogućena je zaključavanjem tredova uz pomoć *mutex* i *lock* instanci. Kada dobije signalizaciju da može podatke da pošalje ka klijentu, klijentski soket koristi neblokirajuću funkciju za slanje. Ukoliko bi došlo do diskonekcije sa klijentom, a koju bi prouzrokovala klijentska strana, unutar beskonačne petlje primenjuje se hvatanje izuzetka (eng. *Exception*) za bezbedan izlazak iz petlje i uspešno zatvaranje treda.

Da bi komunikacija sa klijentima bila moguća, tredovi zaduženi za interakciju sa klijentima se inicijalizuju u još jednom tredu u kojem se “dočekuju” novi klijenti. Naime, u ovom tredu nalazi se blokirajuća funkcija prihvatanja novog zahteva za konekciju od strane drugih uređaja. U tom trenutku funkcija zadužena za prihvatanje (eng. *Accept*) kao povratnu vrednost vraća uređeni par koji čine *IP* adresa uređaja i brojčana vrednostotvorenog soketa.

def clientReceivement():

    print ("\nWaiting for new clients...\n")

    while True:

        try:

# Blokirajuća funkcija koja prihvata nove klijente

# Ukoliko se poveže novi klijent vraćaju se dve promenljive

# Promenljiva client je jedinstvena identifikacija klijentskog soketa

# Promenljiva address je IP adresa klijentskog uređaja

           (client, address) = serversock.accept()

# Ukoliko dođe do neke sistemske greške izađi iz petlje

        except OSError:

            break

# Dodaj listi tredova th novu Thread instancu u kojoj će se izvršavati

# funkcija listener (za komunikaciju sa klijentskim soketom)

# argumenti funkcije su preuzete promenljive

        th.append(Thread(target=listener, args = (client,address)) )

# Započni rad poslednjeg dodatog treda od malopre

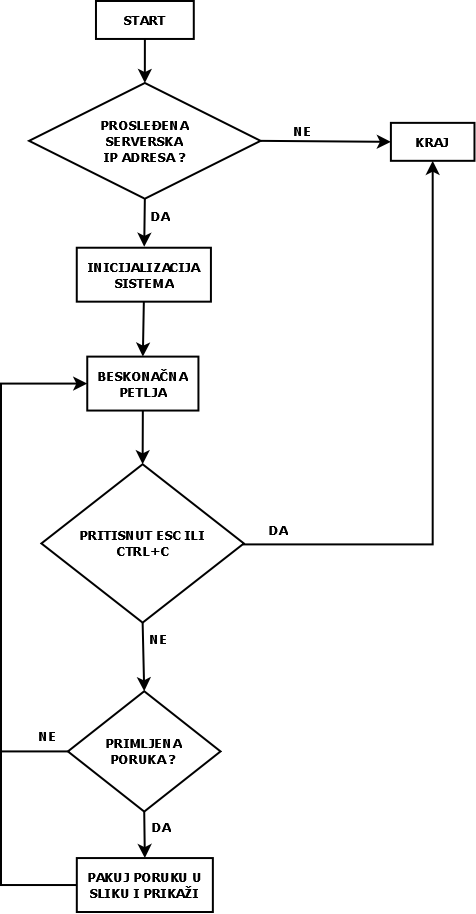
        th[-1].start()

*Deo koda u okviru kojeg se izvršava prijem novih klijenata*

Svaka nova detekcija provalnika zapisuje se u video zapisu koji nosi naziv trenutka njegovog kreiranja (eng. *Timestamp*), a sam video je u *MJPEG* (eng. *Motion JPEG*)formatu i skladišti se u eksternoj memoriji uređaja *SD* kartici. Administrator može u bilo kom trenutku prekinuti rad aplikacije “ubijajućim” signalom koji se, između ostalog, može napraviti pritiskom tastera *Ctrl* + *C* na tastaturi. U tom trenutku se izvršava bezbedan izlazak iz aplikacije, što znači da se svi objekti deinicijalizuju, a tredovi zatvore. Da bi se “ubila” prihvatajuća funkcija, šalje se signal za prekid osluškivanja (eng. *Shutdown*) ka glavnoj serverskoj soket instanci, ali i ka ostalim aktivnim klijentima, te se izvršava prekid (eng. *Break*) beskonačnih *While* petlji.

