Brojač ponavljanja vježbi korištenjem računalnog vida

Orlić, Mauro

Sveučilište u Rijeci – Odjel za informatiku

Informacijski i komunikacijski sustavi

Mauro Orlić

Brojač ponavljanja vježbi korištenjem računalnog vida

Diplomski rad

Mentor: doc. dr. sc. Miran Pobar

Rijeka, prosinac 2020

ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

Rijeka, 01.06.2020.

Zadatak za diplomski rad

Pristupnik: Mauro Orlić

Naziv diplomskog rada: Brojač ponavljanja vježbi korištenjem računalnog vida

Naziv diplomskog rada na eng. jeziku: Computer vision-based exercise reptetition counter

Sadržaj zadatka:

Predstaviti mogućnosti brojanja ponavljanja vježbi korištenjem algoritama iz područja računalnog vida poput algoritma za procjenu optičkog toka ili drugih algoritama za procjenu pokreta. Opisati primjere postojećih rješenja. Implementirati i opisati prototip rješenja za automatsko brojanje ponavljanja vježbi jedne osobe na video snimci snimljenoj nepomičnom kamerom. Testirati rješenje na skupu video snimaka izvođenja čučnjeva i sklekova.

Mentor: doc. dr. sc. Miran Pobar

Voditeljica za diplomske radove: izv. prof. dr. sc. Ana Meštrović

Komentor:

Zadatak preuzet: 01.06.2020.

(potpis pristupnika)

Orlić

Sadržaj

Sažetak

Računalni vid je područje podatkovnih znanosti koje je unutar zadnjih par desetljeća značajno napredovalo zahvaljujući raznim tehničkim napredcima. Postoji mnogo rješenja za različite tipove problema, od jednostavnih algoritama koji izvlače površne značajke slika do složenih konvolucijskih neuronskih mreža. Računalni vid danas ima razne primjene u područjima poput zdravstva, transporta, proizvodnje i sporta. U ovom radu obrađujemo postupak kojim pomoću računalnog vida rješavamo problem brojanja repeticija tjelovježbi koristeći dva pristupa. Jedan pristup se oslanja na gusti optički tok, a drugi koristi detekciju ljudske poze. Rješenja su implementirana uporabom Python programskog jezika te biblioteke OpenCV i Googleove Mediapipe biblioteke za procjenu ljudske poze.

Ključne riječi

OpenCV, optički tok, procjena ljudske poze, Mediapipe, Farneback, računalni vid, prebrojavanje vježbi

1 Uvod

Kako tehnologija napreduje, sve više i više ljudi očekuje da se svaki aspekt njihovog života može pratiti, bilježiti i na pristupačan način prezentirati. Postoji mnogo takvih tehnologija: osobni asistenti (Siri, Cortana, Alexa…), aplikacije za praćenje unesenih nutrijenata i fitness narukvice za praćenje raznih indikatora zdravlja. Jedno područje koje nije dovoljno istraženo jest brojanje ponavljanja tjelovježbi. U ovom radu ćemo istražiti kako riješiti taj problem uporabom računalnog vida uporabom dvaju pristupa: jedan koji koristi gusti optički tok i drugi koji koristi procjene ljudske poze.

U prvom dijelu rada detaljnije opisujemo sam problem koji ćemo pokušati riješiti i ukratko objasniti područje računalnog vida te koje su česte primjene. Istražujemo i druge radove koji su se bavili sličnom problematikom te kako su oni pristupili problemu i riješili ga. Problemi koje vezani radovi rješavaju nisu nužno isti kao i problem koji mi pokušavamo riješiti ali imaju neke aspekte ili inspiracije koje bi se mogle primijeniti na rješavanje našeg problema.

Nakon toga se opisuju dva odabrana generalna pristupa kojima će se rješavati problem. Opisuje se opća teorija za svaki pristup i kako funkcionira. Također se objašnjava nekoliko relevantnih tehničkih termina i njihovo značenje u kontekstu pristupa. Opisuju se i konkretni algoritmi/rješenja koja će se koristiti te njihove prednosti, mane i osobitosti.

Prije same implementacije navodimo računalno okruženje u kojemu će se rješenje implementirati te koji programi i alati će se koristiti i zašto. Naveden je programski jezik i ključne biblioteke potrebne za implementaciju te korištene verzije istih.

U poglavlju koje se odnosi na samu implementaciju prvo opisujemo generalna pravila koja će se koristiti kod oba pristupa i koji će biti glavni izazovi kod implementacije. Poslije toga se u dubinu objašnjavaju konkretna pravila korištena u svakom od pristupa i demonstriraju njihove implementacije u samom kodu.

Nakon toga se opisuje način na koji se testirala efektivnost obje implementacije. Opisan je skup podataka nad kojim smo testirali implementacije, način na koji su podaci sakupljeni te njihova svojstva. Dobiveni rezultati testova se tada dalje razrađuju i analiziraju te dolazimo do raznih zaključaka o efektivnostima dvaju pristupa ovisno o raznim parametrima i na kraju ih međusobno usporedimo.

2 Automatizacija brojanja ponavljanja vježbi

2.1 Opis Problema

Tjelovježba je fizička aktivnost čiji je cilj na siguran i efikasan način poboljšati fizičku snagu, izdržljivost, zdravlje i kontrolu nad mišićima osobe. Da bi vježbanje bilo efikasno razvijeno je mnogo programa i planova vježbanja koji zahtijevaju da se određena vježba ponavlja određeni broj puta bez zaustavljanja. Obavljanje određenog broja ponavljanja neke vježbe bez prestanka zovemo jednom serijom te vježbe.

Te repeticije i serije je potrebno brojati ukoliko osoba želi postići dobre i konzistentne rezultate. Ova naizgled trivijalna akcija je značajno teža tijekom izvođenja vježbi pošto se izvođač mora istovremeno koncentrirati na pravilno izvođenje svake repeticije i brojanja repeticija.

Da bi oslobodio dio koncentracije koju izvođač može upotrijebiti za izvođenje same vježbe bilo bi potrebno automatizirati proces brojanja repeticija na način koji je jeftin i dostupan svima. Zbog tog razloga potrebno je rješenje koje može pomoću izlaznog signala videokamere brojati repeticije. Ne smije zahtijevati posebnu opremu. Također je bitno da program nije računalno zahtjevan, odnosno da se može izvršavati i na mobilnom uređaju. (Khurana, i dr. 2018)

U ovom radu cilj je ostvariti automatsko brojanje repeticija korištenjem računalnog vida koje se izvršava pomoću procesora na osobnom računalu.

2.2 Područje računalnog vida

Računalni vid je interdisciplinarno područje računalnih znanosti čiji je cilj kroz slike i/ili video zapise opisati viđeni svijet i njegova svojstva.

Česte primjene uključuju:

- **OCR** - optičko prepoznavanje znakova odnosno teksta (*engl. optical character recognition*)
- **3D rekonstrukcija** - trodimenzionalno modeliranje objekata i scena uporabom dvodimenzionalnih videa ili slika iz više perspektiva
- **Klasifikacija objekta** – klasifikacija objekta na slici, određivanje vjerojatnosti da se

Računalni vid ima mnoge pod-domene koje se bave različitim problemima ali za našu primjenu interesantna su nam domene vezane za procjenu kretnje i procjenu ljudske poze (pozicije zglobova i udova ljudskog tijela). (Klette i Reinhard 2014) (Shapiro i Stockman 2001) (Szeliski 2011)

2.3 Vezani radovi

Automatizacija brojanja repeticija vježbi nije nov problem te postoji mnogo rješenja s različitim pristupima, razinama kompleksnosti, efektivnosti i računalne zahtjevnosti. Neki od čestih pristupa uključuju detekciju kretnje, detekciju ljudske poze, pristupe koji koriste konvolucijske neuronske mreže.

2.3.1 Gymcam

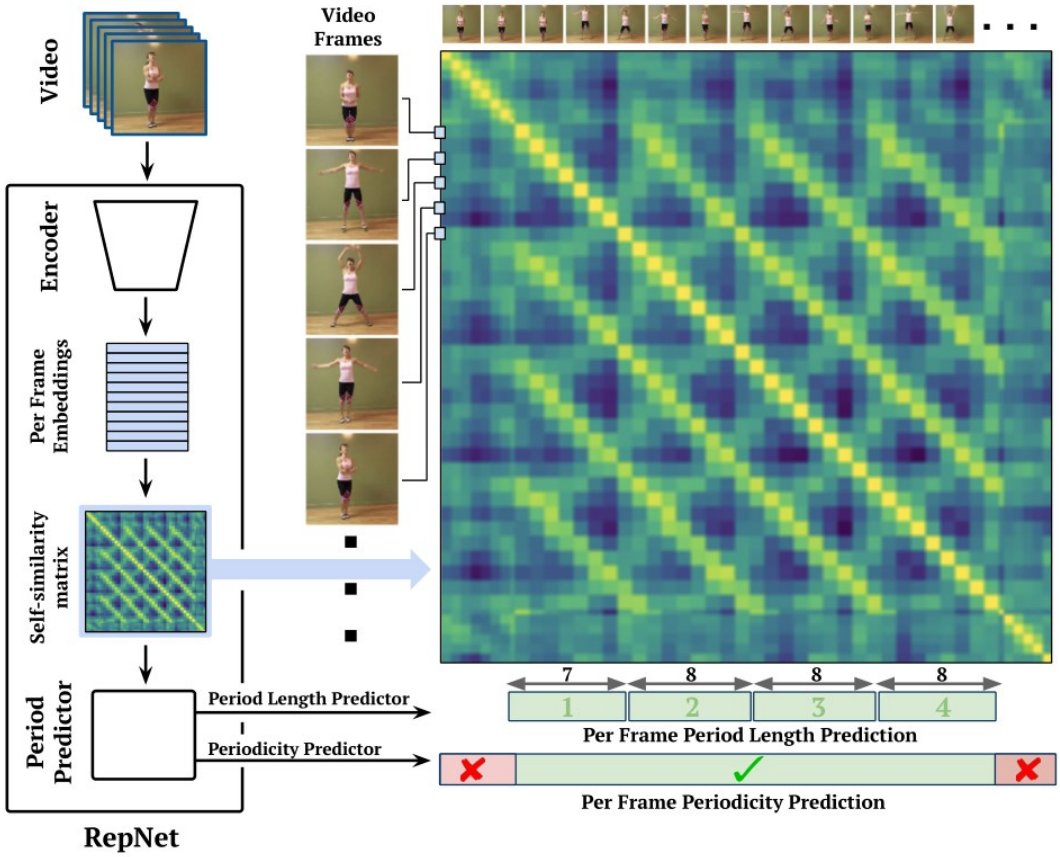
GymCam je sustav za automatsku detekciju, klasifikaciju i praćenje vježbi sa više ljudi odjednom koristeći jednu normalnu videokameru. Bazira se na principu detekcije kretnje i pretpostavci da je svaka repetitivna akcija u teretani vježba. Glavni cilj sustava je dati svakom korisniku mogućnost praćenja obavljenih tjelovježbi i vezanih statistika. (Khurana, i dr. 2018)

Slika 1 Vizualni prikaz rada sustava GymCam

Sa razvijenim sustavom uspijevaju razlikovati vježbe od ostalih aktivnosti sa točnošću od 84.6%, klasificiraju tip vježbe sa 93.6% i brojati repeticije sa prosječnim odstupanjem od ±1.7 repeticija. Na slici 1 možemo vidjeti vizualizaciju rada sustava GymCam. Skup podataka na temelju kojega je sustav izgrađen se sastoji od 42 sata videa vježbanja u teretani od kojih je 15 sati ručno označeno i sadrži 597 instanca vježbanja. U suštini sustav funkcionira na način da prate veliki broj točaka na videu pomoću optičkog toka tražeći repetitivne kretnje kroz vrijeme. Točke se grupiraju u klastere ovisno o međusobnoj blizini i o tome koliko imaju sličan ritam i smjer kretnje. Nakon toga se klasificira da li su dani repetitivni pokreti vježba ili ne. (Khurana, i dr. 2018)

