Sustav za detekciju čučnjeva

Duvnjak, Matej

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

SUSTAV ZA DETEKCIJU ČUČNJEVA

Rijeka, ožujak 2023.

Matej Duvnjak
00669074025

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

SUSTAV ZA DETEKCIJU ČUČNJEVA

Mentor: prof. dr. sc. Mladen Tomić

Rijeka, ožujak 2023.

Matej Duvnjak
00669074025

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 14. ožujka 2022.

Zavod: **Zavod za računarstvo**
Predmet: **Programiranje ugradbenih sustava**
Grana: **2.09.02 informacijski sustavi**

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **Matej Duvnjak (0069074025)**
Studij: Diplomski sveučilišni studij računarstva
Modul: Računalni sustavi

Zadatak: **Sustav za detekciju čučnjeva / Squat Detection System**

Opis zadatka:

Razviti ugradbeni sustav za detekciju čučnjeva. Za dobivanje pozicija ključnih točaka tijela koristiti MoveNet. Za detekciju čučnja koristiti relativne pozicije ključnih točaka. Napraviti korisničko sučelje za upravljanje aplikacijom.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Izv. prof. dr. sc. Mladen Tomić

Predsjednik povjerenstva za diplomski ispit:

Prof. dr. sc. Kristijan Lenac

IZJAVA

Izjavljujem da sam samostalno, pod vodstvom prof. dr. sc. Mladena Tomića, primjenjujući znanja stečena tijekom studija, te koristeći navedenu literaturu izradio diplomski rad naslova "Sustav za detekciju čučnjeva".

Rijeka, ožujak 2023.

Matej Duvnjak

ZAHVALA

Želim se zahvaliti svojim roditeljima i sestri na potpori tijekom ovog studija, ali i cjelokupnog školovanja. Zahvala i kolegama s fakulteta s kojima sam dijelio ovo divno iskustvo, a neki su mi od njih postali jako dragi i bliski prijatelji. Posebna zahvala mentoru, profesorima i ostalim zaposlenicima fakulteta na savjetima i prenesenom znanju.

SADRŽAJ

1. UVOD 1
2. SKLOPOVLJE 3
2.1. Raspberry Pi 3
2.1.1. System-on-Chip 5
2.2. Kamera 6
3. PROGRAMSKI STOG 7
3.1. Raspberry Pi OS 7
3.2. Python 8
3.3. Knjižnice 8
3.2.1. TensorFlow 8
3.2.2. OpenCV 9
3.4. Ostale knjižnice i alati 10
4. ARHITEKTURA SUSTAVA 12
4.1. Klijentska strana 12
4.2. Korisničko sučelje 13
5. POSLUŽITELJSKA STRANA 15
5.1. App skripta 15
5.2. Tensor skripta 17
5.2.1. Detekcija orijentira i udaljenosti 20
5.2.2. Detekcija čučnja 22
5.2.3. Detekcija naklona 25
6. USPOREDBA RAZLIČITIH TF MODELA I GRANICA 28
6.1. MoveNet 28
6.2. Odabrane granice 29
7. ZAKLJUČAK 33
8. SAŽETAK 34
Literatura 35
Popis oznaka i kratica 36
Popis slika, tablica i programskog koda 38

1. UVOD

Danas se mogu pronaći sustavi, koji na mjestima gdje je puno ljudi, kao što su trgovine, restorani, zračne luke i stadioni, poboljšavaju prodaju, usluge ili procese praćenjem korisnika. Takvi sustavi prate korisnike i tako daju toplinske karte (engl. *heat maps*) koje će prikazivati gdje su se korisnici najdulje zadržali (trgovine i trgovački centri), koji su stolovi najpopularniji (restorani) ili gdje se u zračnim lukama ili stadionima stvaraju najveće gužve. Tu je i primjer Amazon Go trgovina koje nemaju blagajne i gdje sustav prati koje je proizvode korisnik uzeo, tako ih stavlja u virtualnu košaricu i naposljetku vrši naplatu. Takvi sustavi imaju na desetke kamera i mogu pratiti na stotine ljudi. Samim time, ne samo da su sustavi kompleksniji, nego imaju na raspolaganju i više računalne snage te specijalizirano sklopovlje koje će dodatno rasteretiti CPU i/ili GPU i tako poboljšati mogućnosti, jer program koristi instrukcije i sklopovlje specijalizirano za strojno učenje (npr. *Advanced Vector Extensions* i Google-ov *Tensor Processing Unit*). Sustav koji ćemo upoznati u ovom radu je manjih mogućnosti upravo jer je limitiran sklopovljem na kojem se izvršava, ali i dalje je dovoljnih performansa da se, uz manje preinake, može koristiti u rekreativne (sportske) ili čak rehabilitacijske (medicinske) svrhe.

Tema ovog diplomskog rada je izgradnja sustava koji će pratiti korisnika i tako brojati korisnikove čučnjeve. Sklopovlje koje će pokretati programski stog je Raspberry Pi, a sam program će se bazirati na Python programskom jeziku te TensorFlow i OpenCV knjižnicama. Prvotna ideja rada došla je od sličnog projekta koji se može pronaći na *Towards Data Science* internetskoj stranici[1]. Ono po čemu se ovaj rad razlikuje od *Towards Data Science* članka je u tome što nećemo raditi i uređivati vlastiti set fotografija koje ćemo koristiti za učenje modela, niti ćemo koristiti model za klasifikaciju fotografija, već ćemo korištenjem TensorFlow knjižnice i MoveNet modela pokušati detektirati korisnikove točke (zglobove) te tako na temelju dobivenih vrijednosti izračunati je li korisnik napravio čučanj ili ne. Postoji nekoliko načina kako se može detektirati čučanj. Prvi način je klasifikacija čučnja i taj način je opisan u *Towards Data Science* članku. Ukratko, neuronsku mrežu (engl. *Neural Network*) se na velikom setu fotografija uči detektirati čučanj te se dana fotografija klasificira ako se na njoj nalazi čučanj ili ne. Drugi način je detekcijom orijentira (zglobova) korisnika i računanjem čučnja, a taj se način još može podijeliti na dvije metode. Prva metoda detekcije čučnja je računanjem kuteva između tri točke te vrijednost dobivenog kuta koji mora biti manji od kuta čučnja. Druga metoda je računanjem duljina između točaka i postavljanjem granice koja će dijeliti čučanj od stojećeg

položaja. Upravo je ova druga metoda odabrana za korištenje u ovom radu te ćemo je kasnije detaljnije pogledati.

Kako bi se olakšalo korištenje tu je korisničko sučelje preko kojega korisnik upravlja sustavom. Korisniku su dostupna dva načina rada: slobodan način rada i izazov u kojem korisnik mora u vremenskom roku napraviti što više čučnjeva. Tu je i ljestvica s bodovima u kojoj se mogu pretraživati imena korisnika i vidjeti najveći rezultati. Korisničkom sučelju se pristupa pomoću internetskog preglednika i tu su korisniku nadohvat ruke kontrole za pokretanje i zaustavljanje sustava te prozor u kojem se prikazuje snimka s kamere koja služi kao povratna veza korisniku za lakše pozicioniranje. Proći ćemo kroz sustav, vidjeti kako je povezan i koji su njegovi dijelovi te detaljnije pogledati najzanimljivije dijelove.

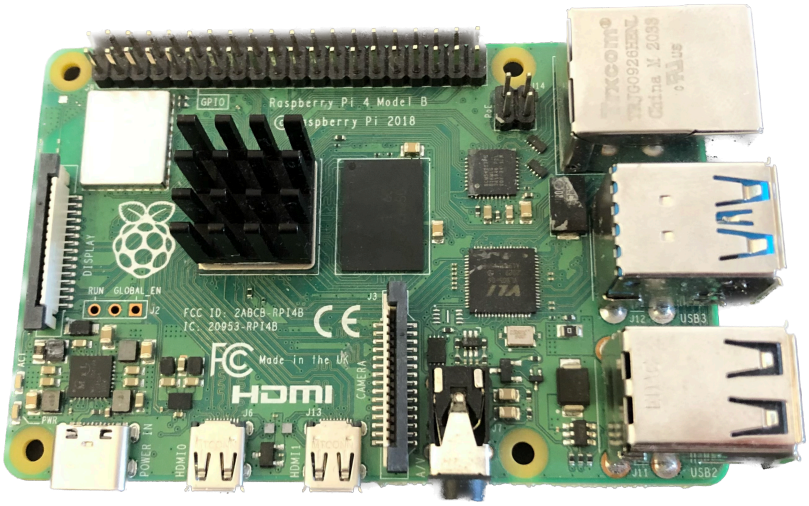
U sljedećem poglavlju proći ćemo kroz sklopovlje koje čini ovaj rad: SBC kategoriju računala u koju spada i Raspberry Pi, pogledat ćemo SoC koji ga pokreće i kameru koja će davati unos (video) Raspberry Pi-u za daljnju obradu. Treće poglavlje opisuje programske jezike i knjižnice koje su se koristile u izradi sustava. U četvrtom poglavlju sagledavamo cjelokupni sustav i korisničko sučelje. Peto poglavlje opisuje nam detaljnije poslužiteljsku stranu. Šesto poglavlje prolazi kroz *TensorFlow* knjižnicu koja se koristi za pokretanje *MoveNet* modela i opisuje dobivene granice za detekciju čučnja; u konačnici donosimo sam zaključak.

2. SKLOPOVLJE

Sklopovski dio ovog projekta sastoji se od dvije glavne komponente, a to je Raspberry Pi koji se koristi za pokretanje programa, posluživanje weba stranice i praćenje korisnika. Druga komponenta je web kamera koja daje unos (sliku) Raspberry Pi-ju.

2.1. Raspberry Pi

Raspberry Pi je osobno računalo malo veće od kreditne kartice i spada u klasu SBC računala. SBC (engl. *Single Board Computer*) kategorija računala zamišljena je da se koristi u edukativne, razvojne, demonstrativne i ugradbene svrhe radi svog omjera performansi i cijene. SBC računala na sebi imaju, osim procesora i radne memorije, po nekoliko portova kao što su USB, HDMI, Ethernet i GPIO za lakše korištenje[2][3]. Povijest SBC računala seže sve do kasnih 70-ih godina 20. stoljeća, a od najznačajnijih računala toga vremena su Acorn Electron (1983. godina) i BBC Micro (1981. godina). Kasnih 90-ih godina 20. stoljeća, kako se razvojem tehnologije dio funkcionalnosti prebacuje na druge komponente; grafičke i mrežne kartice, memorijski kontroleri na čvrstim diskovima (engl. *hard drive*), tako popularnost SBC računala pada sve dok se kasnije dio tih funkcionalnosti opet ne krene integrirati unutar SoC-a. Raspberry Pi se smatra duhovnim nasljednikom ranije spomenutih SBC računala (Acorn Electron i BBC Micro).



Slika 2.1. Raspberry Pi 4 Model B s pasivnim hladnjakom

Postoji više generacija i izvedbi Rasberry Pi-a. Trenutna generacija Raspberry Pi-a, korištena u ovom radu je četvrta generacija koja ima četverojezgreni procesor, 2 micro HDMI priključka, Ethernet priključak, četiri USB priključka i do 8GB radne memorije. Trenutna četvrta generacija može se pronaći i u izvedbama kao cjelokupno osobno računalo (Raspberry Pi 400 Personal Computer Kit)[1], računalni modul (engl. *Compute module*) i kao SBC. Razlike u performansama su minimalne jer se u sve tri verzije nalazi isti SoC, a razlike se vide u izvedbi i načinima upotrebe. Raspberry Pi PC kit ima sve potrebno da se može odmah prispojiti na monitor i početi koristiti kao normalno računalo uz minimalno tehničko znanje korisnika (engl. *plug-and-play*). Izvedba u obliku SBC-a ima iste priključke kao i PC Kit, no nije jednostavno započeti s korištenjem jer se od korisnika traži da prvo ima sve potrebne komponente (micro SD kartica, miš, tipkovnica) te da samostalno učita operacijski sustav na memorijsku karticu. Računalni modul je predviđen da se koristi u ugradbene svrhe (IoT, ručni uređaji, industrijski kontroleri i računala, automatizacija doma i sl.). Modul nema priključaka na sebi nego se za njegov razvoj i korištenje mora koristiti dodatna pločica (engl. *carrier board*) na kojoj će se nalaziti svi potrebni priključci. U ovom radu korištena je SBC izvedba te se u tablici 2.1. mogu vidjeti detaljne specifikacije korištenog Raspberry Pi-a (podebljane vrijednosti se odnose na korištenu konfiguraciju).