# 8 Algoritam rada klijentske aplikacije

Klijentska aplikacija, kako joj samo ime kaže, zadužena je da se poveže na privatni server i omogući korisniku da isprati rad servera, ukoliko je došlo do provale. Ova aplikacija se može izvršavati na bilo kojoj mašini koja radi na *Linux* ili *Windows* platformi. Na osnovu ovih uloga može se zaključiti da je klijentska aplikacija znatno jednostavnija od serverske. Blok dijagram algoritma rada klijentske aplikacije prikazan je na slici 19.



Slika . Blok dijagram algoritma rada klijentske aplikacije

U slučaju ove aplikacije, startna tačka isključuje inicijalizaciju senzora, odnosno *GPIO* pinova, te pod nju spada inicijalizacija klijentskog soketa i zavinosti parsiranja argumenata koji se prosleđuju pri pokretanju aplikacije. U slučaju klijentske aplikacije obavezno je proslediti *IP* adresu serverskog uređaja i komunikacionog porta sa kojeg se komunikacija očekuje, ukoliko se ne prosledi pravilna *IP* adresa dolazi do izlazka iz aplikacije usled njenog “pucanja” zbog bačenog izuzetka (eng. *Exception*). Nakon što je klijentski soket uspešno povezan sa serverom započinje komunikacija.

# Instanciranje Argument Parsera koji je zadužen za parsiranje prosleđenih

# argumenata pri pokretanju aplikacije na terminalu

parser = argparse.ArgumentParser(description='set the IP address.')

# Dodaj argument –IP kao mogući argument za prosleđivanje i dodeli mu string tip

parser.add\_argument('--IP',

    type=str,

    help='set the IP address of the rPi (server device)')

# Pokreni parsiranje prosleđenih argumenata

args = parser.parse\_args()

# Instanciranje soketa sa konstantama koje definišu njegov način rada

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

# Podešavanje port i IP adrese servera na osnovu prosleđenog IP argumenta

port = 21000

host = str(args.IP)

# Podnesi zahtev za povezivanje sa serverom sa prosleđenim parametrima

client\_socket.connect((host, port) )

*Deo koda u okviru kojeg se izvršava inicijalizacija sistema*

U ustaljenom režimu rada, klijentska aplikacija očekuje prijem nove slike (niza bajtova) uz pomoć prijemske funkcije (eng. *Receive*), koja je, kao i slušajuća funkcija na serverskoj strani, blokirajuća, te svi ostali procesi čekaju da ova funkcija primi novu poruku. Kada je server poslao poruku, poruka se ponovo pakuje u matricu u kojoj svaki bajt predstavlja boju nekog piksela. Takva slika se dalje prosleđuje na ekranizaciju uz pomoć *OpenCV* funkcije za prikaz (eng. *Image* *Show*).

# Promenljiva data je postavljena tako da dobijene karaktere tumači kao binarni niz

data = b""

# Promenljiva payload\_size skladišti izračunatu veličinu za ">L"

# Znak > označava ‘Big-Endian’ raspored bitova

# Karakter L se odnosi na Unsigned Long

payload\_size = struct.calcsize(">L")

while True:

    while len(data) < payload\_size:

# Dodaj nove bajtove koje je soket poslao (do 4096 bajtova)

        data += client\_socket.recv(4096)

# Prebaci sve bajtove iz data niza u novi niz

    packed\_msg\_size = data[:payload\_size]

# Isprazni data niz da bi bio spreman za prijem novih bajtova

    data = data[payload\_size:]

# Pokupi nekoliko (Long) prvih bajtova iz pomoćnog niza po Big-Endian(>) rasporedu

# kako bi dobio vrednost veličine poruke

    msg\_size = struct.unpack(">L", packed\_msg\_size)[0]

    while len(data) < msg\_size:

# Dodaj nove bajtove koje je soket poslao (do 4096 bajtova)

        data += client\_socket.recv(4096)

# Prebaci sve bajtove iz data niza u novi niz

    frame\_data = data[:msg\_size]