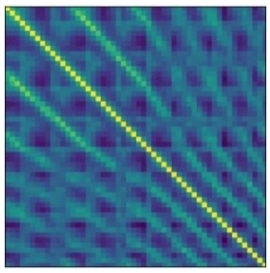
2.3.2 RepNet

RepNet je arhitektura neuronske mreže koja procjenjuje broj ponavljanja neke akcije u videu. Dizajnirana je na način da može pratiti bilo kakvu vrstu ponavljanja, odnosno nije trenirana ni za jedan specifičan tip akcije. (Dwibedi, i dr. 2020)

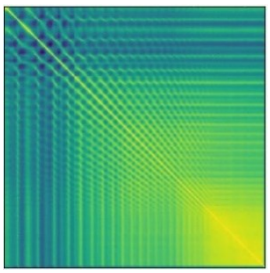


Slika 2 Pregled RepNet arhitekture

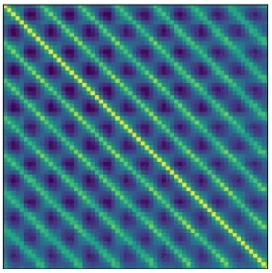
To postiže uporabom tzv. matrica vremenske samo-sličnosti matrica (engl. *Temporal Self-similarity matrices*). Za neki video zapis uzme se niz svih kadrova i odredi se sličnost sa svakim drugim kadrom. Na slici 2 Slika 2gore vidimo prikaz matrice samo-sličnosti za video osobe koja ponavlja skok s raznožajem. Žuta boja označava visoku sličnost između kadrova, dok tamno plava označava nisku sličnost. Na slici također možemo primijetiti iznimno žutu dijagonalnu liniju iz gornjeg lijevog kuta u donji desni. Ta linija označava sličnost svakog kadra sa samim sobom. Ostale paralelne dijagonale predstavljaju različite kadrove u kojima je osoba koja vježba u istoj pozi različitih ponavljanja vježbe. Vodoravna udaljenost između tih paralelnih žutih dijagonala predstavlja vrijeme potrebno za izvršavanje jedne repeticije, a time i samu frekvenciju i broj ponavljanja. (Dwibedi, i dr. 2020)



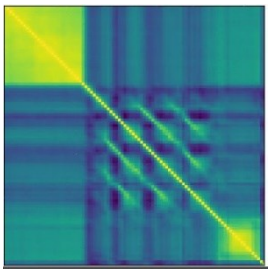
Slika 3 Bacanje kladiva



Slika 4 Odskakutanje lopte



Slika 5 Skakanje



Slika 6 Miješalica

Na gornjim slikama možemo vidjeti još nekoliko primjera repetitivnih akcija, slijeva na desno:

- Atletičara koji sve brže i brže vrti kladivo oko sebe, sa svakim okretom ima sve manji period (Slika 3)
- Lopta koja skakuće, s vremenom gubi energiju i povećava frekvenciju skakutanja lopte (Slika 4)
- Ujednačeno skakanje na mjestu (Slika 5)
- Miješalica cementa koja iz stacionarne pozicije obavi nekoliko okretaja te ponovno stane (Slika 6)

3 Odabrani pristupi problemu

Postoji mnogo mogućih pristupa za rješavanje problema brojanja ponavljanja tjelovježbe, ali većina njih je ili previše računalno zahtjevna ili zahtijeva preveliki veliki skup podataka koji jedna osoba ne može samostalno sakupiti u razumnom vremenu. Takav preduvjet automatski isključuje ručno trenirane neuronske mreže, ali je i dalje moguće koristiti već gotova rješenja koja nisu iznimno zahtjevna. Za rješavanje problema odabrana su dva pristupa koja će se odvojeno implementirati. Jedno koje koristi gusti optički tok za detekciju pokreta i drugo koje koristi gotovo rješenje za detekciju poze ljudskog tijela.

Svaki od pristupa ima svoje prednosti i mane ovisno o raznim varijablama koje će se opisati u daljnjem tekstu. Također svaka implementacija ima dodatno ugrađena pravila kako bi se poboljšala točnost.

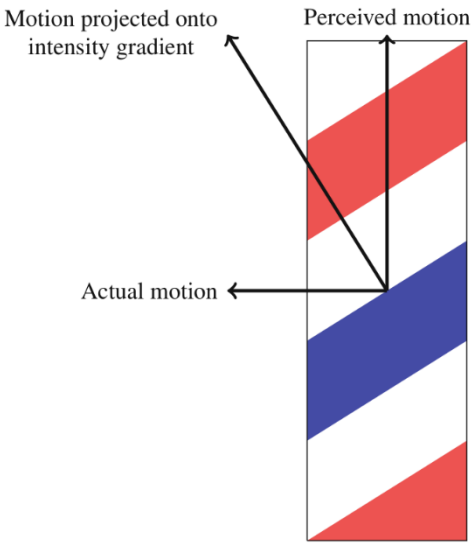
3.1 Optički tok

U knjizi *Optical Flow and Trajectory Estimation methods* autori Joel Gibson i Oge Marques opisuju optički tok (engl. o*ptical flow*) na slijedeći način:

Optički tok možemo razmatrati kao projekciju trodimenzionalnog kretanja na dvodimenzinalnu površinu (sliku). Generalno zadatak nam je na temelju dvodimenzionalnih projekcija fizičkih objekta u različitim trenutcima u vremenu odrediti količinu i smjer kretnje objekta između tih projekcija. (Marques i Gibson 2016)

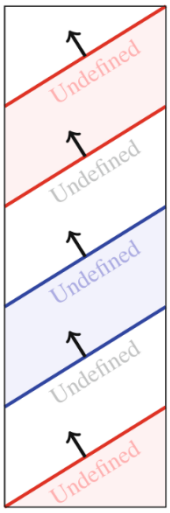
Na prvu ruku problem se ne čini iznimno kompleksan ali u stvarnom svijetu ima mnogih varijabli koje značajno otežavaju procjenu kretnje. Primarni pristup kojim se optički tok ostvaruje koristi boje slike za detekciju, odnosno princip konstantnosti boja (engl. *color constancy*) koji nalaže da za više različitih projekcija jedne te iste fizičke točke (u trodimenzionalnom prostoru) boja te točke će biti ista neovisno o perspektivi. Za slike generirane računalom ovo možda i je istina, ali u stvarnom svijetu percipirana boja neke točke se često mijenja ovisno o promjeni kuta pod kojim se svijetlo odbija od te točke i intenzitetom svjetla. Osim toga čest je slučaj gdje se identična boja pojavljuje na velikom broju piksela tako da ne možemo direktno mapirati poziciju neke fizičke točke isključivo na temelju boja piksela. Jedno dodatno svojstvo koji potpomaže ovom pristupu je konstantnost gradijenta (engl. *gradient constancy*), što je također pod nazivom podudaranjem ruba (engl. *edge matching*) koji gleda ne samo jedan piksel nego i njegove susjede. To iznimno pomaže u

slikama gdje postoji mnogo rubova, odnosno gdje grupa piksela jedne boje oštro prelazi u grupu piksela druge boje. Pretpostavljamo da će taj prijelaz, odnosno gradijent boja biti konstantan između različitih kadrova. Time više ne pokušavamo pratiti pomak istobojnog piksela, već pomak određenog prijelaza boja. (Marques i Gibson 2016)



Slika 7 Primjer efekta otvora blende

Još jedan problem kod optičkog toka koji značajno otežava detekciju smjera kretnje je tzv. efekt otvora blende (engl. *Aperture effect*). Efekt blende se opisuje kao dvosmislenost koja nastaje posljedicom promatranja homogenog ruba većeg objekta kroz manji otvor odnosno blendu. Vizualizaciju tog efekta možemo vidjeti na slici 7. Iako je smjer okretanja brijačkog stupa vodoravan, zbog nakošenosti crvenih i plavih pruga na stupu percipiramo da se pruge penju po stupu. (Marques i Gibson 2016)



Slika 8 Primjer nemogućnosti detekcije
kretnje jednobojnih segmenta

Na slici 8 vidimo još jednu manu optičkog toka. Pošto se stup sastoji od malog broja jednobojnih površina jedino mjesto na kojemu može doći do ikakve detekcije kretnje je sam prijelaz iz jednog u drugi segment (rub). Ukupna površina stupa nad kojim je detektirana kretnja je značajno manja od ukupne vidljive površine stupa. (Marques i Gibson 2016)

3.1.1 Farnebackov algoritam optičkog toka

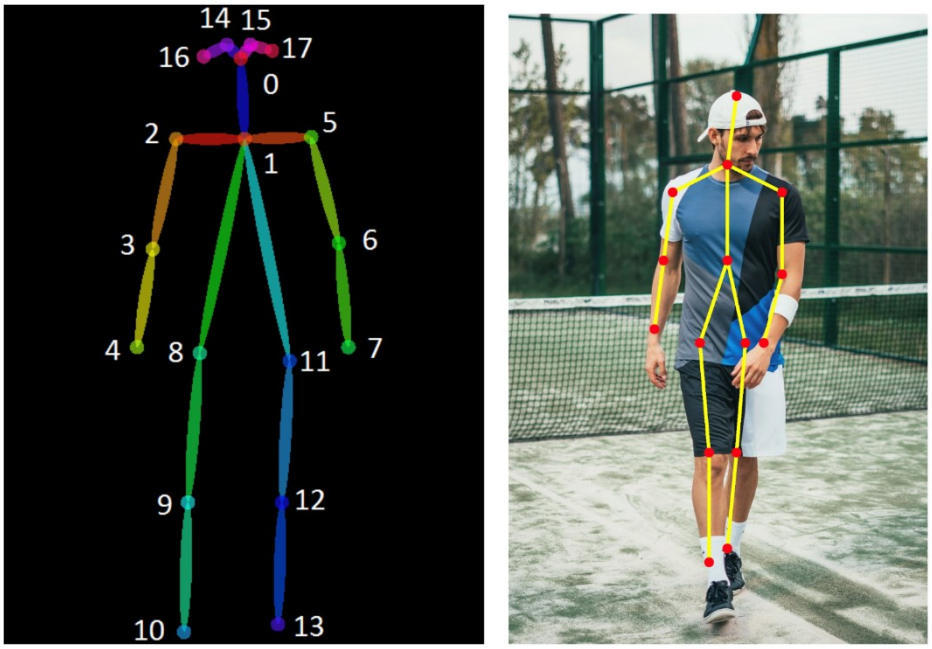
Konkretan algoritam koji će se koristiti za pristup sa optičkim tokom je algoritam koji je Gunnar Farneback opisao u svom radu „*Two-Frame Motion Estimation Based on Polynomial Expansion*“. Ovaj algoritam je odabran zato što je bio jedini dostupan algoritam za gusti optički tok unutar OpenCV-a. Relativno je brz pošto vrši procjene kretnje na temelju samo dva kadra (Farneback 2003) (Joshi, Mendonca i Escriva 2018)

Slika 9 Primjer OpenCV Farneback algoritma

Na slici 9 gore možemo vidjeti primjer rada OpenCV implementacije Farneback algoritma. Algoritmu se proslijede dvije slike za koje želimo odrediti kretnju, kao rezultat dobijemo 2D matricu 2D vektora. Svaki vektor u matrici odgovara smjeru kretnje jednog piksela na slici. Često je potrebno nakon toga dodatno procesirati dobivene vektore da bi se dobili uporabljivi rezultati (npr. kod kamere lošije kvalitete šum u videu se prepozna kao kretnja). (OpenCV 2020)