	Raspberry Pi 3, model B+	**Raspberry Pi 4, model B**
SoC	Broadcom BCM2837B0, Quad core Cortex-A53 (ARMv8) 64-bit SoC @ 1.4GHz	Broadcom BCM2711, Quad core Cortex-A72 (ARMv8) 64-bit SoC @ 1.5GHz
RAM	1GB LPDDR2 SDRAM	1GB, 2GB, 4GB ili **8GB LPDDR4-3200 SDRAM**
WLAN	2.4GHz, 5GHz IEEE 802.11.b/g/n/ac	2.4 GHz, 5.0 GHz IEEE 802.11ac
Bluetooth	4.2, BLE	5.0, BLE
Ethernet	Gigabit Ethernet (preko USB 2.0 maksimalna propusnost 300 Mbps)	Gigabit Ethernet
Portovi	4×USB 2.0, HDMI	2×USB 3.0, 2×USB 2.0, 2×micro-HDMI
GPIO	40-pin	40-pin
Ostalo	CSI, DSI, Micro SD, PoE	CSI, DSI, Micro SD, PoE

Tablica 2.1. Usporedba dvije verzije Raspberry Pi-a

[1] RPi 400 PC Kit dolazi u obliku tipkovnice i sliči na modernu verziju BBC Micro, Acorn Electron i ZX Spectrum računala

2.1.1. System-on-Chip

Takozvani SoC (engl. *System-on-Chip*) na sebi objedinjuje više računalnih komponenti kako bi se olakšao dizajn PCB (engl. *Printed Circuit Board*) pločice i smanjila cijena cijelog sustava. SoC koji se koristi u Raspberry Pi-u je dizajniran od strane Broadcom-a i ima četiri A72 jezgre bazirane na ARMv8 setu instrukcija. Unutar SoC-a se nalaze WiFi i Bluetooth prijamnici kao i integrirani grafički podsustav. Integrirani grafički podsustav je dovoljan za prikaz korisničkog sučelja do dva ekrana od 4K rezolucije i može prikazivati 720p video bez preskočenih sličica[2][4] [5][6].

Samo treniranje i pokretanje modela strojnog učenja radi na diskretnim i integriranim grafičkim podsustavima (engl. *Graphics Processing Unit*) zbog velikog broja jezgri i paralelizacije koja je moguća. U ovom radu grafički podsustav će biti korišten za prikaz korisničkog sučelja i uglavnom neiskorišten zbog nedostatka podrške da se sam model strojnog učenja pokrene na grafičkom podsustavu. Grafički podsustav Raspberry Pi-a ionako nije zamišljen da se koristi u ML (engl. *Machine Learning*) svrhe zbog nedovoljnih mogućnosti i performansi, tako da će se Movenet model izvršavati na CPU komponenti SoC-a. To i nije toliki nedostatak jer ćemo kasnije u radu vidjeti da je CPU dorastao zadatku da pogoni program bez značajnih gubitaka performansi.

Ipak, treba napomenuti da je temperatura SoC-a prilikom pokretanja i izvođenja programa dosezala i do 85° Celzijevih. Operativna temperatura Raspberry Pi-a je do 85°, ali već na 80° dolazi do termalnog prigušenja (engl. *Thermal throttling*) koje smanjuje frekvenciju procesora kako bi se spriječilo pregrijavanje SoC-a. Termalno prigušenje i smanjenje frekvencije utječe na performanse sustava koje se manifestiraju kao usporavanje cjelokupnog sustava i preskakanje slika za obradu i detekciju čučnja. Pojava termalnog prigušenja je najizraženija preko ljeta kada je temperatura zraka (okoline) veća i u skučenim kućištima gdje je loš protok svježeg (hladnijeg) zraka. Rješenje ovog problema se može postići korištenjem boljeg hlađenja; pasivno ili aktivno. Pasivno hlađenje se postiže korištenjem termalne paste i hladnjaka. Termalna pasta prenosi toplinu sa SoC-a na hladnjak te se tako povećava termalni kapacitet. Do termalnog prigušenja i dalje dolazi, samo sporije nego da pasivnog hlađenja nema. Aktivno hlađenje je slično pasivnom,

[2] Testirano reprodukcijom videa u 720p i 1080p rezolucijama unutar YouTube-a te provjereno korištenjem opcije “Stats for nerds”

samo što se tu dodaje i ventilator koji dovodi hladan zrak na hladnjak i odvodi topli zrak. Kod aktivnog hlađenja puno teže dolazi do termalnog prigušenja, ali treba imati na umu da se dio GPIO pinova treba iskoristiti za rad ventilatora. U ovom radu korišteno je pasivno hlađenje.

2.2. Kamera

Premda je bilo kakva web kamera primjerena za ovaj projekt, svaka kamera ima neke svoje karakteristike koje se trebaju uzeti u obzir. Jedna takva karakteristika je u leći. Općenito, u web kamerama koristi se leća od 75° pa sve do 110° širine. Kamera koja se koristila za ovaj projekt ima leću od 160° što daje veću (širu) sliku, samim time znači da kamera može biti bliže pojedincu kako bi ga "uhvatila" u cijelosti. Isto tako, kod kamera s većim vidnim poljem (engl. *Field of view*) češće zna dolaziti do distorzije rubova, a u određenim slučajevima i same slike. Tijekom ovog projekta zabilježene su uobičajene distorzije na rubovima slike koje nisu utjecale na detekciju korisnikovih zglobova. U nastavku se mogu vidjeti ostale bitne specifikacije kamere.

Senzor	Rezolucija	Rezolucija slike	Vidno polje	Video
1/4"	5MP 1080p	2952×1944	160°	VGA@90 fps, 720p@60 fps, 960p@45 fps, 1080p@15/30 fps

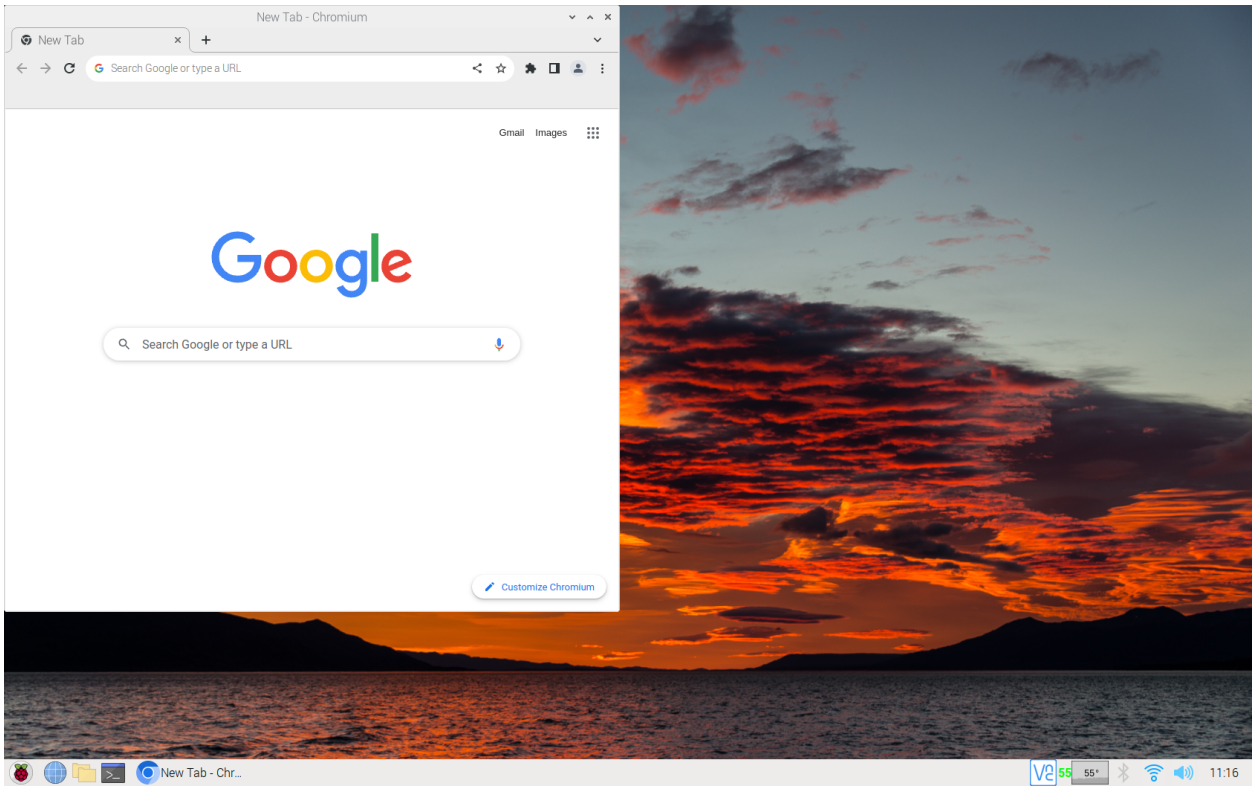
Tablica 2.2. Specifikacije kamere

3. PROGRAMSKI STOG

Cilj ovog poglavlja je proći kroz programski dio samog projekta; od operativnog sustava na Raspberry Pi-u do programskog jezika i korištenih knjižnica. Opisat ćemo prednosti i mane pojedinih dijelova i obrazložiti njihov odabir.

3.1. Raspberry Pi OS

Prethodno je zvan Raspbian; iz starog i novog imena zaključujemo da je to operacijski sustav predviđen za izvršavanje na Raspberry Pi-ju. Bazira se na Debian distribuciji Linuxa, a od veljače 2022. godine službeno je izašla i 64-bitna verzija. Operacijski sustav dolazi s grafičkim sučeljem i nekoliko predinstaliranih programa za lakše korištenje. Neki od programa su Chromium internetski preglednik, VLC player, Wolfram Mathematica i VNC server[7][8]. Tu su podrška i brojne druge mogućnosti za mrežnu komunikaciju i skriptiranje; SSH, FTP/SFTP, bash, PHP koje olakšavaju rad s poslužiteljima kao Apache server i NGINX.



Slika 3.1. Prikaz Raspberry Pi OS-a s otvorenim Chromium preglednikom

3.2. Python

Python je jedan od najpopularnijih programskih jezika na svijetu i većina ovog rada je napravljena u Python programskom jeziku. Jezik podržava velik broj knjižnica za rad s datotekama, obradu i prikaz podataka te je kompatibilan s TensorFlow knjižnicom. Spada u kategoriju skriptnih jezika što znači da se programski kod izvršava u onom trenutku kada interpreter dođe do te linije koda. Python podržava programske paradigme uključujući strukture, objektno i funkcionalno programiranje. Korištena verzija je 3.9.2 koja ima službenu podršku do listopada 2025. godine. Tijekom pisanja ovog rada izašle su novije verzije Pythona, ali izmjene koje su predstavljene u tim verzijama nisu značajne za ovaj rad.

3.3. Knjižnice

Kao što je ranije spomenuto, Python je poznat po velikom broju knjižnica koje mogu uveliko olakšati razvoj programa. Neke od domena koje Python može pokriti s knjižnicama su strojno učenje, GUI aplikacije (engl. *Graphical User Interface*), web development, obrada slike, znanstveno računanje, multimedija, obrada teksta i dr.[9]. Neke od knjižnica koje su se koristile u ovom radu su: *NumPy*, *pandas*, *Matplotlib*, *OpenCV* i *TensorFlow*. Detaljnije ćemo obraditi zadnje dvije knjižnice, dok ćemo ostale korištene knjižnice samo ukratko spomenuti.

3.2.1. TensorFlow

TensorFlow je besplatna knjižnica otvorenog koda razvijena od strane Google Brain tima. Javnosti postaje dostupna u studenom 2015. godine, a službena verzija 1.0.0 izlazi u veljači 2017. godine. Otkako je *TensorFlow* postao dostupan javnosti, neprestano izlaze nove inačice koje donose nove setove funkcionalnosti i optimizacije. *TensorFlow* je napravljen kako bi olakšao izgradnju ML (engl. *Machine Learning*) modela. *TensorFlow* API (engl. *Application Programming Interface*) olakšava učenje i implementaciju strojnog učenja, dubokog učenja i znanstvenog računanja te nudi širok set alata za izgradnju modela. To uključuje unos i obradu podataka, evaluaciju modela i vizualizaciju. Zamišljen je da bude prijenosan i da se izvršava na različitim uređajima i platformama. Lako se skalira s procesora na grafičku karticu ili klaster grafičkih kartica pa sve do veličine podatkovnih centara.

Postoji nekoliko verzija *TensorFlowa* koje su prilagođene uređajima koji ga pokreću. Postoji puna *TensorFlow Core* verzija koja podržava treniranje i pokretanje modela strojnog učenja. Programer odabire želi li koristiti procesor ili grafičku karticu.

TensorFlow.js je predviđen za korištenje u internetskom pregledniku i *Node.js* platformama. U internetskom pregledniku TF.js podržava mobilne i desktop uređaje. Svaki od tih uređaja automatski određuje i konfigurira sučelja kao *WebGL* i sl. U *Node.js* verziji povezuje se TF API koji se spaja na server što povećava same performanse jer se zahtjevi obrađuju na udaljenom serveru, a to ujedno dovodi i do veće latencije zato što je server udaljen od klijenta.