# Isprazni data niz da bi bio spreman za prijem novih bajtova

    data = data[msg\_size:]

# Zapakuj primljene bajtove u sliku uz pomoć pickle biblioteke

    frame=pickle.loads(frame\_data, fix\_imports=True, encoding="bytes")

# Dekoduj sliku u format koji odgovara instanci OpenCV matrice

    finalFrame = cv2.imdecode(frame, cv2.IMREAD\_COLOR)

# Ekranizacija dekodovane slike

    cv2.imshow('ImageWindow',finalFrame)

# Prosledi ASCII vrednost pritisnutog dugmeta tastature

    k = cv2.waitKey(1) & 0xff

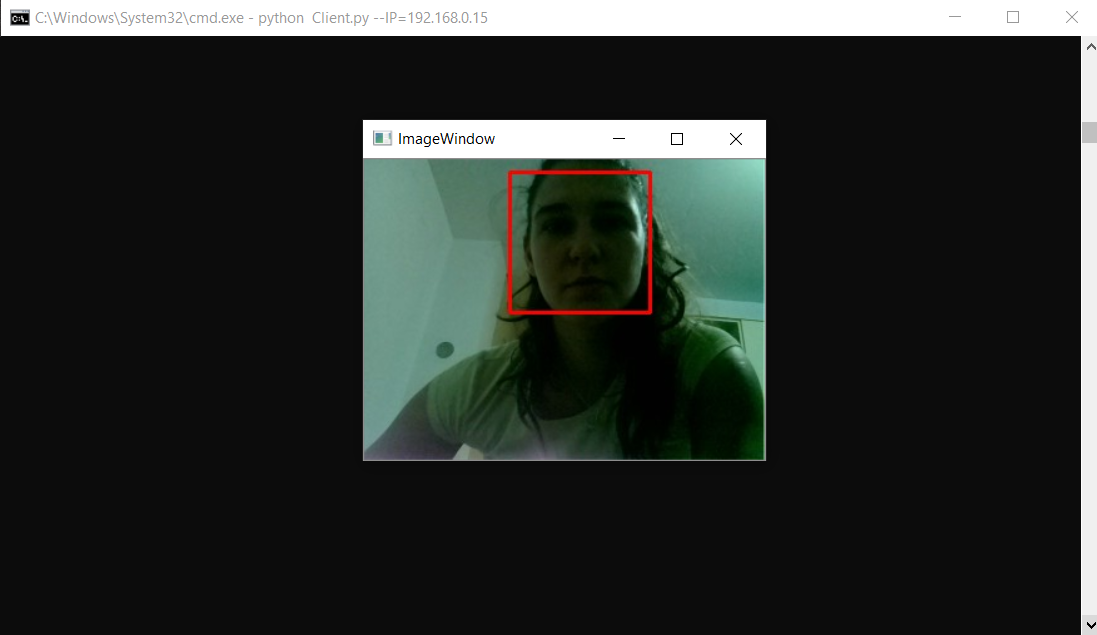
# Ukoliko je pritisnuto ESC dugme (vrednost 27) izađi iz petlje

    if k == 27:

        break

*Deo koda u okviru kojeg se izvršava prijem frejmova od servera i njihovo prikazivanje*

*Printscreen* prijema paketa i prikaza sa klijentske aplikacije predstavljen je na slici 20. Uokvireni deo slike predstavlja detekciju lica, odnosno detektovanje provalnika.



Slika . Printscreen prijema paketa i prikaza sa klijentske aplikacije, uokvireni deo slike predstavlja detekciju lica “provalnika”

Izlazak iz aplikacije moguć je u bilo kom trenutku pritiskom tastera *ESC* ili “ubijanjem” sa *Ctrl + C* tastera tastature.

# 9 Instaliranje sistema

Da bi mogao koristiti ovaj sistem, korisnik bi trebao da nađe adekvatno mesto na kojem bi mogao postaviti nadzorni uređaj – *RaspberryPi*, kamera mora biti postavljena u položaj koji bi pokrivao dobar deo prostorije koja se posmatra, a senzor udaljenosti je neophodno pozicionirati tako da eventualni provalnik prođe pored njega i načini mu prepreku koja uređaj “budi” te započinje proces zapisivanja i slanja slika (frejmova).