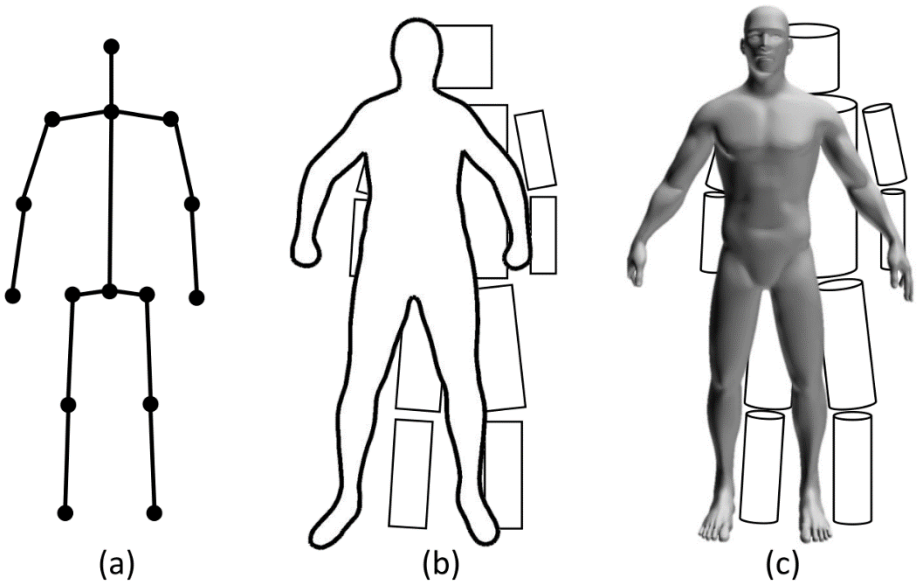
3.2 Procjena ljudske poze

Procjena poze ljudskog tijela (engl. *human pose estimation,* skraćeno *HPE*) se opisuje kao problem kod kojeg je cilj na nekoj slici sa jednom ili više ljudskih osoba odrediti lokacije ključnih točki ljudskog tijela (zglobovi, laktovi, koljena, kukovi, zapešća, gležnjevi) neovisno o pozi u kojoj se čovjek nalazi. (Babu 2018)



Slika 10 Primjer procjene ljudske poze

Na slici 10 možemo vidjeti jedan primjer procjene ljudske poze. Rješenja za detekciju ljudske poze uglavnom koriste konvolucijske neuronske mreže. Ovisno o implementaciji moguće je dobiti procjene pozicije zglobova kao dvodimenzionalne koordinate na slici, ali i kao trodimenzionalne pozicije zglobova u prostoru što značajno proširuje domenu problema na koju možemo primijeniti procjenu poze. Sam problem je dosta složen pošto zahtijeva više različitih tehnika računalnog vida da bi se dobio uporabljiv rezultat. Često korištene tehnike uključuju klasifikaciju slika, detekciju objekata i semantičko segmentiranje. (Chen, Tian i He 2020)



Slika 11 Pristupi modeliranju ljudskog tijela

a) kosturni model; b) model na bazi obrisa; c) model na bazi volumena

Prije bilo kakvog treniranja modela potrebno je odrediti kojim pristupom će se ljudsko tijelo modelirati. Na slici 11 gore možemo vidjeti tri često korištena pristupa.

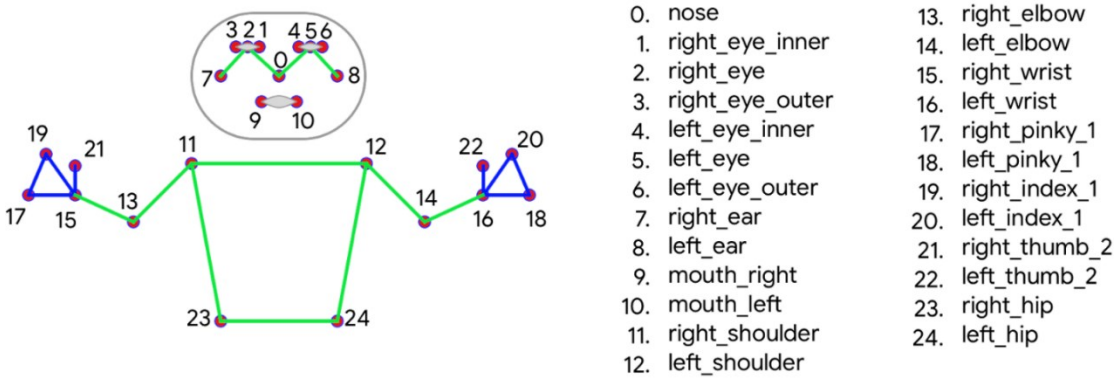
Model na bazi kostura se sastoji od skupa zglobova (tipično između 10 i 30) i odgovarajućih orijentacija udova u obliku ljudskog kostura. Može se opisati kao graf gdje vrhovi predstavljaju zglobove, a bridovi predstavljaju kosti koje ih povezuju. Model je iznimno jednostavan, fleksibilan i jako često se koristi za dvodimenzionalnu i trodimenzionalnu procjenu ljudske poze. Nedostatak ovog pristupa je to što uopće nemamo informaciju o obliku i debljini udova. (Chen, Tian i He 2020)

Model na bazi obrisa se često koristio u ranijim implementacijama procjene poze. Sadrži informacije o udovima i trupu. Dijelovi ljudskog tijela su aproksimirani sa pravokutnicima ili siluetama koje se odnose na taj dio tijela. (Chen, Tian i He 2020)

Modeli na bazi volumena se prikazuju kao geometrijska tijela ili mreže (engl. *meshevi*). U ranijim danima bi se dijelovi ljudskog tijela aproksimirali sa valjcima, stošcima, kuglama i sličnim pravilnim geometrijskim tijelima s kojima bi se procjenjivala ljudska poza. Pristup je od tada značajno sazrio te se danas modeli prikazuju u obliku trodimenzionalne mreže koji predstavlja ljudsko tijelo u cjelini, najčešće pomoću trodimenzionalnog skeniranja osobe. (Chen, Tian i He 2020)

3.2.1 Mediapipe

Mediapipe je *framework* za izradu pipelineova za tumačenje proizvoljnih senzornih podataka razvijen od strane Google-a. Sastoji se od više gotovih rješenja utemeljenih na strojnom učenju za česte probleme. Ideja je da se kod rješavanja nekog kompleksnog, specifičnog problema taj problem razbije na više manjih dijelova koje je pojedinačno moguće riješiti sa jednim od generičkih rješenja. Neka od dostupnih rješenja uključuju detekciju lica, detekciju objekata, te detekciju poze, koja je nama najinteresantnija. (Lugaresi, i dr. 2019)



Slika 12 Ključne točke tijela koje prati MediaPipe

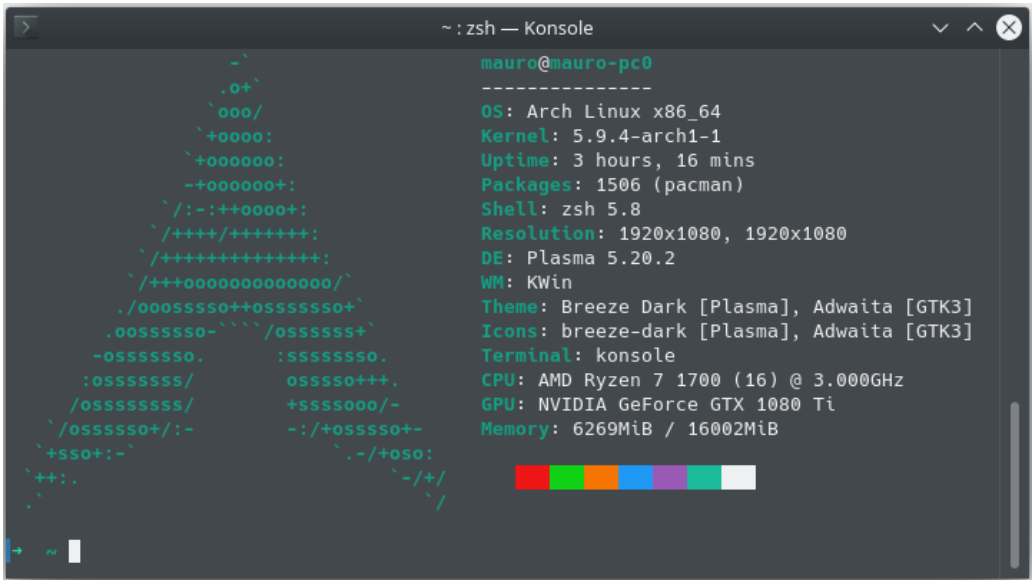
Mediapipe za detekciju poze u pozadini koristi BlazePose model neuronske mreže koji je također razvijen od strane Google-a. BlazePose koristi kosturni model za ljudsko tijelo i podržava do 32 ključne točke koje obuhvaćaju cijelo ljudsko tijelo. Međutim implementacija za MediaPipe koristi njih samo 24 koje se odnose na gornji dio tijela (trup, ruke, dlanovi i glava). Konkretne točke koje su nam dostupne i njihove indekse možemo vidjeti na slici 12 gore. Glavni razlog zašto je odabran MediaPipe za procjenu poze je to što je dostupan za veliki broj platformi i to što jednostavno radi bez ikakvih greški/dodatnih podešavanja. (Mediapipe n.d.)

4 Korišteni programi i alati

4.1.1 Operativni sustav i hardver

Iako je rješenje napravljeno na način da bude kompatibilno sa više platformi, razvoj se izvodio na GNU/Linux operativnom sustavu. Konkretna Linux distribucija koja se koristila je Arch Linux. Nema konkretne verzije koja se koristila pošto je distribucija razvija po modelu tekućeg razvoja (engl. *rolling-release*) što znači da nema konkretne verzije koje se periodično objavljuju. Čim se neki dio distribucije nadogradi i objavi operativni sustav se odmah ažurira.

Ovakav način razvoja nam daje operativni sustav koji nikad nema zastarjeli softver, te su najnovije inačice programa uvijek dostupne bez dodatnih ručnih instalacija, što je iznimno korisno za razvoj softvera.



Slika 13 Specifikacije računala i operativnog sustava

Na slici 13 možemo vidjeti konfiguraciju operativnog sustava za vrijeme razvoja. Korišten je tada aktualan Linux kernel verzije 5.9.4. Što se tiče samog hardvera jedino nam je bitan procesor pošto su programi dizajnirani da se mogu izvoditi na procesoru. Konkretan procesor koji se koristio za vrijeme razvoja je AMD Ryzen 7 1700 sa 8 jezgri i 16 dretvi koji koristi x86_64 arhitekturu.

