Razvoj sustava za prilagođavanje ponašanja robota temeljen na analizi ljudskih kretnji

Pongračić, Matija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Matija Pongračić

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentor: Student:

Doc. dr. sc. Marko Švaco, mag. ing. mech. Matija Pongračić

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru doc. dr. sc. Marku Švaci na prijedlogu i zadavanju teme, asistentici Tari Knežević na pruženim savjetima i usmjeravanju tijekom izrade završnog rada te asistentu Branimiru Ćaranu na pomoći i savjetima oko korištenja robota.

Zahvaljujem svima koji su mi pomogli u prikupljanju testnih podataka, čime su značajno doprinijeli realizaciji ovog rada.

Veliko hvala mojoj obitelji, prijateljima i kolegama na iskrenoj i bezuvjetnoj podršci i motivaciji te svim lijepim trenucima tijekom studiranja.

Matija Pongračić

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Matija Pongračić** JMBAG: **0035239624**

Naslov rada na hrvatskom jeziku: **Razvoj sustava za prilagođavanje ponašanja robota temeljen na analizi ljudskih kretnji**

Naslov rada na engleskom jeziku: **Development of a system for adapting robot behavior based on the analysis of human movements**

Opis zadatka:

Integracija robota u svakodnevne aktivnosti i radne procese neizbježno nosi sa sobom izazove vezane uz sigurnost i interakciju čovjeka i robota. Napredak u robotici omogućio je razvoj kolaborativnih robota, poznatih pod terminom „koboti" (engl. *cobot – collaborative robot*), koji s ljudima rade na suradnički način. Suradnja s ljudima postavlja visoke zahtjeve za praćenje i interpretaciju ljudskih pokreta. Ovo je posebno važno u situacijama gdje roboti moraju prilagoditi svoje ponašanje u stvarnom vremenu kako bi osigurali sigurnost ljudi u svom okruženju. Prednost takvog pristupa očituje se u sposobnosti robota da reagiraju na ljudske geste, ali i da predviđaju ljudske namjere, smanjujući rizike.

U ovom završnom radu cilj je razviti eksperimentalni postav koji kombinira prepoznavanje ljudskih gesti i predikciju kretanja kako bi se omogućila sigurna i efikasna interakcija između čovjeka i robota. U radu je potrebno:

- napraviti analizu i implementaciju javno dostupnih biblioteka skeletonizacije za precizno praćenje ljudskog tijela i kretnji čovjeka
- napraviti programsku podršku za analizu prostornih putanja te trajektorija i procjenu namjera kretanja čovjeka, koja će omogućiti robotu predviđanje radnje čovjeka u prostoru i dinamički prilagodi svoje ponašanje za unapređenje sigurnosne interakcije
- razviti algoritam za procjenu kutova ljudskog tijela koji će omogućiti detekciju specifičnih gesta i njihovu klasifikaciju
- integrirati razvijene algoritme na robotu, s ciljem adekvatne reakcije na ljudske pokrete i geste.

Funkcionalnosti iz završnog rada potrebno je demonstrirati na odabranom industrijskom robotu u Laboratoriju za autonomne sustave, Regionalnog centra izvrsnosti za robotske tehnologije.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan: 24. 4. 2024.

Zadatak zadao: doc. dr. sc. Marko Švaco

Datum predaje rada:
2. rok (izvanredni): 11. 7. 2024.
3. rok: 19. i 20. 9. 2024.

Predviđeni datumi obrane:
2. rok (izvanredni): 15. 7. 2024.
3. rok: 23. 9. – 27. 9. 2024.

Predsjednik Povjerenstva:
izv. prof. dr. sc. Petar Ćurković

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Jedinica	Opis
d	px	Udaljenost detektirane i stvarne točke
d_t	px	Stvarna udaljenost desnog ramena i lijevog kuka
$F1$	-	F1-mjera
H	m	Vertikalna duljina vidnog polja kamere
h	m	Vertikalna pozicija (visina) kamere
L	m	Udaljenost na kojoj stopala ulaze u vidno polje kamere
P	-	Preciznost
R	-	Osjetljivost
r_0	m	Granica zone 0
r_1	m	Granica zone 1
r_2	m	Granica zone 2
r_k	m	Granica radnog/kolizijskog prostora robota
v	m/s	Stvarna brzina točke
v_{bh}	m/s	Prosječna brzina brzog hodanja
v_h	m/s	Prosječna brzina hodanja
v_{in}	-	Inicijalna brzina robota
v_m	m/s	Granična brzina mirovanja
v_{out}	-	Izlazna brzina robota
v_t	m/s	Prosječna brzina trčanja
w	px/s	Brzina točke na slici po y-koordinati
x	m	Udaljenost čovjeka od kamere
x'	m	Udaljenost čovjeka od točke ulaska stopala u vidno polje kamere
y	px	y-koordinata slike
y'	m	Vertikalna udaljenost stopala od donje točke vidnog polja kamere
φ	°	Polovica vertikalnog kuta vidnog polja kamere

POPIS KRATICA

Kratica	Opis
2D	Dvodimenzionalno
3D	Trodimenzionalno
API	*Application Programming Interface* (sučelje za programiranje aplikacija)
CRTA	Regionalni centar izvrsnosti za robotske tehnologije
FN	*False Negative* (lažno negativno)
FP	*False Positive* (lažno pozitivno)
FPS	*Frames Per Second* (kadrovi po sekundi)
GUI	*Graphical User Interface* (grafičko korisničko sučelje)
HPE	*Human Pose Estimation* (procjena ljudske poze)
IP	*Internet Protocol* (internet protokol)
PDJ	*Percentage of Detected Joints* (postotak detektiranih zglobova)
PyPI	*Python Package Index* (indeks *Python* paketa)
RGB	*Red-Green-Blue* (crveno-zeleno-plavo)
SDK	*Software Development Kit* (komplet za razvoj softvera)
TCP	*Transmission Control Protocol* (protokol za kontrolu prijenosa)
TN	*True Negative* (istinito negativno)
TP	*True Positive* (istinito pozitivno)

SAŽETAK

Napretkom u robotici, tradicionalni industrijski roboti sve više ustupaju mjesto kolaborativnim robotima, tzv. „kobotima“, koji su namijenjeni da dijele radni prostor s ljudima i zajednički izvršavaju zadatke.

U završnom radu razvijen je sustav koji kombinira tehnologije prepoznavanja ljudskih gesti i procjene namjera kretanja čovjeka s ciljem prilagođavanja ponašanja robota kako bi se osigurala robusna interakcija i povećala sigurnost u radnom okruženju.

Provedena je analiza i evaluacija javno dostupnih biblioteka za procjenu ljudske poze i skeletonizaciju ljudskog tijela, nakon čega su koncipirani algoritmi za implementaciju u sustavu te je dodatno razvijen i evaluiran vlastiti model za prepoznavanje gesti. Konačno, sustav je implementiran integracijom na robotu u Regionalnom centru izvrsnosti za robotske tehnologije (CRTA-i).

Ključne riječi: kobot, procjena ljudske poze, namjere kretanja, geste, algoritam

SUMMARY

With advancements in robotics, traditional industrial robots are increasingly being replaced by collaborative robots, also known as „cobots”, designed to share workspace with humans and perform tasks collaboratively.

In the Bachelor's thesis, a system has been developed that combines human gesture recognition technologies and movement intention estimation to adjust the robot’s behavior, ensuring robust interaction and enhanced safety in the working environment.

An analysis and evaluation of open-source libraries for human pose estimation and body skeletonization has been conducted, followed by the conceptualization of algorithms for implementation in the system and, additionally, the development and evaluation of a custom model for gesture recognition. Finally, the system has been implemented through integration with the robot at the Regional Center of Excellence for Robotic Technology (CRTA).

Key words: cobot, human pose estimation, movement intentions, gestures, algorithm

1. UVOD

Ubrzanim razvojem tehnologije i automatizacije, roboti su postali dio mnogih industrijskih i svakodnevnih procesa. Tradicionalni industrijski roboti, koji su iz sigurnosnih razloga obično izolirani od ljudi, sve više ustupaju mjesto kolaborativnim robotima, poznatima kao „koboti“ (eng. *cobot*). Koboti su namijenjeni da dijele radni prostor s ljudima omogućujući direktnu interakciju na način koji poboljšava učinkovitost, ali i sigurnost u radnom okruženju. Integracija kobota u radne procese nosi sa sobom mnogo zahtjeva među kojima su najvažniji sigurnost ljudi i robusna interakcija, stoga oni moraju biti sposobni pratiti i interpretirati ljudske kretnje te prilagođavati svoje ponašanje u stvarnom vremenu.

Shodno tome, cilj ovog završnog rada je razviti sustav koji kombinira tehnologije prepoznavanja ljudskih gesti i procjene namjera kretanja čovjeka. Rad obuhvaća analizu i evaluaciju javno dostupnih biblioteka za skeletonizaciju i praćenje ljudskog tijela i kretnji, razvoj algoritma za prilagođavanje brzine rada robota s obzirom na kretanje čovjeka u radnom prostoru, razvoj algoritama za detekciju i klasifikaciju specifičnih gesti te, konačno, integraciju razvijenih rješenja na robotu u Laboratoriju za autonomne sustave Regionalnog centra izvrsnosti za robotske tehnologije (CRTA-e).

Svi algoritmi i programska rješenja razvijeni u sklopu ovog rada implementirani su u programskom jeziku *Python*, a svi korišteni kôdovi dostupni su u priloženim *GitHub* repozitorijima.

2. PREGLED I ODABIR BIBLIOTEKA ZA SKELETONIZACIJU

Kako bi robot mogao pratiti i analizirati ljudske kretnje te samim time u stvarnom vremenu prilagođavati svoje ponašanje, u njegov sustav potrebno je implementirati algoritam koji omogućuje detekciju i praćenje ključnih točaka ljudskog tijela (eng. *Human Pose Estimation* – HPE), odnosno skeletonizaciju ljudskog tijela. Ti algoritmi mogu se dijeliti s obzirom na:

- dimenzionalnost prostora detekcije – dvodimenzionalno (2D) ili trodimenzionalno (3D)
- mogući broj detektiranih osoba – jedna osoba (eng. *single-person*) ili više osoba (eng. *multi-person*)
- način detekcije točaka – „od vrha prema dolje“ (eng. *top-down*) ili „od dna prema gore“ (eng. *bottom-up*)

Kod metode *top-down* algoritam najprije detektira broj osoba na slici te za svaku od njih kreira zaseban okvir, a zatim unutar svakog okvira procjenjuje pozicije ključnih točaka. Suprotno tome, metoda *bottom-up* prvo detektira ključne točke na cijeloj slici te ih nakon toga grupira kako bi formirale skelet za svaku osobu [1].

HPE tehnologija danas se koristi u raznovrsnim područjima, npr. fizioterapiji [2], analizi pokreta u sportovima [3] i video nadzoru [4]. Zbog sve veće potražnje, razvijene su biblioteke za korištenje HPE algoritama koje pružaju podršku za različite programske jezike (primarno *Python* i C++) te korisnicima znatno olakšavaju implementaciju tih algoritama u svojim projektima.

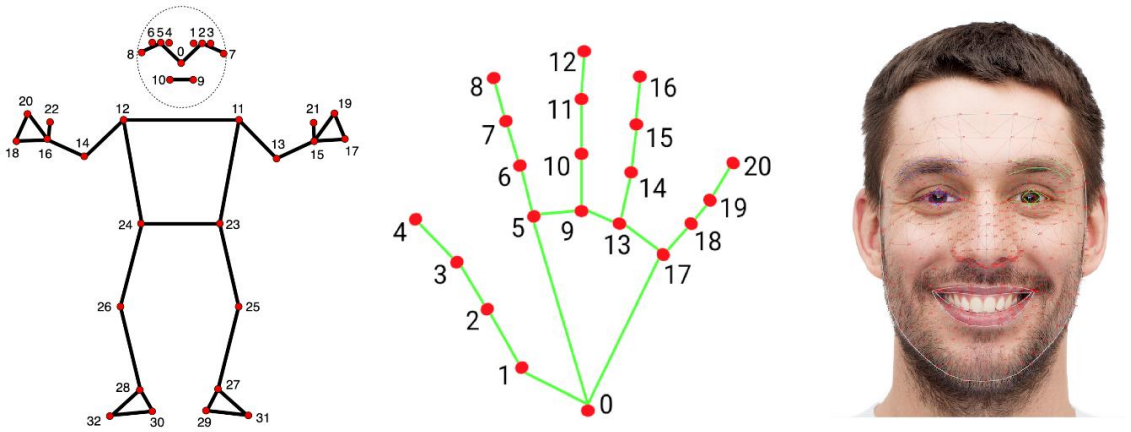
2.1. Javno dostupne biblioteke

Biblioteke koje su korisnicima javno dostupne za preuzimanje, korištenje, modificiranje i komercijalnu upotrebu najčešće pružaju dovoljno funkcionalnosti i upotrebljive su za raznovrsne znanstvene i istraživačke projekte. S druge strane, biblioteke čije licence zahtijevaju plaćanje ili dozvolu vlasnika, nude naprednije značajke, ali su optimizirane za specifične zadaće, dakle ne upotrebljavaju se u bilo kakve svrhe.

U sklopu ovog rada usporedit će se performanse 5 javno dostupnih 2D HPE biblioteka/modela: *MediaPipe, OpenPose, YOLOv8* te *MoveNet* sa svoja dva modela, *Lightning* i *Thunder*.

2.1.1. MediaPipe [5]

MediaPipe je biblioteka razvijena od strane *Google*-a za analizu i obradu vizualnih podataka koja nudi rješenja za izazove poput detekcije objekata, segmentacije slika i sl., a u kontekstu skeletonizacije ljudskog tijela nudi modele za detekciju ključnih točaka tijela (33 točke), šake (21 točka) i lica (478 točaka).



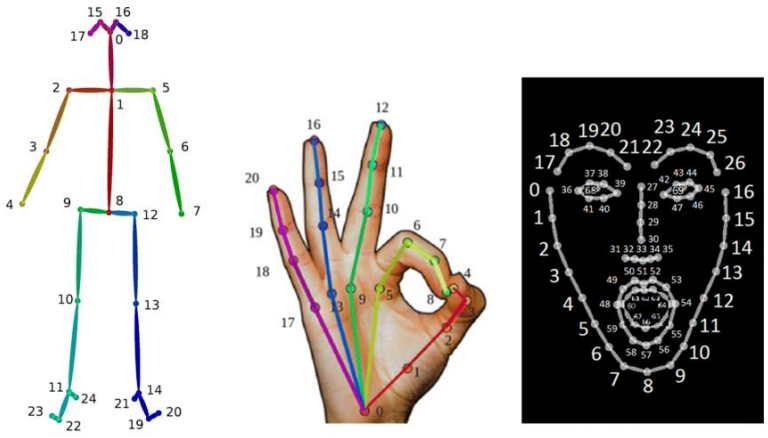
Slika 1. *MediaPipe* – pozicije ključnih točaka tijela, šake i lica [5]

MediaPipe koristi *top-down* metodu detekcije točaka te njegovi modeli mogu detektirati samo jednu osobu na slici (*single-person*). Biblioteka je dostupna na PyPI (eng. *Python Package Index*), glavnom repozitoriju za *Python* pakete te se može instalirati u naredbenom retku (eng. *command prompt*) pomoću naredbe *pip install mediapipe*[1].

2.1.2. OpenPose [6]

Carnegie Mellon University 2017. godine razvio je biblioteku *OpenPose* za praćenje položaja tijela (HPE) koja može detektirati više osoba na istoj slici (*multi-person*) metodom *bottom-up* i davati informacije o ključnim točkama tijela, šake, lica i stopala za svaku od njih. Za detekciju tijela koristi se model *BODY_25* (zadani, detektira 25 točaka) ili rjeđe *COCO* (sporiji, manje točnosti, ali resursno manje zahtjevan, detektira 17 točaka). *OpenPose* također detektira 21 točku za svaku šaku, 70 točaka za lice te 6 za stopala.

[1] Verzije starije od 0.9.1.0 više nisu dostupne na PyPI



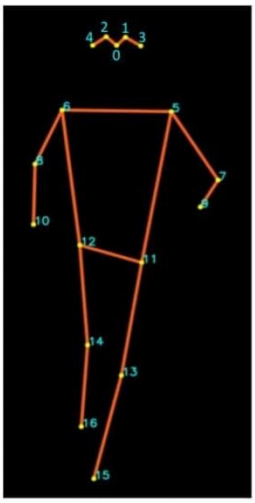
Slika 2. *OpenPose* - pozicije ključnih točaka tijela, šake i lica [7]

Budući da biblioteka nije dostupna na PyPI, potrebno ju je preuzeti s repozitorija navedenog u dokumentaciji. Može se izgraditi iz izvornog kôda (*„build from source“*) ili instalirati portabilna demo verzija za *Windows*, koja je upotrijebljena u ovom radu.

2.1.3. YOLOv8 [8]

Inicijalno zamišljen kao model za detekciju objekata i segmentaciju slika, tvrtka *Ultralytics* razvila je prvu verziju modela *YOLO* (*You Only Look Once*) 2016. godine. U novijim verzijama dodane su mnoge druge značajke te tako verzija *YOLOv8* ima mogućnost skeletonizacije ljudskog tijela (HPE) i praćenja u stvarnom vremenu. Za upotrebu se nudi nekoliko sličnih HPE modela koji su unaprijed izrađeni uz pomoć biblioteke *PyTorch* te su dostupni u dokumentaciji. Modeli detektiraju 17 ključnih točaka na tijelu.

Index	Key point
0	Nose
1	Left-eye
2	Right-eye
3	Left-ear
4	Right-ear
5	Left-shoulder
6	Right-shoulder
7	Left-elbow
8	Right-elbow
9	Left-wrist
10	Right-wrist
11	Left-hip
12	Right-hip
13	Left-knee
14	Right-knee
15	Left-ankle
16	Right-ankle

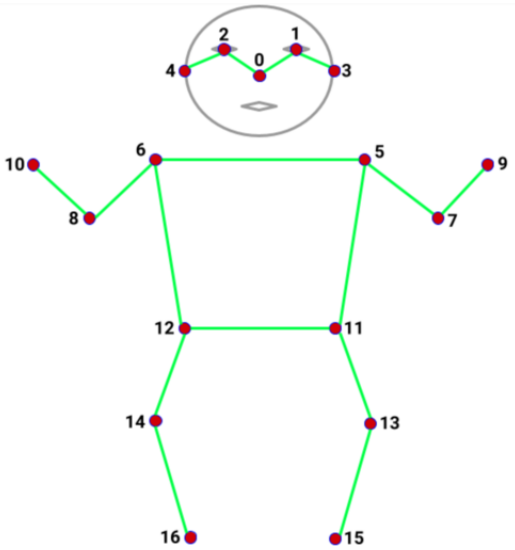


Slika 3. *YOLOv8* - pozicije ključnih točaka tijela [9]

Za detekciju se koristi *top-down* metoda te se može detektirati više osoba na slici (*multi-person*). *Ultralytics* paket dostupan je na PyPI te se pomoću naredbe *pip install ultralytics* u naredbenom retku instaliraju *YOLOv8* i sve potrebne ovisnosti za njegovo funkcioniranje.

2.1.4. MoveNet [10]

Osim *MediaPipe*-a, *Google* je razvio i *MoveNet* modele za preciznu i brzu detekciju i praćenje točaka ljudskog tijela u stvarnom vremenu. Implementirani su i optimizirani za rad unutar *TensorFlow*-a, biblioteke za strojno učenje i razvoj modela, podržavajući i njegove varijante poput *TensorFlow Lite* (za uređaje s ograničenim resursima) i *TensorFlow.js* (za izvođenje u *web* preglednicima). Postoje dvije varijante *MoveNet*-a – *Lightning* (brži, manje točnosti) i *Thunder* (sporiji, veće točnosti). Razlikuju im se rezolucije ulaznih slika, dok je implementacija gotovo identična te oba detektiraju 17 ključnih točaka tijela.