Zadnja verzija, ujedno i ona koja je korištena u ovom radu je *TensorFlow Lite* koja je predviđena za izvršavanje na mobilnim i IoT (engl. *Internet of Things*) uređajima. Ova verzija je pojednostavljena i nema mogućnosti za treniranje modela strojnog učenja. Programerima je dostupan samo interpreter koji može pokrenuti postojeće modele koji su prethodno istrenirani na drugim verzijama *TensorFlowa* koje podržavaju učenje. Dok se sve tri verzije mogu izvršavati lokalno na uređaju, ova verzija je nešto lakša i samim time manje opterećuje uređaj na kojem je pokrenut *TensorFlow*. Lokalno pokretanje TF znači manju latenciju jer se ne mora kontaktirati udaljeni poslužitelj, isto tako štiti i privatnost jer podatci ne napuštaju uređaj i nije potrebno biti povezan na internet. Veličina modela je smanjena i samim time zauzima manje memorije. Manja je potrošnja snage zbog optimiziranog modela, ali manja je potrošnja i jer mrežni uređaji ne moraju biti stalno uključeni. Ipak, postoji i par nedostataka. Jedan od njih je da postojeći *TensorFlow Core* modeli nisu kompatibilni s *Lite* verzijom i trebaju prvo biti pretvoreni u *Lite* verziju[10][11].

3.2.2. OpenCV

Druga korištena knjižnica je *OpenCV* koja je izvorno izašla 1999. godine te je napravljena od strane Intela. *OpenCV* stoji za *Open Source Computer Vision Library* i nudi detaljan set najnovijih algoritama za računalni vid i strojno učenje. Algoritmi mogu biti korišteni za detekciju predmeta i lica, praćenje predmeta u pokretu, spajanja više fotografija u jednu, s ciljem kreiranja jedne fotografije veće rezolucije i ostale napredne manipulacije slike. U ovom radu *OpenCV* je korišten radi lakog pristupa kameri kao i velikog broja funkcija za manipulaciju slike. Ipak, u

radu se već koristi druga knjižnica za detekciju i praćenje korisnika pa je dio *OpenCV* ostao neiskorišten.

OpenCV ima podršku za C++, Python, Java i Matlab programske jezike te je podržan na svim većim operacijskim sustavima (Mac OS, Windows, Linux i Android). Zamišljen je za korištenje u programima i aplikacijama u stvarnom vremenu te se za njega aktivno razvija podrška za *Compute Unified Device Architecture* (CUDA) i *Open Computing Language* (OpenCL).

3.4. Ostale knjižnice i alati

Od ostalih knjižnica mogu se spomenuti sistemske knjižnice: *os*, *time*, *threading*, *pathlib*, *datetime*. Ove knjižnice implementiraju većinu funkcionalnosti jezgre operacijskog sustava kako bi se moglo pristupati datotečnom sustavu, vremenu i kreiranju novih dretvi.

Kako bi se lakše radilo s dobivenim TF podacima tu su *NumPy*, *pandas* i *Matplotlib*. NumPy olakšava rad s poljima i matricama koje se dobivaju kao rezultat od TF. *Pandas* olakšava rad s .csv datotekama kada se zapisuju i čitaju korisnikovi rezultati čučnjeva. Matplotlib se nije direktno koristio u programu za detekciju čučnjeva, ali ipak ga vrijedi spomenuti jer je bio ključan za prikaz korisnikovih vrijednosti čučnjeva i samim time za analizu korisnikovih čučnjeva i odlučivanju granice čučnja.

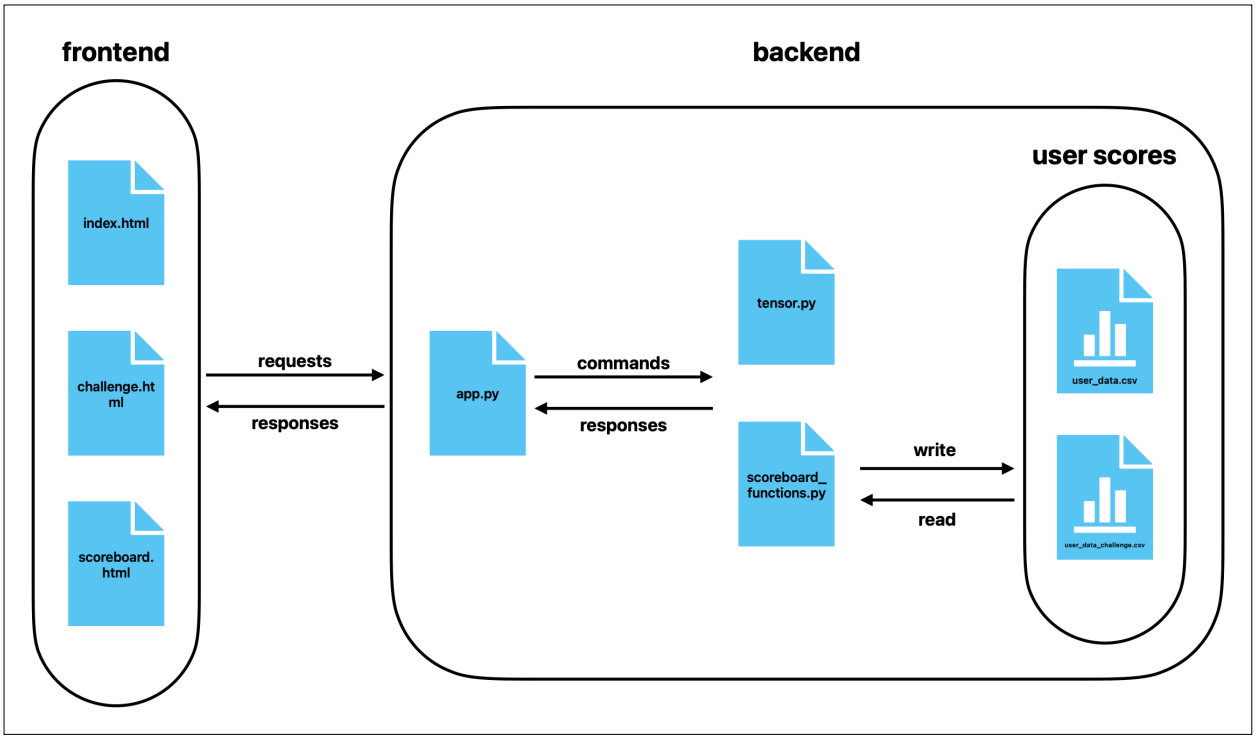
Flask je *framework* koji je osmišljen kako bi u početku razvoja bio brz i jednostavan za implementirati, a kasnije, kako projekti rastu, skalirao do složenih aplikacija. Postoje brojni alati i knjižnice koje rade neometano s *Flaskom* te ga to čini odličnim izborom za programere početnike na malim projektima, ali i za iskusnije programere na većim projektima. Flask se u ovom radu koristi za posluživanje internetske stranice (korisničkog sučelja) korisniku te kao sučelje između klijentske strane i ostatka sustava. Flask kao sučelje, između klijenta i ostatka sustava, postignuto je korištenjem route dekoratora koji povezuju URL s funkcijama koje se nalaze u glavnoj poslužiteljskoj skripti (app.py).

Visual Studio Code je besplatni uređivač koda napravljen od strane Microsofta i podržava debugiranje, isticanje sintakse, inteligentno dovršavanje koda, refaktoriranje koda, Git. Tu je i prilagođavanje VS Code-a prema potrebi korisnika - od odabira teme do prečaca i raznih

proširenja koja dodaju nove funkcionalnosti. VS Code podržava širok raspon programskih jezika uključujući C, C#, C++, Go, Java, JavaScript, Python, Rust te brojni drugi. Ovaj diplomski rad napravljen je upravo u VS Code s IntelliCode, Pylance i Python dodacima.

4. ARHITEKTURA SUSTAVA

Prije nego što krenemo detaljnije gledati same cjeline sustava, prvo moramo objasniti na što se odnosi koji dio. Klijentska strana podrazumijeva web sučelje preko kojega korisnik može kontrolirati sustav i pristupati različitim načinima rada. Klijentska strana se sastoji od tri .html datoteke. Za sve ostalo možemo reći da je pozadinska strana (engl. *backend*). *Backend* čine tri .py skripte i dvije .csv datoteke. Glavna app.py skripta pokreće Flask i web server te se nakon uspješne inicijalizacije servera korisnik može spojiti na Raspberry Pi. app.py skripta je poveznica između klijentske strane i ostatka pozadinskog sustava te sve zahtjeve koje dobije od korisnika na korisničkom sučelju obrađuje i prosljeđuje na preostale dvije .py skripte (tensor.py i scoreboard_functions.py). Bitno je za napomenuti da app skripta ne radi zahtjeve prema klijentskoj strani nego samo odgovara na pristigle zahtjeve.



Slika 4.1. Logički prikaz sustava

4.1. Klijentska strana

Kada se korisnik spoji na Raspberry Pi putem internetskog preglednika dočeka ga sučelje preko kojega može upravljati sustavom (slika 4.1.). Korisnik može navigirati na tri postojeća sučelja.

Početno sučelje, odnosno stranica ujedno je i jedan od načina rada gdje korisnik može pokrenuti sustav i raditi čučnjeve bez vremenskog limita. Korisnik se spaja na stranicu preko računala koje se nalazi u istoj lokalnoj mreži kao i Raspberry Pi i u adresnu traku upisuje “http://raspberry-pi-ip-adresa:7000” ili se direktno spaja preko samog Raspberry Pi-a gdje se može opet upisati ip adresa i port ili “http://localhost:7000”.

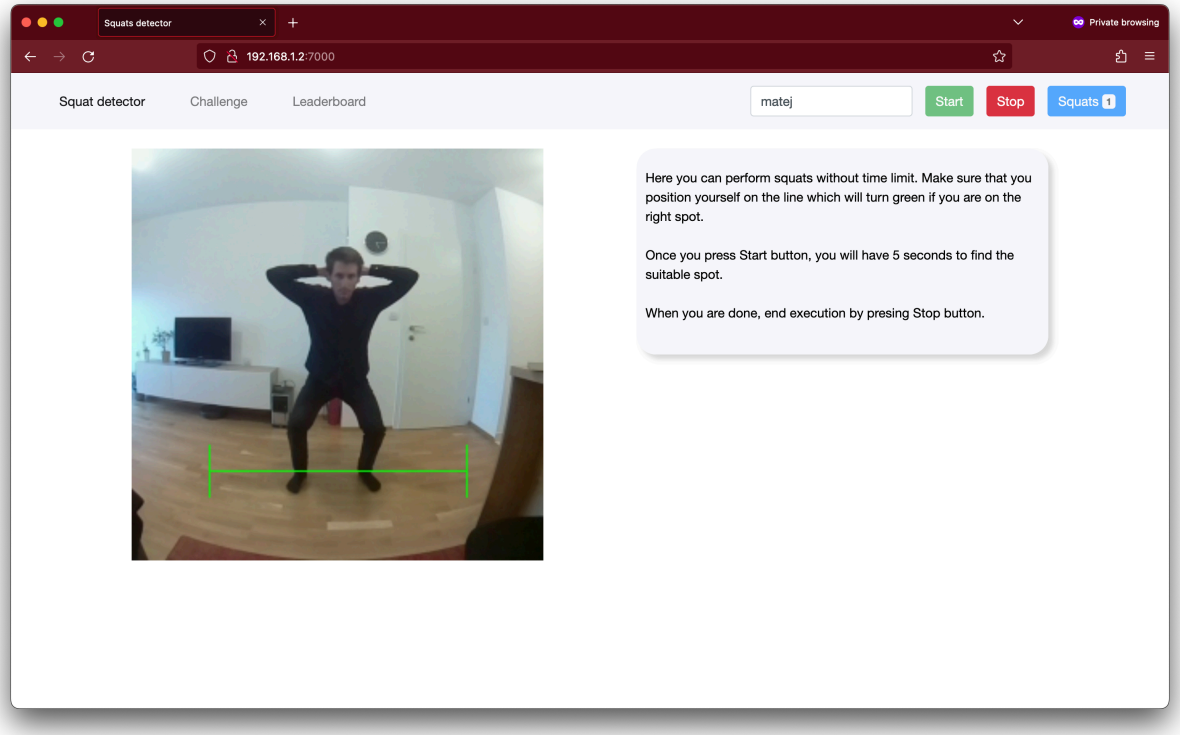
Druga stranica je *challenge* (engl. izazov) i dosta je slična *index* stranici (slika 4.2.). Razlika se krije u logici programa. Tu korisnik ima vremensko ograničenje od 30 sekundi da napravi što više čučnjeva. Korisnik može navigirati do stranice preko *index* i *scoreboard* sučelja ili upisivanjem novog stringa u adresnu traku (“http://raspberry-pi-ip-adresa:7000/challenge").

Zadnja stranica je *scoreboard* stranica i tu se nalazi ljestvica s brojem čučnjeva svakog korisnika/ igrača (slika 4.3.). Postoje dvije ljestvice, a isto toliko je i načina rada sustava (slobodni način i izazov). Tu korisnik može pretraživati postojeće korisnike i gledati stanje na ljestvici. Rezultati ljestvice se pohranjuju u dvije .csv[3] datoteke. U prvu datoteku se pohranjuju vrijednosti iz slobodnog moda, a u drugu rezultati iz izazova. Pohrana rezultata je različita između modova jer se u slobodnom modu pohranjuje tek kada korisnik pritisne “Stop” gumb, a u izazovu tek kada istekne brojač (engl. timer). Ako korisnik pritisne “Stop” u izazovu, rezultat se neće bilježiti jer se smatra da korisnik nije odradio izazov do kraja.