4.1.2 Programski jezik i alati

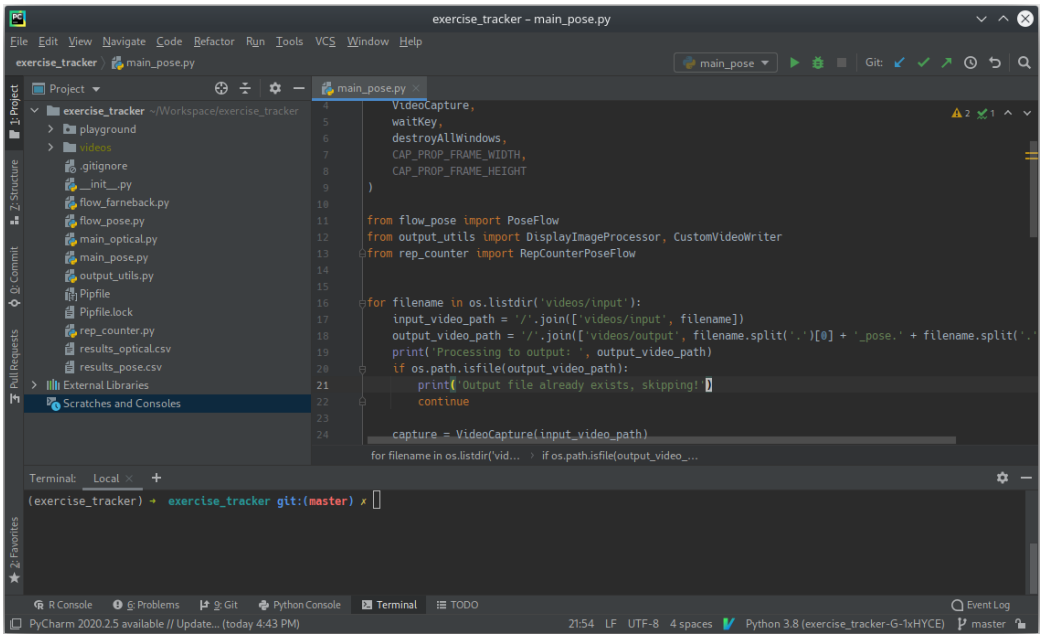
Za razvoj aplikacije je odabran programski jezik Python zbog svoje fleksibilnosti, lakoće pisanja i velikog broja dostupnih paketa vezanih za računalnu znanost i računalni vid. Korištena je 3.8.6 verzija Pythona.

Za upravljanje Python projektom korišten je program Pipenv (verzija 2020.11.4). Njegova uloga je bila upravljanje instalacijom paketa i njihovim zavisnim paketima. Također je korišten za stvaranje i vođenje reda o Python virtualnom okruženju (engl. *virtual environment*) unutar kojega su se instalirali paketi.

Kao sustav za upravljanje kontrolom verzija (engl. *Version control system, VCS*) odabran je git (verzija 2.29.2) zbog svoje popularnosti. Dvije su aktivne grane repozitorija, *master* na kojem se uvijek nalazi najnovija inačica stabilnog softvera i *develop* na kojemu se razvijaju nove značajke dok ne postanu dovoljno stabilne nakon čega se sjedine (engl. *merge*) u *master* granu. Repozitorij je dostupan na servisu Github-u i može mu se pristupiti pomoću slijedeće poveznice: https://github.com/MauroOrlic/exercise_tracker/

4.1.3 Integrirano razvojno okruženje

Za razvojno okruženje je odabran PyCharm Community Edition (verzija 2020.2.5). PyCharm je odabran zato što je sam alat jako dobro uglađen. Ima izvrstan sustav za statičku analizu koda i automatsko upotpunjavanje koda. Na slici 14 dolje možemo vidjeti izgled razvojnog okruženja PyCharm.



Slika 14 Izgled razvojnog okruženja PyCharm

4.1.4 Korišteni Python paketi

Projekt koristi nekolicinu biblioteka, neke od bitnijih uključuju:

- **mediapipe** (0.7.10)- koristi se za praćenje ljudske poze iz slika, ključan preduvjet za implementaciju brojanja repeticija vježbi uporabom procjene poza
- **opencv-python** (3.4.11.43) – korišten je za generalno procesiranje slika i videozapisa, računanje optičkog toka iz dva susjedna kadra uporabom Farneback metode. Odabrana je verzija 3.4.11 umjesto najnovije 4.4.0 pošto na novoj verziji ne funkcionira automatsko popunjavanje koda
- **numpy** (1.19.2) – koristi se za razne matematičke operacije nad ogromnim poljima koja predstavljaju piksele slika

Osim ta tri glavna paketa koriste se još *opencv-contrib-python* za automatsko popunjavanje OpenCV koda, nptyping za bilježenje tipova (engl. *type hinting*) NumPy klasa i objekta.

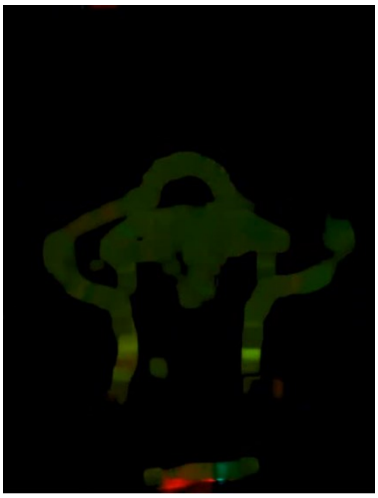
5 Pristup i implementacija

5.1 Generalni pristup

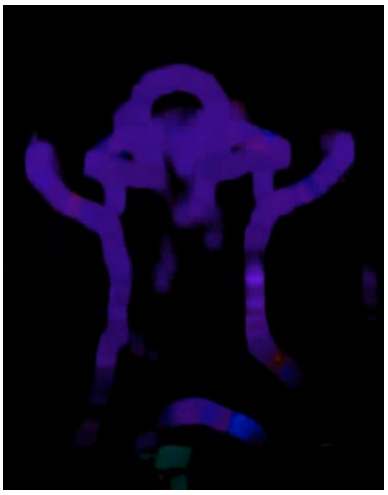
Generalni pristup koji koriste oba rješenja svodi se na problem detekcije iznenadne promjene u smjeru kretanja osobe. To se odvija na slijedeći način:

Prvo se odrede pomaci osobe između svaka dva kadra te se određeni broj najnovijih pomaka uvijek drži u memoriji. Taj keširani skup pomaka se podijeli u dva skupa, jedan koji sadrži starije pomake, drugi koji sadrži novije pomake. Za svaki skup se odredi prosječan smjer kretnje. Na primjer, ako keširamo smjerove i intenzitet kretnje za svaki od posljednjih 12 kadrova, za svaki kadar bismo odredili prosječan vektor kretnje te bi imali skupove vektora od 6 starijih i 6 novijih kadrova. Za svaki od skupova se izračuna prosječni vektor za taj skup.

Slika 15 Spust u čučanj



Slika 16 Mirovanje



Slika 17 Uzlaz iz čučnja

Uzmimo situaciju gdje osoba koju snimamo izvodi čučanj i spušta se do pozicije čučnja (Slika 15Slika 15 Spust u čučanj), prosječan smjer kretnje će pokazivati prema podu. Od točke mirovanja (Slika 16) i kroz uzlaz iz čučnja (Slika 17) prosječan vektor kretnje će pokazivati prema gore. Vizualizaciju toga možemo vidjeti na slikama gore. Ako je trenutak mirovanja točno između 6 starijih i novijih kadrova onda nam stariji kadrovi predstavljaju spust u čučanj, a noviji kadrovi uzlaz iz čučnja. Prosječan vektor starijih kadrova će pokazivati prema dolje a novijih prema gore. Ako uzmemo ta dva vektora i odredimo njihov vektorski umnožak dobiti ćemo negativan broj pošto je kut između njih veći od 90 stupnjeva. Na taj način možemo detektirati svaku iznenadnu, suprotnu promjenu smjera kretanja osobe te takvu pojavu bilježimo kao polovicu jedne repeticije vježbe.

5.2 Zajednički kod

Prije nego što krenemo na specifične implementacije svakog pristupa ukratko ćemo opisati dva dijela potpornog koda koji se koriste za prikazivanje onoga što „program vidi“ i zapisivanje procesiranih kadrova u video.

```
class DisplayImageProcessor:
    COLOR_CHANNEL_COUNT = 3

    def __init__(
            self,
            width: int, height: int,
            window_name: str = 'Excercise Tracker'
    ):
        self._width = width
        self._height = height
        print(f"({self._width}x{self._height})")
        self._hsv_mask = np.zeros(
            (self._height, self._width, self.COLOR_CHANNEL_COUNT),
            dtype=np.uint8
        )
        self._hsv_mask[..., 1] = 255
        self._window_name = window_name
        self._current_frame = cvtColor(self._hsv_mask, COLOR_HSV2BGR)

    @classmethod
    def from_video_capture(cls, capture: VideoCapture):
        return cls(
            int(capture.get(CAP_PROP_FRAME_WIDTH)),
            int(capture.get(CAP_PROP_FRAME_HEIGHT))
        )

    @property
    def current_frame(self):
        return self._current_frame

    def display_frame(
            self,
            *,
            rep_count: int,
            magnitude: np.ndarray = None,
            angle: np.ndarray = None,
            frame: np.ndarray = None,
            landmarks: Tuple[Landmark] = None
    ):
        if magnitude is not None and angle is not None and frame is None and landmarks is None:
            self._display_frame_optical_flow(rep_count, magnitude, angle)
        elif magnitude is None and angle is None and frame is not None:
            self._display_frame_pose_flow(rep_count, frame, landmarks)
```

Isječak koda 1 Implementacija DisplayImageProcessora

U isječku koda 1 gore vidimo implementaciju *DisplayImageProcessor* klase. Njezina uloga je uzeti izlazne vrijednosti brojača repeticija i vizualizirati ih u obliku procesiranog videa. Podaci dobiveni procesiranjem se prosljeđuju metodi *display_frame()*. Parametri koje je potrebno proslijediti ovise o tome radi li se radi o implementaciji koja koristi optički tok ili implementaciji koja koristi procjenu poze. Parametar *rep_count* je obavezan pošto po njemu objekt zna koji je trenutan broj ponavljanja vježbe. U slučaju da se radi o implementaciji s optičkim tokom prosljeđuju se parametri *magnitude* i *angle*, dok se u slučaju implementacije koja koristi procjenu poze prosljeđuju parametri *frame* i *landmarks*.

```
class CustomVideoWriter(VideoWriter):

  @classmethod
  def from_video_capture(cls, capture: VideoCapture, file='untitled.mp4'):
    fps = capture.get(CAP_PROP_FPS)
    fourcc = VideoWriter_fourcc(*'mp4v')
    resolution = (
      int(capture.get(CAP_PROP_FRAME_WIDTH)),
      int(capture.get(CAP_PROP_FRAME_HEIGHT))
    )
    return cls(file, fourcc, fps, resolution)
```

Isječak koda 2 Implementacija CustomVideoWritera

U isječku koda 2 iznad vidimo *CustomVideoWriter*. On je mala modifikacija OpenCV *cv2.VideoWriter* klase koja se koristi za zapisivanje NumPy NDArray-eva u obliku slika. Funkcionira identično kao normalan *VideoWriter*, ali je olakšano instanciranje s time što je moguće uzeti potrebna svojstva ulaznog videa direktno iz OpenCV *cv2.VideoCapture* objekta.