Slika 4. *MoveNet* - pozicije ključnih točaka tijela [11]

Obje varijante koriste *bottom-up* metodu za detekciju točaka te nude modele za praćenje jedne osobe (*single-person*) ili više osoba (*multi-person*). Kako bi se *MoveNet* mogao koristiti, potrebno je instalirati *TensorFlow* pomoću naredbe *pip install tensorflow==2.10.0* u naredbenom retku[2].

[2] 2.10 je posljednja verzija *TensorFlow*-a koja podržava operativni sustav *Windows*

2.2. Usporedba biblioteka

Kako bi se mogla odabrati optimalna biblioteka za implementaciju u algoritme u daljnjim koracima rada, potrebno je usporediti njihovu točnost i brzinu. Budući da biblioteke imaju različite značajke, definirat će se zajednički kriteriji prema kojima će se one uspoređivati i kriteriji koje testni podaci moraju zadovoljiti.

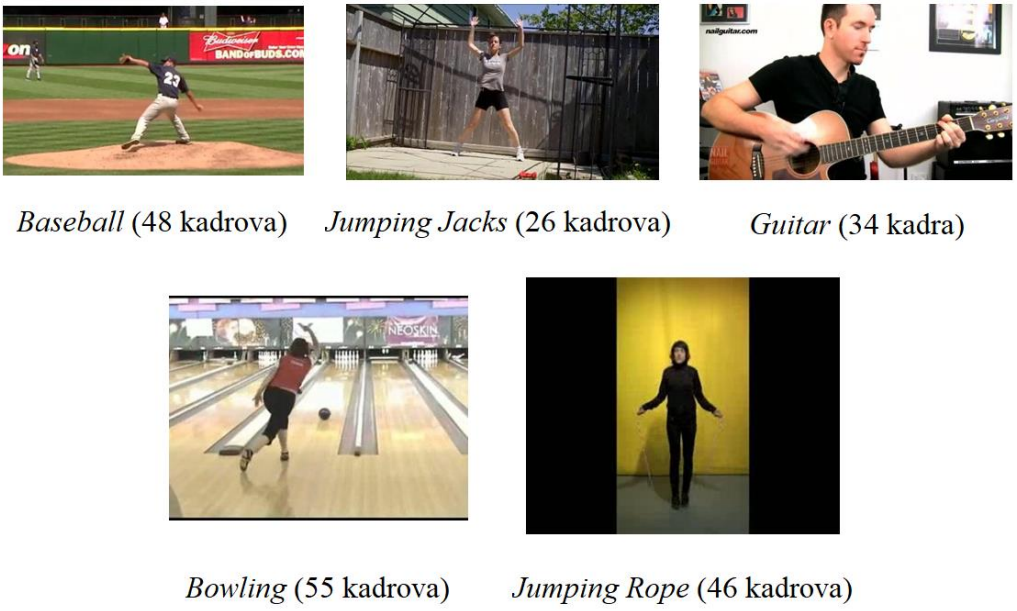
Tablica 1. Usporedba značajki biblioteka

BIBLIOTEKA	MOGUĆI BROJ DETEKTIRANIH OSOBA	MODELI	BROJ KLJUČNIH TOČAKA TIJELA
MediaPipe	*single-person*	tijelo, šake, lice	33
OpenPose	*multi-person*	tijelo, šake, lice	25
YOLOv8	*multi-person*	tijelo	17
MoveNet (*Lightning* i *Thunder*)	*single-person* ili *multi-person*	tijelo	17

Kako bi sve biblioteke bile ravnopravne tijekom usporedbe, prema podacima iz Tablice 1 vidljivo je da se moraju odabrati testni podaci gdje će se detektirati i pratiti samo jedna osoba te tražiti isključivo onih 17 ključnih točaka tijela koje su zajedničke svakoj od njih. Uzevši u obzir da je svrha algoritama izrađenih u daljnjem dijelu rada detekcija i praćenje u stvarnom vremenu, potrebno je da testni podaci budu u obliku videozapisa, tj. pojedinačnih kadrova iz svakog od njih.

Penn Actions [12] je javno dostupan skup podataka koji sadrži 2326 video sekvenci koje prikazuju 15 različitih radnji snimljenih iz različitih perspektiva. Svaka video sekvenca spremljena je u obliku kadrova i pridružena joj je datoteka formata *.mat* u kojoj su navedeni podaci o stvarnim položajima 13 ključnih točaka na slici (eng. *ground truth*) izraženim u pikselima te njihova vidljivost izražena *Booleovom* vrijednošću (istina/neistina, odnosno *true/false*). Jedna od 13 ključnih točaka je „glava“, a budući da svaka biblioteka detektira minimalno 5 točaka na glavi, ne može se odrediti koja od njih je ekvivalentna onome što predstavlja točka glave u *Penn Actions* podacima. Shodno tome, u obzir će se uzimati podaci o ostalih 12 točaka koje mogu detektirati sve biblioteke koje se uspoređuju. Kako su video sekvence primarno fokusirane na detekciju jedne osobe, *Penn Actions* zadovoljava sve ranije nabrojane kriterije.

Od ponuđenih 2326 video sekvenci odabrano je 5 koje će se upotrijebiti za testiranje biblioteka. Svaka od njih predstavlja različitu radnju, snimljena je iz različite perspektive i s različitom udaljenošću osobe.



Slika 5. Odabrane video sekvence

2.2.1. Usporedba brzine

Jedan od zahtjeva za biblioteke jest da njihova brzina obrade slika i detekcije točaka bude što veća kako bi se praćenje osobe moglo učinkovito odvijati u stvarnom vremenu te kako bi robot na vrijeme mogao reagirati i prilagoditi svoje ponašanje.

Mjera brzine za usporedbu je FPS (eng. *Frames Per Second*, kadrovi po sekundi). Biblioteke obrađuju i detektiraju točke na svih 5 video sekvenci te se za svaku od njih računa prosječni FPS. Mjeri se ukupno vrijeme obrade jedne video sekvence, a zatim se broj kadrova iste podijeli izmjerenim vremenom. Veći FPS znači da je brzina obrade i detekcije veća i obrnuto.

$$FPS = \frac{broj\ kadrova}{vrijeme\ obrade} \quad \textbf{(1)}$$

Iznimku čini biblioteka *OpenPose*, čiji je FPS prilagodljiv promjenom veličine ulazne slike koju neuronska mreža obrađuje, a to se ostvaruje pomoću naredbe *--net_resolution -1xN* (gdje je *N* višekratnik broja 16) tijekom pokretanja obrade u naredbenom retku. Korištenjem većeg broja *N* smanjuje se brzina i povećava točnost detekcije, dok njegovo smanjenje rezultira

povećanjem brzine i smanjenjem točnost. Prema zadanim postavkama *OpenPose* pokazuje FPS vrijednosti 1,6. Pokretanjem obrade podataka s različitim vrijednostima broja *N* ustanovljeno je da se naredbom *--net_resolution -1x80* ostvaruje maksimalna vrijednost FPS-a za koju će *OpenPose* detektirati ljudski skelet, a ona iznosi 10,2. Spomenute vrijednosti FPS-a očitane su tijekom obrade slika zato što portabilna demo verzija ne nudi *Python* programsko sučelje za *OpenPose*, ali automatski ispisuje vrijednost FPS-a.

Tablica 2 prikazuje izmjerene vrijednosti FPS-a za ostale biblioteke.

Tablica 2. Izmjereni FPS biblioteka

BIBLIOTEKA	***BASEBALL***	***JUMPING JACKS***	***GUITAR***	***BOWLING***	***JUMPING ROPE***	***PROSJEČNI FPS***
MediaPipe	29,52	29,60	29,38	29,40	30,12	**29,60**
YOLOv8	6,93	6,63	6,79	6,51	6,72	**6,72**
MoveNet Lightning	52,57	53,39	53,88	52,43	51,98	**52,85**
MoveNet Thunder	15,88	15,83	15,82	15,72	15,70	**15,79**

MoveNet Lightning uvjerljivo pokazuje najveću brzinu, dok *YOLOv8* pokazuje najmanju. *MediaPipe* je gotovo duplo brži od *MoveNet Thunder*-a, ali sporiji od *MoveNet Lightning*-a. *MoveNet Thunder* je također sporiji od *MoveNet Lightning*-a, no značajno brži od *YOLOv8*. Uključivanjem *OpenPose*-a u usporedbu, zaključujemo da je prema zadanim postavkama njegova brzina daleko najmanja, a konfiguriranjem ulazne rezolucije da se postigne najveća brzina, ona je i dalje mala u usporedbi s ostalim bibliotekama i veća je tek od *YOLOv8*.

2.2.2. Usporedba točnosti

Na isti način kao što je slučaj s brzinom biblioteka, uspoređuje se njihova točnost, tj. koliko točno mogu detektirati i odrediti pozicije ključnih točaka. Biblioteke obrađuju i detektiraju ključne točke na kadrovima video sekvenci, a zatim se njihovi rezultati uspoređuju sa stvarnim pozicijama točaka navedenima u odgovarajućim *.mat* datotekama, odnosno računa se apsolutna udaljenost detektirane i stvarne točke u pikselima, prema izrazu:

$$d = \sqrt{(x_{\mathrm{sv}} - x_{\mathrm{b}})^2 + (y_{\mathrm{sv}} - y_{\mathrm{b}})^2} \quad \textbf{(2)}$$

Kao mjerilo točnosti uzima se stvarna udaljenost desnog ramena i lijevog kuka, a u slučaju da lijevi kuk nije vidljiv na slici (kao u video sekvenci *Guitar*), uzima se stvarna udaljenost desnog i lijevog ramena.

$$d_{\mathrm{t}} = \sqrt{(x_{\mathrm{dr}} - x_{\mathrm{lk}})^2 + (y_{\mathrm{dr}} - y_{\mathrm{lk}})^2} \quad \textbf{(3)}$$

Ako su stvarna vrijednost i detektirana točka međusobno udaljeni za manje od jedne desetine d_{t}, detektirana točka smatra se ispravnom. Suprotno tome, ako je udaljenost veća ili jednaka jednoj desetini d_t, smatra se pogrešnom.

Kao što je navedeno, u *.mat* datotekama postoje podaci o vidljivosti točaka izraženi *Booleovom* vrijednošću. U slučaju da neka od točaka prema tim podacima nije vidljiva, ona se ne uzima u obzir kod testiranja zato što modeli često (ispravno) procjenjuju njezinu stvarnu poziciju, stoga takve detekcije nije korektno nazivati pogrešnima.

U kadru se računa udaljenost svakog od 12 parova detektiranih točaka i stvarnih vrijednosti. Nakon toga određuje se broj ispravno detektiranih točaka te se na temelju toga izračunava PDJ (eng. *Percentage of Detected Joints,* postotak detektiranih zglobova). To je konačna mjera uspješnosti detekcije točaka u jednom kadru, a računa se kao broj ispravno detektiranih točaka podijeljen s brojem vidljivih točaka te se izražava kao postotak. Kada su sve točke ispravno detektirane, PDJ je jednak 100 %, a kada su sve pogrešno detektirane, PDJ je 0 %.

$$PDJ = \frac{\sum_{i=1}^{n} bool(d_i < 0{,}1 \cdot d_t)}{n} \quad \textbf{(4)}$$

Budući da točnost *OpenPose*-a varira ovisno o njegovoj ulaznoj rezoluciji, testiraju se dva slučaja: prema zadanim postavkama (kada je FPS = 1,6) te s maksimalnom postignutom brzinom (kada je FPS = 10,2).

Tablica 3 prikazuje broj kadrova koje biblioteke prepoznaju u svakom specifičnom rasponu PDJ vrijednosti u pojedinoj video sekvenci.

Tablica 3. Broj kadrova video sekvenci s obzirom na PDJ za svaku biblioteku

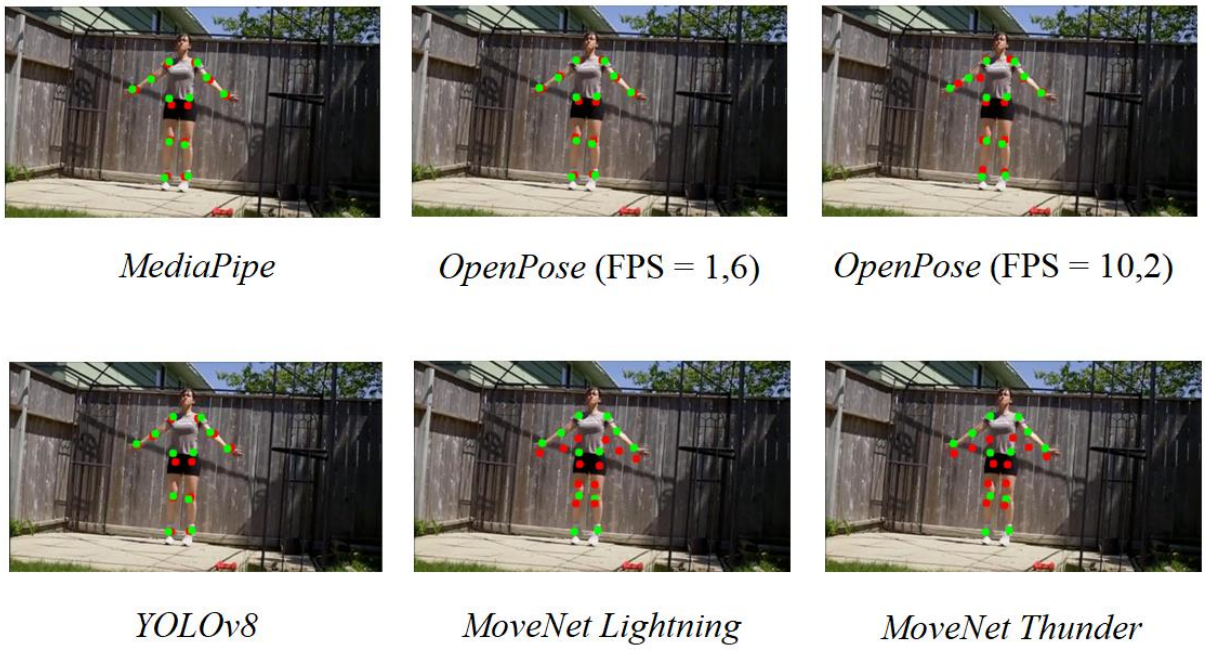
VIDEO SEKVENCA	***MediaPipe***			
	0 % ≤ PDJ < 25 %	**25 % ≤ PDJ < 50 %**	**50 % ≤ PDJ < 75 %**	**75 % ≤ PDJ ≤ 100 %**
Baseball	2	9	21	16
Jumping Jacks	1	6	12	7
Guitar	0	0	20	14
Bowling	3	16	33	3
Jumping Rope	0	4	17	25
UKUPNO	**6**	**35**	**103**	**65**
	2,87 %	**16,75 %**	**49,28 %**	**31,10 %**
VIDEO SEKVENCA	***OpenPose* (FPS = 1,6)**			
	0 % ≤ PDJ < 25 %	**25 % ≤ PDJ < 50 %**	**50 % ≤ PDJ < 75 %**	**75 % ≤ PDJ ≤ 100 %**
Baseball	0	1	14	33
Jumping Jacks	0	4	10	12
Guitar	0	0	11	23
Bowling	0	7	33	15
Jumping Rope	4	11	7	24
UKUPNO	**4**	**23**	**75**	**107**
	1,91 %	**11,00 %**	**35,89 %**	**51,20 %**
VIDEO SEKVENCA	***OpenPose* (FPS = 10,2)**			
	0 % ≤ PDJ < 25 %	**25 % ≤ PDJ < 50 %**	**50 % ≤ PDJ < 75 %**	**75 % ≤ PDJ ≤ 100 %**
Baseball	12	24	12	0
Jumping Jacks	3	14	9	0
Guitar	0	2	29	3
Bowling	27	20	8	0
Jumping Rope	19	23	4	0
UKUPNO	**61**	**83**	**62**	**3**
	29,19 %	**39,71 %**	**29,67 %**	**1,44 %**

VIDEO SEKVENCA	***YOLOv8***			
	0 % ≤ PDJ < 25 %	**25 % ≤ PDJ < 50 %**	**50 % ≤ PDJ < 75 %**	**75 % ≤ PDJ ≤ 100 %**
Baseball	8	19	16	5
Jumping Jacks	1	8	12	5
Guitar	0	13	21	0
Bowling	9	18	26	2
Jumping Rope	1	3	17	25
UKUPNO	**19**	**61**	**92**	**37**
	9,09 %	**29,19 %**	**44,02 %**	**17,70 %**

VIDEO SEKVENCA	***MoveNet Lightning***			
	0 % ≤ PDJ < 25 %	**25 % ≤ PDJ < 50 %**	**50 % ≤ PDJ < 75 %**	**75 % ≤ PDJ ≤ 100 %**
Baseball	48	0	0	0
Jumping Jacks	23	3	0	0
Guitar	4	30	0	0
Bowling	36	16	3	0
Jumping Rope	3	33	10	0
UKUPNO	**114**	**82**	**13**	**0**
	54,55 %	**39,23 %**	**6,22 %**	**0 %**

VIDEO SEKVENCA	***MoveNet Thunder***			
	0 % ≤ PDJ < 25 %	**25 % ≤ PDJ < 50 %**	**50 % ≤ PDJ < 75 %**	**75 % ≤ PDJ ≤ 100 %**
Baseball	47	1	0	0
Jumping Jacks	24	2	0	0
Guitar	0	34	0	0
Bowling	35	16	4	0
Jumping Rope	0	24	22	0
UKUPNO	**103**	**80**	**26**	**0**
	49,28 %	**38,28 %**	**12,44 %**	**0 %**

Slika 6 prikazuje šesti kadar iz video sekvence *Jumping Jacks* i na njemu detektirane točke svake pojedine biblioteke sa stvarnim točkama. Crvenom bojom označene su detektirane točke, a zelenom stvarne pozicije na slici.



Slika 6. *Jumping Jacks* - kadar 6

OpenPose pokazuje najveću točnost u usporedbi s ostalim bibliotekama, ali samo u slučaju kada je konfiguriran prema zadanim postavkama, odnosno kada je FPS = 1,6. Više od polovice broja kadrova postiže PDJ veći ili jednak 75 %, dok samo 4 kadra u jednoj video sekvenci postižu PDJ manji od 25 %. Promjenom na ulaznu rezoluciju za koju se postiže maksimalni FPS, točnost se znatno smanjuje budući da gotovo 70 % kadrova postiže PDJ manji od 50 %. *MediaPipe* pokazuje izvrsnu točnost budući da u više od 80 % kadrova PDJ vrijednost dostiže ili prelazi 50 %, a gotovo trećina njih dostiže i 75 %. S nešto manjom točnošću ističe se *YOLOv8,* koji pokazuje solidne rezultate, no za razliku od *MediaPipe*-a i *OpenPose*-a na zadanim postavkama, njegov PDJ u puno manje slučajeva postiže vrijednosti veću ili jednaku 75 %. *MoveNet Lightning* i *Thunder* pokazuju se kao izrazito slabi kada je točnost detekcije u pitanju jer ni u jednom kadru postignuta PDJ vrijednost ne dostiže 75 %, a okvirno u polovici kadrova ne dostiže ni 25 %.