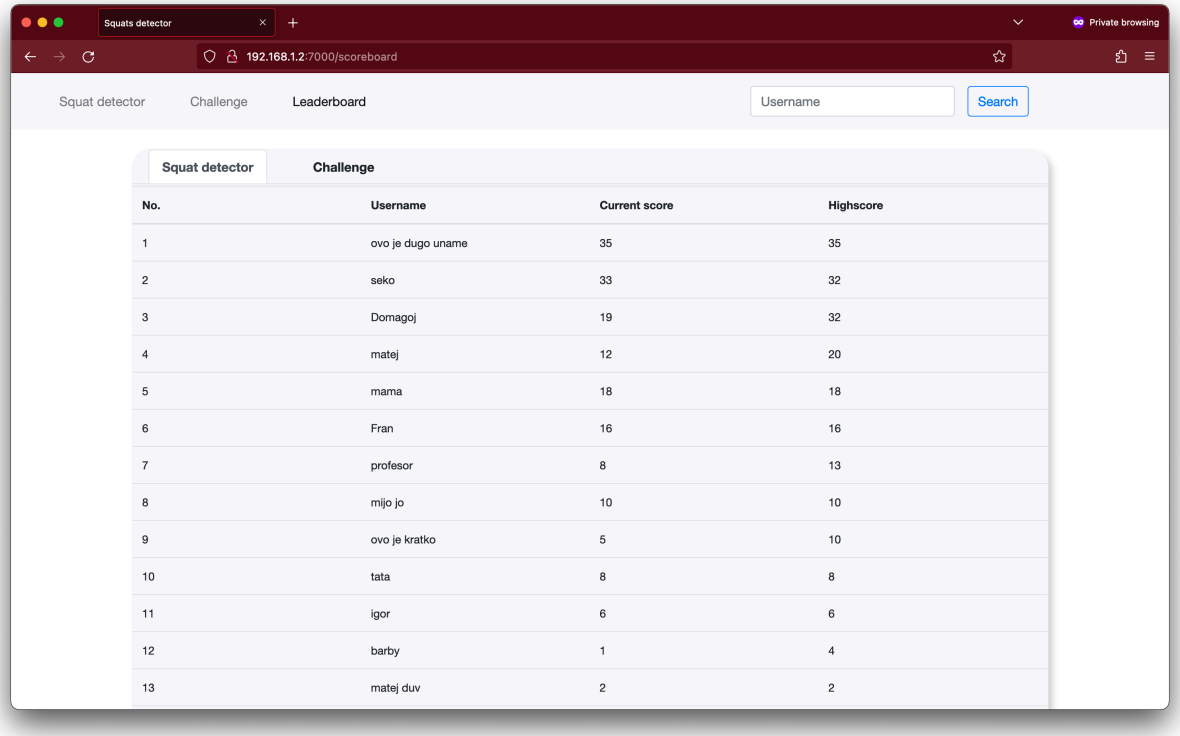
4.2. Korisničko sučelje

Korisnik pristupa sučelju pomoću internetskog preglednika. Sve tri stranice imaju navigacijsku traku koja sadrži poveznice na druge dijelove sustava i unosno polje za upis korisničkog imena. Unosno polje ima filter koji pretražuje zabranjene znakove kao što su: “!#$%&/()=?’+*-.:,;{}|\”\ \<>”. Filter je posebno bitan upravo radi zareza, jer u suprotnom, zarez može uzrokovati krivo čitanje podataka iz datoteke. Osim filtera za zabranjene znakove, postavljeno je i ograničenje na duljinu korisničkog imena koje ne može biti veće od 24 znaka ili prazan string.
Poslužiteljska logika koja pokreće slobodan mod ista je kao i u izazovu. Prevelike razlike u sučelju između slobodnog moda i izazova ne postoje, osim dodatnog brojača koji se aktivira prilikom izazova kako bi informirao korisnika koliko izazov još traje.

[3] .csv stoji za C*omma Separated Value*



Slika 4.2. Prikaz sučelja u slobodnom modu



No.	Username	Current score	Highscore
1	ovo je dugo uname	35	35
2	seko	33	32
3	Domagoj	19	32
4	matej	12	20
5	mama	18	18
6	Fran	16	16
7	profesor	8	13
8	mijo jo	10	10
9	ovo je kratko	5	10
10	tata	8	8
11	igor	6	6
12	barby	1	4
13	matej duv	2	2

Slika 4.3. Prikaz ljestvice

5. POSLUŽITELJSKA STRANA

Poslužiteljska strana (engl. *backend*) se sastoji od nekoliko datoteka od kojih su dvije najvažnije i koje ćemo detaljnije sagledati - app.py i tensor.py skripte. *Backend* je zamišljen da app skripta služi kao čvor na koji će dolaziti zahtjevi s klijentske strane, a logika u app skripti, na temelju tih zahtjeva, poziva funkcije koje upravljaju tensor skriptom. Rezultati tensor skripte prosljeđuju se app skripti koja šalje odgovor na pristigli zahtjev s klijentske strane. Komunikacija između klijentske strane i app skripte ide preko HTTP POST i GET zahtjeva. Uvoženjem preostalih poslužiteljskih skripti u app skriptu, ostvaruje se komunikacija unutar poslužiteljske strane.

5.1. App skripta

Kao što je ranije spomenuto, app.py je čvor koji prima zahtjeve od korisnika i na temelju tih zahtjeva upravlja tensor.py skriptom koja ima pristup OpenCV-u (video stream) i TensorFlow-u (detekcija korisnika i čučnja).

Ispod se nalazi glavna ulazna točka u kod. Run metoda pokreće Flask poslužitelj na portu 7000 koji je javno dostupan svima unutar mreže.

```
from flask import Response, Flask, jsonify, request, send_from_directory

app = Flask(__name__)
app.config['TEMPLATES_AUTO_RELOAD'] = True

                                        ⋮


if __name__ == '__main__':
    app.run(host='0.0.0.0',port=7000,debug=True,threaded=True,use_reloader=False)
```

Programski kod 5.1. Postavljanje i pokretanje Flask poslužitelja

Kako bi se korisnik mogao spojiti kada upiše u adresnu traku “http://raspberry-pi-ip-adresa:7000”, potrebno je povezati njegov URL (engl. Uniform Resource Locator) zahtjev s funkcijama. To upravo radi *route* dekorator u isječku programskog koda 5.2. gdje svaki dekorator ima specifičnu funkciju obrade zahtjeva pristiglih s klijentske strane. Sve tri funkcije su slične, prvo se izvrši stop_thread funkcija koja se pobrine da ne postoji pokrenuta TensorFlow dretva (ako postoji - terminira je) te se pokreće nova i učitava se korisničko sučelje (web stranica).

```
@app.route('/', methods=["GET"])
def main():
    stop_thread()
    my_thread = Thread(target=tensor.start, args=("index",), name="index")
    my_thread.daemon = True
    my_thread.start()
    return send_from_directory("static", "index.html")

@app.route('/challenge', methods=["GET"])
def challenge():
    stop_thread()
    my_thread = Thread(target=tensor.start, args=("challenge",), name="challenge")
    my_thread.daemon = True
    my_thread.start()
    return send_from_directory("static", "challenge.html")

@app.route('/scoreboard', methods=["GET"])
def scoreboard():
    stop_thread()
    return send_from_directory("static", "scoreboard.html")
```

Programski kod 5.2. Povezivanje URL zahtjeva s odgovarajućim funkcijama

U app skripti se nalazi još par funkcija (programski kod 5.3.) koje povezuju klijentsku stranu s tensor skriptom. Te funkcije postavljaju zastavice preko kojih tensor skripta zna kada treba započeti ili prestati brojati čučnjeve te se tu dohvaćaju ljestvice i rezultati čučnjeva. Ove funkcije su povezane s gumbima na korisničkom sučelju, a app skripta služi kao posrednik koji obrađuje pristigle zahtjeve te ih prosljeđuje tensor skripti.

```
@app.route('/status')
def status():
    return jsonify(tensor.get_status())

@app.route('/start', methods=['POST'])
def start():
    global uname
    global mode
    global pause_flag
    pause_flag = True
    uname = request.form["uname"]
    mode = request.form["mode"]
    time.sleep(6)
    if(uname != None and pause_flag == True):
        tensor.TIMER = time.time()
        tensor.USERNAME = uname
        tensor.STOP_FLAG = False

    return "ok", 200

@app.route('/user_search', methods=['POST'])
def user_search():
    return scoreboard_functions.user_search(request.form["uname"],
request.form["mode"])

@app.route('/stop', methods=['POST'])
def stop():
    global pause_flag
    pause_flag = False
    tensor.stop()
    return "ok", 200

@app.route('/update_scoreboard', methods=['POST'])
def update_scoreboard():
    mode = request.get_data().decode("utf-8")
    return scoreboard_functions.scoreboard(mode)
```

Programski kod 5.3. Ostale funkcije app.py skripte koje obrađuju klijentske zahtjeve

5.2. Tensor skripta

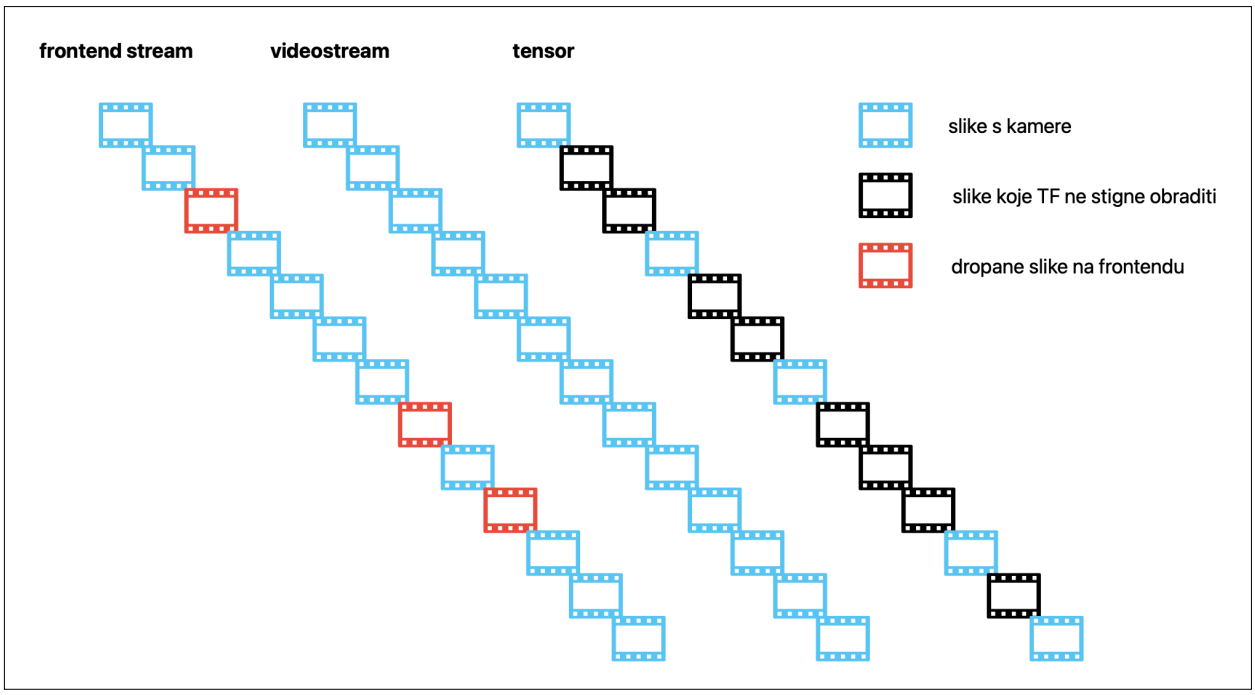
Kao što joj samo ime kaže tensor.py skripta zadužena je za TensorFlow interpreter. U njoj se osim TF interpretera nalazi i VideoStream klasa koja, kada se pozove, instancira pristup kameri pomoću OpenCV knjižnice i tako započinje snimanje. Unutar klase nalazi se nekoliko funkcija kojima se kontrolira klasa, odnosno videozapis. Klasa prilikom inicijalizacije prima dva argumenta, rezoluciju i broj sličica u sekundi (engl. Frames per Second - FPS). FPS ovisi o mogućnostima kamere i podržanim formatima. Standard je da to bude 30 ili 60 sličica u sekundi i na ovom projektu je korišteno 30 FPS-a radi korištenja nestandardne rezolucije snimanja (192×192 ili 256×256 piksela).