5.3 Rješenje uporabom optičkog toka

Rješenje koje koristi optički tok se može svesti na slijedeći postupak:

1. Za kadar se izračuna optički tok između njega i njegovog prethodnog kadra.
2. Pikseli sa jako malom količinom kretnje se ignoriraju (postavljaju na 0).
3. Kadar se potpuno ignorira ukoliko ima manje od 5% piksela koji se kreću.
4. Izračuna se prosječan vektor kretnje na temelju svih piksela kadra.
5. Posljednjih 13 prosječnih vektora kretnje se kešira.
6. Detekcija koja provjerava je li se dogodila polovica repeticije vježbe se preskače ukoliko je detektirana repeticija unutar zadnjih 8 kadrova. U suprotnom algoritam se nastavlja.
7. Od 13 keširanih prosječnih vektora kadrova, za 5 najstarijih i 5 najnovijih se računaju prosječni vektori.
8. Između prosjeka najstarijih i najnovijih vektora se računa produkt vektora.
9. Ukoliko je produkt vektora manji od nule, broj detektiranih repeticija vježbe se povećava za 0.5.
10. Ovaj algoritam se sekvencijalno izvodi za svaki kadar.

```
capture = VideoCapture(input_video_path)
rep_counter = RepCounterOpticalFlow(
   int(capture.get(CAP_PROP_FRAME_WIDTH)),
   int(capture.get(CAP_PROP_FRAME_HEIGHT)),
   min_nonzero_pixel_percentage=0.05,  # 0.0 to 100.0
   frames_to_cache=13,  # 2 or more
   cached_frames_to_sample=5,  # 1 to frames_to_cache//2
   dot_product_detection_threshold=0.0  # equal or close to 0.0
)
image_processor = DisplayImageProcessor.from_video_capture(capture)
output = CustomVideoWriter.from_video_capture(capture, file=output_video_path)

for magnitude, angle in generate_flow_from_capture(capture):

   rep_counter.update_rep_count(magnitude, angle)
   image_processor.display_frame(
      rep_count=rep_counter.rep_count,
      magnitude=magnitude,
      angle=angle
   )
   output.write(image_processor.current_frame)
```

Isječak koda 3 Optički tok - glavni dio koda

U isječku koda 3 iznad možemo vidjeti glavni dio koda koji prikazuje okvirnu ideju o tome kako se program izvodi. Na početku se instanciraju četiri glavna objekta koja obavljaju većinu posla:

- *capture* - dohvaća video datoteku ili signal sa web kamere
- *rep_counter* - ključni objekt koji na temelju dobivenog optičkog toka kadrova detektira ponavljanje vježbi
- *image_processor* - ukratko smo ga objasnili na početku, koristi se za pretvaranje matrice vektora optičkog toka u vidljivu sliku i prikaza broja ponavljanja,
- *output* - također opisan iznad,, koristi se za zapisivanje dobivenog videa

U for petlji možemo vidjeti generalni tok programa. Funkcija *generate_flow_from_capture()* uzima prethodno stvoren *capture* objekt i iz njega generira matrice magnitude i kuta vektora optičkog toka za svaki kadar. Nakon toga *rep_counter* procesira dobiveni optički tok i odredi je li se u ovom kadru povećao broj obavljenih repeticija vježbe. Zatim *image_processor* pretvori matrice magnitude i kuta u sliku i nadoda očitanje prebrojanih repeticija u gornji lijevi kut. Procesirani video se tada sprema u datoteku.

5.3.1 Računanje optičkog toka

```
def generate_flow_from_capture(
    capture: VideoCapture, magnitude_threshold=2
) -> Generator[np.ndarray, None, None]:
  # Get first frame and convert it to grayscale
  frame_current = cvtColor(init_frame(capture), COLOR_BGR2GRAY)

  while capture.isOpened():
    frame_previous = frame_current
    # Get new frame and check if capture is working (detects last frame in a video file).
    status, frame_current = capture.read()
    if not status:
      break
    # Convert new frame to grayscale
    frame_current = cvtColor(frame_current, COLOR_BGR2GRAY)
    # Calculate optical flow (movement) between the previous and current frame
    flow = calcOpticalFlowFarneback(
      prev=frame_previous,
      next=frame_current,
      flow=None,
      pyr_scale=0.5,
      levels=3,
      winsize=15,
      iterations=3,
      poly_n=5,
      poly_sigma=1.2,
      flags=0
    )
    magnitude, angle = cartToPolar(flow[..., 0], flow[..., 1])
    # Ignores very small movements by setting magnitude to 0 (magnitude is in range 0.0 - 100.0)
    magnitude[magnitude < magnitude_threshold] = 0.0
    magnitude[magnitude > 100.0] = 100.0
    yield magnitude, angle
```

Isječak koda 4 Optički tok - računanje optičkog toka

U isječku koda 4 možemo vidjeti funkciju *generate_flow_from_capture()* koja je zadužena za generiranje optičkog toka iz dohvaćenih kadrova.

Kadar dobiven iz OpenCV *cv2.VideoCapture* objekta se pretvara u optički tok na temelju kojega radimo detekciju repeticija. Prvo dohvatimo kadar iz proslijeđenog *capture* objekta i pretvorimo ga u crno-bijelu sliku sa *cvtColor()* funkcijom. Dobiveni kadar, zajedno sa njegovim prethodnim prosljeđujemo u *calcOpticalFlowFarneback()* funkciju.

Prije prosljeđivanja matrica magnitude i kuta vektora za taj kadar čistimo šum iz matrice magnitude. Kroz eksperimentiranje određeno je da u većini slučajeva magnituda vektora ima vrijednost između 0 i 100. Ukoliko je za ikoji vektor magnituda manja od 2, magnituda tog vektora se postavlja na 0, odnosno zanimaju nas isključivo vektori magnitude 2

ili više. Minimalna valjana vrijednost magnitude može se podesiti kroz parametar *magnitude_threshold*.

Pri kraju također postavljamo sve magnitude veće od 100 na 100, odnosno ograničujemo maksimalnu vrijednost magnitude. U principu ovo nije potrebno za pravilno funkcioniranje, ali kod izračuna optičkog toka se ponekad javi greška (mogući izvori su implementacija Farnebackovog algoritma ili neke nekompatibilnosti sa sustavom) gdje desetak piksela poprimi magnitudu maksimalnog mogućeg cijelog broja (2^{31} -1 za 32-bitne sustave, 2^{63} -1 za 64-bitne sustave). Pošto se u daljnjem programu podaci agregiraju pomoću prosjeka iznimno je bitno ograničiti utjecaj ovakvih pojava.

5.3.2 Brojač ponavljanja

```
def update_rep_count(self, magnitude: np.ndarray, angle: np.ndarray):
    # Ignores low/no activity frames (useful when subject is
    # resting for a few frames at the end of a rep)
    if np.count_nonzero(magnitude) / self._count_vector_total \
            <= self.min_nonzero_pixel_percentage:
        return

    # Calculates average vector for the frame and inserts it to the
    # beginning of the self._cache_avg_vectors cache
    avg_vector = self.get_avg_vector(magnitude, angle)
    if np.isnan(avg_vector[0]):
        print(magnitude, angle)
        print(avg_vector)
    self._cache_avg_vectors = np.insert(
        self._cache_avg_vectors, 0,
        avg_vector,
        axis=0)

    # This part of code doesn't execute until there are self.frames_to_cache items cached
    if len(self._cache_avg_vectors) == self.frames_to_cache + 1:
        self._cache_avg_vectors = self._cache_avg_vectors[:-1]

        # Updates rep count if a repetition is detected
        self._calculate_rep_count()
```

Isječak koda 5 Optički tok - brojač ponavljanja

U isječku koda 5 vidimo glavnu metodu klase *RepCounterOpticalFlow* čiji je zadatak odrediti broj ponavljanja vježbi iz dobivenih matrica magnitude i kuta optičkog toka.

Metoda prvo provjeri imaju li za dobivenu matricu magnituda barem 5% vektora čija je vrijednost veća od 0. Svrha ove funkcionalnosti je osigurati da postoji dovoljno veliki objekt koji se kreće u kadru (osoba) pošto ne želimo vršiti detekciju dok se unutar kadra kreću manji objekti koji nas ne zanimaju (lagano lepršanje zavjese u pozadini, šum u samoj slici

zbog loše kvalitete kamere). Ovo je također vrlo korisno kod same detekcije pošto će program ignorirati kadrove u kojima izvođač vježbe miruje.

Nakon toga se izračunava vektor koji je jednak prosjeku svih vektora piksela unutar kadra pomoću metode *get_avg_vector()*. Dobiveni vektor se kešira na početak polja *_cache_avg_vectors* i najstariji vektor u polju se briše. Na kraju se poziva metoda *_calculate_rep_count()* koja pomoću keširanih vektora odredi broj repeticija vježbe.

Koriste se dvije pomoćne metode: *get_avg_vector()* i *_calculate_rep_count()*.

```
def get_avg_vector(self, magnitude: np.ndarray, angle: np.ndarray):
    # Converts magnitude and angle 2D arrays to a single 2D array containing
    # [x, y] vectors, then calculates and returns the avg [x, y] vector
    return np.add.reduce(
        np.reshape(
            np.dstack((
                np.cos(angle) * magnitude,
                np.sin(angle) * magnitude
            )),
            (self._count_vector_total, self.VECTOR_COMPONENT_COUNT)
        )
    ) / self._count_vector_total
```

Isječak koda 6 Optički tok - određivanje prosječnog vektora kretnje

U isječku koda 6 metoda *get_avg_vector()* obavlja dvije stvari: pretvara vektore piksela iz euklidskog u koordinatni oblik i računa njihov prosjek. Sa *np.cos* i *np.sin* dobijemo polje x i polje y komponenata vektora. Pomoću funkcije *np.dstack* iz ta dva polja dobijemo jedno polje sa svim vektorima kadra nad kojim onda izračunamo prosječan vektor.

```
def _calculate_rep_count(self):
    # Prevents detecting the same change of movement multiple frames in a row
    if self._last_increment_frames_ago >= self.frames_to_cache - self.cached_frames_to_sample:
        # Takes the 'oldest' and 'newest' self.cached_frames_to_sample number of vectors,
        # calculates the average vector for the 'oldest' and 'newest' group of vectors (smoothing)
        # and then calculates the dot product of the two vectors.
        dot_product = np.dot(
            np.add.reduce(
                self._cache_avg_vectors[:self.cached_frames_to_sample]
            ) / self.cached_frames_to_sample,
            np.add.reduce(
                self._cache_avg_vectors[-self.cached_frames_to_sample:]
            ) / self.cached_frames_to_sample
        )
        # If the dot product is negative that means the subject suddenly changed the
        # direction of movement (we interpret this as doing half of a rep)
        if dot_product < self.dot_product_detection_threshold:
            self._rep_count += 0.5
            self._last_increment_frames_ago = 0
    else:
        self._last_increment_frames_ago += 1
```

Isječak koda 7 Optički tok - detekcija ponavljanja

Metoda *_calculate_rep_count()* koju vidimo u isječku koda gore broji repeticije vježbe na temelju prosječnih vektora kadrova zapisanih u varijabli *_cache_avg_vectors* i zapisuje rezultat u varijablu *_rep_count*.

U prvom koraku određujemo hoće li se uopće pokušati detektirati repeticiju vježbe. Od kadra kada je povećanje broja ponavljanja detektirano za slijedećih 8 nećemo ni pokušati. Razlog proizlazi iz toga što dobiveni video ima brzinu od 30 kadrova u sekundi te bez ovakvog ograničenja jedan te isti pokret bi se detektirao više puta uzastopno. Broj kadrova koji ćemo ignorirati mora biti dovoljno velik da ne detektiramo isti pokret više puta, ali dovoljno malen da ne ignoriramo slijedeći pokret. Trenutno zadana vrijednost je 8 kadrova, ali ovisi o drugim parametrima.

U slijedećem koraku su nam potrebne varijable *frames_to_cache* i *cached_frames_to_sample*. Varijabla *frames_to_cache* nam govori koliko ukupno prosječnih vektora kadrova je keširano, a *cached_frames_to_sample* nam govori koliko vektora sa svake strane cache-a treba uzeti. Trenutne zadane vrijednosti su 13 i 5 što nama znači da ćemo od 13 keširanih vektora uzeti najnovijih 5 i najstarijih 5, a srednja 3 ćemo ignorirati.

Kod iznenadnih promjena smjera kretanja za vrijeme vježbanja dolazi do deceleracije i ponovne akceleracije gdje brzina kretanja značajno opadne te to često rezultira i kratkim

mirovanjem. Ignoriranjem vektora u samoj sredini cache-a želimo smanjiti utjecaj perioda mirovanja na detekciju.

Sa *np.add.reduce()* računamo prosjek 5 najnovijih i 5 najstarijih vektora te sa *np.dot()* računamo njihov vektorski produkt. Ukoliko je njihov produkt manji od *dot_product_detection_threshold* (zadana vrijednost je 0) trenutna vrijednost *_rep_count* varijable se poveća za 0.5, odnosno detektirali smo pola repeticije vježbe.