2.3. Zaključak usporedbe i odabir biblioteke

Rezultati testiranja pokazuju da na zadanim postavkama *OpenPose* pokazuje najbolje rezultate kada je u pitanju točnost detekcije, no u tom se slučaju postiže izrazito mala brzina, koja je u

većini situacija neprihvatljivo mala, posebno u projektima gdje se traži praćenje u stvarnom vremenu. Promjenom postavki na ulaznu rezoluciju s maksimalnom brzinom, točnost postaje vrlo niska, a vrijednost FPS-a od 10,2 je i dalje manja od gotovo svih ostalih biblioteka. Uzevši u obzir spomenute probleme, zaključuje se da, bez obzira na konfiguraciju, *OpenPose* nije upotrebljiv u nastavku rada.

Iako je u dokumentaciji navedeno da je *MoveNet Thunder* model s većom točnošću od *MoveNet Lightning*-a, provedeno testiranje pokazalo je da je razlika vrlo mala te da *MoveNet Thunder* daje gotovo jednako loše rezultate kada je u pitanju točnost. S druge strane, navedeno je da je *MoveNet Lightning* brži model od *MoveNet Thunder*-a, što se testiranjem potvrdilo pokazavši da je njegova brzina trostruko veća, dakako najveća u usporedbi s ostalim bibliotekama. Iako se *MoveNet Lightning* može upotrijebiti u projektima gdje je brzina izrazito važna, a točnost detekcije može imati veće tolerancije, za potrebe ovog rada to nije zadovoljavajuće s obzirom na to da su oba parametra ključna. Jednako tako, *MoveNet Thunder* nije dobar odabir zbog ispodprosječne točnosti i ispodprosječne brzine.

YOLOv8 pokazao se kao biblioteka male brzine i solidne točnosti. Može se razmatrati kao opcija za korištenje u nastavku rada, ali budući da je svrha algoritama praćenje u stvarnom vremenu, mala brzina nije poželjna karakteristika. Nadalje, testiranje je pokazalo da postoji biblioteka koja nadmašuje *YOLOv8* u oba parametra, stoga se svrha njegovog korištenja dovodi u pitanje.

Spomenuta biblioteka jest *MediaPipe*, koji je karakteriziran velikom brzinom i visokom točnošću pa time predstavlja rješenje za balansiranje tih dvaju zahtjeva. Time se omogućava praćenje u stvarnom vremenu, a ujedno i osigurava da neće dolaziti do velikih pogrešaka kod procjene ljudske poze. Shodno tome, *MediaPipe* se odabire kao HPE biblioteka koja će se u nastavku implementirati.

3. IZRADA ALGORITMA ZA PROCJENU NAMJERA KRETANJA ČOVJEKA

U okruženjima gdje robot samostalno izvršava zadatke i dijeli radni prostor s ljudima, najveću opasnost predstavlja mogućnost neželjenog fizičkog kontakta, pri čemu bi robot čovjeku mogao nanijeti ozljede. Kako bi se takve situacije minimizirale ili potpuno spriječile, sigurnost ljudi mora biti prioritet u interakciji s robotom, što iziskuje razvoj i implementaciju sustava koji će robotu omogućiti dinamičko prilagođavanje njegovog ponašanja (prvenstveno brzine) s obzirom na čovjekove kretnje.

Ideja je razviti algoritam koji će uz pomoć kamere i implementirane HPE biblioteke (*MediaPipe*) procjenjivati čovjekovu stvarnu udaljenost i brzinu kretanja. Na temelju toga, robotu će se putem računala slati naredbe za promjenu njegove brzine, s ciljem da ne dođe do neželjenog fizičkog kontakta kada se čovjek približi njegovom kolizijskom prostoru. Algoritam se sastoji od triju dijelova:

- određivanje stvarne udaljenosti i brzine čovjeka na temelju slike s kamere
- postavljanje inicijalne brzine robota na temelju stvarne udaljenosti i brzine čovjeka
- određivanje izlazne brzine koja se šalje robotu na temelju inicijalne brzine

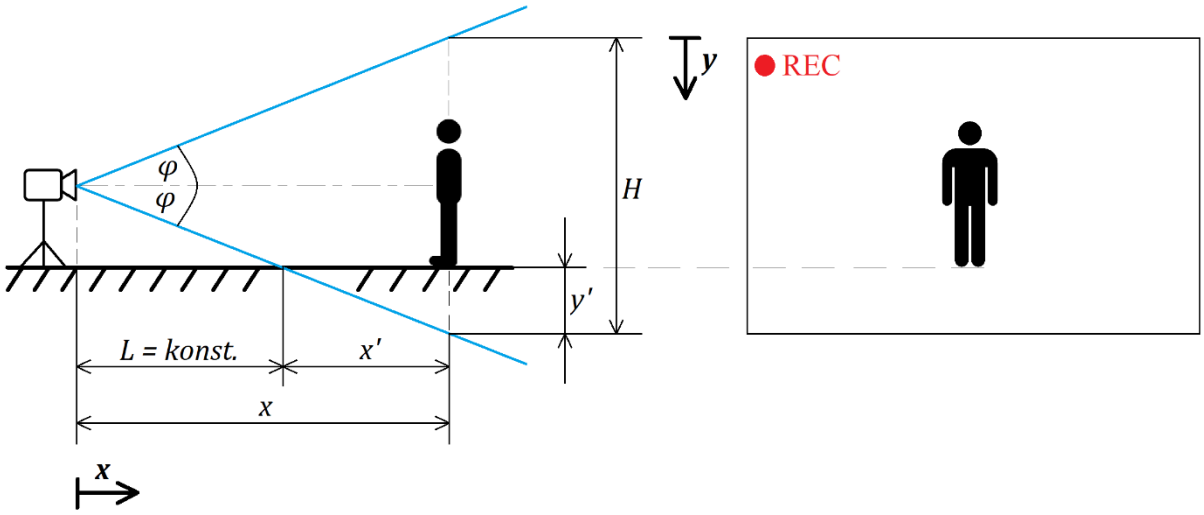
U ovom je poglavlju naglasak stavljen isključivo na proces određivanja brzine robota, a implementacija u kôd, način slanja brzine i integracija algoritma na robotu bit će objašnjeni u dijelu rada koji slijedi.

3.1. Određivanje stvarne udaljenosti i brzine čovjeka

Korištenjem dubinske (3D) kamere, parametri kretanja čovjeka u prostoru (položaj, brzina, akceleracija itd.) mogu se jednostavno odrediti, ali karakteristike takvih kamera su da imaju ograničen domet te generiraju i obrađuju velike količine podataka, čime se izaziva kašnjenje i smanjenje performansi, što je nepovoljno za sustave u kojima je potrebna obrada u stvarnom vremenu.

S ciljem da se navedeni problemi izbjegnu, 2D kamera predstavlja bolje rješenje. S druge strane, određivanje položaja i brzine time postaje puno izazovnije budući da 2D kamera ne pruža izravne informacije o dubini, odnosno udaljenosti objekta od kamere. *MediaPipe* može generirati vrijednosti z-koordinate, odnosno procijeniti udaljenost točke, no te su vrijednosti relativne u odnosu na neki objekt koji *MediaPipe* odabire kao referentnu (srednju) dubinu. Iz tog razloga, time se ne dobivaju precizne informacije o stvarnoj udaljenosti točke od kamere.

Iako postoje tehnička rješenja poput stereo vizije ili kombiniranja više kamera, parametri kretanja mogu se približno odrediti isključivo iz dobivene 2D slike s kamere tako da se stvarna udaljenost (položaj) i brzina promatranog objekta prikažu kao funkcije položaja i brzine na slici.



Slika 7. Položaj čovjeka u vidnom polju kamere

Spomenute funkcije bit će izvedene korištenjem prikaza na Slici 7.

Prije svega, potrebno je odrediti točku na ljudskom tijelu koja će se promatrati te će ista služiti kao referentna za procjenu udaljenosti i brzine čovjeka. Budući da su stopala na istoj visini s površinom tla, što ujedno ne ovisi o čovjekovoj visini ni građi, ona će služiti kao referentna točka. Kod izvoda funkcija pokreti nogu ne uzimaju se u obzir, tj. čovjek se promatra kao da u svakom trenutku drži stopala jedno kraj drugog zato što je važna udaljenost i brzina isključivo jedne točke.

Funkcije vrijede samo za slučajeve kada su stopala unutar vidnog polja kamere, tj. kada vrijedi $x \geq L$.

Visina (vertikalna duljina) vidnog polja kamere na udaljenosti x može se izraziti pomoću trigonometrije pravokutnog trokuta:

$$H = 2 \cdot x \cdot \tan \varphi \quad \textbf{(5)}$$

te na isti način dobivamo izraz za vertikalnu udaljenost stopala od donje točke vidnog polja.

$$y' = x' \cdot \tan \varphi \quad \textbf{(6)}$$

Varijabla x' definira se kao razlika između stvarne udaljenosti od kamere x i udaljenosti L, na kojoj stopala ulaze u vidno polje kamere.

$$x' = x - L \tag{7}$$

Uvrštavanjem jednadžbe (7) u jednadžbu (6) dobiva se izraz:

$$y' = (x - L) \cdot \tan\varphi \tag{8}$$

Sada je potrebno dobiti izraz za *y*-koordinatu promatrane točke na slici. Kada bi se tražila stvarna udaljenost od vrha vidnog polja do stopala, izraz bi se jednostavno dobio kao razlika vrijednosti H *i* y', no *MediaPipe*, odabrana HPE biblioteka, vrijednosti *y*-koordinate daje u rasponu od 0 do 1, odnosno $y \in [0, 1]$, što su zapravo relativne vrijednosti proporcionalne visini slike gdje 0 predstavlja vrh slike, a 1 dno. Ako je neka točka udaljena 30 % od vrha slike, *MediaPipe* će za *y*-koordinatu dati vrijednosti 0,3. Shodno tome, *y*-koordinata se pomoću vrijednosti H i y' izražava kao

$$y = \frac{H - y'}{H} \tag{9}$$

Uvrštavanjem jednadžbi (5) i (8) u jednadžbu (9) dobiva se *y*-koordinata kao funkcija udaljenosti *x*:

$$y = \frac{2 \cdot x \cdot \tan\varphi - (x - L) \cdot \tan\varphi}{2 \cdot x \cdot \tan\varphi} = \frac{2x - (x - L)}{2x}$$

$$y(x) = \frac{x + L}{2x} \tag{10}$$

Cilj je prikazati udaljenost *x* kao funkciju *y*-koordinate na slici, dakle potrebno je pronaći inverznu funkciju one prikazane u jednadžbi (10).

$$y = \frac{x + L}{2x}$$

$$2xy = x + L$$

$$2xy - x = L$$

$$(2y - 1)x = L$$

$$x(y) = \frac{L}{2y - 1} \tag{11}$$

Sada kada je dobivena ovisnost udaljenosti *x* o *y*-koordinati, može se izvesti izraz za stvarnu brzinu *v* kao funkciju *y*-koordinate i *w*, brzine po *y*-koordinati promatrane točke na slici.

Budući da je *y* funkcija vremena, odnosno $y = y(t)$, brzina *w* može se izraziti kao derivacija te funkcije po vremenu. Međutim, računalo ne može primati kontinuirani niz vrijednosti, već radi s diskretnim podacima, dakle prima vrijednosti samo u određenim trenucima pa se zbog toga brzina *w* numerički aproksimira tako da se mjeri promjena *y*-koordinate između dva uzastopna kadra.

$$w(t) = \frac{dy}{dt} = \frac{\Delta y}{\Delta t} \tag{12}$$

Stvarna brzina *v* definira se kao derivacija stvarnog položaja (udaljenosti) po vremenu.

$$v(t) = \frac{dx}{dt} \tag{13}$$

Međutim, funkcija *x* nije direktno izražena u ovisnosti o *t*, stoga se primjenjuje pravilo ulančanog deriviranja, pri čemu je potrebno pronaći derivaciju funkcije *x* po *y*-koordinati.

$$v(t) = \frac{dx}{dy} \cdot \frac{dy}{dt} \tag{14}$$

$$\frac{dx}{dy} = -\frac{L \cdot (2y-1)'}{(2y-1)^2}$$

$$\frac{dx}{dy} = -\frac{2L}{(2y-1)} \tag{15}$$

Konačno, uvrštavanjem jednadžbi (15) i (12) u jednadžbu (14) dobiva se traženi izraz:

$$v(y,w) = -\frac{2L}{(2y-1)} \cdot w \tag{16}$$

Prema definiciji, brzina je pozitivnog predznaka kada njezin vektor ima istu orijentaciju kao vektor položaja, što znači da će brzina *v* biti pozitivna kada se točka udaljava od kamere, a brzina *w* kada se točka na slici spušta prema dolje. Iz iskustva je poznato da se udaljavanjem od kamere točka na slici podiže, a približavanjem prema kameri spušta, dakle predznaci brzina *v* i *w* su u svakom trenutku suprotni čime se može objasniti negativni predznak u jednadžbi (16).

3.2. Postavljanje inicijalne brzine robota

Sljedeći korak u izradi algoritma jest definiranje pravila prema kojima će se, na temelju stvarne udaljenosti i brzine čovjeka, u svakom kadru određivati inicijalna (željena) brzina robota u promatranom trenutku. Ona nije konačna izlazna naredba koju robot prima zato što se radi o velikim i naglim promjenama brzine u malim vremenskim intervalima, što je nepovoljno da robot izvršava zbog mogućnosti gubitka kontrole i oštećenja mehaničkih komponenti.

Kako bi se izbjegli izvodi kompleksnih funkcija ovisnosti inicijalne brzine robota o udaljenosti i brzini čovjeka, ta su dva parametra podijeljena u klase na temelju čijih će se kombinacija donositi odluka.

Prije svega, potrebno je jasno definirati koja će točka služiti kao referentna tijekom praćenja čovjeka. U prethodnom je poglavlju objašnjeno zašto je poželjno odabrati točku na stopalu, no nije konkretno definirano koja je to točka.

MediaPipe može prepoznati 3 ključne točke stopala: gležanj, petu i nožne prste. Proučavanjem u prirodnim uvjetima ustanovljeno je da su nožni prsti dio stopala koji po vertikalnoj osi najmanje oscilira tijekom hodanja, za razliku od pete i gležnja čije je osciliranje jasno uočljivo. Dodatno opažanje jest da je tijekom hodanja u svakom trenutku jedno stopalo u pokretu, a drugo miruje te kad bi se proučavala točka samo na jednom stopalu, dobilo bi se periodično gibanje, koje ne daje konkretne informacije o brzini čovjeka. Povlačenjem dužine od nožnih prstiju jednog stopala do drugog i praćenjem njezine točke polovišta, dobiva se približno kontinuirano gibanje. Ta će točka u svakom trenutku biti referentna za određivanje pozicije i brzine čovjeka.

3.2.1. Klasifikacija brzina čovjeka

Nakon što je izračunata vrijednost brzine kretanja čovjeka v, ona se svrstava u jednu od 5 klasa (kategorija):

- „trčanje“
- „brzo hodanje“
- „hodanje“
- „mirovanje“
- „kretanje unazad“

Budući da granice između ovih klasa nisu jednoznačno definirane, potrebno je precizno odrediti njihove intervale.

„Mirovanje“ se može jednostavno definirati kao trenutak u kojem je brzina *v* jednaka 0, ali kao i sve druge HPE biblioteke, *MediaPipe* nije savršeno precizan u procjeni položaja točaka ljudskog tijela. To znači da u jednom kadru može odrediti *y*-koordinatu točke na jednoj poziciji, a u sljedećem kadru na poziciji pomaknutoj za jedan piksel više ili niže, što će rezultirati izračunatom brzinom koja nije jednaka nuli, stoga je potrebno odrediti apsolutnu vrijednost stvarne brzine do koje će se ona svrstavati u klasu „mirovanje“. Ne postoji univerzalni način za određivanje te brzine pa će se kao početna vrijednost granične brzine mirovanja v_m uzeti 0,1 m/s.

„Kretanje unazad“ je klasa u koju se svrstavaju brzine *v* pozitivnog predznaka, što je objašnjeno u prethodnom poglavlju. Apsolutna vrijednost te brzine ne utječe na odluku o inicijalnoj brzini robota.

Intervali za klase „hodanje“, „brzo hodanje“ i „trčanje“ određeni su izračunavanjem prosječne brzine spomenutih kretanja na temelju mjerenja brzina triju osoba. Na stazi od 15 m svakoj osobi izmjereno je vrijeme potrebno za prelazak, a zatim izračunata srednja brzina promatranog kretanja. S ciljem izbjegavanja utjecaja zaleta i pogrešaka mjerenja vremena, mjerenje je započeto nakon što osoba prijeđe prvih 5 metara te je konačno izmjereno vrijeme prelaska posljednjih 10 metara bez zaleta i zaustavljanja.

Rezultati mjerenja prikazani su u Tablici 4.

Tablica 4. Rezultati mjerenja brzina kretanja

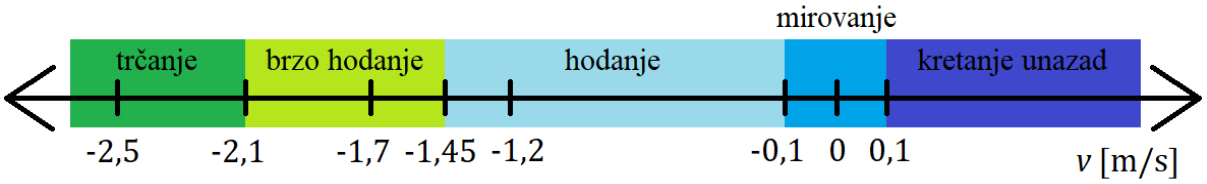
KRETNJA	OSOBA 1	OSOBA 2	OSOBA 3	PROSJEK
Hodanje [m/s]	1,1	1,0	1,4	**1,17**
Brzo hodanje [m/s]	1,7	1,5	1,9	**1,70**
Trčanje [m/s]	2,4	2,3	2,7	**2,47**

Na temelju prikazanih mjerenja određuju se prosječne brzine te se dodaje negativan predznak:

- brzina hodanja, $v_h = -1{,}2$ m/s
- brzina brzog hodanja, $v_{bh} = -1{,}7$ m/s
- brzina trčanja, $v_t = -2{,}5$ m/s

Granice intervala navedenih klasa mogu se proizvoljno odrediti, no u ovom slučaju bit će definirane kao aritmetička sredina prosječnih vrijednosti. Npr. granica između trčanja i brzog hodanja definirana je kao $\frac{v_{bh}+v_t}{2} = \frac{-1{,}7-2{,}5}{2} = -2{,}1$ m/s.

Slika 8 kvalitativno prikazuje raspone brzina za svaku od klasa.



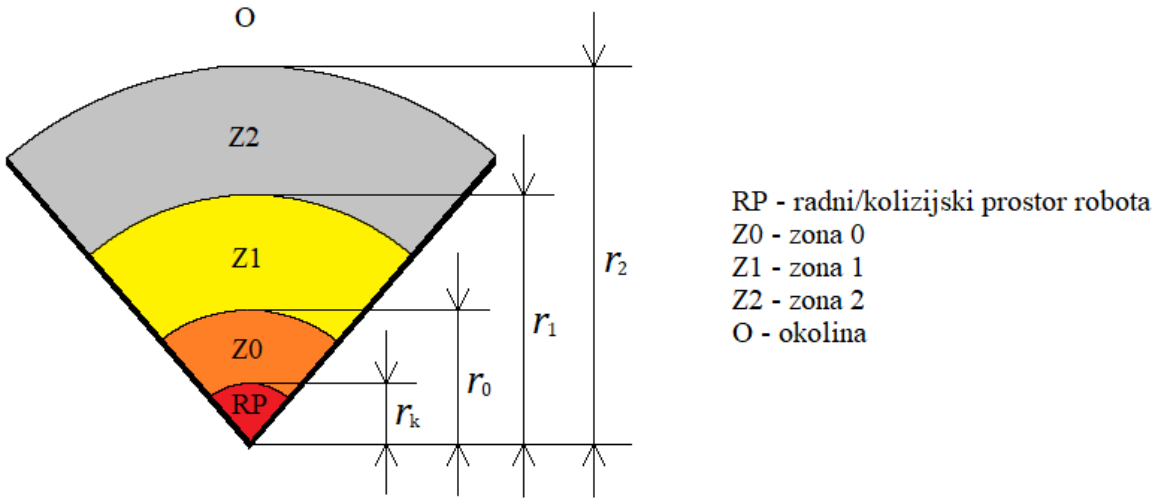
Slika 8. Granice intervala za sve klase brzina

Prethodno je spomenuto da nožni prsti najmanje osciliraju tijekom hodanja, no poznato je da se tijekom trčanja cijelo stopalo podiže visoko od tla pa je tada osciliranje prisutno i kod njih. Veće oscilacije referentne točke stoga mogu biti alternativni pokazatelj da čovjek u promatranom trenutku trči.