Pored toga, uređaju je neophodno obezbediti napajanje i internet za komunikaciju sa klijentima. Aplikacija se jednostavno može pokrenuti preko terminala uređaja, te je neophodno uređaju pristupiti ili daljinski (preko računara putem *SSH*) ili direktno na uređaj povezati miš, tastaturu i monitor.

Dokle god se korisnik nalazi u svom domu ili objektu koji želi da sačuva, odnosno na istoj je mreži njegov klijentski uređaj, korisnik može pri dolasku provalnika da dobije prenešenu sliku, u suprotnom bi mogao naknadno da vidi snimke.

# 10 Zaključak

Nakon završene implementacije celokupnog sistema, a potom i verifikacije funkcionalnosti može se zaključiti da sistem uspešno izvršava sve predviđene zadatke. Serverski uređaj uspešno prima nove klijente i pohranjuje ih potrebnim podacima u gotovo realnom vremenu, a pritom rukovodi i detekcijom sa senzora udaljenosti, kao i prepoznavanja uz pomoć kaskadnog klasifikatora. Performanse uređaja su na zavidnom nivou, s obzirom da se izvršavaju na malom računaru znatno slabijih performansi od modernih mašina, a protok podataka ne trpi zagušenja na većem broju aktivnih klijenata. Klijentska aplikacija podjednako dobro izvršava svoju funkcionalnost na obe predviđene platforme (*Windows* i *Linux*), te korisniku “olakšava život” brzim pristupom i otvaranjem iste.

U ovom projektu korišćena je moderna verzija *Python* programskog jezika (*Python3*) na obe strane, gde je najveći izazov zapravo bio sama oprimizacija sistema i paralelnog izvršavanja više funkcija. Pre svega, *Multi-threading* je kumovao takvom rezultatu.

Projekat pruža velik dijapazon potencijalnih unapređenja. Pre svega, na serverskoj strani bi se mogao povećati bezbednosni sistem različitim hardverskim dodacima, kako senzorima (npr. veći broj kamera), tako i aktuatorima (npr. zvučnim alarmima). Softverska strana bi mogla da uključi i prepoznavanje ukućana, te da ne pravi uzbunu pri njihovoj detekciji što bi moglo biti rađeno uz pomoć različitih biblioteka, među kojima je najpopularnija *Tensorflow* biblioteka.

Sa druge strane, klijentska aplikacija bi pre svega mogla da omogući pristup i sa mobilnih uređaja, odnosno da bude podržana na *Android* operativnom sistemu, ali i da trenutna aplikacija preraste sa konzolne u *GUI* aplikaciju.

# 11 Literatura

1. <https://sr.wikipedia.org/sr-ec/Raspberry_Pi> – *RaspberryPi*, *Wikipedia* članak, pristupljeno: septembar 2020.
2. [https://sr.wikipedia.org/sr-ec/Internet\_soket](https://sr.wikipedia.org/sr-ec/Internet_soket/) – Soket, *Wikipedia* članak, pristupljeno: septembar 2020.
3. <https://realpython.com/python-sockets/?fbclid=IwAR3ZFysBQGK0TBJp3MLAdu-GqH-IXn5S2SuLKivGhrG71LlyWvY_hMIdD5Q> – Soket programiranje u *Python* programskom jeziku, *RealPython* članak, Nathan Jennings, pristupljeno: septembar 2020.
4. <https://en.wikipedia.org/wiki/Network_socket> - Mrežni soket, *Wikipedia* članak, pristupljeno: septembar 2020.
5. <https://opencv.org/about/> - O *OpenCV* biblioteci, zvanična *OpenCV* stranica, pristupljeno: septembar 2020.
6. <https://towardsdatascience.com/a-guide-to-face-detection-in-python-3eab0f6b9fc1> – Uputstvo za primenu detekcije lica na *Python* programskom jeziku, *Towards Data Science* članak, Mael Fabian, pristupljeno: septembar 2020.
7. <https://www.raspberrypi.org/products/camera-module-v2/> - *RaspberryPi Camera Module V2*, zvanična *RaspberryPi* stranica o proizvodu, pristupljeno: septembar 2020.