5.3.3 Prikaz i zapis procesiranog videa

```
def _display_frame_optical_flow(self, rep_count: int, magnitude: np.ndarray, angle: np.ndarray):
    self._hsv_mask[..., 0] = angle * 180 / np.pi / 2
    self._hsv_mask[..., 2] = normalize(magnitude, None, 0, 255, NORM_MINMAX)

    frame = putText(
        cvtColor(self._hsv_mask, COLOR_HSV2BGR),
        f"Rep count: {int(rep_count)}",
        (0, 50), FONT_HERSHEY_SIMPLEX, 2, 255
    )

    imshow(self._window_name, frame)

    self._current_frame = frame
```

Isječak koda 8 Optički tok - zapisivanje kadrova

Nakon što smo za dani kadar odredili koji je ukupan broj ponavljana vježbe rezultat je potrebno zapisati. Iznad možemo vidjeti isječak koda 8 u kojem je ta funkcionalnost implementirana. Za prikaz optičkog toka u obliku slike pretvaramo matrice magnitude i kuta u HSV (engl. *Hue, Saturation, Value*) model boja gdje nam se matrice kutova mapiraju na *Hue*, a magnituda u *Value* (*Saturation* smo na početku postavili na maksimalnu vrijednost). Nakon toga HSV model boje pretvaramo u RGB model i nadodajemo ispis trenutnog broja ponavljanja u gornjem lijevom kutu. Na slici 18 dolje možemo vidjeti izgled procesiranog kadra.

Slika 18 Primjer kadra procesiranog optičkim tokom

5.4 Rješenje uporabom procjene ljudske poze

Rješenje koje koristi procjenu ljudske poze se može svesti na slijedeći postupak:

1. Za svaki kadar se odrede ključne točke ljudskog tijela.
2. Keširaju se točke trenutnog kadara i 2 koja mu prethode, kao i točke prethodnog kadra i 2 koja njemu prethode.
3. Za skup od 3 trenutne i za skup od 3 prethodne točke kadra računaju se prosječne točke za svaki skup.
4. Između njih se izračuna razlika koja predstavlja vektore pomaka ključnih točki, odnosno predstavlja tok poze.
5. Ako je prosječna magnituda toka poze manja od 0.5 tok poze za taj kadar se potpuno ignorira.
6. Za svaki od posljednjih 13 kadrova kešira se tok poze.
7. Detekcija koja provjerava je li se dogodila polovica repeticije vježbe se preskače ukoliko je detektirana repeticija unutar zadnjih 8 kadrova. U suprotnom algoritam se nastavlja.
8. Iz 13 keširanih vektora poze za 5 najstarijih i 5 najnovijih računaju se prosječni tokovi poze.
9. Za svaki par vektora između starijeg i novijeg toka (npr. vektor oka na starijem i novijem, vektor lakta na starijem i novijem…) računa se njihov produkt. Sa time dobivamo vektorski produkt vektora za svaku ključnu točku tijela.
10. Ukoliko je prosjek produkta manji od -0.25 broj detektiranih ponavljanja vježbe se povećava za 0.5.
11. Ovaj algoritam se sekvencijalno izvodi za svaki kadar.

```
capture = VideoCapture(input_video_path)
rep_counter = RepCounterPoseFlow(
   min_movement_amplitude_per_frame=0.5,
   frames_to_cache=13,
   cached_frames_to_sample=6,  # 1 to frames_to_cache//2
   dot_product_detection_threshold=-0.25  # equal or close to 0.0
)
image_processor = DisplayImageProcessor.from_video_capture(capture)
output = CustomVideoWriter.from_video_capture(capture, file=output_video_path)

for frame, landmarks, landmarks_flow in PoseFlow.generate_flow_from_capture(capture):

   rep_counter.update_rep_count(landmarks_flow)
   image_processor.display_frame(rep_count=rep_counter.rep_count, frame=frame,
landmarks=landmarks)
   output.write(image_processor.current_frame)
```

Isječak koda 9 Procjena poze - glavni dio koda

Gore vidimo glavni dio koda koji je vrlo sličan implementaciji sa optičkim tokom. I dalje imamo *capture* objekt koji iz kojeg povlačimo kadrove, *rep_counter* ima malo drugačije parametre i implementaciju od prethodnog pošto mora brojati ponavljanja na temelju poze umjesto optičkog toka, isto vrijedi i za *image_processor*.

Klasa *PoseFlow* ima metodu *generate_flow_from_capture()* koja ima istu ulogu kao i *generate_flow_farneback()* iz prethodnog rješenja samo što umjesto optičkog toka vraća trenutnu sliku kadra, pozicije ključnih točaka i njihov tok, odnosno vektor kretnje.

Vektori kretnje se proslijede u *rep_counter* objekt i broj ponavljana vježbe se ažurira. Potom *image_processor* prikaže rezultate i procesirani kadar se zapisuje u datoteku uporabom *output* objekta.

5.4.1 Računanje toka poze

```
@classmethod
def generate_flow_from_capture(
      cls,
      capture: VideoCapture,
      smoothing: int = 3
) -> Generator[
   Tuple[NDArray, Optional[Landmark], Optional[Tuple[LandmarkFlow, ...]]]
   , Any, Any]:
   tracker = UpperBodyPoseTracker()

   landmarks_previous = None
   landmarks_current = None

   landmarks_group_previous = []
   landmarks_group_current = []

   while capture.isOpened():
      if landmarks_current is not None:
         landmarks_previous = landmarks_current
         landmarks_group_previous.insert(0, landmarks_previous)
         if len(landmarks_group_previous) > smoothing:
            landmarks_group_previous.pop()

      status, frame = capture.read()
      if not status:
         print(f"Capture status returned: {status}")
         break
      landmarks_current, _ = tracker.run(frame)
      if landmarks_current is not None:
         landmarks_current = Landmark.from_mediapipe_landmarks(landmarks_current)
         landmarks_group_current.insert(0, landmarks_current)
         if len(landmarks_group_current) > smoothing:
            landmarks_group_current.pop()

      if landmarks_current is None or landmarks_previous is None:
         yield frame, None, None
      else:
         landmarks_current_mean = cls.mean_landmark_group(landmarks_group_current)
         landmarks_previous_mean = cls.mean_landmark_group(landmarks_group_previous)
         flow = cls._calculate_flow(
            landmarks_previous_mean,
            landmarks_current_mean
         )
         yield frame, landmarks_current_mean, flow
```

Isječak koda 10 Procjena poze - generator toka poze

Za razliku od pristupa s optičkim tokom koji ima postojeću metodu za računanje optičkog toka u ovom slučaju toga nema. Zbog toga moramo mi sami odrediti smjer kretnje osobe iz detektiranih pozicija na kadrovima, to možemo vidjeti u isječku koda 10.

Varijabla *landmarks_group_current* je lista koja sadrži pozicije ključnih točki trenutnog kadra i dva kadra prije njega, a *landmarks_group_previous* sadrži pozicije ključnih točki prethodnog kadra i dva kadra prije njega. Pomoću metode *mean_landmark_group* iz prethodne dvije liste računamo prosječne ključne točke za svaku od listi točaka. Time uglađujemo detektirane točke jer one nemaju konzistentnu poziciju kadar za kadrom neovisno o kretnji osobe u videozapisu.

Za dobiveni par pozicija ključnih točaka (*landmarks_previous_mean* i *landmarks_current_mean*) računamo razliku njihovih pozicija. Dobiveni rezultat nam označava vektor pomaka ključnih točaka između prethodnog i trenutnog kadra.

```
@dataclass
class Landmark:
  x: float
  y: float
  z: float
  visibility: float

  @classmethod
  def from_mediapipe_landmarks(cls, landmarks) -> Tuple['Landmark']:
    return tuple(
      Landmark(
        data_point.x, data_point.y, data_point.z,
        data_point.visibility
      )
      for data_point in landmarks.landmark
    )

  def as_vector(self):
    return np.ndarray([self.x, self.y, self.z])
```

Isječak koda 11 Procjena poze - Podatkovna klasa Landmark

Ugrađeni Mediapipe tipovi u kojima su spremljene ključne točke su vrlo nezgrapni i loše dokumentirani. Zbog tog razloga stvaramo vlastiti tip objekta koji koristimo za zapisivanje. Njegovu implementaciju možemo vidjeti gore u isječku koda 11. Sadrži metodu *from_mediapipe_landmarks()* pomoću koje pretvaramo točke iz mediapipe tipa u naš tip.

```
@dataclass
class LandmarkFlow:
    x: float
    y: float
    z: float
    visibility: float

    @classmethod
    def from_landmarks(cls, position_previous: Landmark, postition_current: Landmark):
        return LandmarkFlow(
            (postition_current.x - position_previous.x) * 100,
            (postition_current.y - position_previous.y) * 100,
            (postition_current.z - position_previous.z) * 100,
            mean((postition_current.visibility, position_previous.visibility))
        )

    def as_np_vector(self):
        return np.array([self.x, self.y])
```

Isječak koda 12 Procjena poze - podatkovna klasa LandmarkFlow

Osim tipa za same točke potreban nam je i tip za tok (vektore kretnje) ključnih točaka čiju implementaciju vidimo u isječku koda 12. Klasa također sadrži metodu *from_landmarks()* pomoću koje, iz prethodno spomenutih varijabli *landmarks_previous_mean* i *landmarks_current_mean*, za svaki par točaka dobivamo vektor koji predstavlja pomak iz prethodne u trenutnu točku.

5.4.2 Brojač repeticija

```
def update_rep_count(self, landmarks_flow: Tuple[LandmarkFlow]):
    if landmarks_flow is None or \
            self._frame_avg_flow_magnitude(landmarks_flow) < self.min_movement_amplitude_per_frame:
        return

    print(self._frame_avg_flow_magnitude(landmarks_flow))

    self._cached_frames_landmarks.insert(0, landmarks_flow)

    if len(self._cached_frames_landmarks) == self.frames_to_cache + 1:
        self._cached_frames_landmarks.pop()

        self._calculate_rep_count()
```

Isječak koda 13 Procjena poze - brojač ponavljanja

U isječku koda 13 vidimo glavnu metodu koja se koristi unutar *rep_counter* objekta – *update_rep_count*. U prvom koraku kadar se ignorira ukoliko za trenutni kadar tok poze nije bio detektiran. Kadar se također ignorira ukoliko je količina kretnje iznimno mala (npr. mali pomaci za vrijeme mirovanja). Nakon toga, kao i kod prethodnog pristupa, keširamo tokove poza za posljednjih 13 kadrova i pozivamo *_calculate_rep_count()* metodu.

```
def _calculate_rep_count(self):
    # Prevents detecting the same change of movement multiple frames in a row
    if self._last_increment_frames_ago >= self.frames_to_cache - self.cached_frames_to_sample:

        previous = self._cached_frames_landmarks[:self.cached_frames_to_sample]
        previous_avg_frame = tuple(LandmarkFlow(0, 0, 0, 0) for i in range(self.MARKERS_PER_FRAME))
        previous_frame_count = len(previous)

        for frame_landmarks_flow in previous:
            for i in range(previous_frame_count):
                previous_avg_frame[i].x += frame_landmarks_flow[i].x / previous_frame_count
                previous_avg_frame[i].y += frame_landmarks_flow[i].y / previous_frame_count
                previous_avg_frame[i].z += frame_landmarks_flow[i].z / previous_frame_count
                previous_avg_frame[i].visibility += \
                    frame_landmarks_flow[i].visibility / previous_frame_count

        current = self._cached_frames_landmarks[-self.cached_frames_to_sample:]
        current_avg_frame = [LandmarkFlow(0, 0, 0, 0) for i in range(self.MARKERS_PER_FRAME)]
        current_frame_count = len(current)

        for frame_landmarks_flow in current:
            for i in range(previous_frame_count):
                current_avg_frame[i].x += frame_landmarks_flow[i].x / current_frame_count
                current_avg_frame[i].y += frame_landmarks_flow[i].y / current_frame_count
                current_avg_frame[i].z += frame_landmarks_flow[i].z / current_frame_count
                current_avg_frame[i].visibility += \
                    frame_landmarks_flow[i].visibility / current_frame_count

        frame_marker_dot_products = [
            np.dot(
                previous_avg_frame[i].as_np_vector(),
                current_avg_frame[i].as_np_vector()
            )
            for i in range(self.MARKERS_PER_FRAME)]

        if sum(frame_marker_dot_products) < self.dot_product_detection_threshold:
            self._rep_count += 0.5
            self._last_increment_frames_ago = 0
    else:
        self._last_increment_frames_ago += 1
```

Isječak koda 14 Procjena poze - detekcija ponavljanja

U isječku koda 14 možemo vidjet implementaciju metode *_calculate_rep_count()* koja je zadužena za detekciju ponavljanja. U prvom koraku, isto kao i kod pristupa sa optičkim

tokom, ne pokušavamo detektirati ponavljanje vježbe ukoliko se detekcija dogodila unutar zadnjih 8 kadrova.