3.2.2. Zone udaljenosti

Osim brzina, potrebno je odrediti način na koji će se klasificirati položaj čovjeka u odnosu na kameru. Jedna od metoda jest određivanje zona udaljenosti, tj. područja na temelju udaljenosti od robota koja predstavljaju različite razine opasnosti od mogućeg fizičkog kontakta.

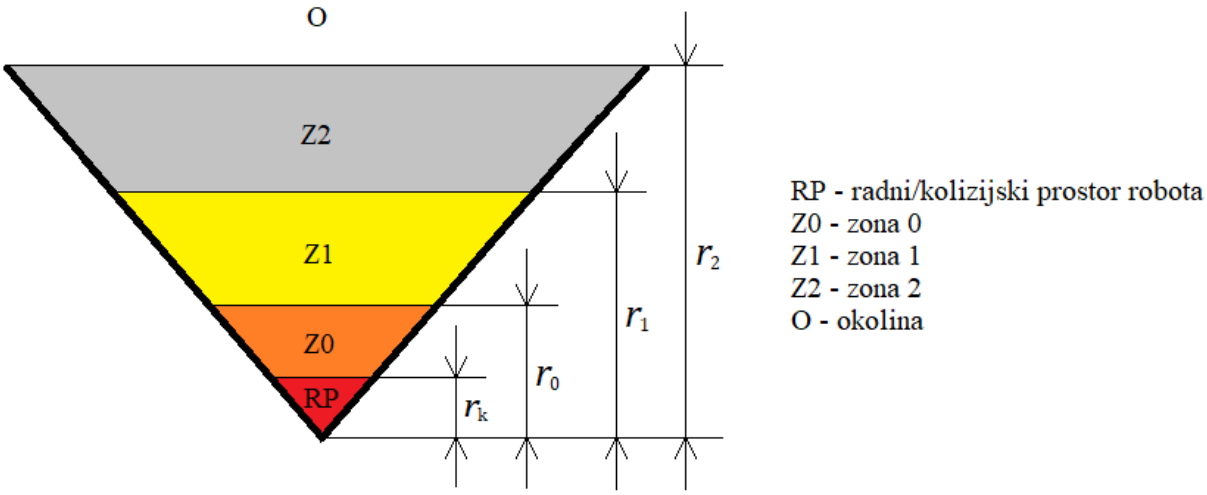
Određene su četiri zone kako je prikazano na Slici 9.



Slika 9. Inicijalni raspored zona udaljenosti

Granice zona prikazane su kao kružni lukovi koji spajaju sve točke jednake udaljenosti od robota/kamere. Iako je ovakav raspored zona najtočniji, kod računanja stvarne udaljenosti čovjeka tada dolazi do problema jer ona također ovisi i o horizontalnom položaju, tj. o *x*-

koordinati na slici s kamere, no budući da je za potrebe ovog algoritma važna isključivo uzdužna udaljenost, odnosno *y*-koordinata na slici s kamere, zone se mogu rasporediti na način koji prikazuje Slika 10.



Slika 10. Konačni raspored zona udaljenosti

Činjenica koja dodatno opravdava ovakav raspored jest da svaka točka u prostoru ostaje u istoj zoni ili prelazi u „opasniju“ zonu, tj. zonu bližu robotu, čime se dodatno povećava sigurnost. Sada preostaje odrediti granice prikazanih zona.

Kolizijski radijus ili doseg r_k je veličina koja ovisi o specifikaciji samog robota, ali i o programu koji izvršava, stoga ona nikada nije fiksna. Kako bi se osigurala stabilnost kamere, zamišljeno je da bude postavljena u blizini baze robota, odnosno da po uzdužnoj osi budu na istoj koordinati, a poprečni razmak da bude što manji te je iz tog razloga važno uzeti u obzir kolizijski prostor robota. Maksimalni doseg kolaborativnih robota tipično je u rasponu od 0,5 do 1,9 metara [13] [14]. Budući da roboti u rijetkim slučajevima koriste svoj maksimalni doseg, za potrebe ovog rada uzet će se 1 m kao vrijednost veličine r_k.

Veličina r_0 granica je zone 0 i zone 1 te se definira kao udaljenost na kojoj se robot može zaustaviti u vremenskom intervalu od 0,5 sekundi kada čovjek brzo hoda prema njemu, bez da dođe do kolizije jer pretpostavka je da čovjek na toliko maloj udaljenosti neće svjesno naglo potrčati prema robotu. Prema tome, r_0 se računa na sljedeći način:

$$r_0 = r_\mathrm{k} + 0{,}5 \cdot v_\mathrm{bh} \qquad \textbf{(17)}$$

Kao brzina za određivanje r_0 može se uzeti apsolutna vrijednost granične brzine brzog hodanja (1,45 m/s) ili prosječne brzine brzog hodanja (1,7 m/s), no uzimanjem veće vrijednosti povećava se sigurnost jer će r_0 biti veći, dakle uzima se da je v_{bh} jednak 1,7 m/s.

$$r_0 = 1 + 0{,}5 \cdot 1{,}7 = 1{,}85 \text{ m}$$

Dobivenu vrijednost poželjno je radi sigurnosti dodatno povećati, stoga je odabrana vrijednost r_0 = 2 m.

Udaljenost r_2, granica zone 2 i okoline, najmanja je udaljenost na kojoj se čovjek može kretati bez ikakve opasnosti te se ona bira proizvoljno, no poželjno je da bude minimalno 5 metara. Za potrebe ovog rada odabrana je vrijednost r_2 = 6 m.

Na temelju odabranih vrijednosti r_0 i r_2 određuje se r_1 tako da udaljenosti budu proporcionalne ili da njihova razlika bude jednaka. Na primjeru ovog algoritma, r_1 će se izračunati kao aritmetička sredina r_0 i r_2 te će se tako postići jednake razlike udaljenosti.

$$r_1 = \frac{r_0 + r_2}{2} = \frac{2 + 6}{2} = 4 \text{ m} \quad \textbf{(18)}$$

3.2.3. Odabir inicijalne brzine robota

Posljednji korak ovog dijela algoritma jest definiranje pravila prema kojima se željena brzina robota odabire na temelju klase čovjekove brzine (kategorije kretnji) i zone u kojoj se nalazi. Pravila se proizvoljno mogu odrediti, no važno je da brzina robota bude manja kada se čovjek nalazi u bližoj zoni i kada mu je brzina približavanja veća, tj. kada je apsolutna vrijednost negativne brzine veća, te suprotno tome, kada je čovjek u daljoj zoni i kada miruje ili se kreće unazad, brzina robota mora biti veća.

U Tablici 5 prikazane su definirane vrijednosti inicijalne brzine robota koje će se u nastavku koristiti, s obzirom na spomenute parametre. Brzina robota izražava se u vrijednostima od 0 do 1 ili u postocima, gdje 1 (100 %) označava maksimalnu brzinu robota, a 0 (0 %) mirovanje.

Tablica 5. Odabir inicijalne brzine robota

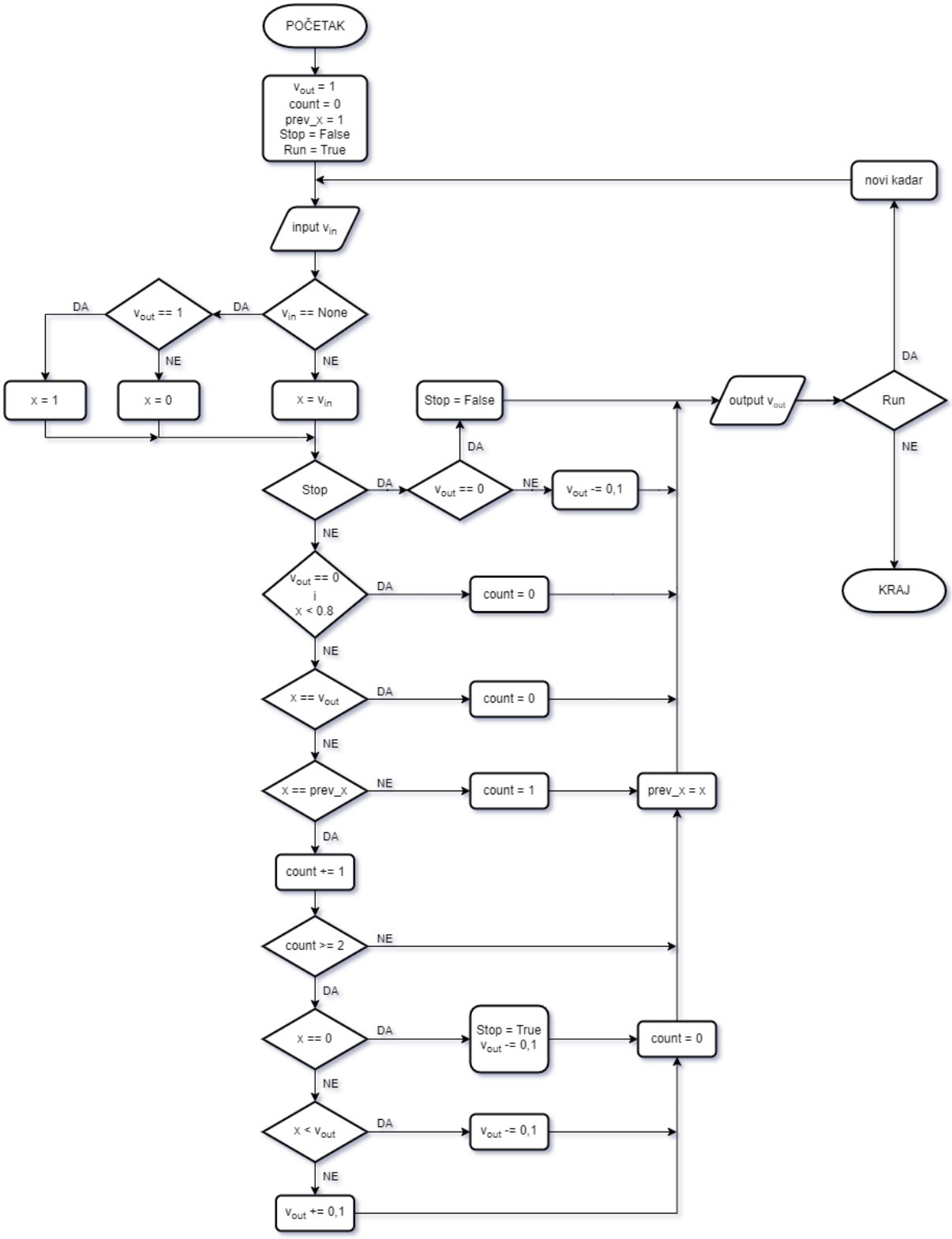
ZONA	KLASA BRZINE				
	Trčanje	Brzo hodanje	Hodanje	Mirovanje	Kretanje unazad
Zona 0	0	0	0	0	0
Zona 1	0	0,3	0,6	0,8	0,8
Zona 2	0	0	0,1	0,3	0,3
Okolina	1	1	1	1	1

U prethodnom je poglavlju navedeno da se čovjekov položaj i brzina mogu računati ako i samo ako se njegova referentna točka nalazi u vidnom polju kamere. Iz tablice je vidljivo da kada se čovjek nalazi u zoni 0, inicijalna brzina robota je 0, bez obzira kreće li se čovjek ili miruje. Shodno tome, ako se odredi da je udaljenost L na Slici 7 jednaka udaljenosti granice r_0, ulazak u zonu 0 rezultira izlaskom referentne točke iz vidnog polja kamere te se tada inicijalna brzina postavlja na 0. Međutim, isto se događa kada čovjek u okolini izlazi iz vidnog polja kamere, a inicijalna brzina tada je 1. Jedno od mogućih rješenja tog problema jest da ako *MediaPipe* detektira neku drugu točku na čovjeku, to znači da se nalazi u zoni 0, a ako ne detektira nijednu, tada je čovjek izašao iz okoline, no to može predstavljati problem kada čovjek iz okoline ne uđe u vidno polje kamere cijelim tijelom jer postoji mogućnost da ni tada *MediaPipe* neće detektirati referentnu točku.

Iz navedenog razloga, kada *MediaPipe* ne detektira referentnu točku (nožni prsti nisu u vidnom polju kamere), konkretna inicijalna (željena) brzina robota ne može se odrediti na temelju jednog kadra te se zato vrijednost postavlja na *„None“*. Kada posljednji dio algoritma primi spomenutu vrijednost umjesto brojčane vrijednosti inicijalne brzine, on će imati mogućnost odrediti pravu, izlaznu brzinu robota na temelju prethodnih kadrova.

3.3. Određivanje izlazne brzine robota

Postavljena inicijalna brzina u prošlom dijelu algoritma robotu služi kao referentna brzina koju nastoji postići u nekom vremenskom intervalu. Kao što je navedeno, nagle promjene brzine robota nisu poželjne, stoga je potrebno odrediti pravila prema kojima će se na temelju postavljene inicijalne brzine dinamički mijenjati brzina robota tijekom vremena.



Slika 11. Detaljan dijagram toka algoritma za određivanje izlazne brzine

Algoritam prikazan dijagramom toka (Slika 11) može se prikazati i pojednostavljenim pseudokodom:

```
 1:  v_out = 1
 2:  dok program nije prekinut činiti
 3:      novi kadar()
 4:      ulaz(v_in)
 5:      ako v_in = None onda
 6:          ako v_out = 1 onda
 7:              v_in = 1
 8:          inače
 9:              v_in = 0
10:          završi ako
11:      završi ako
12:      ako v_in = 0 onda
13:          ako su prošla dva kadra s istim v_in onda
14:              dok v_out != 0 činiti
15:                  v_out = v_out - 0.1
16:                  novi kadar()
17:              završi dok
18:              dok čovjek nije u zoni 2 činiti
19:                  novi kadar()
20:              završi dok
21:          završi ako
22:      inače ako v_in != v_out onda
23:          ako su prošla dva kadra s istim v_in onda
24:              ako v_in < v_out onda
25:                  v_out = v_out - 0.1
26:              inače
27:                  v_out = v_out + 0.1
28:              završi ako
29:          završi ako
30:      završi ako
31:      izlaz(v_out)
32:  završi dok
```

Da bi algoritam funkcionirao, formulirane su dvije pretpostavke. Jedna od njih je da čovjek u vidno polje kamere može ući samo iz okoline, odnosno ne može se pojaviti u vidnom polju i momentalno biti u zoni 0, 1 ili 2. Druga pretpostavka jest da se čovjek na početku programa nalazi u okolini, zbog čega robot radi na maksimalnoj brzini.

Bit ovog dijela algoritma jest da se izlazna brzina robota mijenja postepeno korakom od 0,1 (10 %) i to samo kada se u dvama uzastopnim kadrovima postavi jednaka inicijalna brzina. U tom trenutku uspoređuju se spomenute brzine te u slučaju da se ne podudaraju u vrijednostima, izlazna brzina se smanjuje, odnosno povećava za 0,1 ovisno o tome je li postavljena inicijalna brzina manja ili veća od trenutne izlazne. Npr. ako čovjek prvotno miruje u okolini te zatim brzinom hodanja uđe u zonu 2, inicijalna brzina robota momentalno se mijenja s 1 na 0,6. Početna izlazna brzina robota bila je 1, no ona se počinje smanjivati tek kada je inicijalna brzina od 0,6 postavljena u dvama uzastopnim kadrovima. Najprije se smanji na 0,9 te se nakon toga ponovno čekaju dva uzastopna kadra u kojima će biti postavljena jednaka inicijalna brzina, sve do trenutka kada se spomenute brzine međusobno ne podudaraju. Ovakav pristup osigurava da se izlazna brzina robota mijenja dovoljno glatko, bez naglih promjena. Također, *MediaPipe* ponekad može pogrešno procijeniti položaj točaka u jednom kadru, što može izazvati „trzaj“ koji bi uzrokovao lažne promjene brzine kada bi se uzimala u obzir inicijalna brzina u svakom kadru, što se korištenjem ove metode uspješno sprječava.

Iznimka se događa kada se u dvama uzastopnim kadrovima postavi inicijalna brzina 0 te se tada izlazna brzina robota u svakom kadru smanjuje za 0,1 dok se robot ne zaustavi, tj. izlazna brzina poprimi vrijednosti 0, bez obzira na postavljene inicijalne brzine tijekom tog vremena. Vrijednost izlazne brzine ostaje 0, odnosno robot miruje, sve dok se čovjek ne vrati u zonu 2, kada se postepeno vraća na vrijednost postavljene inicijalne brzine. Ovaj dio algoritma koristi se kako bi se osiguralo da se robot zaustavi što je brže moguće kada je to potrebno, ali bez naglih promjena brzine, i ponovno se počne kretati tek kada je čovjek na dovoljno velikoj udaljenosti.

U slučaju da je umjesto brojčane vrijednosti inicijalna brzina postavljena na „*None*“, algoritam će je prilagoditi u ovisnosti o izlaznoj brzini robota u prethodnom kadru. Ako je ta brzina bila 1, znači da je čovjek u okolini izašao iz vidnog polja kamere, stoga se vrijednost postavlja na 1. U suprotnom, čovjek je ušao u zonu 0, dakle vrijednost inicijalne brzine postavlja se na 0.

Algoritam se izvodi u beskonačnoj petlji sve dok ga korisnik ručno ne prekine.

4. IZRADA, USPOREDBA I IMPLEMENTACIJA MODELA ZA DETEKCIJU I KLASIFIKACIJU SPECIFIČNIH GESTI

Dijeljenje radnog prostora s ljudima, osim sigurnosti, od robota zahtjeva i sposobnost razumijevanja i pravilnog reagiranja na ljudske naredbe izražene putem gestikulacija, što znači da robot mora biti opremljen sustavom za detekciju i klasifikaciju gesti kako bi čovjek mogao intuitivno i efikasno komunicirati s njime u zajedničkom radnom okruženju. Iz tog je razloga ključan izazov razviti, odnosno odabrati model koji precizno i točno prepoznaje različite geste u stvarnom vremenu, omogućujući time robotu da se prilagodi dinamici radnog prostora i osigura produktivnu suradnju s čovjekom.

Odabrani model bit će implementiran tako da svaka gesta predstavlja određenu naredbu koju robot izvršava čim je model prepozna te na nju reagira na odgovarajući način, dajući time mogućnost čovjeku da putem kamere aktivno upravlja ponašanjem robota.

Modeli su ograničeni na geste koje uključuju jednu šaku, stoga se u svakom od njih za skeletonizaciju koristi *MediaPipe*-ov model za detekciju ključnih točaka ljudske šake.