Rezolucija može poprimiti samo dvije vrijednosti koju određuju korišteni *TensorFlow* modeli. *Lightning* model za unos očekuje sliku rezolucije 192×192 piksela, dok *Thunder* očekuje sliku rezolucije 256×256 piksela. U programskom kodu 5.4. može se vidjeti postavljanje rezolucije ovisno o korištenom TF modelu i *VideoStream* klase s pripadajućim funkcijama.

```
if(MODEL_NAME.rsplit('/', 1)[1] == "lite-
model_movenet_singlepose_lightning_tflite_int8_4.tflite" or MODEL_NAME.rsplit('/',1)
[1] == "lite-model_movenet_singlepose_lightning_tflite_float16_4.tflite"):
    tf_res = 192
elif(MODEL_NAME.rsplit('/', 1)[1] == "lite-
model_movenet_singlepose_thunder_tflite_int8_4.tflite" or MODEL_NAME.rsplit('/', 1)
[1] == "lite-model_movenet_singlepose_thunder_tflite_float16_4.tflite"):
    tf_res = 256


class VideoStream:
    """Camera object that controls video streaming from the OpenCV"""
    def __init__(self, resolution = (tf_res, tf_res), framerate = 30):
        self.stream = cv2.VideoCapture(0)
        self.stream.set(cv2.CAP_PROP_FPS, framerate)
        logging.info("Camera initiated.")
        _ = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
        _ = self.stream.set(3, resolution[0])
        _ = self.stream.set(4, resolution[1])

        (self.grabbed, self.frame) = self.stream.read()
        self.stopped = False

    def start(self):
        Thread(target = self.update, args = ()).start()
        return self

    def update(self):
        while True:
            if self.stopped:
                self.stream.release()
                return

            (self.grabbed, self.frame) = self.stream.read()

    def read(self):
        return self.frame

    def stop(self):
        self.stopped = True
```

Programski kod 5.4. Definicija VideoStream klase

Jednom, kada se VideoStream klasa instancira, *stream* postaje dostupan i počinje se slati na klijentsku stranu. Isti *stream* koristi se i za detekciju čučnjeva u TensorFlow dijelu koda. Tu nastaje problem, jer ako se koristi isti *stream* za prikaz na korisničkom sučelju i za detekciju čučnjeva, *framerate* će pasti zbog TensorFlow-a te koliko mu treba da obradi jednu sliku. Ako TF obradi 4 slike u sekundi onda će se i *stream* na klijentskoj strani prikazivati u 4 FPS-a. Da bismo izbjegli da TF usporava cijeli *framerate*, svaki dio radi u svojoj dretvi kako bi se izbjegla međusobna interferencija.



Slika 5.1. Pojednostavljeni prikaz odvojenosti streamova na frontendu i backendu

U teoriji, klijentska strana bi trebala ići u korak sa snimkom koja dolazi s kamere, ali zbog pozadinskih procesa i termalnog prigušivanja (engl. thermal throttling) događa se da klijentska strana preskoči i izgubi dio slika za prikaz. To se najčešće manifestira kao *stutter* u prikazu koji prolazi nakon nekoliko slika. Zbog nedovoljnih performansi RPi-a da stabilno obrađuje pristigle snimke, određene slike se ponovno prikazuju korisniku na ekranu dok nova slika ne bude spremna za prikaz, to ponavljanje slike, propuštanje jedne slike za crtanje dok druga ne bude spremna zove se *stutter*. Na žalost, konkretno rješenje za ovaj problem ne postoji, ali se može umanjiti boljim hlađenjem Raspberry Pi-a ili dodatnim smanjivanjem *framerate*-a što kod nas nije opcija jer kamera ne podržava format snimanja s manjom frekvencijom osvježavanja slike[4].
S druge strane, *TensorFlow* model koji se koristio za ovaj rad je *MoveNet.Thunder* (FP16) koji po TF dokumentaciji nije moguće obrađivati više od 2 slike u sekundi na Raspberry Pi-u (više o tome u idućem poglavlju)[12]. U stvarnosti, s ovim je modelom na Raspberry Pi-u je moguće obrađivati 3-4 slike u sekundi[5], što je dovoljno da se pouzdano detektiraju točke za detekciju čučnja. Iako se na prvu čini da 4 slike u sekundi nije dovoljno da se detektira čučanj to i nije baš točno jer za samu detekciju čučnja potrebno je samo detektirati točke i njihovu promjenu u visini

[4] Danas je uobičajno imati reprodukciju videa u 30 sličica u sekundi ili više. Frekvencija osvježavanja koja je ispod 30 FPS-a smatra se da je loša za korisničko iskustvo

[5] Brzina obrade slike ovisi o pozadinskim procesima, temperaturi SoC-a i drugim čimbenicima

tijekom vremena. Četiri slike u sekundi s kojima raspolažemo je dovoljno za trajanje jednog čučnja da se uoči ta razlika u visini i tako detektira čučanj.

5.2.1. Detekcija orijentira i udaljenosti

Ono što tensor skriptu čini zanimljivom je upravo implementacija TensorFlowa i logika koja detektira čučanj. Kako bi čučanj bio detektiran trebamo dobiti točke orijentira, točnije točke gležnja, koljena, kuka i ramena. Za to su zadužene dvije funkcije gdje movenet funkcija dobiva sliku i na toj slici pomoću TensorFlow interpretera detektira korisnikove točke. Druga funkcija, run_inference poziva prvu funkciju (movenet) i dobiveni rezultat točaka translatira u koordinate same slike.

```
def movenet(input_image):
    """
    Runs detection on an input image.
    Args:
        input_image: A [1, height, width, 3] tensor represents the input image
        pixels. Note that the height/width should already be resized and match
        the expected input resolution of the model before passing into this
        function.

    Returns:
        A [1, 1, 17, 3] float numpy array representing the predicted keypoint
        coordinates and scores.
    """
    interpreter.set_tensor(input_details[0]['index'], input_image.astype(np.uint8))
    interpreter.invoke()
    keypoints_with_scores = interpreter.get_tensor(output_details[0]['index'])

    return keypoints_with_scores


def run_inference(movenet, image):
    """
    Runs model inferece on the cropped region.
    The function runs the model inference on the cropped region and updates the
    model output to the original image coordinate system.
    """
    image_height, image_width, _ = image.shape
    keypoints_with_scores = movenet(np.expand_dims(image, axis = 0))
    for idx in range(17):
        keypoints_with_scores[0, 0, idx, 0] = (image_height + image_height *
                keypoints_with_scores[0, 0, idx, 0]) / image_height
        keypoints_with_scores[0, 0, idx, 1] = (image_width + image_width *
                keypoints_with_scores[0, 0, idx, 1]) / image_width

    return keypoints_with_scores
```

Programski kod 5.5. Funkcija za translataciju točaka (run_inference) i funkcija zaključka (movenet)

Slika 5.2. Detektirani orijentiri s translatacijom i signalnom linijom

Orijentiri služe još jednoj svrsi, a to je signalizacija korisniku ako se nalazi na optimalnoj udaljenosti. Udaljenost korisnika od kamere je bitna kako bi se osiguralo da su sve bitne točke korisnika vidljive (da ne budu izvan kadra) jer je korisnik preblizu ili predaleko od kamere. Linija koja služi za signalizaciju udaljenosti korisniku nalazi se na fiksnoj poziciji na slici. Udaljenost od kamere koju će signalna linija prikazivati ovisi o lokaciji (visini) kamere. Ako se kamera nalazi na visini od 90-120 cm, udaljenost koju će linija predstavljati bit će 150-200 cm od kamere i korisnik će imati oko 50 cm manevarskog prostora da se pravilno pozicionira na liniju. Ako se korisnik previše udalji ili približi liniji ona postaje crvena. Linija je inače zelene boje i ona signalizira korisniku da je na pravilnoj udaljenosti te da može započeti s čučnjevima. U kodu, funkcija koja crta liniju je ista funkcija koja je zadužena za slanje videozapisa korisničkom sučelju.

```
def get_jpg():
    """
    upscales 192x192 image four times (to 768x768) or upscales 256x256 image
    three times (to 768x768) depending on tfmodel. Upscaled image returned to web
    """
    global line_color
    color = (217,105,0)
    thickness = 1
    if line_color == "green":
        color = (0, 255, 0)
    else:
        color = (0, 0, 255)

    frame = cv2.line(videostream.read(), (48,200), (208,200), color, thickness)
    frame = cv2.line(frame, (48,184), (48,216), color, thickness)
    frame = cv2.line(frame, (208,184), (208,216), color, thickness)
    return cv2.imencode(".jpg", cv2.resize(frame, (768, 768)))
```

Programski kod 5.6. Funkcija koja crta signalnu liniju na sliku i prosljeđuje je klijentskoj strani

5.2.2. Detekcija čučnja

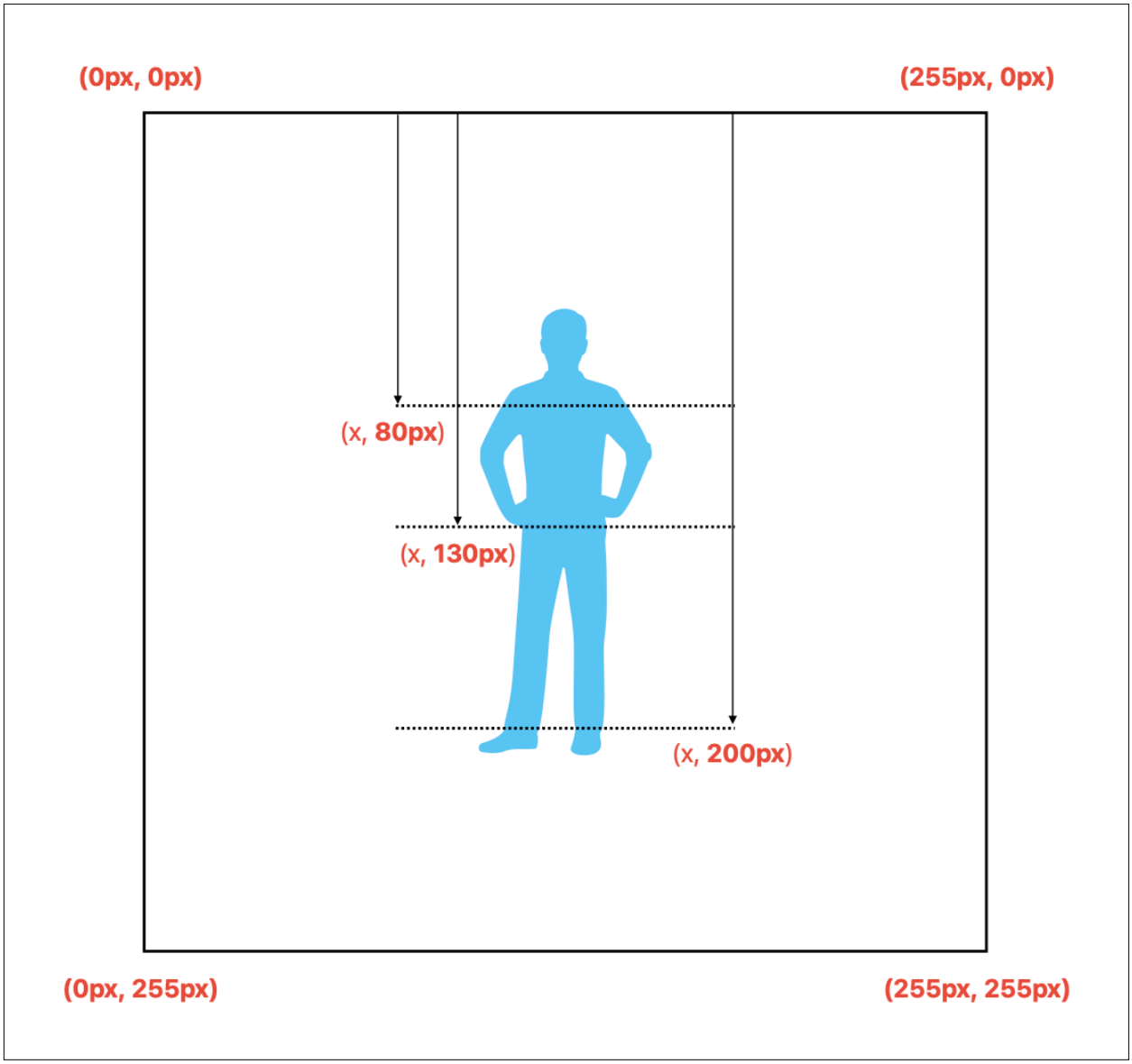
Jednom kada su orijentiri i njihove međusobne udaljenost izračunate, slijedi detekcija čučnja. Detekcija se radi računajući omjere duljine između orijentira i kako se njihove vrijednosti s vremenom mijenjaju. Prvo se računaju duljine rame-gležanj i kuk-gležanj. Gležanj služi kao polazna točka od koje se promatraju sve ostale. Razlog odabira gležnja leži u tome što gležnjevi najmanje mijenjaju svoj položaj tijekom izvođenja čučnjeva. Ramena i kukovi su odabrani jer najviše mijenjaju svoju visinu prilikom izvođenja čučnja i tako pridonose mijenjaju duljina između točaka. Ostale točke se ne uzimaju u obzir jer ne pridonose značajno računanju duljina pa samim time i detekciji čučnja te se ne mogu koristiti s pouzdanošću. Najbolji primjer su ruke koje mogu biti ispružene ispred korisnika, oslonjene na kukove ili korisniku iza glave. Položaj ruku ne ovisi o izvođenju čučnja pa ih nema smisla uračunavati u detekciju čučnja. Glava je izbačena iz računice zbog praktičnih razloga, primjerice ako izađe iz kadra zbog korisnikove visine.

```
try:
    left_shoulder = keypoint_locs[15][1]-keypoint_locs[5][1]
    left_hip = keypoint_locs[15][1]-keypoint_locs[11][1]
    max_ratio_left_hk = ((keypoint_locs[15][1]/255)**3)*(keypoint_locs[15][1]/
                left_shoulder)*(keypoint_locs[15][1]/left_hip)
except:
    logging.warning("%s %s", datetime.datetime.now().strftime('%d%m%Y-%H:%M.%S'),
                " -> Not enough data for left side ratio!")
    max_ratio_left_hk = na_tresh

try:
    right_shoulder = keypoint_locs[16][1]-keypoint_locs[6][1]
    right_hip = keypoint_locs[16][1]-keypoint_locs[12][1]
    max_ratio_right_hk = ((keypoint_locs[16][1]/255)**3)*(keypoint_locs[16][1]/
                right_shoulder)*(keypoint_locs[16][1]/right_hip)
except:
    logging.warning("%s %s", datetime.datetime.now().strftime('%d%m%Y-%H:%M.%S'),
                " -> Not enough data for right side ratio!")
    max_ratio_right_hk = na_tresh
```

Programski kod 5.7. Računanje duljina između orijentira

Kako bismo bolje razumjeli prikazani kod (5.7), prvo trebamo razumjeti koordinatni sustav OpenCV-a i bilo koje druge slike po tom pitanju. Naime, koordinatni sustav kod fotografija je malo izmijenjen i ishodište (koordinate [0, 0]) se nalaze u gornjem lijevom kutu, a maksimalne vrijednosti se nalaze u donjem desnom kutu (u našem slučaju [255, 255]). Formula (5.1) prikazuje računanje duljina između orijentira. U slučaju kada TensorFlow uspije detektirati bitne točke, potrebna je samo y komponenta koordinate, jer x komponenta govori da je korisnik na

lijevoj-desnoj strani kadra ili u centru. X koordinata ne pridonosi računanju dužine između točaka te se zbog toga izostavlja. Pojednostavljeni primjer se može vidjeti na slici 5.3.