Iz varijable *_cached_frames_landmarks* od 13 keširanih tokova poze, njih 5 najstarijih i 5 najnovijih spremamo u varijable *previous* i *current*. Za obje varijable računamo njihov prosjek. Rezultate spremamo u *previous_avg_frame* i *current_avg_frame* varijable.

Nakon toga za svaki par *LandmarkFlow* objekta gdje je prvi iz *previous_avg_frame*, a drugi iz *current_avg_frame* računamo produkt vektora i spremamo rezultirajuće polje produkta u varijablu *frame_market_dot_products*.

Ukoliko je prosječan produkt vektora manji od -0.25 (odnosno došlo je do iznenadne promjene smjera kretnje osobe) povećavamo broj obavljenih ponavljanja vježbe za 0.5.

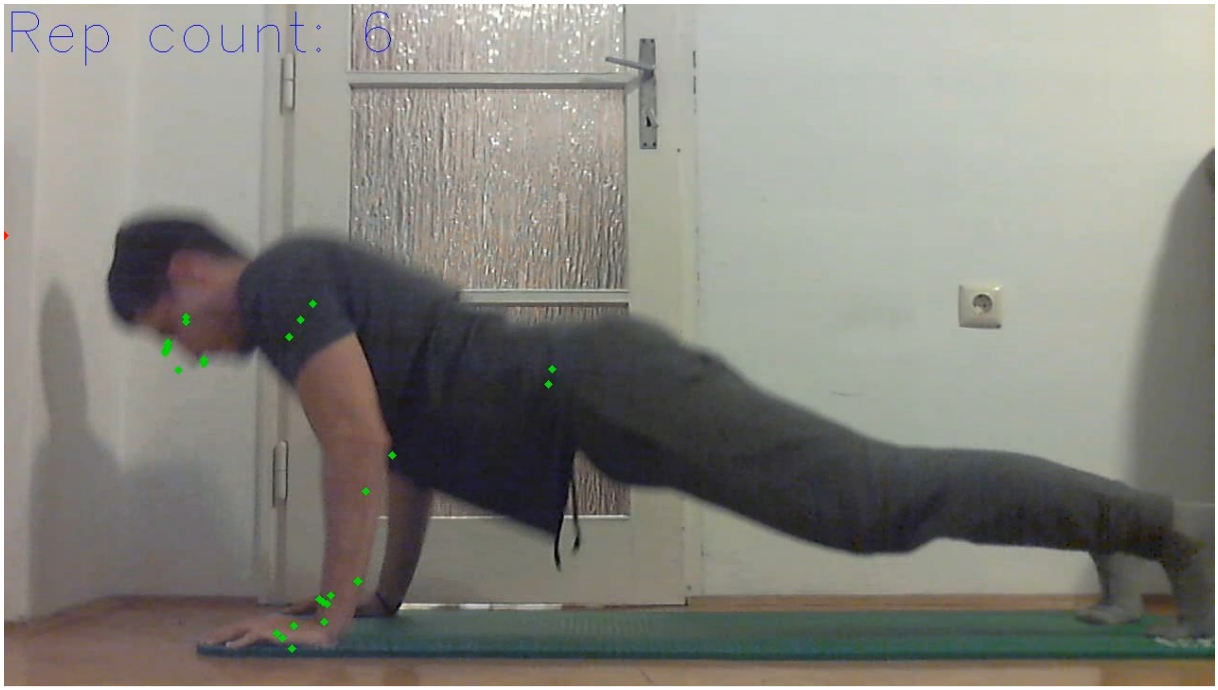
5.4.3 Prikaz i zapis procesiranog videa

```
def _display_frame_pose_flow(
    self, rep_count: int, frame: np.ndarray, landmarks: Collection[Landmark]
):
  if landmarks is not None:
    for landmark in landmarks:
      x = max(0, min(
        self._width,
        int(landmark.x * self._width)
      ))
      y = max(0, min(
        self._height,
        int(landmark.y * self._height)
      ))

      if not (0 <= landmark.x <= 1) or \
          not (0 <= landmark.y <= 1):
        marker_color = (0, 0, 255)
      else:
        marker_color = (0, 255, 0)
      frame = circle(frame, center=(x, y), radius=2, color=marker_color, thickness=3)

  frame = putText(
    frame,
    f"Rep count: {int(rep_count)}",
    (0, 50), FONT_HERSHEY_SIMPLEX, 2, 255
  )

  imshow(self._window_name, frame)

  self._current_frame = frame
```

Isječak koda 15 Procjena poze - zapisivanje kadrova

Procesirani podaci se vizualiziraju na drugačiji način u usporedbi sa prethodnim pristupom. Implementaciju možemo vidjeti u isječku koda 15. Kroz parametre dobijemo sliku kadra (*frame*), pozicije ključnih točaka poze (*landmarks*) i broj detektiranih ponavljanja (*rep_count*).

Za svaku ključnu točku pretvaramo njezinu relativnu poziciju u apsolutnu poziciju koja odgovara specifičnom pikselu na slici te sa metodom *cv2.circle()* dodajemo zelenu točku na te koordinate. Ukoliko je pozicija ključne točke van kadra ona se prikazuje kao crvena točka na rubu kadra. Na slici 19 dolje možemo vidjeti primjer obrađenog kadra.



Slika 19 Primjer kadra procesiranog tokom poze

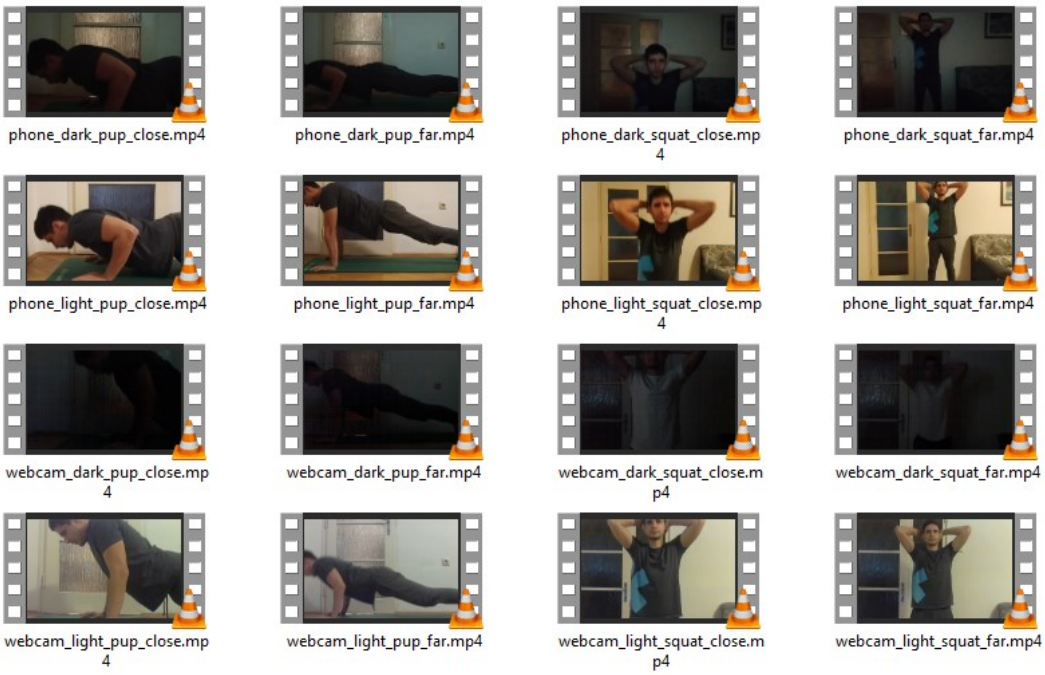
6 Rezultati

6.1 Testni podaci

Da bi mogli procijeniti efektivnost i mogućnosti razvijenih programa potrebno je testirati oba rješenja. Za test je snimljeno 16 videozapisa u kojima se obavlja tjelovježba u raznim uvjetima. Cilj je sa svakim od rješenja procesirati videozapise i odrediti njihovu razinu točnosti te odrediti kakav utjecaj imaju koji od uvjeta.

Nažalost nisu se mogli iskoristiti videozapisi sa interneta pošto kod većine od njih postoji neki od problema:

- kamera nije stacionarna nego se kreće
- nema dovoljno dugih perioda snimke bez promjene perspektive
- često je kamera iznimno blizu subjektu i ne vidi se cijelo tijelo



Slika 20 Prikaz testnih podataka

Kroz 16 videa (koji su prikazani na slici 20 iznad) ćemo provjeriti generalnu efektivnost programa, kao i efektivnost ovisno o četiri različita parametra:

- **Izvor videa** – kamere sa kojima su video zapisi snimani su web kamera (Logitech C270) i mobilni telefon (Xiaomi Redmi Note 8T, stražnja kamera)
- **Osvjetljenje** – razina osvjetljenja nije strogo kontrolirana te je svaki video sniman u „tamnim“ ili „svjetlim“ uvjetima
- **Tip vježbe** – tip vježbe koji se obavlja u videozapisu
- **Udaljenost** – slično kao kod osvjetljenja udaljenost subjekta od kamere nije strogo kontrolirana, već su korištene određene udaljenosti „blizu“ i „daleko“

Nadalje, radi konzistentnosti, u svim videozapisima prisutan je isti subjekt. Neovisno o kameri videozapisi su snimani u 720p kvaliteti brzinom od 30 kadra po sekundi. Radi jednostavnosti svaki video sadrži točno 10 ponavljanja dane vježbe.

6.2 Rezultati optičkog toka

Tablica 1 Rezultati optičkog toka

Izvor	Osvjetljenje	Vježba	Udaljenost	Detektirana ponavljanja (od 10)
Mobitel	Svjetlo	Sklekovi	Daleko	**9**
Mobitel	Svjetlo	Čučnjevi	Blizu	**9**
Web kamera	Tamno	Sklekovi	Blizu	**0**
Mobitel	Tamno	Čučnjevi	Blizu	**0**
Web kamera	Svjetlo	Čučnjevi	Blizu	**9**
Web kamera	Tamno	Čučnjevi	Daleko	**0**
Web kamera	Svjetlo	Sklekovi	Daleko	**10**
Web kamera	Svjetlo	Sklekovi	Blizu	**9**
Mobitel	Tamno	Sklekovi	Daleko	**3**
Mobitel	Svjetlo	Čučnjevi	Daleko	**9**
Mobitel	Tamno	Sklekovi	Blizu	**3**
Mobitel	Svjetlo	Sklekovi	Blizu	**9**
Mobitel	Tamno	Čučnjevi	Daleko	**0**
Web kamera	Tamno	Sklekovi	Daleko	**0**
Web kamera	Svjetlo	Čučnjevi	Daleko	**9**
Web kamera	Tamno	Čučnjevi	Blizu	**0**

Na tablici 1 iznad vidimo postignute rezultate s ovim pristupom. Prosječan broj detektiranih vježbi je 4.94 sa standardnom devijacijom od 4.29. Čak i bez daljnje analize vidimo da program funkcionira iznimno loše u mračnim uvjetima zbog toga što je za većinu mračnih snimki detektirao 0 ponavljanja. Također možemo primijetiti da imamo veliki broj slučajeva gdje se detektiralo 9 ponavljanja. Pretpostavljamo da je to zato jer je pri zadnjem ponavljanju vježbe subjekt završio vježbu u stanju mirovanja. Odnosno nakon mirovanja ne bi došlo do kretnje u suprotnom smjeru te se zadnja repeticija ne bi detektirala do kraja.