4.1. Modeli i geste za usporedbu točnosti

Prije izrade vlastitog modela potrebno je odabrati unaprijed izrađene javno dostupne modele s kojima će se njegova točnost uspoređivati, s ciljem da se utvrde specifične geste koje treba prepoznavati.

Odabrana su dva javno dostupna modela te su navedena njihova imena i nazivi gesti koje svaki od njih prepoznaje.

Tablica 6. Javno dostupni modeli i geste koje prepoznaju

MODEL	GESTE[3]	
TechVidvan [15]	*Closed fist* (Zatvorena šaka)	*OK*
	Thumbs up (Palac gore)	*Call me* (Nazovi me)
	Thumbs down (Palac dolje)	*Smile* (Smiješak)
	Peace (Mir)	*Stop*
	Rock	*Live long* (Živi dugo)

[3] Imena gesti se razlikuju od modela do modela. U tablici su navedena univerzalna imena koja će biti korištena na svim testiranim modelima.

MediaPipe HandGestureClassifier [16]	*Pointing up* (Pokazivanje prema gore) *Open palm* (Otvoreni dlan) *Closed fist* (Zatvorena šaka) *Thumbs up* (Palac gore)	*Thumbs down* (Palac dolje) *Peace* (Mir) *Rock* *None* (Ništa)

Modeli zajedno prepoznaju ukupno 12 gesti prikazanih na Slici 12, no samo 5 njih je zajedničko obama. *MediaPipe HandGestureClassifier* također sadrži klasu *„None"* u koju svrstava geste koje ne prepoznaje među ostalim definiranima. 5 gesti je premalo za međusobnu usporedbu točnosti pa će se iz tog razloga svaki model testirati zasebno s gestama koje prepoznaje. U skladu s time, kod izrade vlastitog modela razmotrit će se svaka od navedenih gesti.

Slika 12. Geste za usporedbu točnosti

4.2. Izrada vlastitog modela

Na primjeru odabranih modela za usporedbu, jasno je da se radi o prediktivnoj metodi strojnog učenja, konkretno o klasifikaciji, jer računalo na temelju postojećih podataka svrstava nepoznati podatak (pozu šake) u jednu od predefiniranih klasa (gesti). Kod prediktivnih metoda računalu se daje skup podataka s nezavisnim varijablama (ulazima) i zavisnim varijablama (izlazima), tj. varijablama koje na neki način ovise o danim ulazima. Takve zavisne varijable još se nazivaju labele, a vrsta strojnog učenja u kojoj se primjenjuju prediktivne metode naziva se nadzirano učenje (eng. *supervised learning*). Konačan cilj takve metode jest da model nauči

pravila prema kojima se ulazi povezuju sa zavisnom varijablom te ih primijeni na nepoznatim podacima.

Za prikupljanje podataka te trening i izradu vlastitog modela postoji unaprijed izrađeno javno dostupno okruženje čijim se korištenjem proces znatno olakšava [17]. Okruženje se sastoji od sljedećih datoteka koje su važne za izradu modela:

- *app.py* – glavna skripta za realizaciju klasifikacije gesti pomoću *web* kamere i prikupljanje podataka za trening modela
- *keypoint_classification.ipynb* – skripta za trening modela
- *model/keypoint_classifier* – direktorij s datotekama vezanima uz model
    - *keypoint.csv* – prikupljeni podaci za učenje
    - *keypoint_classifier.hdf5* – datoteka za skladištenje strukture i težine modela za njegovo ponovno korištenje
    - *keypoint_classifier.tflite* – model za implementaciju u kôdu
    - *keypoint_classifier_label.csv* – nazivi klasa
    - *keypoint_classifier.py* – skripta za primjenu modela na novim podacima
- *utils/cvfpscalc.py* – skripta za izračunavanje FPS-a

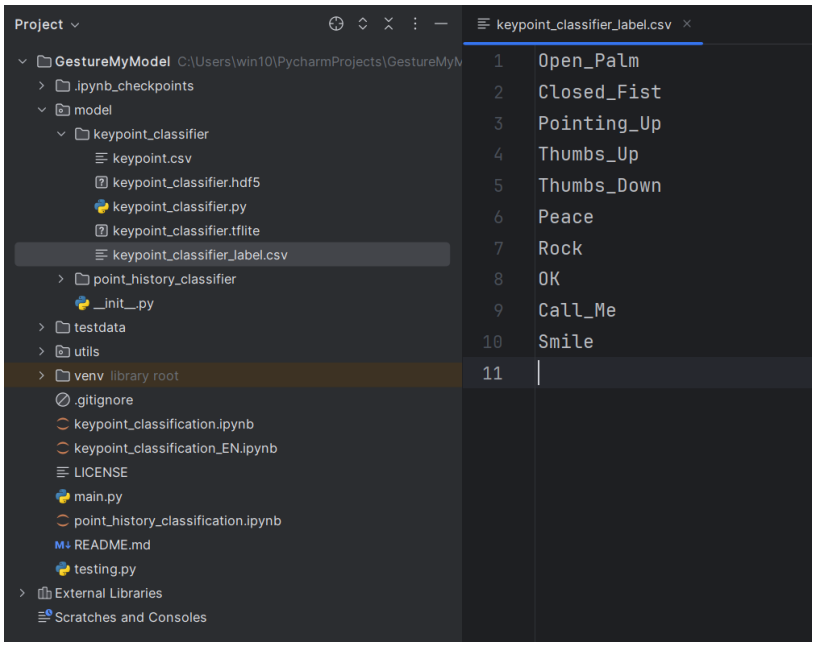
Ovo se okruženje može koristiti i u druge svrhe te su zbog toga, osim nabrojanih, prisutne i druge datoteke, no one nisu korištene u izradi modela za prepoznavanje gesti. Bez ikakvih modifikacija i treninga, model prepoznaje 4 zadane geste, no ako se taj broj želi povećati ili neka od zadanih gesti mijenjati, optimalno je obrisati sve podatke iz datoteke *keypoint.csv* i samostalno ih prikupljati od početka.

Korisnik može imenovati i stvoriti maksimalno 10 klasa u koje će njegov model naučiti svrstavati podatke, stoga je od prethodnih 12 gesti potrebno odabrati dvije koje vlastiti model neće upotrebljavati. Geste *„Stop“* i *„Live long“* vrlo su slične gesti *„Open palm“* te bi time mogle uzrokovati zabunu tijekom treninga modela, što ih čini optimalnim izborom za izbacivanje.

4.2.1. Prikupljanje podataka za učenje

Nakon što su odabrane geste koje model treba naučiti, potrebno je otvoriti datoteku *keypoint_classifier_label.csv* te u svaki redak upisati ime jedne klase (maksimalno 10) pri čemu

svaka od njih dobiva svoj indeks od 0 do 9. Indeks klase u prvom retku je 0, a klase u zadnjem retku 9. Slika 13 prikazuje klase upisane u datoteci.



Slika 13. Otvaranje i prikaz datoteke *keypoint_classifier_label.csv*

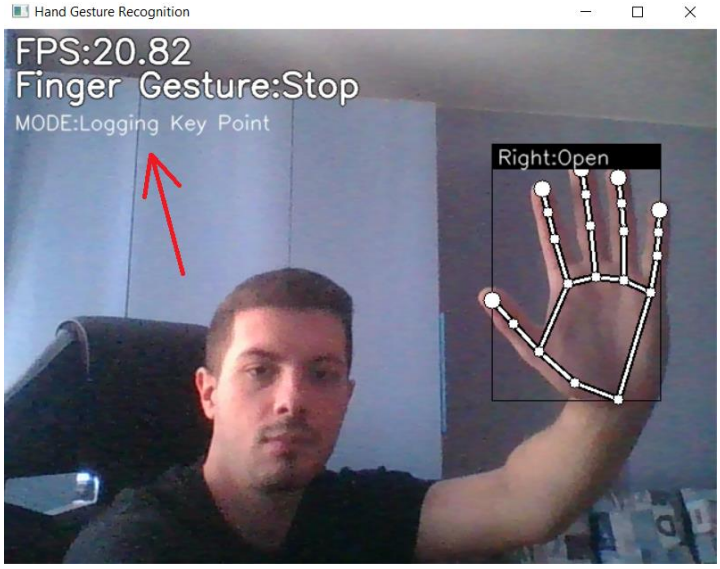
Pokretanjem skripte *app.py* uključuje se kamera i prikazuje sučelje u kojem se mogu vidjeti rezultati implementacije modela u stvarnom vremenu. Pritiskom tipke „K“ na tipkovnici, na sučelju se ispisuje poruka „*MODE:Logging Key Point*“ (Slika 14), što pokazuje da je pokrenut dio programa za prikupljanje podataka za učenje. Ispred kamere je potrebno šakom napraviti gestu čiji se podatak želi spremiti te na tipkovnici pritisnuti brojčanu tipku koja odgovara indeksu dotične klase (geste). Prikupljeni podaci sadrže indeks klase i relativne koordinate ključnih točaka šake u odnosu na točku zapešća te se spremaju u dokument *keypoint.csv*.

Npr. ako je cilj prikupiti 100 podataka koji će biti označeni kao gesta *„Open palm”* te ako je ona u dokumentu *keypoint_classifier_label.csv* upisana u prvi redak, dakle sadrži indeks 0, potrebno je pokrenuti skriptu *app.py*, pritisnuti tipku „K“, pred kameru staviti otvoreni dlan bilo koje šake te pritisnuti tipku „0“ kako bi se spremio jedan podatak. Kontinuiranim pritiskanjem tipke „0“, uz blago mijenjanje poze šake, spremaju se različiti podaci za učenje označeni kao ista klasa.

Prednost korištenja ovog okruženja je što model istovremeno uči za obje šake, tj. podaci za učenje mogu sadržavati koordinate ključnih točaka samo jedne šake, no model će prepoznavati geste na objema. Nadalje, podaci ne ovise o pozadini slike ni o položaju gabarita šake u kadru

zbog toga što koordinate ključnih točaka nisu izražene kao apsolutne, već kao relativne u odnosu na točku zapešća.

Kod izrade vlastitog modela prikupljeno je ukupno 8182 podatka. U početnoj fazi prikupljeno je 5000, s približno 500 podataka za svaku klasu, no uzastopnim treniranjem modela uočeno je da neke klase pokazuju lošije rezultate prema evaluacijskim metrikama, stoga je prikupljen dodatan broj podataka kako bi model podjednako naučio sve klase. Iz navedenog razloga broj podataka po klasama kod treniranja vlastitog modela nije uravnotežen.



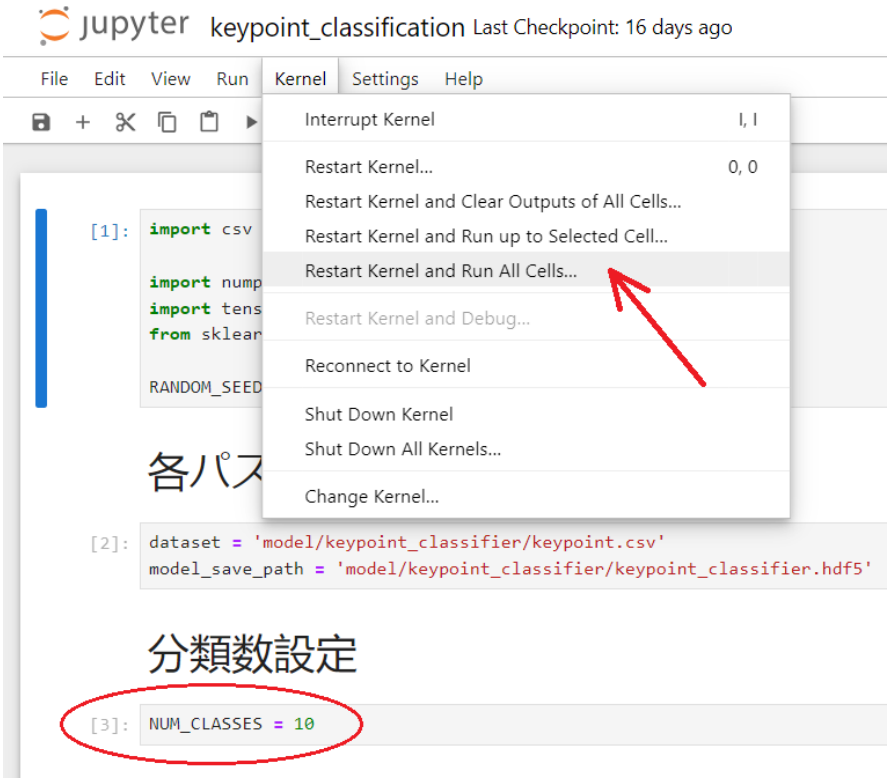
Slika 14. Sučelje za prikupljanje podataka

4.2.2. Trening modela

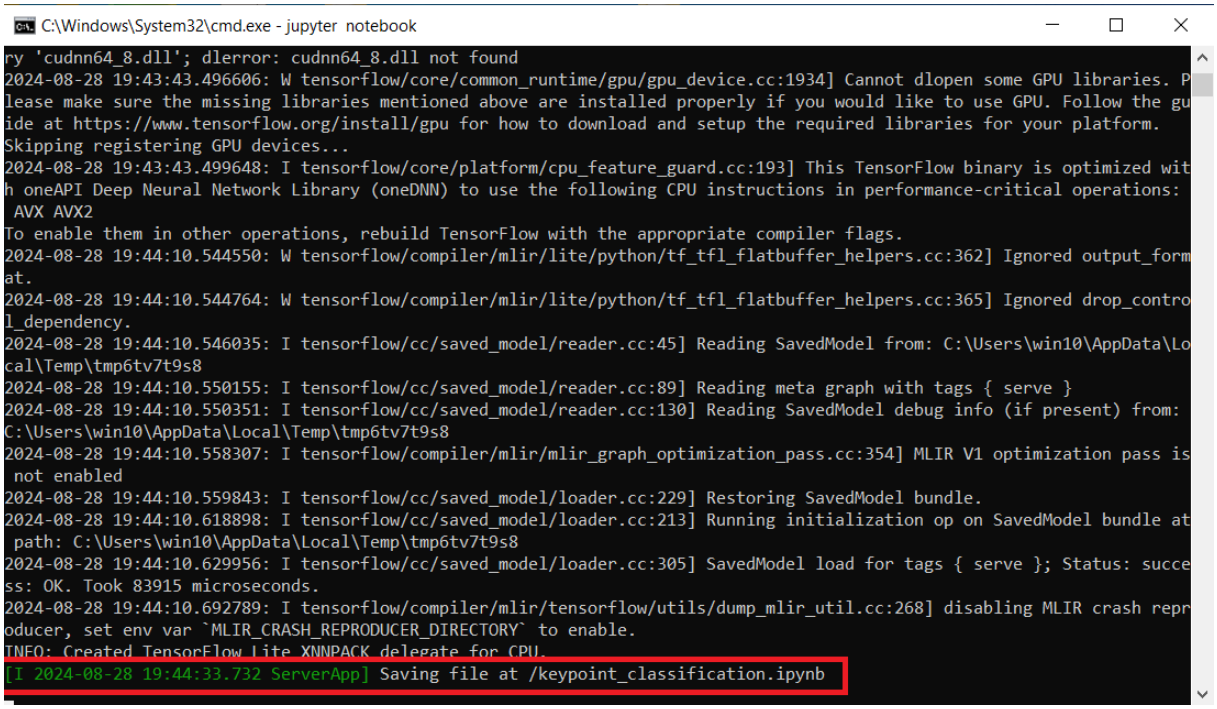
Kada je prikupljen dovoljan broj podataka, započinje proces treniranja modela. U naredbenom retku potrebno je doći u mapu projekta te u pregledniku otvoriti *Jupyter Notebook* sučelje pomoću naredbe *jupyter notebook*. U slučaju da *Jupyter* nije instaliran, potrebno je to učiniti naredbom *pip install jupyter*.

Otvaranjem dokumenta *keypoint_classification.ipynb* prikazuje se skripta koja sadržava kôd za treniranje modela. Prije pokretanja samog procesa, u trećem odlomku potrebno je promijeniti vrijednost varijable *NUM_CLASSES* u odgovarajući broj klasa koje model treba naučiti. U ovom se slučaju vrijednost varijable *NUM_CLASSES* postavlja na 10 jer model uči 10 različitih klasa. Proces se pokreće pritiskom na sekciju *Kernel* na traci oznaka te odabirom opcije *Restart Kernel and Run All Cells* u padajućem izborniku.

Kada se u naredbenom retku ispiše poruka da je model spremljen, proces je završen i preglednik se može zatvoriti.



Slika 15. Pokretanje treninga modela



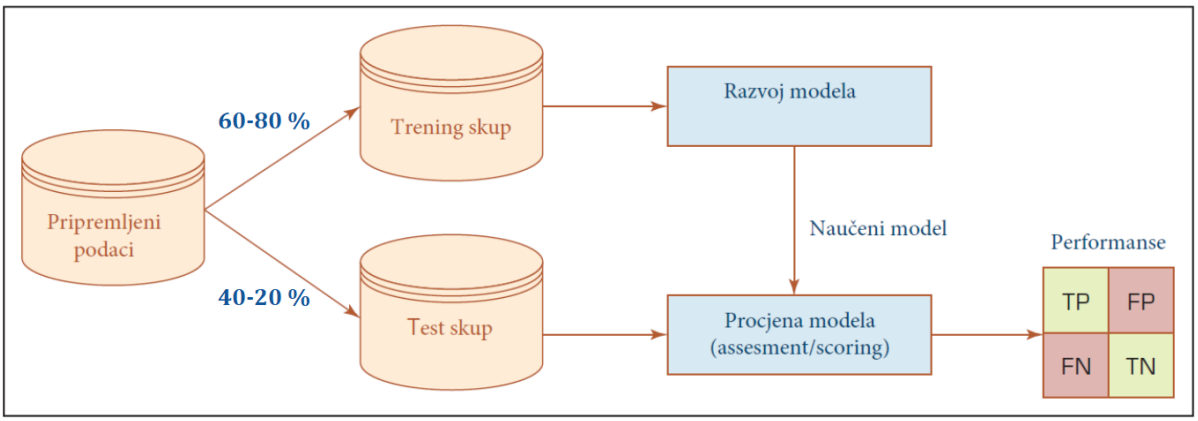
Slika 16. Naredbeni redak - poruka o spremanju modela

Korisnik doprinosi klasifikaciji isključivo određivanjem broja i naziva klasa, spremanjem koordinata ključnih točaka šake i dodjeljivanjem oznake klase kojoj pripadaju pritiskom odgovarajuće brojčane tipke na tipkovnici. Cilj je da model na temelju zadanih koordinata ključnih točaka šake svrsta gestu u odgovarajuću klasu, što znači da su koordinate točaka ulazni podaci, a prepoznate geste izlazni rezultati.

Za trening modela koristi se metoda dubokog učenja, konkretno neuronska mreža implementirana putem *TensorFlow* biblioteke. Pravila prema kojima model donosi odluke i povezuje ulaze s izlazima su kompleksna, teško razumljiva i ne mogu se lako interpretirati. Takve metode kod kojih se mogu promatrati ulazi i izlazi, ali je proces između njih zamršen, nazivaju se *black box* metodama. Duboko učenje vrsta je *black box* metode jer svaki od slojeva neurona sadrži ogroman broj parametara poput težina i pristranosti (eng. *bias*), koji se tijekom treninga konstantno optimiziraju, a interakcija među slojevima je vrlo kompleksna.

4.2.3. Evaluacija modela

Prije početka procesa treniranja, prikupljeni podaci dijele se na trening skup, iz čijih podataka model uči, i test skup, pomoću kojeg se izvodi evaluacija modela, odnosno procjenjuje uspješnost klasifikacije. Podaci su podijeljeni metodom *simple split* (jednostavna podjela) tako da 75 % njih čini trening skup, a 25 % test skup. Kod takve je metode poželjno da skupovi budu slični kako bi se izbjegla pristranost modela te kako bi evaluacija bila vjerodostojnija.