Slika 5.3. Prikaz koordinata u slici

$$L_R = G_y - R_y$$
$$L_K = G_y - K_y \qquad \textbf{(5.1)}$$

Gdje su:

L_R	duljina između gležnja i ramena,
L_K	duljina između gležnja i kuka,
G_y	y vrijednost koordinate gležnja,

R_y y vrijednost koordinate ramena,

K_y y vrijednost koordinate kuka.

Sada se s detektiranim orijentirima i izračunatim duljinama mogu izračunati omjeri za detekciju čučnja. Formula je zamišljena da svaki čučanj izazove skok u vrijednosti kako bi granica čučnja mogla lakše prepoznati čučanj. Izračun omjera za detekciju čunja se može vidjeti u izrazu (5.2):

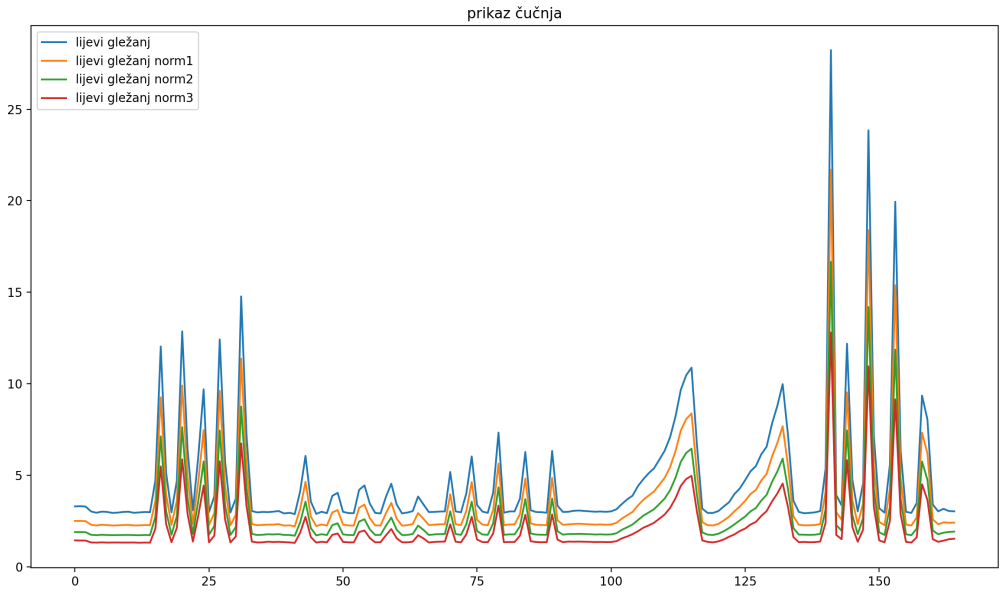
$$omjer = \left(\frac{G_y}{255}\right)^3 \left(\frac{G_y}{L_R}\right)\left(\frac{G_y}{L_K}\right) \qquad \textbf{(5.2)}$$

Gdje je:

G_y y vrijednost koordinate gležnja,

L_R duljina između gležnja i ramena,

L_K duljina između gležnja i kuka.



Slika 5.4. Detekcija čučnjeva i utjecaj normalizacije na vrijednosti

Jednom, kad se izračunaju omjeri, sustav započinje s detekcijom čučnja. Granica čučnja u ovom radu je postavljena na 3.7, a sve granice u intervalu [3.3, 4.0] smatraju se dobrim granicama (više o tome u 6. poglavlju). Odabir same vrijednosti granice ovisit će o poziciji visini kamere, a ako je kamera na većoj visini, poželjno je da granica čučnja bude bliža 4.0, a ako je kamera na nižoj visini, granica čučnja bi trebala biti bliže 3.3. Vrijednost granice od 3.7 je blizu sredini intervala

[3.3, 4.0] i dobro razdvaja čučanj od stojećeg položaja kada se kamera nalazi između 90-110 cm visine. Sustav računa omjere na lijevoj i desnoj strani i provjerava ako su obje strane prešle granicu čučnja. Bez provjere lijeve i desne strane sustav može detektirati čučanj kada korisnik ne radi čučanj (podizanje jedne noge blizu prsa može izazvati slično ponašanje omjera kao i kada se radi pravi čučanj). Problem može nastati ako izračunati omjeri zajedno ne prijeđu granicu čučnja, npr. zbog korištenog modela za detekciju točaka ili ako zbog lošeg osvjetljenja samo jedna strana (omjer) prekorači granicu čučnja, dok se druga strana (omjer) nedovoljno približi granici. Zbog toga je napravljena *tolerance* funkcija koja će kao argument primiti vrijednost čučnja, dozvoljeno odstupanje od granice i granicu čučnja. Funkcija će vratiti *True* čak i kada je omjer manji od granice čučnja, a kada mu se pridoda dozvoljeno odstupanje, tada prelazi granicu.

```
def tolerance(shoulder_knee_ratio, squat_tresh, range):
    if(shoulder_knee_ratio + range >= squat_tresh):
        return True
    if(shoulder_knee_ratio - range >= squat_tresh):
        return True
    return False


if((tolerance(max_ratio_left_hk, squat_tresh, 0) and
    tolerance(max_ratio_right_hk, squat_tresh, 0)) and not squat_flag):
    if((left_ankle-left_hip)/(left_ankle-left_shoulder) <= 0.65 and
        (right_ankle-right_hip)/(right_ankle-right_shoulder) <= 0.65):
        squat_flag = True
        SQUAT_COUNTER += 1
        logging.info("%s %s %f", datetime.datetime.now()
            .strftime('%d%m%Y-%H:%M.%S'), " -> squat, left: ", max_ratio_left_hk)
        logging.info("%s %s %f", datetime.datetime.now()
            .strftime('%d%m%Y-%H:%M.%S'), " -> squat, right: ", max_ratio_right_hk)
        logging.info("%s %s %f", datetime.datetime.now()
            .strftime('%d%m%Y-%H:%M.%S'), " -> Squat counter: ", SQUAT_COUNTER)

if((max_ratio_left_hk < stand_tresh or max_ratio_right_hk < stand_tresh)
    and squat_flag):
    squat_flag = False
    logging.info("%s %s %f", datetime.datetime.now()
        .strftime('%d%m%Y-%H:%M.%S'), " -> stand, left: ", max_ratio_left_hk)
    logging.info("%s %s %f", datetime.datetime.now()
        .strftime('%d%m%Y-%H:%M.%S'), " -> stand, right: ", max_ratio_right_hk)
```

Programski kod 5.8. Funkcija tolerance i provjera za detekciju čučnja

5.2.3. Detekcija naklona

Nakon što sustav detektira čučanj dolazi još jedna provjera vrijednosti koja će provjeriti radili li se uistinu o čučnju. Naime, korisnik može prevariti sustav tako da se nakloni te da taj naklon uzrokuje dovoljan skok u vrijednostima omjera kako bi se detektirao lažni čučanj. U toj dodatnoj provjeri radi se novi omjer pomoću duljina kuk-gležanj i rame-gležanj. Ako je uvjet zadovoljen,

tek onda se detektira čučanj. Provjera naklona je nešto jednostavnija od one koja se koristi za samu detekciju čučnja gdje je dovoljno dobiti duljine između gležnjeva, kukova i ramena, te se nakon dobivenih duljina podijeli duljina kuk-gležanj s duljinom rame-gležanj (formula 5.3).

$$\frac{G_y - K_y}{G_y - R_y} \leq 0.65 \qquad \textbf{(5.3)}$$

Gdje je:

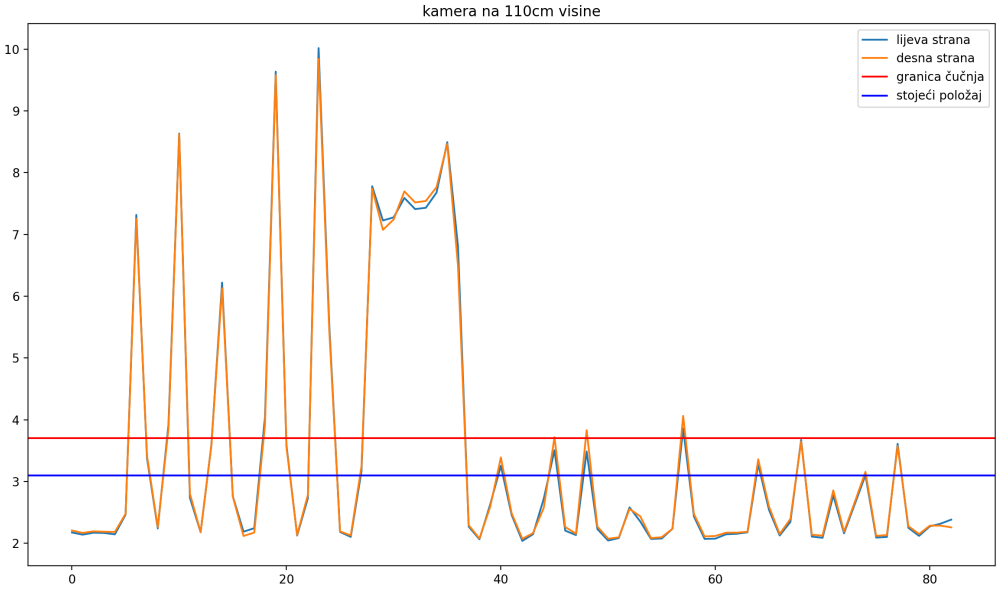
G_y	y vrijednost koordinate gležnja,
K_y	y vrijednost koordinate kuka,
R_y	y vrijednost koordinate ramena.

Ako je dobiveni omjer manji ili jednak 0.65 tada možemo s pouzdanošću reći da se radi o čučnju. Detekcijom čučnja ažuriraju se vrijednosti zastavica i varijabli za praćenje broja čučnjeva koje preuzima app skripta svaki put kada šalje odgovor na zahtjev *frontenda*.

Za kraj, da bismo u potpunosti razumjeli programski kod 5.8 moramo objasniti čemu služe zastavice (engl. Flags) i zašto se koristi provjera stojećeg položaja. Provjere u programskom kodu 5.8 neprestano se izvršavaju u *while* petlji. Ako korisnik napravi i ostane u pozi čučnja, uvjet za detekciju čučnja će uvijek biti zadovoljen unutar petlje, a petlja se tako pretvara u brojač sličica u sekundi koje je TensorFlow uspio obraditi. Kako bismo to izbjegli, u prvu se provjeru dodaje *squat_flag* varijabla čija se vrijednost promijeni jednom kada se detektira čučanj. Ako korisnik odluči ostati u pozi čučnja duže vrijeme, čučanj će se brojati samo jednom - sve dok se *squat_flag* zastavica ne poništi tj. dok se korisnik ne vrati u stojeći položaj.

Provjera čučnja, koristeći zastavice i granice stojećeg položaja i čučnja, na prvu se može učiniti nepotrebnim, odnosno da se isti rezultat može postići samo granicom čučnja i zastavicom. Ipak, *TensorFlow* prilikom detekcije točaka može vratiti različite vrijednosti za istu pozu kada se obrađuju slike. Slika 5.5 prikazuje pet kratkih čučnjeva, jedan dulji čučanj, pet naklona i pet polovičnih čučnjeva. Obratite pozornost kako šesti čučanj (dulji) ima oscilaciju u vrijednostima iz slike u sliku, iako korisnik zapravo miruje u pozi čučnja. Zbog tog rubnog slučaja koristi se provjera tranzicije iz čučnja u stojeći položaj, jer ako korisnik ostane u vremenski duljoj pozi

čučnja, a dobivene vrijednosti osciliraju oko granice čučnja iz slike u sliku, tu bi se mogao detektirati isti čučanj više puta.