Tablica 2 Analiza rezultata optičkog toka

Izolirani parametar	Prosjek	Standardna devijacija	Korelacija
Web kamera	**4.6**	**4.64**	**-0.073**
Mobitel	**5.3**	**3.90**	**0.073**
Tamno	**0.8**	**1.30**	**-0.975**
Svjetlo	**9.1**	**0.33**	**0.975**
Čučnjevi	**4.5**	**4.50**	**-0.102**
Sklekovi	**5.4**	**4.03**	**0.102**
Blizu	**4.9**	**4.23**	**-0.015**
Daleko	**5.0**	**4.36**	**0.015**

Na tablici 2 gore možemo vidjeti analizu dobivenih rezultata kada fiksiramo vrijednosti svakog parametra. Kao što smo prethodno pretpostavili, broj detektiranih ponavljanja ima pozitivnu korelaciju od 0.975 sa svijetlim uvjetima i -0.975 sa mračnim uvjetima. Osim toga čini se da rješenje preferira sklekove preko čučnjeva, ali pošto je korelacija samo 0.102 i pošto imamo samo 16 videozapisa ovo nije značajno zapažanje.

6.3 Rezultati procjene poze

Tablica 3 Rezultati procjene poze

Izvor	Osvjetljenje	Vježba	Udaljenost	Detektirana ponavljanja (od 10)
Mobitel	Svjetlo	Sklekovi	Daleko	**9**
Mobitel	Svjetlo	Čučnjevi	Blizu	**10**
Web kamera	Tamno	Sklekovi	Blizu	**9**
Mobitel	Tamno	Čučnjevi	Blizu	**9**
Web kamera	Svjetlo	Čučnjevi	Blizu	**9**
Web kamera	Tamno	Čučnjevi	Daleko	**1**
Web kamera	Svjetlo	Sklekovi	Daleko	**10**
Web kamera	Svjetlo	Sklekovi	Blizu	**9**
Mobitel	Tamno	Sklekovi	Daleko	**9**
Mobitel	Svjetlo	Čučnjevi	Daleko	**9**
Mobitel	Tamno	Sklekovi	Blizu	**10**
Mobitel	Svjetlo	Sklekovi	Blizu	**9**
Mobitel	Tamno	Čučnjevi	Daleko	**9**
Web kamera	Tamno	Sklekovi	Daleko	**11**
Web kamera	Svjetlo	Čučnjevi	Daleko	**9**
Web kamera	Tamno	Čučnjevi	Blizu	**8**

Na tablici 3 iznad vidimo postignute rezultate s ovim pristupom. Prosječan broj detektiranih vježbi je 8.75 sa standardnom devijacijom od 2.10. Na tablici gore vidimo da su skoro svi rezultati točni ili za jedno ponavljanje udaljeni od 10, osim jednog videozapisa gdje je detektirano samo jedno ponavljanje. Također imamo i pojavu gdje veliki broj prebrojavanja ima vrijednost 9. Osim toga rezultati su većinom konzistentni i ne možemo razaznati ikakav uzorak bez dodatne analize.

Tablica 4 Analiza rezultata procjene poze

Izolirani parametar	Prosjek	Standardna devijacija	Korelacija
Web kamera	**8.3**	**2.86**	**0.237**
Mobitel	**9.3**	**0.43**	**-0.237**
Tamno	**8.3**	**2.86**	**-0.237**
Svjetlo	**9.3**	**0.43**	**0.237**
Čučnjevi	**8.0**	**2.69**	**-0.356**
Sklekovi	**9.5**	**0.71**	**0.356**
Blizu	**9.1**	**0.60**	**0.178**
Daleko	**8.2**	**2.87**	**-0.178**

Na tablici 4 iznad vidimo analizu prethodnih podataka koju možemo opisati u nekoliko točaka:

- ne postoji niti jedan parametar koji ima izniman utjecaj na sposobnost detekcije
- prosjek prebrojanih vježbi za pojedine parametre je jako blizak općem prosjeku
- standardna devijacija je za svaki od parametra ili manja od 1 ili veća od 2.

Na prvi pogled izgleda kao da smo pronašli parametre za koje pristup nije toliko učinkovit, ali ovo je očekivan rezultat pošto parametri koji imaju velike vrijednosti standardne devijacije se ujedno nalaze u slučaju gdje je izbrojano samo jedno ponavljanje.

6.4 Usporedba rezultata

Pristup koji koristi procjenu poze ima daleko bolju točnost prebrojavanja vježbi (8.75 prosjek) u usporedbi sa pristupom koji koristi optički tok (prosjek 4.94). Pristup sa pozom je također konzistentniji sa svojim rezultatima (2.10 standardna devijacija) nego optičkim tokom (4.29 standardna devijacija).

Veliki razlog tome je zato što pristup sa optičkim tokom nije učinkovit kod tamnih videa. To vidimo po samoj korelaciji (negativna korelacija od -0.975 između broja detektiranih ponavljanja i toga da li je videozapis mračan), ali možemo i pretpostaviti po tome što su prosjek i devijacija oboje blizu broja 5. Time možemo zaključiti da je većina rezultata ili jako blizu 0 ili jako blizu 10.

Ako usporedimo oba pristupa isključivo u svijetlim uvjetima efektivnost pristupa sa optičkim tokom se značajno mijenja. U svijetlim uvjetima pristup s optičkim tokom ima prosječan broj ponavljanja od 9.1 i standardnu devijaciju od 0.33 dok pristup sa pozom ima prosjek od 9.3 i standardnom devijacijom od 0.43. Vidimo da u svijetlim uvjetima optički tok ima većinom slične rezultate kao i pristup sa pozom. Zbog toga što su rezultati bazirani na temelju samo 8 videozapisa nemamo dovoljno velik uzorak podataka na temelju kojih možemo ustanoviti dodatne zaključke.

Kod oba pristupa dolazi do istog problema, zbog mirovanja na kraju posljednje repeticije program ne uspije detektirati repeticiju vježbe do kraja. Također, oba pristupa su imala malo bolje rezultate sa sklekovima nego sa čučnjevima (0.102 korelacija optičkog toka sa čučnjevima, 0.356 korelacija poze sa čučnjevima), no zbog malog broja videa i zbog male vrijednosti korelacije ne možemo biti sigurni je li je to slučajnost, je li zapravo lakše pratiti sklekove nego čučnjeve ili je u pitanju neka druga varijabilnost koju nismo uzeli u obzir.

7 Zaključak

U ovom radu smo pristupili rješavanju problema brojanja repeticija vježbi koristeći računalni vid. Ukratko smo opisali što je područje računalnog vida, koje su njegove mogućnosti i primjene u stvarnom svijetu. Opisali smo generalne algoritme i tehnologije koje smo koristili za implementaciju naših rješenja, nakon čega smo u detalje opisali korake i pravila za svaki od pristupa.

Dobiveni rezultati su poprilično dobri, pogotovo što se tiče pristupa sa procjenom poza, ali pristup koji koristi optički tok je imao problema sa tamnim scenama. Oba pristupa su imala problema sa brojanjem zadnjeg ponavljanja vježbe pošto na kraju subjekt završava kretnju sa mirovanjem, a ne promjenom smjera.

Postoji mnogo područja gdje bi se pojedini pristupi mogli poboljšati. Pristup sa optičkim tokom bi se mogao poboljšati na način da se bliski pikseli koji se kreću u istom smjeru grupiraju i onda gleda smjer kretnje grupa umjesto pojedinih piksela. Na ovaj način bi bilo mnogo lakše razlikovati pozadinski šum od kretnji ljudskog tijela. Pretpostavljamo da postoji način na koji možemo ulaznu sliku pred procesirati prije nego odredimo optički tok nad njom. Mogla bi se ubrzati detekcija ljudske poze ako bismo zamijenili detektor poze sa nekim koji procjenjuje pozu u dvodimenzionalnom prostoru umjesto trodimenzionalnom. Mediapipe također ne pruža rješenje koje detektira ne samo pozu glave, ruku i trupa nego i poziciju noga te zbog toga ne možemo uopće raditi detekcije vježbi koje se fokusiraju na noge.

8 Prilozi

Uz ovaj rad priložene su slijedeće datoteke:

- *exercise_tracker-master.zip* – sadrži git repozitorij sa napisanim kodom

Ulazni i izlazni videozapisi nisu priloženi pošto zauzimaju previše prostora.

9 Popis slika i tablica

Slike

Tablice

Isječci koda

10 Literatura

Babu, Sudharshan Chandra. 2018. *A 2019 guide to Human Pose Estimation with Deep Learning.* Pokušaj pristupa 1. 12 2020. https://nanonets.com/blog/human-pose-estimation-2d-guide/.

Chen, Yucheng, Yingli Tian, i Mingyi He. 2020. »Monocular human pose estimation: A survey of deep learning-based methods.« *Computer Vision and Image Understanding* (Elsevier) 192: 1-23.

Dwibedi, Debidatta, Yusuf Aytar, Jonathan Tompson, Pierre Sermanet, i Andrew Zisserman. 2020. »Counting Out Time: Class Agnostic Video Repetition Counting in the Wild.« U *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 10387-10396. New York: Association for Computing Machinery.

Farneback, Gunnar. 2003. »Two-frame motion estimation based on polynomial expansion.« U *Scandinavian conference on Image analysis*, 363-370. Linköping: Springer.

Joshi, Prateek, Vinicius G. Mendonca, i David Millan Escriva. 2018. *Learn OpenCV 4 by Building Projects.* 2nd. Birmingham: Packt Publishing.

Khurana, Rushil, Karan Ahuja, Zac Yu, Jennifer Mankoff, Chris Harrison, i Mayank Goel. 2018. »GymCam: Detecting, recognizing and tracking simultaneous exercises in unconstrained scenes.« *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* (ACM) 2: 1-17.

Klette, i Reinhard. 2014. *Concise Computer Vision: an Introduction into Theory and Algorithms.* London: Springer London.

Lugaresi, Camillo, Jiuqiang Tang, Hadon Nash, Chris McClanahan, Esha Uboweja, Michael Hays, Fan Zhang, i dr. 2019. »Mediapipe: A framework for building perception pipelines.« *arXiv preprint arXiv:1906.08172.*

Marques, Oge, i Joel Gibson. 2016. *Optical Flow and Trajectory Estimation Methods.* Cham: Springer International Publishing.

Mediapipe. n.d. *Pose - mediapipe.* Pokušaj pristupa 2. 12 2020. https://google.github.io/mediapipe/solutions/pose.

OpenCV. 2020. *OpenCV: Optical Flow.* 11. 7. Pokušaj pristupa 2. 12 2020. https://docs.opencv.org/3.4.12/d4/dee/tutorial_optical_flow.html.

Shapiro, Linda G., i George C. Stockman. 2001. *Computer Vision.* Seattle, Washington: Prentice Hall.

Szeliski, Richard. 2011. *Computer Vision: Algorithms and Applications.* London: Springer.