Slika 17. Dijagram *simple split* metode podjele [18]

Po završetku treninga ispisuje se matrica konfuzije zajedno s tablicom rezultata evaluacije pomoću test skupa podataka.

Matrica konfuzije je alat za evaluaciju performansi klasifikacijskog modela koji prikazuje odnos predikcija i stvarnih klasa, čime omogućuje izračun raznih evaluacijskih metrika poput točnosti, preciznosti, osjetljivosti i F1-mjere.

Tablica 7 prikazuje opći oblik binarne matrice konfuzije, gdje postoje samo dvije klase, s četirima osnovnim elementima:

- TP (*True Positive*) – broj slučajeva gdje je model predvidio pozitivan ishod, a stvarni ishod je također pozitivan
- TN (*True Negative*) – broj slučajeva gdje je model predvidio negativan ishod, a stvarni ishod je također negativan
- FP (*False Positive*) – broj slučajeva gdje je model predvidio pozitivan ishod, no stvarni ishod je negativan
- FN (*False Negative*) – broj slučajeva gdje je model predvidio negativan ishod, no stvarni ishod je pozitivan

Tablica 7. Binarna matrica konfuzije

STVARNA KLASA	**PREDIKCIJA**	
	Pozitivno	Negativno
Pozitivno	TP	FN
Negativno	FP	TN

Navedeni elementi čine temelj za izračun evaluacijskih metrika koje se koriste za procjenu uspješnosti modela.

Preciznost (eng. *precision*) je mjera koja pokazuje udio ispravnih pozitivnih predikcija u odnosu na sve predikcije koje je model predvidio kao pozitivne. Računa se prema izrazu:

$$P = \frac{TP}{TP + FP} \tag{19}$$

Osjetljivost ili odziv (eng. *recall*) je mjera koja pokazuje postotak stvarnih pozitivnih vrijednosti koje su ispravno predviđene kao pozitivne. Izraz za računanje glasi:

$$R = \frac{TP}{TP + FN} \tag{20}$$

F1-mjera (eng. *F1-score*) je harmonijska sredina preciznosti i osjetljivosti, a koristi se kako bi se dobila jedinstvena ocjena koja uzima u obzir i točnost modela u predviđanju pozitivnih klasa i njegovu sposobnost da ih sve predvidi.

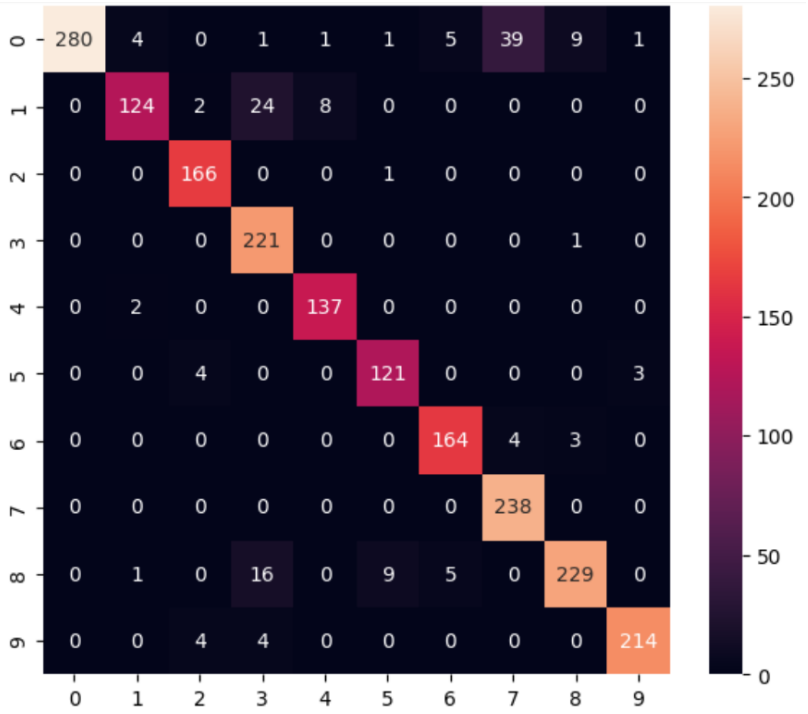
$$F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{21}$$

U slučaju da postoje dvije klase između kojih se ne može odrediti što je pozitivno, a što negativno (npr. pas i mačka), najprije se jedna od njih uzima kao pozitivna, a zatim druga, čime se navedene metrike dobivaju za obje klase.

Točnost (eng. *accuracy*) modela pokazuje udio ispravnih predikcija (i pozitivnih i negativnih) u odnosu na ukupni broj predikcija.

$$Točnost = \frac{TP + TN}{TP + TN + FP + FN} \tag{22}$$

Slika 18 prikazuje matricu konfuzije ispisanu nakon treninga vlastitog modela. Horizontalna os prikazuje indekse predviđenih klasa, a vertikalna indekse stvarnih.

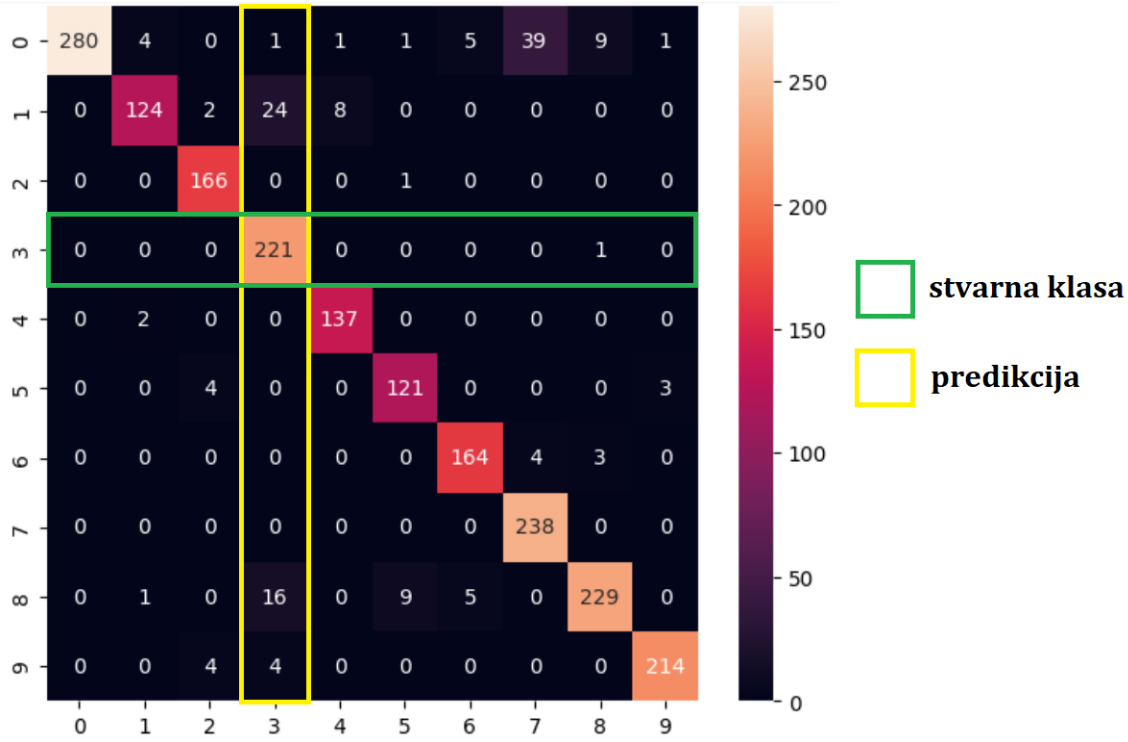


Slika 18. Matrica konfuzije vlastitog modela

U slučaju da klasifikacija nije binarna, tj. postoji više od dvije klase, za izračun evaluacijskih metrika koristi se metoda „jedan protiv svih“ (eng. *One-vs-All, One-vs-Rest*). Cilj spomenute

metode jest napraviti binarnu matricu konfuzije za svaku pojedinu klasu tako da se promatrana klasa uzima kao pozitivna, a negativnu zajedno čine sve ostale klase.

Primjer binarne matrice za jednu klasu prikazan je u Tablici 8, gdje su osnovni elementi određeni za klasu 3.



Slika 19. Promatrane vrijednosti za klasu 3

Tablica 8. Binarna matrica konfuzije za klasu 3

STVARNA KLASA	**PREDIKCIJA**	
	Pozitivno (klasa 3)	Negativno (ostale klase)
Pozitivno (klasa 3)	221 (TP)	1 (FN)
Negativno (ostale klase)	45 (FP)	1779 (TN)

Pomoću ovih vrijednosti, prema jednadžbama (19), (20) i (21) računaju se metrike za promatranu klasu:

$$P_3 = \frac{221}{221 + 45} \approx 0{,}83$$

$$R_3 = \frac{221}{221 + 1} \approx 1{,}00$$

$$F1_3 = \frac{2 \cdot 0{,}83 \cdot 1}{0{,}83 + 1} \approx 0{,}91$$

Na isti se način računaju metrike za svaku pojedinu klasu, čije se vrijednosti, uz matricu konfuzije, također ispisuju po završetku treninga vlastitog modela.

Classification Report

	precision	recall	f1-score	support
0	1.00	0.82	0.90	341
1	0.95	0.78	0.86	158
2	0.94	0.99	0.97	167
3	0.83	1.00	0.91	222
4	0.94	0.99	0.96	139
5	0.92	0.95	0.93	128
6	0.94	0.96	0.95	171
7	0.85	1.00	0.92	238
8	0.95	0.88	0.91	260
9	0.98	0.96	0.97	222
accuracy			0.93	2046
macro avg	0.93	0.93	0.93	2046
weighted avg	0.93	0.93	0.93	2046

Slika 20. Ispisana tablica evaluacijskih metrika

Osim dosad navedenih metrika, prikazane su vrijednosti makro i težinskog prosjeka. Makro prosjek je prosjek svih klasa mjerenih pojedinačno za svaku metriku, tretirajući svaku klasu jednako, dok težinski uzima u obzir broj vrijednosti u svakoj klasi, pri čemu klase s više uzoraka imaju veći utjecaj na rezultat.

Globalna točnost modela je 0,93 (93 %), što je za potrebe ovog rada izrazito visoko. Osim toga, sve klase imaju visoke i približno jednake vrijednosti preciznosti, osjetljivosti i F1-mjere, dakle svaka od njih se može tretirati jednako, što je u slučaju ovog modela povoljno jer su klase podjednako važne.

4.3. Usporedba točnosti modela

Završetkom izrade i evaluacije vlastitog modela, njegovu točnost potrebno je ispitati na novim, nepoznatim podacima te usporediti s točnošću ostalih odabranih javno dostupnih modela. Točnost, odnosno sposobnost modela da ispravno prepoznaje geste, ključna je metrika u

sveobuhvatnoj procjeni performansi modela, stoga će se isključivo ona uzimati u obzir kod usporedbe.

4.3.1. Usporedba pomoću slika

Za svaku od 12 gesti prikupljeno je 13 različitih fotografija od šest osoba, uključujući one prikazane na Slici 12. Fotografije se primarno razlikuju u uporabljenoj šaci (lijeva ili desna), njezinoj udaljenosti i zakrenutosti. Točnost modela ne ovisi o pozadini i vidljivosti šake budući da ulazni podaci nisu same slike, već isključivo koordinate ključnih točaka koje detektira *MediaPipe*. Primjeri različitih fotografija za istu gestu prikazani su na Slici 21.

Slika 21. Primjeri fotografija geste *„peace“*

Kao što je navedeno, nijedan od promatranih modela ne prepoznaje svih 12 gesti, stoga se na svakom od njih ispituje točnost klasifikacije isključivo za one klase u koje može svrstati ulazne podatke.

Tablica 9 prikazuje rezultate ispitivanja točnosti modela. Za svaku stvarnu klasu navedena su dva broja odvojena crticom (–), pri čemu prvi broj označava broj ispravnih, a drugi broj pogrešnih predikcija. Ako model ne prepoznaje određenu gestu, umjesto broja prikazan je znak X, a dotična klasa se za taj model ne uzima u obzir.

Točnost se u ovom slučaju računa prema izrazu:

$$Točnost = \frac{Ispravno}{Ispravno + Pogrešno} \quad \textbf{(23)}$$

Tablica 9. Rezultati ispitivanja točnosti modela pomoću slika

STVARNA GESTA	**MODEL**		
	TechVidvan	*MediaPipe HandGestureClassifier*	Vlastiti model
Pointing up	X	12 – 1	13 – 0
Open palm	X	11 – 2	13 – 0
Closed fist	1 – 12	12 – 1	13 – 0
Thumbs up	2 – 11	13 – 0	13 – 0
Thumbs down	9 – 4	13 – 0	13 – 0
Peace	7 – 6	12 – 1	13 – 0
Rock	6 – 7	11 – 2	13 – 0
OK	1 – 12	X	13 – 0
Call me	3 – 10	X	11 – 2
Smile	0 – 13	X	13 – 0
Stop	4 – 9	X	X
Live long	4 – 9	X	X
Ispravno	37	84	128
Pogrešno	93	7	2
Točnost	**28,46 %**	**92,31 %**	**98,46 %**

Vlastiti model pokazuje iznimno visoku točnost s najmanjim udjelom pogrešnih predikcija, što ga čini najboljim među promatranim modelima. Pogrešne predikcije uočene su tek u jednoj klasi, dok su u svim ostalim slučajevima bile ispravne. *MediaPipe HandGestureClassifier* također se ističe visokom točnošću, ali s većim udjelom pogrešnih predikcija u usporedbi s vlastitim modelom. *TechVidvan* model pokazuje daleko najnižu točnost od svih triju modela, s velikim brojem pogrešnih predikcija u odnosu na ispravne, s činjenicom da gestu *„Smile“* nije ispravno prepoznao ni u jednom testnom primjeru.

4.3.2. Usporedba pomoću video sekvence

Konačan cilj jest da modeli prepoznaju geste u stvarnom vremenu, stoga je njihovu točnost potrebno usporediti i pomoću video sekvence, tj. videozapisa spremljenog u obliku 295 kadrova. Video upotrijebljen za usporedbu samostalno je izrađen te se u njemu određenim redoslijedom uzastopno mijenja 12 promatranih gesti. Svakom kadru pridružena je gesta koja se u njemu pokazuje te se provodi isti postupak kao i kod usporedbe pomoću slika.

Slika 22. Primjeri kadrova iz video sekvence

Budući da se radi o video sekvenci, postoji mogućnost da *MediaPipe* u određenom kadru ne uspije detektirati šaku, zbog čega modeli ne mogu izvršiti klasifikaciju. Zbog toga je potrebno formulirati pravilo prema kojem se u prvom kadru u kojem *MediaPipe* ne detektira šaku kao izlaz modela uzima gesta prepoznata u prethodnom kadru, dok se svi sljedeći uzastopni kadrovi u kojima *MediaPipe* neuspješno detektira šaku ne uzimaju u obzir kod usporedbe točnosti.

Tablica 10. Rezultati ispitivanja modela pomoću video sekvence

STVARNA GESTA	**MODEL**		
	TechVidvan	*MediaPipe HandGestureClassifier*	Vlastiti model
Pointing up	X	28 – 2	30 – 0
Open palm	X	15 – 7	22 – 0
Closed fist	19 – 5	17 – 7	24 – 0
Thumbs up	15 – 11	25 – 1	26 – 0
Thumbs down	15 – 4	20 – 0	18 – 1
Peace	18 – 10	18 – 10	25 – 2

Rock	28 – 2	25 – 5	30 – 0
OK	21 – 7	X	28 – 0
Call me	16 – 6	X	18 – 4
Smile	13 – 11	X	23 – 1
Stop	5 – 20	X	X
Live long	13 – 3	X	X
Ispravno	163	148	244
Pogrešno	79	32	8
Točnost	**67,36 %**	**82,22 %**	**96,83 %**

Vlastiti model ponovno pokazuje najbolje performanse s izrazito visokom točnošću. Iako performanse modela *MediaPipe HandGestureClassifier* na video sekvenci nisu toliko dobre kao na slikama, njegova točnost i dalje se smatra vrlo dobrom. Najveća promjena rezultata zabilježena je kod modela *TechVidvan,* čija je točnost povećana za gotovo 40 % u odnosu na točnost klasifikacije na slikama, zbog čega se njegova upotreba u stvarnom vremenu također može razmatrati, iako i dalje pokazuje najmanju točnost među promatranim modelima.

4.3.3. Zaključak usporedbe i odabir modela

Geste koje modeli prepoznaju su statične, no moraju se klasificirati u stvarnom vremenu, što znači da je važno odabrati model koji pokazuje visoke performanse i na slikama i na videozapisima.

TechVidvan model pokazao je solidnu točnost klasifikacije na video sekvenci, no njegova točnost na slikama je izrazito mala, stoga on nije pouzdan za implementaciju u nastavku rada. Zbog visoke točnosti na slikama i video sekvenci, *MediaPipe HandGestureClassifier* dobar je izbor za klasifikaciju gesti i može se razmatrati kao opcija za korištenje. Međutim, u oba slučaja vlastiti model pokazao je veću točnost, što ga čini boljom opcijom.

Pokazavši se najtočnijim modelom u klasifikaciji gesti s iznimno visokom točnošću, vlastiti model odabire se za implementaciju u sustavu.

4.4. Implementacija modela

Za razliku od algoritma za procjenu namjera kretanja čovjeka, u načinu rada u kojem čovjek upravlja ponašanjem robota pomoću gesti sustav ne mora samostalno donositi odluke, što takav algoritam čini znatno manje kompleksnim. Potrebno je odabrati geste koje će se koristiti, svakoj od njih pridružiti odgovarajuću naredbu koja se šalje robotu kada je gesta prepoznata te definirati potrebna ograničenja u algoritmu, npr. vrijeme reakcije robota na geste.

Za potrebe rada odabrane su 4 geste koje vlastiti model prepoznaje te je svakoj pridružena naredba za promjenu brzine robota, kako je prikazano u Tablici 11.

Tablica 11. Implementirane geste i njihove naredbe

GESTA	NAREDBA
Pointing up	Postavi brzinu robota na 1 (100 %)
Open palm	Postavi brzinu robota na 0 (0 %)
Thumbs up	Povećaj brzinu robota za 0,1 (10 %)
Thumbs down	Smanji brzinu robota za 0,1 (10 %)

Na početku programa, brzina robota postavljena je na 0. S ciljem izbjegavanja slučajnih gestikulacija tijekom upravljanja, naredbe se prihvaćaju tek kada je njihova gesta prepoznata u 10 uzastopnih kadrova. U ovom algoritmu nagle promjene brzine robota također nisu poželjne, stoga je važno da se ona uvijek mijenja postepeno. Kada je prihvaćena naredba da se brzina postavi na 1 ili 0, nakon triju uzastopnih kadrova ona se poveća, odnosno smanji za 0,1 sve dok ne postigne traženu vrijednost. U slučaju da je prepoznata neka od gesti kojima nije pridružena naredba, prekidaju se sve aktivne naredbe i robot nastavlja rad na trenutnoj postavljenoj brzini.

Kao i za procjenu namjera kretanja čovjeka, ovaj algoritam radi u neprekidnoj petlji sve dok ga korisnik ne zaustavi.

5. INTEGRACIJA NA ROBOTU

Konačan cilj koncipiranja i razvoja algoritama jest njihova uspješna integracija u stvarni robotski sustav, čime se omogućuje autonomno prepoznavanje ljudskih gesti i procjena namjera kretanja čovjeka te prilagodba ponašanja robota prema zadanim pravilima.