Slika 5.5. Prikaz čučnjeva i oscilacije u vrijednostima

6. USPOREDBA RAZLIČITIH TF MODELA I GRANICA

Do sada smo prolazili kroz dijelove sustava i često spominjali *TensorFlow*. Sada ćemo detaljnije pogledati modele koji ga pokreću, kao i odabrane parametre koji se koriste prilikom detekcije čučnja. Postoje dvije vrste TensorFlow modela za detekciju ljudskih poza:

- PoseNet, stariji model izašao 2017. godine
- MoveNet, trenutno najnoviji model i koji je korišten tijekom ovog rada.

6.1. MoveNet

MoveNet model dolazi u dvije varijante: *Lightning* i *Thunder*. Lightning je manji, brži i često manje točniji model od Thunder varijante. Iako je Thunder teža i sporija verzija Lightning modela, tako je ipak korisnija za aplikacije koje traže veću točnost detektiranih točaka.
MoveNet je konvolucijska neuralna mreža (engl. Convolutional Neural Network - CNN) koja na ulaz prima RGB (engl. Red Green Blue) sliku, a kao izlaz vraća položaj zglobova na osobi. Rezolucija slike ovisi o korištenoj verziji MoveNeta. Lakša, Lightning verzija koristi oblik slike 192×192×3, dok Thunder koristi 256×256×3 oblik. 192 ili 256 predstavljaju broj piksela, a tri predstavlja broj kanala (boja) unutar slike.
Oba modela vraćaju kao rezultat float32 tenzor oblika [1, 1, 17, 3] gdje prve dvije vrijednosti predstavljaju xy koordinate, treća predstavlja 17 orijentira na korisniku (redoslijedom: nos, lijevo oko, desno oko, lijevo uho, desno uho, lijevo rame, desno rame, lijevi lakat, desni lakat, lijevo zapešće, desno zapešće, lijevi kuk, desni kuk, lijevo koljeno, desno koljeno, lijevi gležanj, desni gležanj) i zadnja vrijednost koja predstavlja razinu pouzdanosti predviđanja.

Varijante Lightning i Thunder modela još se dijele na dvije podvarijante koje koriste int8 ili float16 tip podataka. Int i float slično utječu na performanse kao i rezolucija gdje je int8 model nešto brži pod cijenu točnosti, dok je float16 model točniji, ali ipak nešto sporiji. Neovisno o podvarijanti i odabranom modelu, ako se unese pravilan naziv MoveNet modela, sustav će prepoznati o kojem je modelu riječ i tako prilagoditi rezoluciju kamere.

Model	Veličina (MB)	Točnost	Kašnjenje (ms)		
			Pixel 5 - CPU	Pixel 5 - GPU	RPi 4 - CPU
MoveNet - Thunder (FP16)	12.6 MB	72.0	155ms	45ms	594ms
MoveNet - Thunder (INT8)	7.1 MB	68.9	100ms	52ms	251ms
MoveNet - Lightning (FP16)	4.8 MB	63.0	60ms	25ms	186ms
MoveNet - Lightning (INT8)	2.9 MB	57.4	52ms	28ms	95ms
PoseNet (FP32)	13.3 MB	45.6	80ms	40ms	338ms

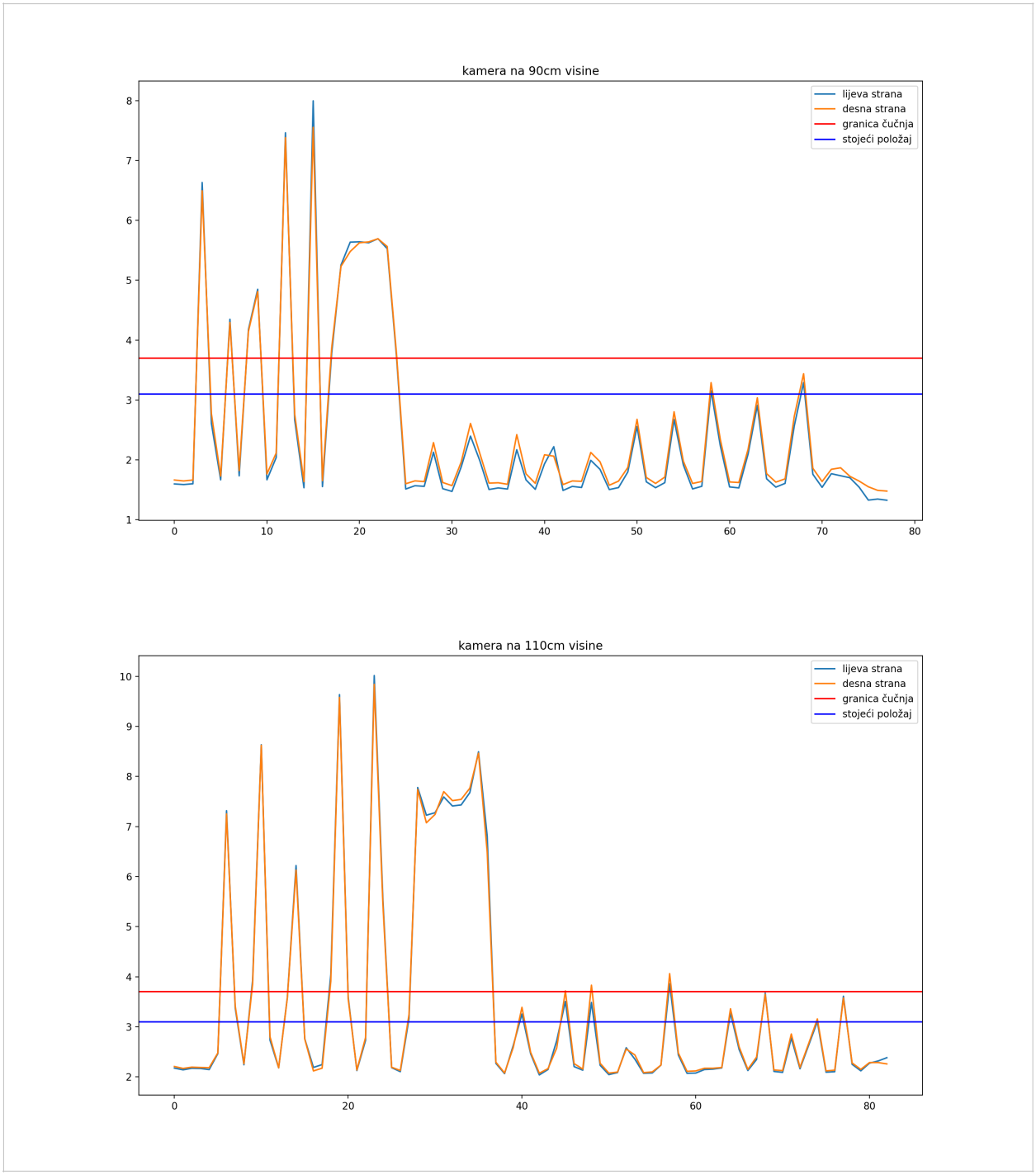
Tablica 6.1. Usporedba različitih TF modela i performansi na različitim uređajima

6.2. Odabrane granice

Kako bismo detektirali čučanj koristimo tri granice: granicu čučnja, granicu stojećeg položaja i granicu naklona. Granica čučnja detektira sam čučanj, a preostale dvije služe za rubne slučajeve kada točke osciliraju blizu granice čučnja ili kada korisnik s naklonom želi prevariti sustav. Kroz detaljnu analizu se došlo do granice čučnja koja u rasponu između 3.3 i 4.0 dobro detektira čučanj. Razlog zašto je u pitanju raspon, a ne konkretna vrijednost, ovisi o nekoliko čimbenika kao što su - korišteni MoveNet model, izvor svjetlosti (svjetlini prostorije) i položaj (visina) kamere. Ako se granica poveća, uvjet za detekciju je postrožen i neki čučnjevi mogu biti nedetektirani. Ako se granica spusti, uvjet je ublažen i polovični čučnjevi se detektiraju kao pravi čučnjevi. Odabrana granica 3.7 je u sredini raspona i dobro detektira čučnjeve.

Za granicu stojećeg položaja odabrana je vrijednost 3.1. Što se ove granice tiče poželjno je da bude strogo manja od granice čučnja jer je u suprotnom nepotrebna i sustav bi se ponašao jednako kao da je i nema. Osim što je poželjno da granica bude manja od granice čučnja, potrebno je da između njih bude razmak od barem 10% upravo zbog oscilacije u detektiranim vrijednostima.

Granica naklona ima fiksnu vrijednost od 0.65 i njena vrijednost je odabrana grafičkom analizom (slika 6.3).



Slika 6.1. Granice čučnja i stojećeg položaja na 2 različite visine kamere

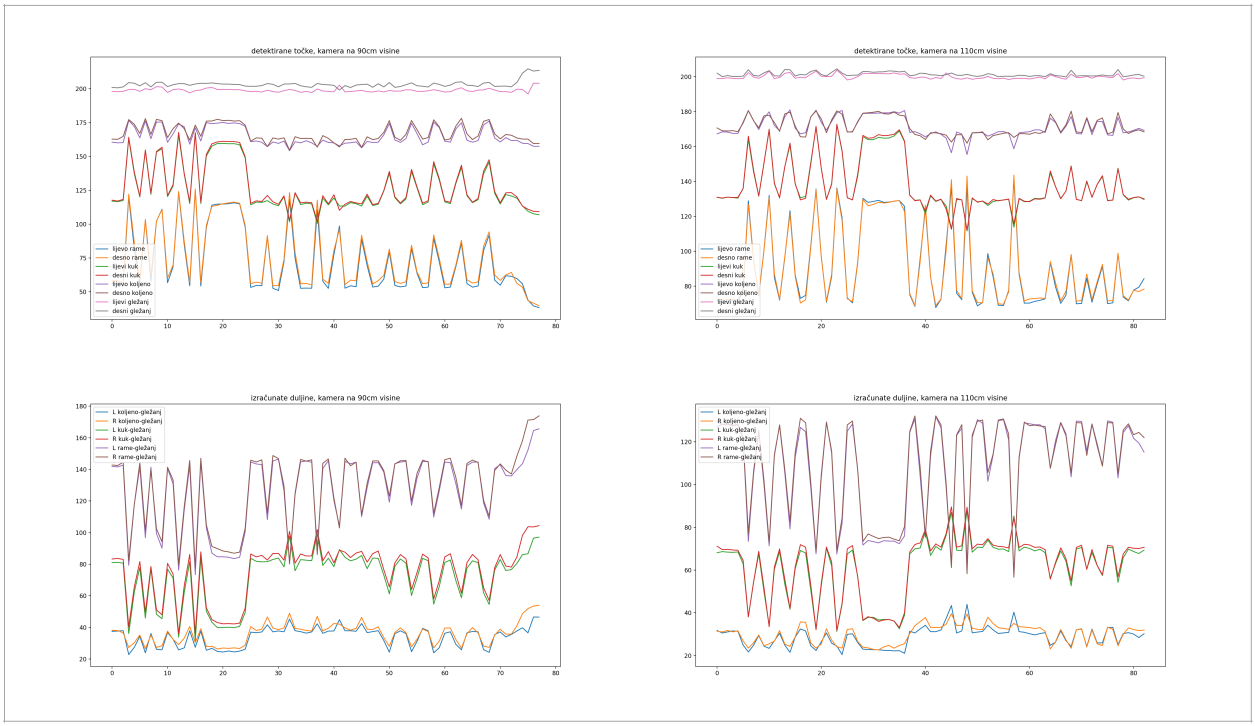
Na slici 6.1. mogu se vidjeti dvije serije čučnjeva na visinama od 90 cm i 110 cm. Redoslijed čučnjeva je redom bio: pet običnih čučnjeva, poza čučnja od pet sekundi, pet naklona i pet polovičnih čučnjeva. Sustav je na kraju uspješno izbrojao šest čučnjeva u oba slučaja, ali bitno je uočiti kako visina kamere može utjecati na TF predikciju točaka i samim time detekciju čučnja. Da je granica bila niže vrijednosti kada je kamera bila povišena, detektiralo bi polovične čučnjeve kao prave.

Razlog različitih vrijednosti, na različitim visinama, u percepciji je kamere i dobivenih duljina između detektiranih točaka. Na slici 6.2. se može vidjeti kako povećanje u visini kamere utječe na detekciju točaka i duljina koje se koriste za izračun omjera. Ako je kamera na nižoj visini, korisnik se mora približiti kameri kako bi stajao na indikacijskoj liniji, a ako se kamera namjesti na veću visinu, korisnik se mora udaljiti od kamere kako bi se našao na indikacijskoj liniji. Ta promjena udaljenosti korisnika od 0.5-1 metra od kamere direktno utječe na detekciju točaka, odnosno, kako je kamera na većoj visini, udaljenosti između detektiranih točaka bit će nešto manje nego kada je kamera na nižoj visini. Kako omjere računamo tako da dobivene duljine između točaka budu u nazivniku, smanjivanjem duljina ćemo proporcionalno povećati rezultat omjera (6.1).

$$omjer = \left(\frac{G_y}{255}\right)^3 \left(\frac{G_y}{L_R}\right)\left(\frac{G_y}{L_K}\right) \qquad \textbf{(6.1)}$$

Gdje je:

G_y	y vrijednost koordinate gležnja,
L_R	duljina između gležnja i ramena,
L_K	duljina između gležnja i kuka.