Glavni izazovi u ovom koraku uključuju odabir odgovarajućeg robota i kamere, određivanje optimalne pozicije kamere, uspostavljanje komunikacije između računala i robota te, konačno, slanje naredbi putem računala koje robot izvršava tijekom rada svog glavnog programa. Osim toga, potrebno je osmisliti rješenje za integraciju oba razvijena algoritma u jedinstveni program te omogućiti nesmetan prijelaz između njih.

Iako su algoritmi u svojoj osnovnoj strukturi nepromjenjivi, mogu biti podložni manjim prilagodbama kako bi se uskladili s tehničkim ograničenjima robota i njegovim specifičnim mogućnostima.

5.1. FANUC CR-15*i*A [19]

Slika 23. FANUC CR-15*i*A [19]

Jedan od robota dostupnih za korištenje u Regionalnom centru izvrsnosti za robotske tehnologije (CRTA-i) jest CR-15*i*A, kolaborativni robot kojeg je razvila tvrtka FANUC. Pripada seriji kolaborativnih robota CR, a ističe se kompaktnom izvedbom, jednostavnim i intuitivnim programiranjem te primjenom u različitim okruženjima. Opremljen je senzorima za detekciju kontakta te ima ugrađene sigurnosne funkcije koje zadovoljavaju međunarodne standarde sigurnosti. Tablica 12 prikazuje njegove najvažnije specifikacije.

Tablica 12. Specifikacije robota FANUC CR-15*i*A [19]

Proizvođač i model	FANUC CR-15*i*A
Masa robota	255 kg
Nosivost	15 kg
Doseg	1441 mm
Stupnjevi slobode gibanja	6
Ponovljivost	$\pm$ 0,02 mm
Maksimalna brzina prihvatnice	800 mm/s

5.1.1. Komunikacija robota s računalom

Da bi se robotu omogućilo primanje naredbi s računala, potrebno je uspostaviti komunikaciju između njih, što je učinjeno putem TCP/IP protokola koristeći *socket* programiranje.

TCP/IP je skup protokola koji omogućuje razmjenu podataka između različitih uređaja putem interneta ili lokalne mreže. Ti se podaci šalju u obliku paketa, za čiji je ispravan prijenos s jednog uređaja na drugi zaslužan TCP (eng. *Transmission Control Protocol*, protokol za kontrolu prijenosa), dok IP (eng. *Internet Protocol*, internet protokol) upravlja njihovim adresiranjem i usmjeravanjem kroz mrežu.

Socket-i predstavljaju krajnje točke veze kroz koje se podaci prenose tijekom komunikacije te svaki uređaj na mreži koristi jedan ili više *socket*-a za slanje, odnosno primanje podataka. Identificiraju se pomoću IP adrese i broja porta, čime se omogućuje precizno usmjeravanje podataka i upravljanje istima.

Ovakav je oblik komunikacije standardiziran, što omogućuje jednostavno povezivanje robota s računalima različitih arhitektura i platformi.

Osnovne uloge koje uređaji mogu imati prilikom razmjene podataka jesu server (poslužitelj) i klijent. Uređaj koji pruža usluge ili resurse drugim uređajima na mreži ima ulogu servera, a onaj koji se povezuje sa serverom kako bi koristio njegove usluge ili resurse ima ulogu klijenta. Klijent inicira vezu prema serveru, koji čeka dolazne veze i zahtjeve od klijenata. Analizom osnovnog *Python* kôda za povezivanje s robotom i slanje naredbi za promjenu brzine jasno se može utvrditi koja je uloga računala, a koja robota.

5.1.2. Slanje naredbe robotu putem Python skripte

```
import socket

def start_server():
    server_socket = socket.socket(socket.AF_INET,
                                  socket.SOCK_STREAM)
    host = '192.168.40.152'
    port = 8000

    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Server listening on {host}:{port}")

    conn, addr = server_socket.accept()
    print(f"Connection established with {addr}")

    while True:
        command = input("Enter a command to send to the client
                        (or 'exit' to quit): ")

        if command.lower() == 'exit':
            break

        conn.send(command.encode())

        response = conn.recv(1024)
        print(f"Response from client: {response.decode()}")

    conn.close()
    print("Connection closed")

if __name__ == "__main__":
    start_server()
```

Na početku programa potrebno je uvesti modul *socket* koji omogućava rad s mrežnim *socket*-ima i komunikaciju putem mreže.

Izvršni dio programa započinje kreiranjem novog *socket* objekta pomoću funkcije `socket.socket()`. Parametar `socket.AF_INET` označava da se koristi IPv4 adresa za mrežnu komunikaciju, dok `socket.SOCK_STREAM` označava korištenje TCP protokola.

Varijabla `host` sadrži IP adresu servera, dok se `port` postavlja na proizvoljnu vrijednost (u ovom slučaju 8000), što označava broj porta na kojem će server slušati dolazne veze.

Metodom `server_socket.bind((host, port))` *socket* se veže na zadanu IP adresu i port, a `server_socket.listen(1)` postavlja ga u način rada gdje sluša dolazne veze. Parametar 1 označava da *socket* može imati samo jednu dolaznu vezu u redu čekanja.

Sve dok se neki klijent ne pokuša povezati se sa serverom, linija `conn, addr = server_socket.accept()` zaustavlja izvođenje programa. Kada klijent uspostavi vezu, `conn` je novi *socket* objekt koji se koristi za komunikaciju s klijentom, a `addr` je adresa klijenta koji se povezao, odnosno IP adresa i port.

Naredba za promjenu brzine robotu se šalje u obliku *string* podatka čiji je opći oblik `'SPD_'`. Umjesto donje crte upisuje proizvoljan broj od 0 do 100, koji predstavlja postotak vrijednosti maksimalne brzine robota.

Odgovor klijenta na naredbu sprema se u varijablu `response` pomoću metode `conn.recv(1024)`, koja čita do 1024 bajta podataka.

Kada petlja završi, veza s klijentom se zatvara pomoću metode `conn.close()`.

Na temelju ovog kôda zaključuje se da računalo ima ulogu servera jer čeka na vezu i omogućuje slanje i primanje podataka, a robot je klijent jer se povezuje na server (računalo) da bi primio i izvršio naredbe.

5.2. *Intel RealSense Deapth Camera* D435

Osim robota, s računalom mora biti spojena kamera koja dohvaća sliku potrebnu za detekciju ključnih točaka tijela, odnosno šake. Zahtjevi koje odabrana kamera mora ispunjavati uključuju brzinu osvježavanja (FPS) koja je veća ili jednaka izmjerenoj brzini *MediaPipe*-a, kao i mogućnost snimanja slike u boji, budući da *MediaPipe* kao ulazni podatak prima sliku u RGB

(eng. *Red-Green-Blue*, crveno-zeleno-plavo) formatu, te domet koji je veći od udaljenosti granice r_2.

Slika 24. *Intel RealSense Deapth Camera* D435

Intel RealSense Deapth Camera D435 je napredna kamera koja nudi mogućnost snimanja 2D i 3D slika te je opremljena dvostrukim senzorima za dubinu, infracrvenim projektorom i RGB kamerom koja omogućava snimanje u boji, čime je zadovoljen jedan od zahtjeva [20]. Budući da ni u kojem trenutku algoritmi ne obrađuju 3D sliku, tijekom izvođenja programa koristit će se samo RGB kamera, čije su specifikacije prikazane u Tablici 13.

Tablica 13. Specifikacije RGB kamere [20]

Rezolucija kadra	1920 × 1080
FPS	30
Tehnologija senzora	*Rolling Shutter*
Vidno polje senzora (H × V)	69° × 42°
Rezolucija senzora	2 MP
Domet	10 m

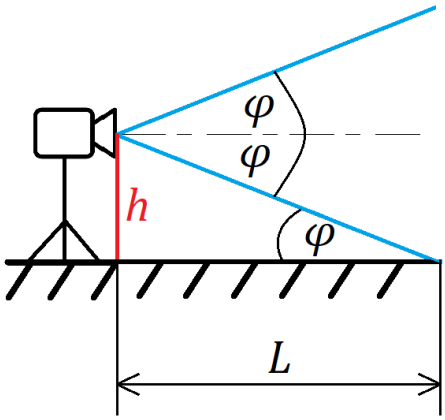
Iz Tablice 2 očitava se da je FPS *MediaPipe*-a 29,60. FPS kamere približno je jednak toj vrijednosti, što je idealno jer neće doći do međusobnog usporavanja.

U ranijem poglavlju određena je vrijednost r_2 koja iznosi 6 m, što domet kamere od 10 m čini dovoljnim za upotrebu.

Shodno tome, odabrana kamera ispunjava sve postavljene zahtjeve i može se upotrijebiti u realizaciji i integraciji razvijenih algoritama na robotu.

5.2.1. Određivanje pozicije kamere

U poglavlju u kojem je opisan algoritam za procjenu namjera kretanja čovjeka određeno je da udaljenost L (Slika 7), na kojoj referentna točka (nožni prsti) izlazi iz vidnog polja kamere, bude jednaka udaljenosti granice r_0, koja iznosi 2 m. Zbog toga udaljenost L također iznosi 2 m, a kako bi se to ostvarilo, kameru je potrebno pozicionirati na odgovarajuću visinu.



Slika 25. Položaj kamere u vertikalnoj ravnini

Primjenom trigonometrije pravokutnog trokuta, dobiva se izraz za vertikalnu poziciju (visinu) kamere:

$$h = L \cdot \tan\varphi \qquad \textbf{(24)}$$

Osim duljine L, potrebno je odrediti konkretnu vrijednost kuta φ, koji predstavlja polovicu vertikalnog kuta vidnog polja kamere. Iz Tablice 13 očitava se da je taj kut jednak 42°, što znači da traženi kut φ iznosi 21°.

$$h = 2 \cdot \tan 21° \approx 0{,}77 \text{ m}$$

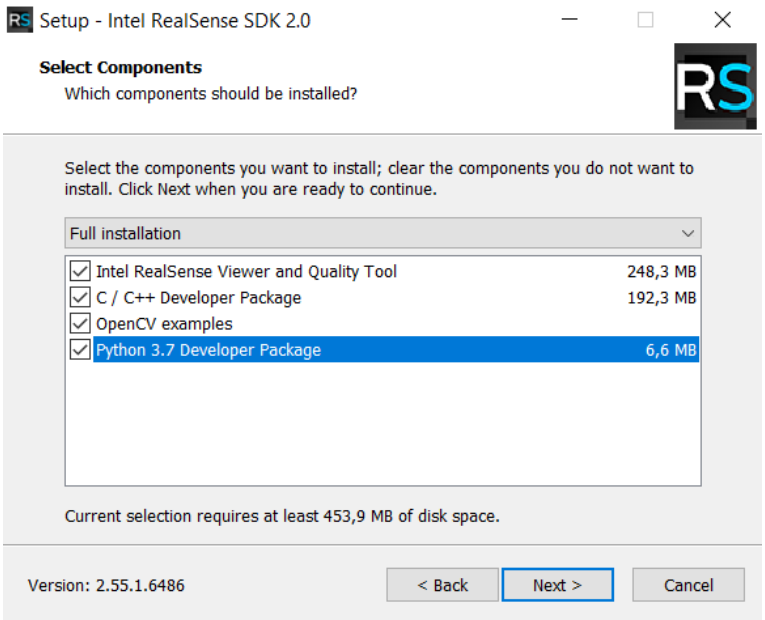
Ovime je određen najvažniji parametar pozicije kamere.

Pozicioniranje u horizontalnoj ravnini je fleksibilno te nisu potrebne konkretne metrike, no poželjno je da kamera bude u blizini baze robota, ali na dovoljnoj udaljenosti kako bi se

izbjegao fizički kontakt. Po uzdužnoj osi robot i kamera trebaju biti na istoj koordinati, tj. kamera se ne bi trebala nalaziti ispred niti iza robota.

5.2.2. Konfiguriranje i pokretanje kamere

Kako bi se kamera mogla pokrenuti putem računala i upravljati putem *Python* skripte, potrebno je preuzeti i instalirati *Intel RealSense* SDK 2.0, komplet za razvoj softvera (eng. *Software Development Kit*) dostupan na službenom *GitHub* repozitoriju [21]. U SDK su uključeni *Intel RealSense Viewer* (aplikacija za brzi pristup snimanju), *Depth Quality Tool* (aplikacija za testiranje dubine kamere), primjeri kôdova te, najvažnije, API (eng. *Application Programming Interface*, sučelje za programiranje aplikacija) za programske jezike poput *Python*-a i C++. Važno je obratiti pažnju da sve potrebne značajke budu uključene u instalaciju, a u ovom je slučaju najvažnija *Python* API.



Slika 26. Odabir značajki za instalaciju *Intel RealSense* SDK 2.0

Nakon uspješne instalacije, potrebno je otvoriti naredbeni redak i unijeti naredbu *pip install pyrealsense2*, čime će se instalirati biblioteka *PyRealSense2* pomoću koje se putem *Python* skripte može upravljati kamerom.

Kamera se s računalom spaja putem USB-C kabla.

5.3. Modifikacija i integracija algoritama

5.3.1. Minimalna brzina robota

Iako je inicijalno zamišljeno da robot može postaviti svoju brzinu na 0, realizacija toga može dovesti do sigurnosnih problema i problema s upravljanjem. Postavljanje brzine na 0 znači da motori potpuno prestaju raditi, što dovodi do nestabilnosti sustava i problema s regulacijom položaja, čime se može prouzročiti šteta na robotu. Iz tog razloga, kada je izlazna brzina robota određena kao 0, robotu se umjesto toga šalje naredba da postavi svoju brzinu na minimalnu vrijednost, čime se izbjegavaju spomenuti problemi, a brzina je dovoljno mala da daje dojam mirovanja robota.

Kada FANUC CR-15*i*A primi naredbu za postavljanje brzine na 0, on će je automatski postaviti na 0,01 (1 %), no u slučaju da se sustav implementira na nekom drugom robotu, treba voditi računa o navedenom ograničenju.

5.3.2. Upotreba dodatne kamere

Pozicija kamere određena za izvođenje algoritma za procjenu namjera kretanja čovjeka nije prikladna za upravljanje robotom pomoću gesti. Kamera je postavljena na visinu od samo 0,77 m, što je prenisko čak i kada korisnik sjedi. Osim toga, čovjek bi u tom slučaju morao stalno biti u neposrednoj blizini robota, što nije idealno iz sigurnosnih razloga. Jedno od mogućih rješenja jest upotreba *web* kamere računala. Osim što je postavljena na prirodnoj visini, tj. u ravnini lica, njezino korištenje uklanja potrebu da čovjek bude fizički blizu robota, čime se povećava sigurnost i smanjuje rizik od neželjenog fizičkog kontakta. Također, zamisao algoritma jest da čovjek može upravljati robotom s udaljenosti, neovisno o svojoj poziciji u prostoru, što dodatno opravdava upotrebu *web* kamere.

Problem pri korištenju dviju kamera jest taj da se *web* kamera ne može aktivirati dok je druga spojena na računalo. Zbog toga prelazak s jednog algoritma na drugi zahtijeva spajanje, odnosno odspajanje *RealSense* kamere kako bi obje kamere mogle neovisno raditi u okviru svojih algoritama.

5.3.3. Prijelaz između algoritama

Razvijeni algoritmi mogu raditi neovisno jedan o drugome, no potrebno je osigurati da prijelaz s jednog na drugi prođe bez greške, uz pravilno postavljanje robota.

Osim što je potrebno spojiti, odnosno odspojiti *RealSense* kameru prije prelaska na drugi algoritam, važno je i postaviti odgovarajuću početnu brzinu robota. Prije pokretanja bilo kojeg algoritma, kao i nakon njegovog prekida, brzina robota postavlja se na 0 (u stvarnosti 0,01)[4]. Kada je algoritam spreman za pokretanje, robot mora imati brzinu jednaku onoj koja je određena pretpostavkama tijekom izrade algoritama – u algoritmu za procjenu namjera kretanja čovjeka početna brzina robota je 1, dok je u algoritmu za upravljanje robotom pomoću gesti postavljena na 0, što se treba realizirati na njihovom samom početku.

[4] U daljnjem tekstu, kada se spominje da je brzina robota 0, podrazumijeva se da je stvarna brzina 0,01.

6. PROGRAMSKA PODRŠKA

Da bi se razvijeni algoritmi integrirali i implementirali u stvarnom radnom okruženju, potrebno je izraditi programsku podršku koja će izvršavati algoritme, omogućiti komunikaciju robota s računalom i prilagođavati njegovo ponašanje u stvarnom vremenu. Također, programska podrška treba sadržavati grafičko sučelje kako bi korisnik mogao upravljati procesom izvršavanja programa.

Zbog opsežnosti samog kôda programa, nepraktično je analizirati svaku njegovu liniju, stoga će se u radu opisati korištene biblioteke, najvažnije vlastite funkcije te korištenje grafičkog sučelja. Čitav kôd dostupan je u priloženom *GitHub* repozitoriju.

6.1. Korištene biblioteke

Osim *MediaPipe*-a, *TensorFlow*-a, *PyRealSense2* i *socket*-a, čije su funkcionalnosti prethodno objašnjene, programska podrška koristi i brojne druge biblioteke/module neophodne za izvršavanje cijelog programa:

- *NumPy* – biblioteka za numeričke proračune u *Python*-u, pruža podršku za matrične operacije i rad s višedimenzionalnim poljima (eng. *array*), uključujući matematičke funkcije, statističke analize i obradu podataka
- *OpenCV* – biblioteka za računalni vid koja omogućuje obradu slike i videozapisa, nudi brojne funkcionalnosti za prepoznavanje objekata, filtriranje, manipulaciju slikama i rad u stvarnom vremenu
- *tkinter* – standardna biblioteka za izgradnju grafičkog korisničkog sučelja (GUI, eng. *Graphical User Interface*) u *Python*-u, omogućuje kreiranje prozora, tekstualnih prikaza, gumba, izbornika i drugih GUI komponenti
- PIL *(Python Imaging Library)* – biblioteka za obradu slika u *Python*-u čiji modul *ImageTk* pruža jednostavan način za integraciju slika u *tkinter* sučelju
- *time* – standardna biblioteka koja omogućuje mjerenje vremena, odgodu izvršavanja dijela kôda (eng. *delay*) i druge funkcionalnosti za rad s vremenom

NumPy, *OpenCV* i PIL dostupni su na PyPI, dok su *tkinter* i *time* standardne biblioteke, što znači da dolaze unaprijed instalirane s *Python* distribucijom.

6.2. Najvažnije funkcije

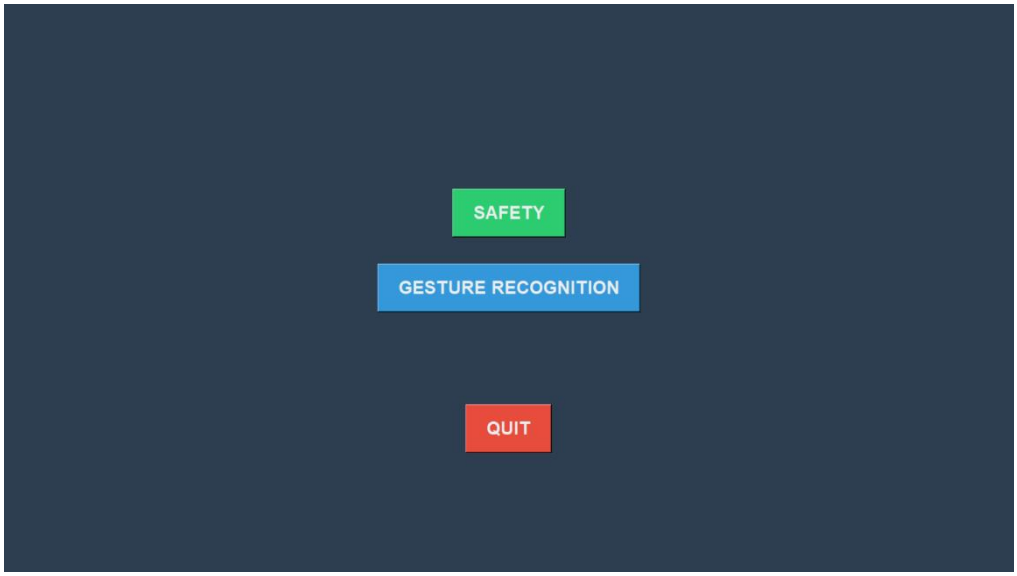
Zbog proceduralnog pristupa programiranju, funkcije su ključne za organizaciju i ponovnu upotrebu kôda, što doprinosi jasnijoj strukturi i efikasnom upravljanju programom. Neke funkcije imaju važne uloge poput slanja naredbi robotu ili promjene aktivnog algoritma, dok druge obavljaju pomoćne zadatke poput računanja FPS-a ili crtanja linija koje povezuju detektirane točke na slici. Nabrojane su funkcije čije su uloge ključne u izvršavanju programa:

- `robot_connection()` – čeka uspješno povezivanje računala s robotom i pokreće funkciju `start()`
- `start()` – kreira glavni prozor GUI aplikacije, dodaje dva gumba u sučelje od kojih jedan pokreće funkciju `safety()`, a drugi `gesture_recognition()`
- `safety()` – aktivira *RealSense* kameru, pokreće algoritam za procjenu namjera kretanja čovjeka
- `set_speed(brzina)` – određuje izlaznu brzinu robota na temelju postavljene inicijalne brzine
- `gesture_recognition()` – aktivira *web* kameru računala, pokreće algoritam za upravljanje robotom pomoću gesti
- `gesture_speed(gesta)` – određuje izlaznu brzinu robota na temelju prepoznate geste
- `send_speed(brzina)` – šalje naredbu robotu za promjenu brzine
- `camera_check()` – provjerava je li *RealSense* kamera spojena na računalo
- `update_frame()` – osvježava i ažurira glavni prozor GUI aplikacije
- `destroy_all_widgets(root)` – uklanja sve grafičke elemente (eng. *widget*) sa sučelja
- `quit()` – oslobađa resurse povezane s *MediaPipe*-ovim modelima, isključuje aktivnu kameru, zatvara sve prozore i prekida vezu s robotom, čime omogućuje siguran završetak programa

6.3. Grafičko korisničko sučelje (GUI)

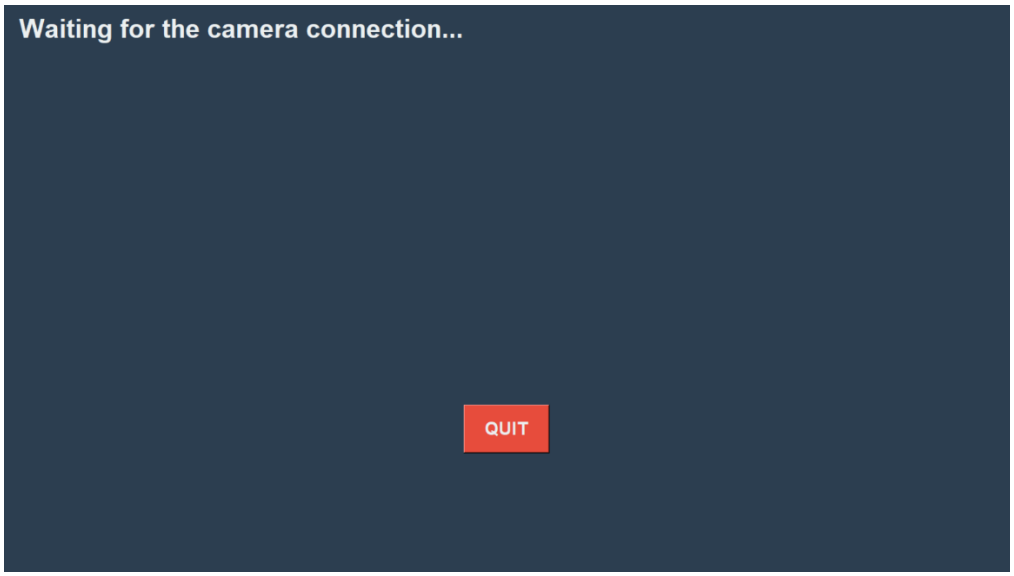
Kako bi se korisnicima omogućila jednostavna i intuitivna interakcija s računalom putem vizualnih elemenata i radi lakšeg pristupanja funkcijama programa, uz pomoću biblioteke *tkinter* kreirano je grafičko korisničko sučelje (GUI) za upravljanje načinom rada sustava. Njegova je uloga da korisniku u svakom trenutku pruža informacije o potrebnim koracima za pokretanje algoritama, trenutnoj brzini robota kada je algoritam aktivan te nudi opciju za prekidanje cijelog programa.

Kada je veza s robotom uspostavljena, kreira se glavni prozor i prikazuje početni zaslon na kojem se nalaze gumbi *SAFETY*, čijim se pritiskom pokreće algoritam za procjenu namjera kretanja čovjeka, i *GESTURE RECOGNTION*, kojim se pokreće algoritam za upravljanje robotom pomoću gesti. Gumb *QUIT* prisutan je tijekom cijelog trajanja programa, a pritiskom na njega u bilo kojem trenutku prekida se izvođenje programa.



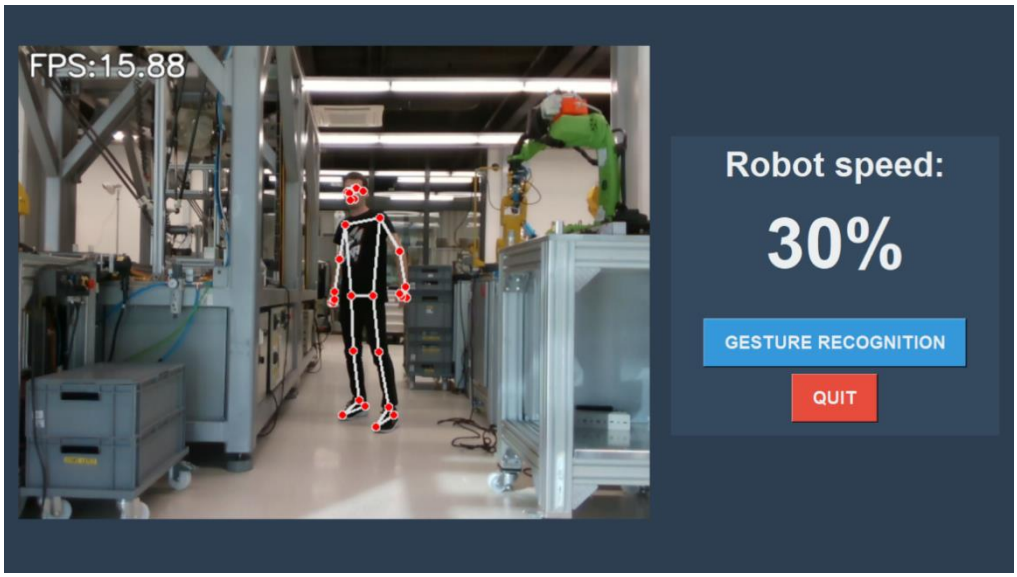
Slika 27. Početni zaslon

Pokretanjem algoritma za procjenu namjera kretanja čovjeka, program najprije provjerava je li *RealSense* kamera spojena s računalom. U slučaju da nije, pojavljuje se zaslon s porukom *„Waiting for the camera connection…“* („Čekanje na povezivanje kamere…).



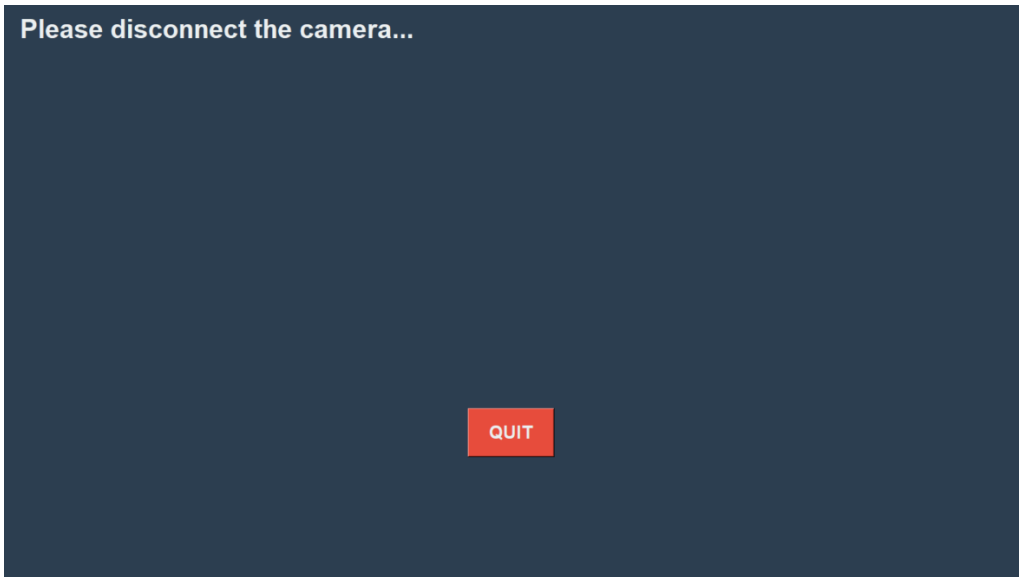
Slika 28. Poruka za povezivanje kamere

Nakon uspješnog spajanja i aktivacije, pokreće se algoritam i otvara zaslon na kojem se prikazuje slika s kamere u stvarnom vremenu, trenutna brzina robota u postocima te gumbi *GESTURE RECOGNITION* za promjenu aktivnog algoritma i *QUIT*.



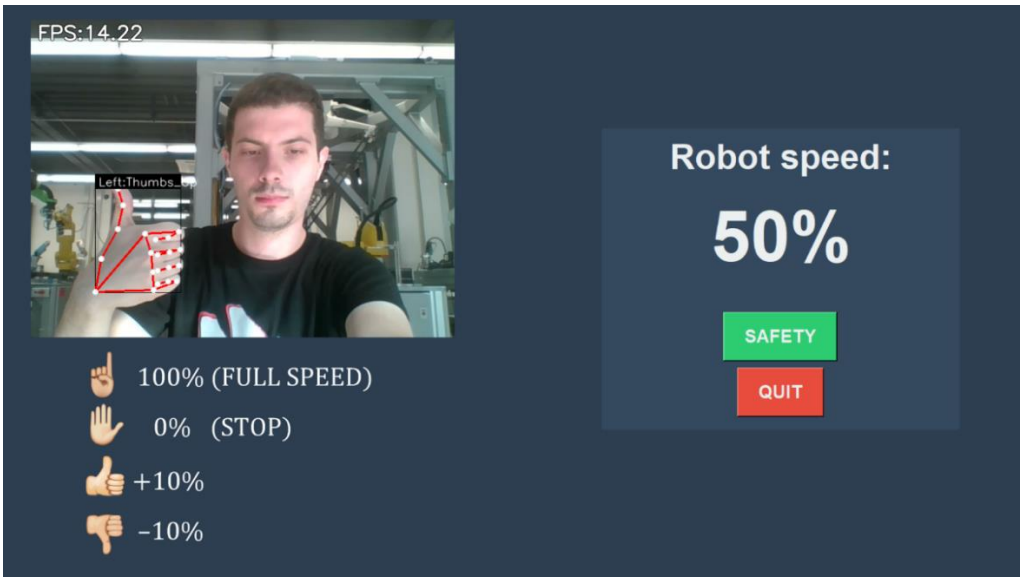
Slika 29. Zaslon tijekom izvođenja algoritma za procjenu namjera kretanja čovjeka

Pritiskom gumba *GESTURE RECOGNTION*, prije pokretanja algoritma za upravljanje robotom pomoću gesti program ponovno provjerava je li *RealSense* kamera spojena s računalom te, ako jest, na zaslonu se ispisuje poruka *„Please disconnect the camera…“* („Molim odspojite kameru…“).



Slika 30. Poruka za odspajanje kamere

Kada je *RealSense* kamera odspojena i *web* kamera uspješno aktivirana, počinje izvođenje algoritma i otvara se zaslon sa sličnim rasporedom grafičkih elemenata kao i u drugom algoritmu. Prikazuju se trenutna brzina robota, gumbi *SAFETY* za promjenu aktivnog algoritma i *QUIT*, slika s kamere u stvarnom vremenu, a ispod nje navedene su naredbe pridružene svakoj od gesti na temelju kojih algoritam prilagođava svoju brzinu.



Slika 31. Zaslon tijekom izvođenja algoritma za upravljanje robotom pomoću gesti

Cijeli program izvodi se u beskonačnoj petlji do trenutka kada ga korisnik ne prekine pritiskom gumba *QUIT*.

7. ZAKLJUČAK

U završnom radu objašnjen je proces odabira optimalne HPE biblioteke te su osmišljeni, koncipirani i razvijeni algoritmi za procjenu namjera kretanja čovjeka i upravljanje robotom pomoću gesti, uz koje je također izrađen vlastiti model strojnog učenja za prepoznavanje specifičnih gesti pomoću šake i uspoređen s postojećim javno dostupnim modelima. Konačno, ti su algoritmi integrirani i implementirani na robotu s ciljem prilagođavanja njegovog ponašanja, tj. brzine, na temelju čovjekovih namjera kretanja, odnosno gestikulacija.

Glavni tehnički izazov u radu jest integracija različitih robotičkih i informatičkih disciplina poput interakcije čovjeka i robota, strojnog učenja i vizijskih sustava, čime se očituje njegova kompleksnost. Algoritmi moraju biti robusni, prilagoditi se ograničenjima u integraciji na robotu te ispravno reagirati na svaku čovjekovu radnju, što zahtjeva razmišljanje izvan okvira, pažljivo promatranje ponašanja čovjeka u prirodnim uvjetima te višestruko testiranje i doradu algoritama.

Nastavak rada na ovom području može uključivati unaprjeđenje sustava korištenjem dviju ili više kamera, dinamičkim prilagođavanjem drugih parametara ponašanja robota osim brzine, detekcijom i praćenjem više od jedne osobe ili prepoznavanjem gesti koje uključuju obje šake. Dodatnim povećanjem robusnosti algoritama i sigurnosti tijekom njihovog izvođenja, sustav bi se mogao implementirati i na tradicionalnim industrijskim robotima. Također, dodatna istraživanja mogu se fokusirati na primjenu ovih tehnologija u različitim industrijama, kao što su proizvodnja i montaža, s ciljem povećanja njihove učinkovitosti i sigurnosti.

LITERATURA

[1] Chung J-L, Ong L-Y, Leow M-C. Comparative Analysis of Skeleton-Based Human Pose Estimation. 2022. https://www.mdpi.com/1999-5903/14/12/380, pristupljeno 18.08.2024.

[2] Hassan HA, Abdallah BH, Abdallah AA, Abdel-Aal RO, Numan RR, Darwish AK, El-Behaidy WH. Automatic Feedback For Physiotherapy Exercises Based On PoseNet. 2020. https://fcihib.journals.ekb.eg/article_116046_4a69938e0cc736a8faae517934669c09.pdf?lang=en, pristupljeno 18.08.2024.

[3] Li Y-C, Chang C-T, Cheng C-C, Huang Y-L. Baseball Swing Pose Estimation Using OpenPose. In: 2021 IEEE International Conference on Robotics, Automation and Artificial Intelligence (RAAI), Hong Kong, 2021., p. 6-9. https://ieeexplore.ieee.org/abstract/document/9507807, pristupljeno 18.08.2024.

[4] Yoo H-R, Lee B-H. An openpose-based child abuse decision system using surveillance video. Journal of the Korea Institute of Information and Communication Engineering, 2019. https://koreascience.kr/article/JAKO201913649329503.page, pristupljeno 18.08.2024.

[5] Google AI Edge, MediaPipe, MediaPipe Solution guide. https://ai.google.dev/edge/mediapipe/solutions/guide, pristupljeno 19.08.2024.

[6] OpenPose Documentation. https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/index.html, pristupljeno 19.08.2024.

[7] Modeling and evaluating beat gestures for social robots - Scientific Figure on ResearchGate. https://www.researchgate.net/figure/Openpose-detected-body-hand-and-face-keypoints_fig2_353668783, pristupljeno 19.08.2024.

[8] Ultralytics YOLO docs. https://docs.ultralytics.com, prisupljeno 20.08.2024.

[9] Medium, YoloV8 Pose Estimation and Pose Keypoint Classification using Neural Net PyTorch. https://alimustoofaa.medium.com/yolov8-pose-estimation-and-pose-keypoint-classification-using-neural-net-pytorch-98469b924525, pristupljeno 20.08.2024.

[10] TensorFlow Hub, Tutorials, MoveNet. https://www.tensorflow.org/hub/tutorials/movenet, pristupljeno 20.08.2024.

[11] GitHub, tensorflow, tfjs-models, pose-detection, README. https://github.com/tensorflow/tfjs-models/blob/master/pose-detection/README.md, pristupljeno 20.08.2024.

[12] Zhang W, Zhu M, Derpanis K. From Actemes to Action: A Strongly-supervised Representation for Detailed Action Understanding. International Conference on Computer Vision (ICCV). 2013. https://dreamdragon.github.io/PennAction/, pristupljeno 20.08.2024.

[13] Universal Robots, Robot arm, Technical specifications. https://www.universal-robots.com/media/1827367/05_2023_collective_data-sheet.pdf, pristupljeno 24.08.2024.

[14] FANUC, Robots, Collaborative robots. https://www.fanuc.eu/uk/en/robots/robot-filter-page/collaborative-robots, pristupljeno 24.08.2024.

[15] TechVidvan, Real-time Hand Gesture Recognition using TensorFlow & OpenCV. https://techvidvan.com/tutorials/hand-gesture-recognition-tensorflow-opencv/, pristupljeno 26.08.2024.

[16] Google AI Edge, MediaPipe, Gesture recognition task guide. https://ai.google.dev/edge/mediapipe/solutions/vision/gesture_recognizer, pristupljeno 26.08.2024.

[17] Github, Kazuhito00, hand-gesture-recognition-using-mediapipe. https://github.com/Kazuhito00/hand-gesture-recognition-using-mediapipe, engleski: https://github.com/kinivi/hand-gesture-recognition-mediapipe, pristupljeno 26.08.2024.

[18] Lisjak D, Kolar D. Predavanja kolegija „Uvod u znanost o podacima“, Modeli rudarenja podataka, FSB, 2024.

[19] FANUC America, CR-15iA Collaborative Robot – Medium Payload Cobot. https://www.fanucamerica.com/products/robots/series/collaborative-robot/cr-15ia-cobot, pristupljeno 06.09.2024.

[20] Intel® RealSense™ Depth Camera D435. https://www.intelrealsense.com/depth-camera-d435/, pristupljeno 07.09.2024.

[21] GitHub, IntelRealSense, librealsense. https://github.com/IntelRealSense/librealsense/releases/tag/v2.55.1, pristupljeno 07.09.2024.

PRILOZI

I. Kôd glavnog programa: https://github.com/MatijaPongracic/ZavrsniRad.git

II. Kôdovi za usporedbu biblioteka: https://github.com/MatijaPongracic/UsporedbaBiblioteka.git

III. Kôdovi za usporedbu modela: https://github.com/MatijaPongracic/UsporedbaModela