Slika 6.2. Promjene u vrijednosti točaka i duljina na različitoj visini kamere

Jednom, kada sustav detektira čučanj, provjerit će radi li se stvarno o čučnju ili o naklonu korisnika. Detekcija naklona radi se prema izrazu (6.2). Ako su vrijednosti manje od 0.65 sustav će detektirati čučanj, u suprotnom, sustav će misliti da se radi o naklonu.

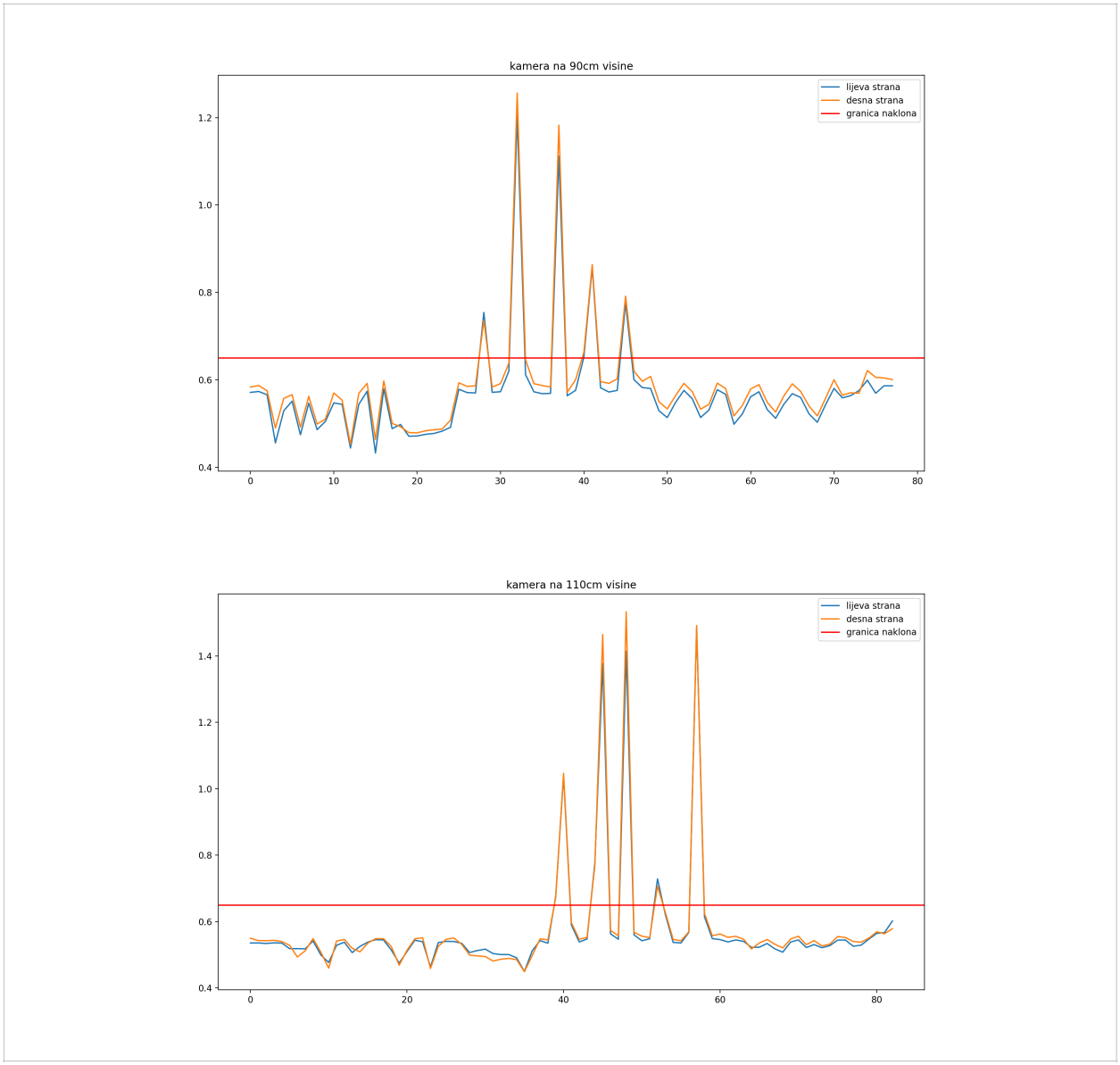
$$\frac{G_y - K_y}{G_y - R_y} \leq 0.65 \qquad \textbf{(6.2)}$$

Gdje je:

G_y y vrijednost koordinate gležnja,

K_y y vrijednost koordinate kuka,

R_y y vrijednost koordinate ramena.



Slika 6.3. Granica naklona na 2 različite visine kamere

7. ZAKLJUČAK

Tema ovog rada bila je izrada cjelovitog rješenja na Raspberry Pi računalu koji će brojati čučnjeve koristeći TensorFlow i OpenCV knjižnice. Izrada podrazumijeva korisničko sučelje kojem se pristupa preko lokalne mreže i/ili localhosta, poslužiteljskih skripti koje detektiraju korisnika i čučanj, te analiza vrijednosti dobivenih tijekom razvoja i odabira najboljih vrijednosti parametara koji će detektirati čučanj.

Napravljeni sustav uspješno detektira čučnjeve, ali s vremenom nastupa degradacija performansi zbog neadekvatnog hlađenja. Ako se sustav koristi u dugim sesijama, temperatura SoC-a dosegne svoj limit od 85° te se počinje pojavljivati termalno prigušenje koje uzrokuje pad broja obrađenih sličica u sekundi na klijentskoj i poslužiteljskoj strani sustava. Pad FPS-a uzrokuje uglavnom manje neugodnosti kao *stutter* te u ekstremnim slučajevima, kada se može propustiti više sličica za obradu za redom, neuspješno detektirani sam čučanj. Sustav najbolje detektira točke i čučanj kada se koriste teži *TensorFlow* modeli (*Thunder* i FP16) koji upravo stvaraju najveći stres na SoC tako da se za najbolje izvođenje sustava preporučuje korištenje aktivnog hlađenja.

Osim *TensorFlow* modela i termalnog prigušenja, na detekciju čučnja mogu utjecati i odabrane granice čučnja, položaj kamere, osvjetljenje okoline te i sam odabir metode detekcije čučnja. U ovom radu je korištena metoda računanja omjera dobivenih duljinama između korisnikovih orijentira. Druga metoda, koja bi mogla dati jednako dobru detekciju čučnja, računanje je kuteva između tri točke. Ta metoda ne bi trebala imati višestruke provjere čučnja (čučanj i naklon), kao metoda korištena u ovom radu, nego samo dobro definiranu granicu čučnja. Postavlja se samo pitanje koliko bi bio točan izračun kuteva ako je korisnik točno okrenut prema kameri i ako se sve točke nalaze na istoj ravnini bez informacije o dubini tih točaka.

Za kraj ostaje reći da napravljeni sustav dobro detektira čučnjeve kada se uzme u obzir sklopovlje, kvaliteta slike, model i metoda za detekciju orijentira i čučnjeva. Do prije nekoliko godina, projekt ovakvih razmjera je mogao biti izvediv samo na puno skupljim računalima ili posebnom sklopovlju predviđenom za strojno učenje.

8. SAŽETAK

Tema rada izrada je cjelovitog rješenja brojača čučnjeva na Raspberry Pi računalu pomoću postojećih TensorFlow i OpenCV knjižnica i MoveNet modela strojnog učenja. Detaljno su se opisali korišteni alati i knjižnice te objasnili dijelovi programskog koda koji pristupaju kameri i detektiraju čučanj.

***Ključne riječi* — Raspberry Pi, Python, OpenCV, TensorFlow, MoveNet, Flask, detekcija položaja**

ABSTRACT

The topic of this paper is the creation of a complete squat counter solution on a Raspberry Pi computer using the existing TensorFlow and OpenCV libraries and MoveNet machine learning models. The tools and libraries used were described in detail as well as parts of the program code that access the camera and detect squat.

***Keywords* — Raspberry Pi, Python, OpenCV, TensorFlow, MoveNet, Flask, pose detection**

Literatura

[1] Toporov, C.; “Squats detector with OpenCV and Tensorflow”, s Interneta, https://towardsdatascience.com/squats-detector-with-opencv-and-tensorflow-ce934f19aeb9, 23. lipnja 2020.

[2] “Single-board computer”, s Interneta, https://en.wikipedia.org/wiki/Single-board_computer

[3] “Raspberry Pi Documentation”, s Interneta, https://www.raspberrypi.com/documentation/computers/raspberry-pi.html

[4] “Raspberry Pi 4 B (Broadcom BCM2711) Benchmark, Test and specs”, s Interneta, https://www.cpu-monkey.com/en/cpu-raspberry_pi_4_b_broadcom_bcm2711, 21. veljače 2022.

[5] “Broadcom VideoCore VI”, s Interneta, https://www.cpu-monkey.com/en/igpu-broadcom_videocore_vi-221

[6] “VideoCore”, s Interneta, https://en.wikipedia.org/wiki/VideoCore

[7] “Raspberry Pi OS”, s Interneta, https://www.raspberrypi.com/documentation/computers/os.html

[8] “Raspberry Pi OS”, s Interneta, https://en.wikipedia.org/wiki/Raspberry_Pi_OS

[9] “Python Programming Language”, s Interneta, https://www.geeksforgeeks.org/python-programming-language/, 23. veljače 2023.

[10] “Introduction to TensorFlow”, s Interneta, https://www.tensorflow.org/learn

[11] “TensorFlow Lite”, s Interneta, https://www.tensorflow.org/lite/guide

[12] “Pose estimation”, s Interneta, https://www.tensorflow.org/lite/examples/pose_estimation/overview

Popis oznaka i kratica

API	Application Programming Interface
ARM	Advanced RISC Machines
BLE	Bluetooth Low Energy
CPU	Central Processing Unit
CSI	Camera Serial Interface
CSV	Comma Separated Value
CUDA	Compute Unified Device Architecture
DL	Deep Learning
DSI	Display Serial Interface
FOV	Field Of View
FPS	Frames Per Second
FTP	File Transfer Protocol
GB	Gigabyte
GPIO	General Purpose Input/Output
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HDMI	High Definition Multimedia Interface
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IOT	Internet of Things
MICRO SD	Micro Secure Digital
ML	Machine Learning
OPENCL	Open Computing Language
OPENCV	Open Source Computer Vision Library
PC	Personal Computer
PCB	Printed Circuit Board
POE	Power Over Ethernet
PY	Python
RPI	Raspberry Pi
SBC	Single Board Computer
SFTP	Secure File Transfer Protocol

SOC	System on Chip
SSH	Secure Shell
TF	TensorFlow
URL	Uniform Resource Locator
USB	Universal Serial Bus
VNC	Virtual Network Computing
VS CODE	Visual Studio Code
WIFI	Wireless Fidelity

Popis slika, tablica i programskog koda

Slike

Slika 2.1. Raspberry Pi 4 Model B s pasivnim hladnjakom

Slika 3.1. Prikaz Raspberry Pi OS-a s otvorenim Chromium preglednikom

Slika 4.1. Logički prikaz sustava

Slika 4.2. Prikaz sučelja u slobodnom modu

Slika 4.3. Prikaz ljestvice

Slika 5.1. Pojednostavljeni prikaz odvojenosti streamova na frontendu i backendu

Slika 5.2. Detektirani orijentiri s translatacijom i signalnom linijom

Slika 5.3. Prikaz koordinata u slici

Slika 5.4. Detekcija čučnjeva i utjecaj normalizacije na vrijednosti

Slika 5.5. Prikaz čučnjeva i oscilacije u vrijednosti

Slika 6.1. Granice čučnja i stojećeg položaja na 2 različite visine kamere

Slika 6.2. Promjene u vrijednosti točaka i duljina na različitoj visini kamere

Slika 6.3. Granica naklona na 2 različite visine kamere

Tablice

Tablica 2.1. Usporedba dvije verzije Raspberry Pi-a

Tablica 2.2. Specifikacije kamere

Tablica 6.1. Usporedba različitih TF modela i performansi na različitim uređajima

Programski kod

Programski kod 5.1. Postavljanje i pokretanje Flask poslužitelja

Programski kod 5.2. Povezivanje URL zahtjeva s odgovarajućim funkcijama

Programski kod 5.3. Ostale funkcije app.py skripte koje obrađuju klijentske zahtjeve

Programski kod 5.4. Definicija VideoStream klase

Programski kod 5.5. Funkcija za translataciju točaka (run_inference) i funkcija zaključka (movenet)

Programski kod 5.6. Funkcija koja crta signalnu liniju na sliku i prosljeđuje je klijentskoj strani

Programski kod 5.7. Računanje duljina između orijentira

Programski kod 5.8. Funkcija tolerance i provjera za detekciju čučnja